ARM® Compiler

Version 6.01

armasm Reference Guide

ARM®

Copyright © 2014 ARM. All rights reserved.
ARM DUI 0802B (ID121814)

ARM Compiler
armasm Reference Guide

Copyright © 2014 ARM. All rights reserved.

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
14 March 2014	A	Non-Confidential	ARM Compiler v6.00 Release
15 December 2014	B	Non-Confidential	ARM Compiler v6.01 Release

Proprietary Notice

Words and logos marked with ™ or ® are registered trademarks or trademarks of ARM® in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Contents
ARM Compiler armasm Reference Guide

Chapter 1 **Conventions and Feedback**

Chapter 2 **armasm Command-line Options**

2.1 armasm command-line syntax 2-2
2.2 armasm command-line options 2-3
2.3 --16 2-5
2.4 --32 2-6
2.5 --apcs 2-7
2.6 --arm 2-9
2.7 --arm_only 2-10
2.8 --bi 2-11
2.9 --bigend 2-12
2.10 --brief_diagnostics 2-13
2.11 --checkreglist 2-14
2.12 --cpu 2-15
2.13 --debug 2-17
2.14 --depend 2-18
2.15 --depend_format 2-19
2.16 --diag_error 2-20
2.17 --diag_remark 2-21
2.18 --diag_style 2-22
2.19 --diag_suppress 2-23
2.20 --diag_warning 2-24
2.21 --dllexport_all 2-25
2.22 --dwarf2 2-26
2.23 --dwarf3 2-27
2.24 --errors 2-28
2.25 --execstack 2-29
2.26 --exceptions 2-30

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.27 --exceptions_unwind ... 2-31
2.28 --fpmode ... 2-32
2.29 --fpu ... 2-33
2.30 -g ... 2-34
2.31 --help ... 2-35
2.32 -i ... 2-36
2.33 --keep ... 2-37
2.34 --length ... 2-38
2.35 --li ... 2-39
2.36 --library_type ... 2-40
2.37 --list ... 2-41
2.38 --littleend ... 2-42
2.39 -m ... 2-43
2.40 --maxcache ... 2-44
2.41 --md ... 2-45
2.42 --no_code_gen ... 2-46
2.43 --no_esc ... 2-47
2.44 --no_execstack ... 2-48
2.45 --no_exceptions ... 2-49
2.46 --no_exceptions_unwind ... 2-50
2.47 --no_hide_all ... 2-51
2.48 --no_reduce_paths ... 2-52
2.49 --no_regs ... 2-53
2.50 --no_terse ... 2-54
2.51 --no_unaligned_access ... 2-55
2.52 --no_warn ... 2-56
2.53 -o ... 2-57
2.54 --pd ... 2-58
2.55 --predefine ... 2-59
2.56 --reduce_paths ... 2-60
2.57 --regnames ... 2-61
2.58 --report-if-not-wysiwyg ... 2-62
2.59 --show_cmdline ... 2-63
2.60 --thumb ... 2-64
2.61 --tool_variant ... 2-65
2.62 --unaligned_access ... 2-66
2.63 --unsafe ... 2-67
2.64 --untyped_local_labels ... 2-68
2.65 --version_number ... 2-69
2.66 --via ... 2-70
2.67 --vsn ... 2-71
2.68 --width ... 2-72
2.69 --xref ... 2-73

Chapter 3 A32 and T32 Instructions

3.1 A32 and T32 instruction summary ... 3-2
3.2 Instruction width specifiers ... 3-8
3.3 Memory access instructions ... 3-9
3.4 General data processing instructions ... 3-11
3.5 Flexible second operand (Operand2) ... 3-12
3.6 Operand2 as a constant ... 3-13
3.7 Operand2 as a register with optional shift ... 3-14
3.8 Shift operations ... 3-15
3.9 Multiply instructions ... 3-18
3.10 Saturating instructions ... 3-19
3.11 Parallel instructions ... 3-20
3.12 Packing and unpacking instructions ... 3-21
3.13 Branch and control instructions ... 3-22
3.14 Coprocessor instructions ... 3-23
3.15 Miscellaneous instructions ... 3-24

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.16 Pseudo-instructions ... 3-25
3.17 Condition codes ... 3-26
3.18 ADD, SUB, RSB, ADC, SBC, and RSC ... 3-27
3.19 ADR (PC-relative) ... 3-32
3.20 ADR (register-relative) ... 3-34
3.21 ADRL pseudo-instruction ... 3-36
3.22 AND, ORR, EOR, BIC, and ORN ... 3-38
3.23 ASR, LSL, LSR, ROR, and RRX ... 3-41
3.24 B, BL, BX, and BLX ... 3-44
3.25 BFC and BFI ... 3-47
3.26 BKPT ... 3-48
3.27 CBZ and CBNZ ... 3-49
3.28 CLREX ... 3-50
3.29 CLZ ... 3-51
3.30 CMP and CMN ... 3-52
3.31 CPS ... 3-54
3.32 CPY pseudo-instruction ... 3-55
3.33 DBG ... 3-56
3.34 DCPS1 (T32 instruction) ... 3-57
3.35 DCPS2 (T32 instruction) ... 3-58
3.36 DCPS3 (T32 instruction) ... 3-59
3.37 DMB, DSB, and ISB ... 3-60
3.38 ERET ... 3-63
3.39 HLT ... 3-64
3.40 HVC ... 3-65
3.41 IT ... 3-66
3.42 LDC and STC ... 3-69
3.43 LDM and STM ... 3-71
3.44 LDR and STR (immediate offset) ... 3-74
3.45 LDR and STR (register offset) ... 3-77
3.46 LDR and STR, unprivileged ... 3-80
3.47 LDR (PC-relative) ... 3-82
3.48 LDR (register-relative) ... 3-84
3.49 LDR pseudo-instruction ... 3-86
3.50 LDA and STL ... 3-89
3.51 LDAEX and STLEX ... 3-91
3.52 LDREX and STREX ... 3-94
3.53 MCR and MCRR ... 3-97
3.54 MOV and MVN ... 3-98
3.55 MOVT ... 3-101
3.56 MOV32 pseudo-instruction ... 3-102
3.57 MRC and MRRC ... 3-103
3.58 MRS (system coprocessor register to ARM register) ... 3-104
3.59 MRS (PSR to general-purpose register) ... 3-105
3.60 MSR (ARM register to system coprocessor register) ... 3-107
3.61 MSR (general-purpose register to PSR) ... 3-108
3.62 MUL, MLA, and MLS ... 3-110
3.63 NEG pseudo-instruction ... 3-112
3.64 NOP ... 3-113
3.65 Parallel add and subtract ... 3-114
3.66 PKHBT and PKHTB ... 3-117
3.67 PLD, PLDW, and PLI ... 3-119
3.68 PUSH and POP ... 3-121
3.69 QADD, QSUB, QDADD, and QDSUB ... 3-123
3.70 REV, REV16, REVSH, and RBIT ... 3-125
3.71 RFE ... 3-127
3.72 SBFX and UBFX ... 3-129
3.73 SDIV and UDIV ... 3-130
3.74 SEL ... 3-131
3.75 SETEND ... 3-133

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.76 SEV, SEVL, WFE, WFI, and YIELD 3-134
3.77 SMC 3-136
3.78 SMLAD and SMLSD 3-137
3.79 SMLALxy 3-139
3.80 SMLALD and SMLSLD 3-140
3.81 SMMUL, SMMLA, and SMMLS 3-142
3.82 SMUAD{X} and SMUSD{X} 3-144
3.83 SMULWy and SMLAWy 3-145
3.84 SMULxy and SMLAxy 3-146
3.85 SRS 3-148
3.86 SSAT and USAT 3-150
3.87 SSAT16 and USAT16 3-152
3.88 SUBS pc, lr 3-154
3.89 SVC 3-156
3.90 SXT, SXTA, UXT, and UXTA 3-157
3.91 SYS 3-159
3.92 TBB and TBH 3-160
3.93 TST and TEQ 3-161
3.94 UMAAL 3-163
3.95 UMULL, UMLAL, SMULL, and SMLAL 3-164
3.96 UND pseudo-instruction 3-166
3.97 USAD8 and USADA8 3-167

Chapter 4 Advanced SIMD and Floating-point Programming (32-bit)

4.1 Advanced SIMD and floating-point instruction summary 4-2
4.2 Shared Advanced SIMD and floating-point instructions 4-7
4.3 Advanced SIMD logical and compare operations 4-8
4.4 Advanced SIMD general data processing instructions 4-9
4.5 Advanced SIMD shift instructions 4-10
4.6 Advanced SIMD general arithmetic instructions 4-11
4.7 Advanced SIMD multiply instructions 4-12
4.8 Advanced SIMD load and store element and structure instructions 4-13
4.9 Interleaving provided by load and store, element and structure instructions 4-14
4.10 Alignment restrictions in load and store, element and structure instructions 4-15
4.11 Advanced SIMD and floating-point pseudo-instructions 4-16
4.12 Floating-point instructions 4-17
4.13 Cryptographic instructions 4-18
4.14 V{Q}{R}SHL (by signed variable) 4-19
4.15 V{Q}ABS and V{Q}NEG 4-20
4.16 V{Q}ADD, VADDL, VADDW, V{Q}SUB, VSUBL, and VSUBW 4-21
4.17 V{R}ADDHN and V{R}SUBHN 4-23
4.18 V{R}HADD and VHSUB 4-24
4.19 V{R}SHR (by immediate) 4-25
4.20 V{R}SHRN (by immediate) 4-26
4.21 V{R}SRA (by immediate) 4-27
4.22 VABA{L} and VABD{L} 4-28
4.23 VABS, VNEG, and VSQRT 4-29
4.24 VACGE and VACGT 4-30
4.25 VACLE and VACLT 4-31
4.26 VADD, VSUB, and VDIV 4-32
4.27 VAND, VBIC, VEOR, VORN, and VORR (register) 4-33
4.28 VAND and VORN (immediate) 4-34
4.29 VBIC and VORR (immediate) 4-35
4.30 VBIF, VBIT, and VBSL 4-36
4.31 VCEQ, VCGE, VCGT, VCLE, and VCLT 4-37
4.32 VCLE and VCLT 4-38
4.33 VCLS, VCLZ, and VCNT 4-39
4.34 VCMP, VCMPE 4-40
4.35 VCVT (between fixed-point or integer, and floating-point) 4-41
4.36 VCVT (from floating-point to integer with directed rounding modes) 4-42

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.37 VCVT (between half-precision and single-precision floating-point) 4-43
4.38 VCVT (between single-precision and double-precision) 4-44
4.39 VCVT (between floating-point and integer) 4-45
4.40 VCVT (from floating-point to integer with directed rounding modes) 4-46
4.41 VCVT (between floating-point and fixed-point) 4-47
4.42 VCVTB, VCVTT (half-precision extension) 4-48
4.43 VCVTB, VCVTT (between half-precision and double-precision) 4-49
4.44 VDUP 4-50
4.45 VEXT 4-51
4.46 VFMA, VFMS 4-52
4.47 VFNMA, VFNMS 4-53
4.48 VLDM, VSTM, VPOP, and VPUSH 4-54
4.49 VLDR and VSTR 4-55
4.50 VLDn and VSTn (single n-element structure to one lane) 4-56
4.51 VLDn (single n-element structure to all lanes) 4-58
4.52 VLDn and VSTn (multiple n-element structures) 4-60
4.53 VLDR pseudo-instruction 4-62
4.54 VLDR and VSTR (post-increment and pre-decrement) 4-63
4.55 VMAX, VMIN, VPMAX, and VPMIN 4-64
4.56 VMAXNM, VMINNM (Advanced SIMD) 4-65
4.57 VMAXNM, VMINNM (floating-point) 4-66
4.58 VMOV 4-67
4.59 VMOV, VMVN (immediate) 4-68
4.60 VMOV, VMVN (register) 4-69
4.61 VMOV (between two ARM registers and one or two extension registers) 4-70
4.62 VMOV (between an ARM register and a scalar) 4-71
4.63 VMOV (between one ARM register and single precision floating-point register) ... 4-72
4.64 VMOV2 4-73
4.65 VMOVL, V{Q}MOVN, VQMOVUN 4-74
4.66 VMRS and VMSR 4-75
4.67 VMUL, VMLA, VMLS, VNMUL, VNMLA, and VNMLS 4-76
4.68 VMUL{L}, VMLA{L}, and VMLS{L} 4-77
4.69 VMUL{L}, VMLA{L}, and VMLS{L} (by scalar) 4-78
4.70 VPADD{L}, VPADAL 4-79
4.71 VQ{R}DMULH (by vector or by scalar) 4-81
4.72 VQ{R}SHR{U}N (by immediate) 4-82
4.73 VQDMULL, VQDMLAL, and VQDMLSL (by vector or by scalar) 4-83
4.74 VRECPE and VRSQRTE 4-84
4.75 VRECPS and VRSQRTS 4-85
4.76 VREV 4-87
4.77 VRINT (Advanced SIMD) 4-88
4.78 VRINT (floating-point) 4-89
4.79 VSEL 4-90
4.80 VSHL, VQSHL, VQSHLU, and VSHLL (by immediate) 4-91
4.81 VSLI and VSRI 4-93
4.82 VSWP 4-95
4.83 VTBL, VTBX 4-96
4.84 VTRN 4-97
4.85 VTST 4-98
4.86 VUZP, VZIP 4-99

Chapter 5 A64 General Instructions

5.1 A64 general instructions in alphabetical order 5-2
5.2 Register restrictions for A64 instructions 5-8
5.3 ADC 5-9
5.4 ADCS 5-10
5.5 ADD (extended register) 5-11
5.6 ADD (immediate) 5-13
5.7 ADD (shifted register) 5-14
5.8 ADDS (extended register) 5-15

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.9 ADDS (immediate) 5-17
5.10 ADDS (shifted register) 5-18
5.11 ADR 5-19
5.12 ADRL pseudo-instruction 5-20
5.13 ADRP 5-21
5.14 AND (immediate) 5-22
5.15 AND (shifted register) 5-23
5.16 ANDS (immediate) 5-24
5.17 ANDS (shifted register) 5-25
5.18 ASR (register) 5-26
5.19 ASR (immediate) 5-27
5.20 ASRV 5-28
5.21 AT 5-29
5.22 B.cond 5-30
5.23 B 5-31
5.24 BFI 5-32
5.25 BFM 5-33
5.26 BFXIL 5-34
5.27 BIC (shifted register) 5-35
5.28 BICS (shifted register) 5-36
5.29 BL 5-37
5.30 BLR 5-38
5.31 BR 5-39
5.32 BRK 5-40
5.33 CBNZ 5-41
5.34 CBZ 5-42
5.35 CCMN (immediate) 5-43
5.36 CCMN (register) 5-44
5.37 CCMP (immediate) 5-45
5.38 CCMP (register) 5-46
5.39 CINC 5-47
5.40 CINV 5-48
5.41 CLREX 5-49
5.42 CLS 5-50
5.43 CLZ 5-51
5.44 CMN (extended register) 5-52
5.45 CMN (immediate) 5-54
5.46 CMN (shifted register) 5-55
5.47 CMP (extended register) 5-56
5.48 CMP (immediate) 5-58
5.49 CMP (shifted register) 5-59
5.50 CNEG 5-60
5.51 CRC32B, CRC32H, CRC32W, CRC32X 5-61
5.52 CRC32CB, CRC32CH, CRC32CW, CRC32CX 5-62
5.53 CSEL 5-63
5.54 CSET 5-64
5.55 CSETM 5-65
5.56 CSINC 5-66
5.57 CSINV 5-67
5.58 CSNEG 5-68
5.59 DC 5-69
5.60 DCPS1 (A64 instruction) 5-70
5.61 DCPS2 (A64 instruction) 5-71
5.62 DCPS3 (A64 instruction) 5-72
5.63 DMB 5-73
5.64 DRPS 5-74
5.65 DSB 5-75
5.66 EON (shifted register) 5-76
5.67 EOR (immediate) 5-77
5.68 EOR (shifted register) 5-78

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.69 ERET 5-79
5.70 EXTR 5-80
5.71 HINT 5-81
5.72 HLT 5-82
5.73 HVC 5-83
5.74 IC 5-84
5.75 ISB 5-85
5.76 LSL (register) 5-86
5.77 LSL (immediate) 5-87
5.78 LSLV 5-88
5.79 LSR (register) 5-89
5.80 LSR (immediate) 5-90
5.81 LSRV 5-91
5.82 MADD 5-92
5.83 MNEG 5-93
5.84 MOV (to or from SP) 5-94
5.85 MOV (inverted wide immediate) 5-95
5.86 MOV (wide immediate) 5-96
5.87 MOV (bitmask immediate) 5-97
5.88 MOV (register) 5-98
5.89 MOVK 5-99
5.90 MOVL pseudo-instruction 5-100
5.91 MOVN 5-102
5.92 MOVZ 5-103
5.93 MRS 5-104
5.94 MSR (immediate) 5-105
5.95 MSR (register) 5-106
5.96 MSUB 5-107
5.97 MUL 5-108
5.98 MVN 5-109
5.99 NEG (shifted register) 5-110
5.100 NEGS 5-111
5.101 NGC 5-112
5.102 NGCS 5-113
5.103 NOP 5-114
5.104 ORN (shifted register) 5-115
5.105 ORR (immediate) 5-116
5.106 ORR (shifted register) 5-117
5.107 RBIT 5-118
5.108 RET 5-119
5.109 REV 5-120
5.110 REV16 5-121
5.111 REV32 5-122
5.112 ROR (immediate) 5-123
5.113 ROR (register) 5-124
5.114 RORV 5-125
5.115 SBC 5-126
5.116 SBCS 5-127
5.117 SBFIZ 5-128
5.118 SBFM 5-129
5.119 SBFX 5-131
5.120 SDIV 5-132
5.121 SEV 5-133
5.122 SEVL 5-134
5.123 SMADDL 5-135
5.124 SMC 5-136
5.125 SMNEGL 5-137
5.126 SMSUBL 5-138
5.127 SMULH 5-139
5.128 SMULL 5-140

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.129 SUB (extended register) 5-141
5.130 SUB (immediate) 5-143
5.131 SUB (shifted register) 5-144
5.132 SUBS (extended register) 5-145
5.133 SUBS (immediate) 5-147
5.134 SUBS (shifted register) 5-148
5.135 SVC 5-149
5.136 SXTB 5-150
5.137 SXTH 5-151
5.138 SXTW 5-152
5.139 SYS 5-153
5.140 SYSL 5-154
5.141 TBNZ 5-155
5.142 TBZ 5-156
5.143 TLBI 5-157
5.144 TST (immediate) 5-158
5.145 TST (shifted register) 5-159
5.146 UBFIZ 5-160
5.147 UBFM 5-161
5.148 UBFX 5-163
5.149 UDIV 5-164
5.150 UMADDL 5-165
5.151 UMNEGL 5-166
5.152 UMSUBL 5-167
5.153 UMULH 5-168
5.154 UMULL 5-169
5.155 UXTB 5-170
5.156 UXTH 5-171
5.157 WFE 5-172
5.158 WFI 5-173
5.159 YIELD 5-174

Chapter 6 A64 Data Transfer Instructions

6.1 A64 data transfer instructions in alphabetical order 6-2
6.2 Register restrictions for A64 instructions 6-5
6.3 LDAR 6-6
6.4 LDARB 6-7
6.5 LDARH 6-8
6.6 LDAXP 6-9
6.7 LDAXR 6-10
6.8 LDAXRB 6-11
6.9 LDAXRH 6-12
6.10 LDNP (SIMD and FP) 6-13
6.11 LDNP 6-14
6.12 LDP (SIMD and FP) 6-15
6.13 LDP 6-17
6.14 LDPSW 6-18
6.15 LDR (immediate, SIMD and FP) 6-19
6.16 LDR (immediate) 6-21
6.17 LDR (literal, SIMD and FP) 6-22
6.18 LDR (literal) 6-23
6.19 LDR pseudo-instruction 6-24
6.20 LDR (register, SIMD and FP) 6-26
6.21 LDR (register) 6-28
6.22 LDRB (immediate) 6-29
6.23 LDRB (register) 6-30
6.24 LDRH (immediate) 6-31
6.25 LDRH (register) 6-32
6.26 LDRSB (immediate) 6-33
6.27 LDRSB (register) 6-34

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.28 LDRSH (immediate) ... 6-35
6.29 LDRSH (register) ... 6-36
6.30 LDRSW (immediate) ... 6-37
6.31 LDRSW (literal) ... 6-38
6.32 LDRSW (register) ... 6-39
6.33 LDTR ... 6-40
6.34 LDTRB ... 6-41
6.35 LDTRH ... 6-42
6.36 LDTRSB ... 6-43
6.37 LDTRSH ... 6-44
6.38 LDTRSW ... 6-45
6.39 LDUR (SIMD and FP) ... 6-46
6.40 LDUR ... 6-47
6.41 LDURB ... 6-48
6.42 LDURH ... 6-49
6.43 LDURSB ... 6-50
6.44 LDURSH ... 6-51
6.45 LDURSW ... 6-52
6.46 LDXP ... 6-53
6.47 LDXR ... 6-54
6.48 LDXRB ... 6-55
6.49 LDXRH ... 6-56
6.50 PRFM (immediate) ... 6-57
6.51 PRFM (literal) ... 6-59
6.52 PRFM (register) ... 6-60
6.53 PRFUM ... 6-62
6.54 STLR ... 6-64
6.55 STLRB ... 6-65
6.56 STLRH ... 6-66
6.57 STLXP ... 6-67
6.58 STLXR ... 6-68
6.59 STLXRB ... 6-69
6.60 STLXRH ... 6-70
6.61 STNP (SIMD and FP) ... 6-71
6.62 STNP ... 6-72
6.63 STP (SIMD and FP) ... 6-73
6.64 STP ... 6-75
6.65 STR (immediate, SIMD and FP) ... 6-76
6.66 STR (immediate) ... 6-78
6.67 STR (register, SIMD and FP) ... 6-79
6.68 STR (register) ... 6-81
6.69 STRB (immediate) ... 6-82
6.70 STRB (register) ... 6-83
6.71 STRH (immediate) ... 6-84
6.72 STRH (register) ... 6-85
6.73 STTR ... 6-86
6.74 STTRB ... 6-87
6.75 STTRH ... 6-88
6.76 STUR (SIMD and FP) ... 6-89
6.77 STUR ... 6-90
6.78 STURB ... 6-91
6.79 STURH ... 6-92
6.80 STXP ... 6-93
6.81 STXR ... 6-94
6.82 STXRB ... 6-95
6.83 STXRH ... 6-96

Chapter 7 A64 Floating-point Instructions

7.1 A64 floating-point instructions in alphabetical order ... 7-2
7.2 FABS (scalar) ... 7-4

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.3 FADD (scalar) 7-5
7.4 FCCMP 7-6
7.5 FCCMPE 7-7
7.6 FCMP 7-8
7.7 FCMPE 7-9
7.8 FCSEL 7-10
7.9 FCVT 7-11
7.10 FCVTAS (scalar) 7-12
7.11 FCVTAU (scalar) 7-13
7.12 FCVTMS (scalar) 7-14
7.13 FCVTMU (scalar) 7-15
7.14 FCVTNS (scalar) 7-16
7.15 FCVTNU (scalar) 7-17
7.16 FCVTPS (scalar) 7-18
7.17 FCVTPU (scalar) 7-19
7.18 FCVTZS (scalar, fixed-point) 7-20
7.19 FCVTZS (scalar, integer) 7-21
7.20 FCVTZU (scalar, fixed-point) 7-22
7.21 FCVTZU (scalar, integer) 7-23
7.22 FDIV (scalar) 7-24
7.23 FMADD 7-25
7.24 FMAX (scalar) 7-26
7.25 FMAXNM (scalar) 7-27
7.26 FMIN (scalar) 7-28
7.27 FMINNM (scalar) 7-29
7.28 FMOV (register) 7-30
7.29 FMOV (general) 7-31
7.30 FMOV (scalar, immediate) 7-32
7.31 FMSUB 7-33
7.32 FMUL (scalar) 7-34
7.33 FNEG (scalar) 7-35
7.34 FNMADD 7-36
7.35 FNMSUB 7-37
7.36 FNMUL 7-38
7.37 FRINTA (scalar) 7-39
7.38 FRINTI (scalar) 7-40
7.39 FRINTM (scalar) 7-41
7.40 FRINTN (scalar) 7-42
7.41 FRINTP (scalar) 7-43
7.42 FRINTX (scalar) 7-44
7.43 FRINTZ (scalar) 7-45
7.44 FSQRT (scalar) 7-46
7.45 FSUB (scalar) 7-47
7.46 SCVTF (scalar, fixed-point) 7-48
7.47 SCVTF (scalar, integer) 7-49
7.48 UCVTF (scalar, fixed-point) 7-50
7.49 UCVTF (scalar, integer) 7-51

Chapter 8 A64 SIMD Scalar Instructions

8.1 A64 SIMD scalar instructions in alphabetical order 8-2
8.2 ABS (scalar) 8-7
8.3 ADD (scalar) 8-8
8.4 ADDP (scalar) 8-9
8.5 CMEQ (scalar, register) 8-10
8.6 CMEQ (scalar, zero) 8-11
8.7 CMGE (scalar, register) 8-12
8.8 CMGE (scalar, zero) 8-13
8.9 CMGT (scalar, register) 8-14
8.10 CMGT (scalar, zero) 8-15
8.11 CMHI (scalar, register) 8-16

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.12 CMHS (scalar, register) 8-17
8.13 CMLE (scalar, zero) 8-18
8.14 CMLT (scalar, zero) 8-19
8.15 CMTST (scalar) 8-20
8.16 DUP (scalar, element) 8-21
8.17 FABD (scalar) 8-22
8.18 FACGE (scalar) 8-23
8.19 FACGT (scalar) 8-24
8.20 FADDP (scalar) 8-25
8.21 FCMEQ (scalar, register) 8-26
8.22 FCMEQ (scalar, zero) 8-27
8.23 FCMGE (scalar, register) 8-28
8.24 FCMGE (scalar, zero) 8-29
8.25 FCMGT (scalar, register) 8-30
8.26 FCMGT (scalar, zero) 8-31
8.27 FCMLE (scalar, zero) 8-32
8.28 FCMLT (scalar, zero) 8-33
8.29 FCVTAS (scalar) 8-34
8.30 FCVTAU (scalar) 8-35
8.31 FCVTMS (scalar) 8-36
8.32 FCVTMU (scalar) 8-37
8.33 FCVTNS (scalar) 8-38
8.34 FCVTNU (scalar) 8-39
8.35 FCVTPS (scalar) 8-40
8.36 FCVTPU (scalar) 8-41
8.37 FCVTXN (scalar) 8-42
8.38 FCVTZS (scalar, fixed-point) 8-43
8.39 FCVTZS (scalar, integer) 8-44
8.40 FCVTZU (scalar, fixed-point) 8-45
8.41 FCVTZU (scalar, integer) 8-46
8.42 FMAXNMP (scalar) 8-47
8.43 FMAXP (scalar) 8-48
8.44 FMINNMP (scalar) 8-49
8.45 FMINP (scalar) 8-50
8.46 FMLA (scalar, by element) 8-51
8.47 FMLS (scalar, by element) 8-52
8.48 FMUL (scalar, by element) 8-53
8.49 FMULX (scalar, by element) 8-54
8.50 FMULX (scalar) 8-55
8.51 FRECPE (scalar) 8-56
8.52 FRECPS (scalar) 8-57
8.53 FRECPX (scalar) 8-58
8.54 FRSQRTE (scalar) 8-59
8.55 FRSQRTS (scalar) 8-60
8.56 MOV (scalar) 8-61
8.57 NEG (scalar) 8-62
8.58 SCVTF (scalar, fixed-point) 8-63
8.59 SCVTF (scalar, integer) 8-64
8.60 SHL (scalar) 8-65
8.61 SLI (scalar) 8-66
8.62 SQABS (scalar) 8-67
8.63 SQADD (scalar) 8-68
8.64 SQDMLAL (scalar, by element) 8-69
8.65 SQDMLAL (scalar) 8-70
8.66 SQDMLSL (scalar, by element) 8-71
8.67 SQDMLSL (scalar) 8-72
8.68 SQDMULH (scalar, by element) 8-73
8.69 SQDMULH (scalar) 8-74
8.70 SQDMULL (scalar, by element) 8-75
8.71 SQDMULL (scalar) 8-76

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.72 SQNEG (scalar) 8-77
8.73 SQRDMULH (scalar, by element) 8-78
8.74 SQRDMULH (scalar) 8-79
8.75 SQRSHL (scalar) 8-80
8.76 SQRSHRN (scalar) 8-81
8.77 SQRSHRUN (scalar) 8-82
8.78 SQSHL (scalar, immediate) 8-83
8.79 SQSHL (scalar, register) 8-84
8.80 SQSHLU (scalar) 8-85
8.81 SQSHRN (scalar) 8-86
8.82 SQSHRUN (scalar) 8-87
8.83 SQSUB (scalar) 8-88
8.84 SQXTN (scalar) 8-89
8.85 SQXTUN (scalar) 8-90
8.86 SRI (scalar) 8-91
8.87 SRSHL (scalar) 8-92
8.88 SRSHR (scalar) 8-93
8.89 SRSRA (scalar) 8-94
8.90 SSHL (scalar) 8-95
8.91 SSHR (scalar) 8-96
8.92 SSRA (scalar) 8-97
8.93 SUB (scalar) 8-98
8.94 SUQADD (scalar) 8-99
8.95 UCVTF (scalar, fixed-point) 8-100
8.96 UCVTF (scalar, integer) 8-101
8.97 UQADD (scalar) 8-102
8.98 UQRSHL (scalar) 8-103
8.99 UQRSHRN (scalar) 8-104
8.100 UQSHL (scalar, immediate) 8-105
8.101 UQSHL (scalar, register) 8-106
8.102 UQSHRN (scalar) 8-107
8.103 UQSUB (scalar) 8-108
8.104 UQXTN (scalar) 8-109
8.105 URSHL (scalar) 8-110
8.106 URSHR (scalar) 8-111
8.107 URSRA (scalar) 8-112
8.108 USHL (scalar) 8-113
8.109 USHR (scalar) 8-114
8.110 USQADD (scalar) 8-115
8.111 USRA (scalar) 8-116

Chapter 9 A64 SIMD Vector Instructions

9.1 A64 SIMD vector instructions in alphabetical order 9-2
9.2 ABS (vector) 9-12
9.3 ADD (vector) 9-13
9.4 ADDHN, ADDHN2 (vector) 9-14
9.5 ADDP (vector) 9-15
9.6 ADDV (vector) 9-16
9.7 AND (vector) 9-17
9.8 BIC (vector, immediate) 9-18
9.9 BIC (vector, register) 9-19
9.10 BIF (vector) 9-20
9.11 BIT (vector) 9-21
9.12 BSL (vector) 9-22
9.13 CLS (vector) 9-23
9.14 CLZ (vector) 9-24
9.15 CMEQ (vector, register) 9-25
9.16 CMEQ (vector, zero) 9-26
9.17 CMGE (vector, register) 9-27
9.18 CMGE (vector, zero) 9-28

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.19 CMGT (vector, register) ... 9-29
9.20 CMGT (vector, zero) ... 9-30
9.21 CMHI (vector, register) ... 9-31
9.22 CMHS (vector, register) ... 9-32
9.23 CMLE (vector, zero) ... 9-33
9.24 CMLT (vector, zero) ... 9-34
9.25 CMTST (vector) ... 9-35
9.26 CNT (vector) ... 9-36
9.27 DUP (vector, element) ... 9-37
9.28 DUP (vector, general) ... 9-38
9.29 EOR (vector) ... 9-39
9.30 EXT (vector) ... 9-40
9.31 FABD (vector) ... 9-41
9.32 FABS (vector) ... 9-42
9.33 FACGE (vector) ... 9-43
9.34 FACGT (vector) ... 9-44
9.35 FADD (vector) ... 9-45
9.36 FADDP (vector) ... 9-46
9.37 FCMEQ (vector, register) ... 9-47
9.38 FCMEQ (vector, zero) ... 9-48
9.39 FCMGE (vector, register) ... 9-49
9.40 FCMGE (vector, zero) ... 9-50
9.41 FCMGT (vector, register) ... 9-51
9.42 FCMGT (vector, zero) ... 9-52
9.43 FCMLE (vector, zero) ... 9-53
9.44 FCMLT (vector, zero) ... 9-54
9.45 FCVTAS (vector) ... 9-55
9.46 FCVTAU (vector) ... 9-56
9.47 FCVTL, FCVTL2 (vector) ... 9-57
9.48 FCVTMS (vector) ... 9-58
9.49 FCVTMU (vector) ... 9-59
9.50 FCVTN, FCVTN2 (vector) ... 9-60
9.51 FCVTNS (vector) ... 9-61
9.52 FCVTNU (vector) ... 9-62
9.53 FCVTPS (vector) ... 9-63
9.54 FCVTPU (vector) ... 9-64
9.55 FCVTXN, FCVTXN2 (vector) ... 9-65
9.56 FCVTZS (vector, fixed-point) ... 9-66
9.57 FCVTZS (vector, integer) ... 9-67
9.58 FCVTZU (vector, fixed-point) ... 9-68
9.59 FCVTZU (vector, integer) ... 9-69
9.60 FDIV (vector) ... 9-70
9.61 FMAX (vector) ... 9-71
9.62 FMAXNM (vector) ... 9-72
9.63 FMAXNMP (vector) ... 9-73
9.64 FMAXNMV (vector) ... 9-74
9.65 FMAXP (vector) ... 9-75
9.66 FMAXV (vector) ... 9-76
9.67 FMIN (vector) ... 9-77
9.68 FMINNM (vector) ... 9-78
9.69 FMINNMP (vector) ... 9-79
9.70 FMINNMV (vector) ... 9-80
9.71 FMINP (vector) ... 9-81
9.72 FMINV (vector) ... 9-82
9.73 FMLA (vector, by element) ... 9-83
9.74 FMLA (vector) ... 9-84
9.75 FMLS (vector, by element) ... 9-85
9.76 FMLS (vector) ... 9-86
9.77 FMOV (vector, immediate) ... 9-87
9.78 FMUL (vector, by element) ... 9-88

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.79 FMUL (vector) 9-89
9.80 FMULX (vector, by element) 9-90
9.81 FMULX (vector) 9-91
9.82 FNEG (vector) 9-92
9.83 FRECPE (vector) 9-93
9.84 FRECPS (vector) 9-94
9.85 FRINTA (vector) 9-95
9.86 FRINTI (vector) 9-96
9.87 FRINTM (vector) 9-97
9.88 FRINTN (vector) 9-98
9.89 FRINTP (vector) 9-99
9.90 FRINTX (vector) 9-100
9.91 FRINTZ (vector) 9-101
9.92 FRSQRTE (vector) 9-102
9.93 FRSQRTS (vector) 9-103
9.94 FSQRT (vector) 9-104
9.95 FSUB (vector) 9-105
9.96 INS (vector, element) 9-106
9.97 INS (vector, general) 9-107
9.98 LD1 (vector, multiple structures) 9-108
9.99 LD1 (vector, single structure) 9-111
9.100 LD1R (vector) 9-112
9.101 LD2 (vector, multiple structures) 9-113
9.102 LD2 (vector, single structure) 9-114
9.103 LD2R (vector) 9-115
9.104 LD3 (vector, multiple structures) 9-117
9.105 LD3 (vector, single structure) 9-118
9.106 LD3R (vector) 9-120
9.107 LD4 (vector, multiple structures) 9-122
9.108 LD4 (vector, single structure) 9-123
9.109 LD4R (vector) 9-125
9.110 MLA (vector, by element) 9-127
9.111 MLA (vector) 9-128
9.112 MLS (vector, by element) 9-129
9.113 MLS (vector) 9-130
9.114 MOV (vector, element) 9-131
9.115 MOV (vector, from general) 9-132
9.116 MOV (vector) 9-133
9.117 MOV (vector, to general) 9-134
9.118 MOVI (vector) 9-135
9.119 MUL (vector, by element) 9-136
9.120 MUL (vector) 9-137
9.121 MVN (vector) 9-138
9.122 MVNI (vector) 9-139
9.123 NEG (vector) 9-140
9.124 NOT (vector) 9-141
9.125 ORN (vector) 9-142
9.126 ORR (vector, immediate) 9-143
9.127 ORR (vector, register) 9-144
9.128 PMUL (vector) 9-145
9.129 PMULL, PMULL2 (vector) 9-146
9.130 RADDHN, RADDHN2 (vector) 9-147
9.131 RBIT (vector) 9-148
9.132 REV16 (vector) 9-149
9.133 REV32 (vector) 9-150
9.134 REV64 (vector) 9-151
9.135 RSHRN, RSHRN2 (vector) 9-152
9.136 RSUBHN, RSUBHN2 (vector) 9-153
9.137 SABA (vector) 9-154
9.138 SABAL, SABAL2 (vector) 9-155

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.139 SABD (vector) 9-156
9.140 SABDL, SABDL2 (vector) 9-157
9.141 SADALP (vector) 9-158
9.142 SADDL, SADDL2 (vector) 9-159
9.143 SADDLP (vector) 9-160
9.144 SADDLV (vector) 9-161
9.145 SADDW, SADDW2 (vector) 9-162
9.146 SCVTF (vector, fixed-point) 9-163
9.147 SCVTF (vector, integer) 9-164
9.148 SHADD (vector) 9-165
9.149 SHL (vector) 9-166
9.150 SHLL, SHLL2 (vector) 9-167
9.151 SHRN, SHRN2 (vector) 9-168
9.152 SHSUB (vector) 9-169
9.153 SLI (vector) 9-170
9.154 SMAX (vector) 9-171
9.155 SMAXP (vector) 9-172
9.156 SMAXV (vector) 9-173
9.157 SMIN (vector) 9-174
9.158 SMINP (vector) 9-175
9.159 SMINV (vector) 9-176
9.160 SMLAL, SMLAL2 (vector, by element) 9-177
9.161 SMLAL, SMLAL2 (vector) 9-178
9.162 SMLSL, SMLSL2 (vector, by element) 9-179
9.163 SMLSL, SMLSL2 (vector) 9-180
9.164 SMOV (vector) 9-181
9.165 SMULL, SMULL2 (vector, by element) 9-182
9.166 SMULL, SMULL2 (vector) 9-183
9.167 SQABS (vector) 9-184
9.168 SQADD (vector) 9-185
9.169 SQDMLAL, SQDMLAL2 (vector, by element) 9-186
9.170 SQDMLAL, SQDMLAL2 (vector) 9-187
9.171 SQDMLSL, SQDMLSL2 (vector, by element) 9-188
9.172 SQDMLSL, SQDMLSL2 (vector) 9-189
9.173 SQDMULH (vector, by element) 9-190
9.174 SQDMULH (vector) 9-191
9.175 SQDMULL, SQDMULL2 (vector, by element) 9-192
9.176 SQDMULL, SQDMULL2 (vector) 9-193
9.177 SQNEG (vector) 9-194
9.178 SQRDMULH (vector, by element) 9-195
9.179 SQRDMULH (vector) 9-196
9.180 SQRSHL (vector) 9-197
9.181 SQRSHRN, SQRSHRN2 (vector) 9-198
9.182 SQRSHRUN, SQRSHRUN2 (vector) 9-199
9.183 SQSHL (vector, immediate) 9-200
9.184 SQSHL (vector, register) 9-201
9.185 SQSHLU (vector) 9-202
9.186 SQSHRN, SQSHRN2 (vector) 9-203
9.187 SQSHRUN, SQSHRUN2 (vector) 9-204
9.188 SQSUB (vector) 9-205
9.189 SQXTN, SQXTN2 (vector) 9-206
9.190 SQXTUN, SQXTUN2 (vector) 9-207
9.191 SRHADD (vector) 9-208
9.192 SRI (vector) 9-209
9.193 SRSHL (vector) 9-210
9.194 SRSHR (vector) 9-211
9.195 SRSRA (vector) 9-212
9.196 SSHL (vector) 9-213
9.197 SSHLL, SSHLL2 (vector) 9-214
9.198 SSHR (vector) 9-215

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.199 SSRA (vector) 9-216
9.200 SSUBL, SSUBL2 (vector) 9-217
9.201 SSUBW, SSUBW2 (vector) 9-218
9.202 ST1 (vector, multiple structures) 9-219
9.203 ST1 (vector, single structure) 9-222
9.204 ST2 (vector, multiple structures) 9-223
9.205 ST2 (vector, single structure) 9-224
9.206 ST3 (vector, multiple structures) 9-225
9.207 ST3 (vector, single structure) 9-226
9.208 ST4 (vector, multiple structures) 9-227
9.209 ST4 (vector, single structure) 9-228
9.210 SUB (vector) 9-230
9.211 SUBHN, SUBHN2 (vector) 9-231
9.212 SUQADD (vector) 9-232
9.213 SXTL, SXTL2 (vector) 9-233
9.214 TBL (vector) 9-234
9.215 TBX (vector) 9-235
9.216 TRN1 (vector) 9-236
9.217 TRN2 (vector) 9-237
9.218 UABA (vector) 9-238
9.219 UABAL, UABAL2 (vector) 9-239
9.220 UABD (vector) 9-240
9.221 UABDL, UABDL2 (vector) 9-241
9.222 UADALP (vector) 9-242
9.223 UADDL, UADDL2 (vector) 9-243
9.224 UADDLP (vector) 9-244
9.225 UADDLV (vector) 9-245
9.226 UADDW, UADDW2 (vector) 9-246
9.227 UCVTF (vector, fixed-point) 9-247
9.228 UCVTF (vector, integer) 9-248
9.229 UHADD (vector) 9-249
9.230 UHSUB (vector) 9-250
9.231 UMAX (vector) 9-251
9.232 UMAXP (vector) 9-252
9.233 UMAXV (vector) 9-253
9.234 UMIN (vector) 9-254
9.235 UMINP (vector) 9-255
9.236 UMINV (vector) 9-256
9.237 UMLAL, UMLAL2 (vector, by element) 9-257
9.238 UMLAL, UMLAL2 (vector) 9-258
9.239 UMLSL, UMLSL2 (vector, by element) 9-259
9.240 UMLSL, UMLSL2 (vector) 9-260
9.241 UMOV (vector) 9-261
9.242 UMULL, UMULL2 (vector, by element) 9-262
9.243 UMULL, UMULL2 (vector) 9-263
9.244 UQADD (vector) 9-264
9.245 UQRSHL (vector) 9-265
9.246 UQRSHRN, UQRSHRN2 (vector) 9-266
9.247 UQSHL (vector, immediate) 9-267
9.248 UQSHL (vector, register) 9-268
9.249 UQSHRN, UQSHRN2 (vector) 9-269
9.250 UQSUB (vector) 9-270
9.251 UQXTN, UQXTN2 (vector) 9-271
9.252 URECPE (vector) 9-272
9.253 URHADD (vector) 9-273
9.254 URSHL (vector) 9-274
9.255 URSHR (vector) 9-275
9.256 URSQRTE (vector) 9-276
9.257 URSRA (vector) 9-277
9.258 USHL (vector) 9-278

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.259 USHLL, USHLL2 (vector) 9-279
9.260 USHR (vector) 9-280
9.261 USQADD (vector) 9-281
9.262 USRA (vector) 9-282
9.263 USUBL, USUBL2 (vector) 9-283
9.264 USUBW, USUBW2 (vector) 9-284
9.265 UXTL, UXTL2 (vector) 9-285
9.266 UZP1 (vector) 9-286
9.267 UZP2 (vector) 9-287
9.268 XTN, XTN2 (vector) 9-288
9.269 ZIP1 (vector) 9-289
9.270 ZIP2 (vector) 9-290

Chapter 10 Directives Reference

10.1 Alphabetical list of directives 10-2
10.2 Symbol definition directives 10-3
10.3 Data definition directives 10-4
10.4 About assembly control directives 10-5
10.5 About frame directives 10-6
10.6 Reporting directives 10-7
10.7 Instruction set and syntax selection directives 10-8
10.8 Miscellaneous directives 10-9
10.9 ALIAS 10-10
10.10 ALIGN 10-11
10.11 AREA 10-13
10.12 ARM, THUMB, CODE16 and CODE32 10-16
10.13 ASSERT 10-17
10.14 ATTR 10-18
10.15 CN 10-19
10.16 COMMON 10-20
10.17 CP 10-21
10.18 DATA 10-22
10.19 DCB 10-23
10.20 DCD and DCDU 10-24
10.21 DCDO 10-25
10.22 DCFD and DCFDU 10-26
10.23 DCFS and DCFSU 10-27
10.24 DCI 10-28
10.25 DCO and DCOU 10-29
10.26 DCQ and DCQU 10-30
10.27 DCW and DCWU 10-31
10.28 END 10-32
10.29 ENDFUNC or ENDP 10-33
10.30 ENTRY 10-34
10.31 EQU 10-35
10.32 EXPORT or GLOBAL 10-36
10.33 EXPORTAS 10-38
10.34 FIELD 10-39
10.35 FRAME ADDRESS 10-41
10.36 FRAME POP 10-42
10.37 FRAME PUSH 10-43
10.38 FRAME REGISTER 10-44
10.39 FRAME RESTORE 10-45
10.40 FRAME RETURN ADDRESS 10-46
10.41 FRAME SAVE 10-47
10.42 FRAME STATE REMEMBER 10-48
10.43 FRAME STATE RESTORE 10-49
10.44 FRAME UNWIND ON 10-50
10.45 FRAME UNWIND OFF 10-51
10.46 FUNCTION or PROC 10-52

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.47 GBLA, GBLL, and GBLS 10-54
10.48 GET or INCLUDE 10-56
10.49 IMPORT and EXTERN 10-57
10.50 INCBIN 10-59
10.51 IF, ELSE, ENDIF, and ELIF 10-60
10.52 INFO 10-62
10.53 KEEP 10-63
10.54 LCLA, LCLL, and LCLS 10-64
10.55 LTORG 10-65
10.56 MACRO and MEND 10-66
10.57 MAP 10-69
10.58 MEXIT 10-70
10.59 NOFP 10-71
10.60 OPT 10-72
10.61 QN, DN, and SN 10-74
10.62 RELOC 10-76
10.63 REQUIRE 10-77
10.64 REQUIRE8 and PRESERVE8 10-78
10.65 RLIST 10-80
10.66 RN 10-81
10.67 ROUT 10-82
10.68 SETA, SETL, and SETS 10-83
10.69 SPACE or FILL 10-85
10.70 TTL and SUBT 10-86
10.71 WHILE and WEND 10-87
10.72 WN and XN 10-88

Appendix A Via File Syntax

A.1 Overview of via files A-2
A.2 Via file syntax A-3

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Chapter 1
Conventions and Feedback

The following describes the typographical conventions and how to give feedback:

Typographical conventions

The following typographical conventions are used:

`monospace` Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.

`mono`space Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.

`monospace italic`

Denotes arguments to commands and functions where the argument is to be replaced by a specific value.

`monospace bold`

Denotes language keywords when used outside example code.

italic Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

bold Highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate, and for ARM® processor signal names.

Feedback on this product

If you have any comments and suggestions about this product, contact your supplier and give:

- Your name and company.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

- The serial number of the product.
- Details of the release you are using.
- Details of the platform you are using, such as the hardware platform, operating system type and version.
- A small standalone sample of code that reproduces the problem.
- A clear explanation of what you expected to happen, and what actually happened.
- The commands you used, including any command-line options.
- Sample output illustrating the problem.
- The version string of the tools, including the version number and build numbers.

Feedback on content

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- The title.
- The number, ARM DUI 0802B.
- If viewing online, the topic names to which your comments apply.
- If viewing a PDF version of a document, the page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

ARM periodically provides updates and corrections to its documentation on the ARM Information Center, together with knowledge articles and *Frequently Asked Questions* (FAQs).

Other information

- ARM Information Center `http://infocenter.arm.com/help/index.jsp`.
- ARM Technical Support Knowledge Articles `http://infocenter.arm.com/help/topic/com.arm.doc.faqs/index.html`.
- ARM Support and Maintenance `http://www.arm.com/support/services/support-maintenance.php`.
- ARM Glossary `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Chapter 2
armasm Command-line Options

The following topics describe the `armasm` command-line syntax and command-line options:

- *armasm command-line syntax* on page 2-2.
- *armasm command-line options* on page 2-3.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.1 armasm command-line syntax

The command for invoking `armasm` is:

`armasm {options} {inputfile}`

where:

options are commands to the assembler. You can invoke the assembler with any combination of options separated by spaces. You can specify values for some options. To specify a value for an option, use either '=' (*option*=*value*) or a space character (*option value*).

inputfile can be one or more assembly source files separated by spaces. Input files must be A64, UAL, or pre-UAL A32 or T32 assembly language source files.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.2 armasm command-line options

`armasm` supports the following command-line options:

- *--16* on page 2-5.
- *--32* on page 2-6.
- *--apcs* on page 2-7.
- *--arm* on page 2-9.
- *--arm_only* on page 2-10.
- *--bi* on page 2-11.
- *--bigend* on page 2-12.
- *--brief_diagnostics* on page 2-13.
- *--checkreglist* on page 2-14.
- *--cpu* on page 2-15.
- *--debug* on page 2-17.
- *--depend* on page 2-18.
- *--depend_format* on page 2-19.
- *--diag_error* on page 2-20.
- *--diag_remark* on page 2-21.
- *--diag_style* on page 2-22.
- *--diag_suppress* on page 2-23.
- *--diag_warning* on page 2-24.
- *--dllexport_all* on page 2-25.
- *--dwarf2* on page 2-26.
- *--dwarf3* on page 2-27.
- *--errors* on page 2-28.
- *--execstack* on page 2-29.
- *--exceptions* on page 2-30.
- *--exceptions_unwind* on page 2-31.
- *--fpmode* on page 2-32.
- *--fpu* on page 2-33.
- *-g* on page 2-34.
- *--help* on page 2-35.
- *-i* on page 2-36.
- *--keep* on page 2-37.
- *--length* on page 2-38.
- *--li* on page 2-39.
- *--library_type* on page 2-40
- *--list* on page 2-41.
- *--littleend* on page 2-42.
- *-m* on page 2-43.
- *--maxcache* on page 2-44.
- *--md* on page 2-45.
- *--no_code_gen* on page 2-46.
- *--no_esc* on page 2-47.
- *--no_execstack* on page 2-48.
- *--no_exceptions* on page 2-49.
- *--no_exceptions_unwind* on page 2-50.
- *--no_hide_all* on page 2-51.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

- *--no_reduce_paths* on page 2-52.
- *--no_regs* on page 2-53.
- *--no_terse* on page 2-54.
- *--no_unaligned_access* on page 2-55.
- *--no_warn* on page 2-56.
- *-o* on page 2-57.
- *--pd* on page 2-58.
- *--predefine* on page 2-59.
- *--reduce_paths* on page 2-60.
- *--regnames* on page 2-61.
- *--report-if-not-wysiwyg* on page 2-62.
- *--show_cmdline* on page 2-63.
- *--thumb* on page 2-64.
- *--unaligned_access* on page 2-66.
- *--unsafe* on page 2-67.
- *--untyped_local_labels* on page 2-68.
- *--version_number* on page 2-69.
- *--via* on page 2-70.
- *--vsn* on page 2-71.
- *--width* on page 2-72.
- *--xref* on page 2-73.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.3 --16

This option instructs `armasm` to interpret instructions as T32 instructions using the pre-UAL syntax. This is equivalent to a `CODE16` directive at the head of the source file. Use the `--thumb` option to specify T32 instructions using the UAL syntax.

Note

This option is not supported for AArch64 state.

2.3.1 See also

- *--thumb* on page 2-64.
- *ARM, THUMB, CODE16 and CODE32* on page 10-16.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.4 --32

This option is a synonym for `--arm`.

Note

This option is not supported for AArch64 state.

2.4.1 See also

- *--arm* on page 2-9.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.5 --apcs

This option specifies whether you are using the *Procedure Call Standard for the ARM Architecture* (AAPCS). It can also specify some attributes of code sections.

The AAPCS forms part of the *Base Standard Application Binary Interface for the ARM Architecture* (BSABI) specification. By writing code that adheres to the AAPCS, you can ensure that separately compiled and assembled modules can work together.

Note

AAPCS qualifiers do not affect the code produced by `armasm`. They are an assertion by the programmer that the code in *`inputfile`* complies with a particular variant of AAPCS. They cause attributes to be set in the object file produced by `armasm`. The linker uses these attributes to check compatibility of files, and to select appropriate library variants.

2.5.1 Syntax

`--apcs=`*`qualifier…qualifier`*

Values for *`qualifier`* are:

`none`
: Specifies that *`inputfile`* does not use AAPCS. AAPCS registers are not set up. Other qualifiers are not permitted if you use `none`.

`/interwork, /nointerwork`
: `/interwork` specifies that the code in the *inputfile* can interwork between ARM32 and Thumb32 safely. The default is `/nointerwork`.

  `/interwork` is not supported for AArch64 state.

`/inter, /nointer`
: Are synonyms for `/interwork` and `/nointerwork`.

  `/inter` is not supported for AArch64 state.

`/hardfp, /softfp`
: Requests hardware or software floating-point linkage. This enables the procedure call standard to be specified separately from the version of the floating-point hardware available through the `--fpu` option. It is still possible to specify the procedure call standard by using the `--fpu` option, but ARM recommends you use `--apcs`. If floating-point support is not permitted (for example, because `--fpu=none` is specified, or because of other means), then `/hardfp` and `/softfp` are ignored. If floating-point support is permitted and the *softfp* calling convention is used (`--fpu=softvfp` or `--fpu=softvfp+vfp-armv8v4`), then `/hardfp` gives an error.

  `/softfp` is not supported for AArch64 state.

Note

You must specify at least one *qualifier*. If you specify more than one *qualifier*, ensure that there are no spaces or commas between the individual qualifiers in the list.

2.5.2 Example

```
armasm --cpu=8-A.32 --apcs=/inter/hardfp inputfile.s
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.5.3 See also

Procedure Call Standard for the ARM Architecture
http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042-/index.html.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.6 --arm

This option instructs `armasm` to interpret instructions as A32 instructions. It does not, however, guarantee A32-only code in the object file. This is the default. Using this option is equivalent to specifying the `ARM` or `CODE32` directive at the start of the source file.

Note

This option is not supported for AArch64 state.

2.6.1 See also

- *--32* on page 2-6.
- *--arm_only* on page 2-10.
- *ARM, THUMB, CODE16 and CODE32* on page 10-16.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.7 --arm_only

This option instructs `armasm` to only generate A32 code. This is similar to `--arm` but also has the property that `armasm` does not permit the generation of any T32 code.

Note

This option is not supported for AArch64 state.

2.7.1 See also

- *--arm* on page 2-9.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.8 --bi

This option is a synonym for `--bigend`.

2.8.1 See also

- *--bigend* on page 2-12.
- *--littleend* on page 2-42.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.9 --bigend

This option instructs `armasm` to assemble code suitable for a big-endian ARM processor. The default is `--littleend`.

2.9.1 See also

- *--littleend* on page 2-42.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.10 --brief_diagnostics

This option instructs `armasm` to use a shorter form of the diagnostic output. In this form, the original source line is not displayed and the error message text is not wrapped when it is too long to fit on a single line. The default is `--no_brief_diagnostics`.

2.10.1 See also

- *--diag_error* on page 2-20.
- *--diag_warning* on page 2-24.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.11 --checkreglist

This option instructs `armasm` to check `RLIST`, `LDM`, and `STM` register lists to ensure that all registers are provided in increasing register number order. A warning is given if registers are not listed in order.

Note

In AArch32 state, this option is deprecated. Use `--diag_warning 1206` instead. In AArch64 state, this option is not supported.

2.11.1 See also

- *--diag_warning* on page 2-24.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.12 --cpu

This option enables code generation for the selected ARM processor or architecture. Some instructions produce either errors or warnings if assembled for the wrong target processor or architecture.

2.12.1 Syntax

```
--cpu=list
--cpu=name
```

Where:

`name` is the name of a processor or architecture. Table 2-1 lists the processor and architecture names that are supported.

Processor and architecture names are not case-sensitive in `armasm`.

Wildcard characters are not accepted.

Table 2-1 Supported ARM processors and architectures

Processor and architecture name	Description
`7`	ARMv7 with Thumb (Thumb-2 technology) only, and without hardware divide
`7-A`	ARMv7 application profile supporting virtual MMU-based memory systems, with ARM and Thumb (Thumb-2 technology), DSP support, and 32-bit Advanced SIMD support
`7-A.security`	Enables the use of the `SMC` instruction (formerly `SMI`) when assembling for the ARMv7-A architecture
`8-A.32`	ARMv8, AArch32 state
`8-A.32.crypto`	ARMv8, AArch32 state with cryptographic instructions
`8-A.32.no_neon`	ARMv8, AArch32 state without Advanced SIMD instructions
`8-A.64`	ARMv8, AArch64 state
`8-A.64.crypto`	ARMv8, AArch64 state with cryptographic instructions
`8-A.64.no_neon`	ARMv8, AArch64 state without Advanced SIMD instructions
`Cortex-A5`	Cortex-A5 processor
`Cortex-A5.vfp`	Cortex-A5 processor with floating point instructions
`Cortex-A5.neon`	Cortex-A5 processor with Advanced SIMD instructions
`Cortex-A7`	Cortex-A7 processor
`Cortex-A7.no_neon`	Cortex-A7 processor with floating point instructions
`Cortex-A7.no_neon.no_vfp`	Cortex-A7 processor without Advanced SIMD instructions and without floating point instructions
`Cortex-A8`	Cortex-A8 processor
`Cortex-A8.no_neon`	Cortex-A8 processor without Advanced SIMD instructions
`Cortex-A9`	Cortex-A9 processor

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 2-1 Supported ARM processors and architectures (continued)

Processor and architecture name	Description
`Cortex-A9.no_neon`	Cortex-A9 processor with floating point instructions
`Cortex-A9.no_neon.no_vfp`	Cortex-A9 processor without Advanced SIMD instructions and without floating point instructions
`Cortex-A12`	Cortex-A12 processor
`Cortex-A12.no_neon.no_vfp`	Cortex-A12 processor without Advanced SIMD instructions and without floating point instructions
`Cortex-A15`	Cortex-A15 processor
`Cortex-A15.no_neon`	Cortex-A15 processor with floating point instructions
`Cortex-A15.no_neon.no_vfp`	Cortex-A15 processor without Advanced SIMD instructions and without floating point instructions
`Cortex-A17`	Cortex-A17 processor
`Cortex-A17.no_neon.no_vfp`	Cortex-A17 processor without Advanced SIMD instructions and without floating point instructions

2.12.2 Usage

Specify `--cpu=list` to list the supported processor and architecture names that you can use with `--cpu=`*name*.

Note

There is no default option for `--cpu`.

2.12.3 Examples

```
armasm --cpu=list
armasm --cpu=8-A.64 inputfile.s
```

2.12.4 See also

Reference

- *--unsafe* on page 2-67.

Other information

- *ARM Architecture Reference Manual* http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.13 --debug

This option instructs `armasm` to generate DWARF debug tables. `--debug` is a synonym for `-g`. The default is DWARF 3.

Note

Local symbols are not preserved with `--debug`. You must specify `--keep` if you want to preserve the local symbols to aid debugging.

2.13.1 See also

- *--dwarf2* on page 2-26.
- *--dwarf3* on page 2-27.
- *--keep* on page 2-37.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.14 --depend

This option instructs `armasm` to save source file dependency lists to *`dependfile`*. These are suitable for use with make utilities.

2.14.1 Syntax

`--depend=`*`dependfile`*

2.14.2 See also

- *--md* on page 2-45.
- *--depend_format* on page 2-19.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.15 --depend_format

This option changes the format of output dependency files to UNIX-style format, for compatibility with some UNIX make programs.

2.15.1 Syntax

`--depend_format=`*`string`*

Where *`string`* is one of:

`unix` Generates dependency files with UNIX-style path separators.

`unix_escaped`
Is the same as `unix`, but escapes spaces with backslash.

`unix_quoted`
Is the same as `unix`, but surrounds path names with double quotes.

2.15.2 See also

- *--depend* on page 2-18.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.16 --diag_error

Sets diagnostic messages that have a specific tag to Error severity.

2.16.1 Syntax

`--diag_error=tag{,tag}`

Where *tag* can be:

- A diagnostic message number to set to error severity.
- `warning` to treat all warnings as errors.

2.16.2 Usage

Diagnostic messages output by `armasm` can be identified by a tag in the form of {*prefix*}*number*, where the *prefix* is A.

You can specify more than one tag with this option by separating each tag using a comma. You can specify the optional assembler prefix `A` before the tag number. If any prefix other than A is included, the message number is ignored.

Table 2-2 shows the meaning of the term *severity* used in the option descriptions.

Table 2-2 Severity of diagnostic messages

Severity	Description
Error	Errors indicate violations in the syntactic or semantic rules of assembly language. Assembly continues, but object code is not generated.
Warning	Warnings indicate unusual conditions in your code that might indicate a problem. Assembly continues, and object code is generated unless any problems with an Error severity are detected.
Remark	Remarks indicate common, but not recommended, use of assembly language. These diagnostics are not issued by default. Assembly continues, and object code is generated unless any problems with an Error severity are detected.

2.16.3 See also

- *--brief_diagnostics* on page 2-13.
- *--diag_warning* on page 2-24.
- *--diag_suppress* on page 2-23.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.17 --diag_remark

Sets diagnostic messages that have a specific tag to Remark severity.

2.17.1 Syntax

`--diag_remark=`*tag*`{,`*tag*`}`

Where *tag* is a comma-separated list of diagnostic message numbers.

2.17.2 Usage

Diagnostic messages output by `armasm` can be identified by a tag in the form of {*prefix*}*number*, where the *prefix* is `A`.

You can specify more than one tag with these options by separating each tag using a comma. You can specify the optional assembler prefix `A` before the tag number. If any prefix other than `A` is included, the message number is ignored.

2.17.3 See also

- *--brief_diagnostics* on page 2-13.
- *--diag_error* on page 2-20.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.18 --diag_style

Specifies the display style for diagnostic messages.

2.18.1 Syntax

`--diag_style=style`

Where *style* is one of:

`arm` Display messages using the ARM assembler style. This is the default if `--diag_style` is not specified.

`ide` Include the line number and character count for the line that is in error. These values are displayed in parentheses.

`gnu` Display messages using the GNU style.

2.18.2 Usage

Choosing the option `--diag_style=ide` implicitly selects the option `--brief_diagnostics`. Explicitly selecting `--no_brief_diagnostics` on the command line overrides the selection of `--brief_diagnostics` implied by `--diag_style=ide`.

Selecting either the option `--diag_style=arm` or the option `--diag_style=gnu` does not imply any selection of `--brief_diagnostics`.

2.18.3 See also

- *--brief_diagnostics* on page 2-13.
- *--diag_style*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.19 --diag_suppress

Suppresses diagnostic messages that have a specific tag.

2.19.1 Syntax

`--diag_suppress=`*tag*`{,`*tag*`}`

Where tag can be:

- A diagnostic message number to be suppressed.
- `error`, to suppress all downgradeable errors.
- `warning`, to suppress all warnings.

Diagnostic messages output by `armasm` can be identified by a tag in the form of {*prefix*}*number*, where the *prefix* is `A`.

You can specify more than one tag with this option by separating each tag using a comma.

2.19.2 Examples

For example, to suppress the warning messages that have numbers `1293` and `187`, use the following command:

```
armasm --cpu=8-A.64 --diag_suppress=1293,187
```

You can specify the optional assembler prefix `A` before the tag number. For example:

```
armasm --cpu=8-A.64 --diag_suppress=A1293,A187
```

If any prefix other than `A` is included, the message number is ignored. Diagnostic message tags can be cut and pasted directly into a command line.

2.19.3 See also

- *--diag_error* on page 2-20.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.20 --diag_warning

Sets diagnostic messages that have a specific tag to Warning severity.

2.20.1 Syntax

`--diag_warning=`*tag*`{,`*tag*`}`

Where tag can be:

- A diagnostic message number to set to warning severity.
- `error`, to set all errors that can be downgraded to warnings.

Diagnostic messages output by `armasm` can be identified by a tag in the form of {*prefix*}*number*, where the *prefix* is `A`.

You can specify more than one tag with these options by separating each tag using a comma. You can specify the optional assembler prefix `A` before the tag number. If any prefix other than `A` is included, the message number is ignored.

2.20.2 See also

- *--diag_error* on page 2-20.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.21 --dllexport_all

This option gives all exported global symbols `STV_PROTECTED` visibility in ELF rather than `STV_HIDDEN`, unless overridden by source directives.

2.21.1 See also

- *EXPORT or GLOBAL* on page 10-36.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.22 --dwarf2

This option can be used with `--debug`, to instruct `armasm` to generate DWARF 2 debug tables.

Note

This option is not supported for AArch64 state.

2.22.1 See also

- *--debug* on page 2-17.
- *--dwarf3* on page 2-27.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.23 --dwarf3

This option can be used with `--debug`, to instruct `armasm` to generate DWARF 3 debug tables. This is the default if `--debug` is specified.

2.23.1 See also

- *--debug* on page 2-17.
- *--dwarf2* on page 2-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.24 --errors

This option instructs `armasm` to output error messages to *`errorfile`*.

2.24.1 Syntax

`--errors=`*`errorfile`*

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.25 --execstack

This option generates a .note.GNU-stack section marking the stack as executable.

You can also use the AREA directive to generate an executable .note.GNU-stack section:

```
AREA    |.note.GNU-stack|,ALIGN=0,READONLY,NOALLOC,CODE
```

In the absence of --execstack and --no_execstack, the .note.GNU-stack section is not generated unless it is specified by the AREA directive.

2.25.1 See also

- *--no_execstack* on page 2-48.
- *AREA* on page 10-13.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.26 --exceptions

This option instructs `armasm` to switch on exception table generation for all functions defined by `FUNCTION` (or `PROC`) and `ENDFUNC` (or `ENDP`).

Note

This option is not supported for AArch64 state.

2.26.1 See also

- *--no_exceptions* on page 2-49.
- *--exceptions_unwind* on page 2-31.
- *--no_exceptions_unwind* on page 2-50.
- *FRAME UNWIND ON* on page 10-50.
- *FUNCTION or PROC* on page 10-52.
- *ENDFUNC or ENDP* on page 10-33.
- *FRAME UNWIND OFF* on page 10-51.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.27 --exceptions_unwind

This option instructs `armasm` to produce unwind tables for functions where possible. This is the default.

For finer control, use `FRAME UNWIND ON` and `FRAME UNWIND OFF` directives.

Note

This option is not supported for AArch64 state.

2.27.1 See also

- *--no_exceptions_unwind* on page 2-50.
- *--exceptions* on page 2-30.
- *--no_exceptions* on page 2-49.
- *FRAME UNWIND ON* on page 10-50.
- *FRAME UNWIND OFF* on page 10-51.
- *FUNCTION or PROC* on page 10-52.
- *ENDFUNC or ENDP* on page 10-33.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.28 --fpmode

This option specifies the floating-point model, and sets library attributes and floating-point optimizations to select the most suitable library when linking.

2.28.1 Syntax

`--fpmode=model`

Where *model* is one of:

`none` Source code is not permitted to use any floating-point type or floating point instruction. This option overrides any explicit `--fpu=name` option.

`ieee_full` All facilities, operations, and representations guaranteed by the IEEE standard are available in single and double-precision. Modes of operation can be selected dynamically at runtime.

`ieee_fixed` IEEE standard with round-to-nearest and no inexact exception.

`ieee_no_fenv` IEEE standard with round-to-nearest and no exceptions. This mode is compatible with the Java floating-point arithmetic model.

`std` IEEE finite values with denormals flushed to zero, round-to-nearest and no exceptions. It is C and C++ compatible. This is the default option.

Finite values are as predicted by the IEEE standard. It is not guaranteed that NaNs and infinities are produced in all circumstances defined by the IEEE model, or that when they are produced, they have the same sign. Also, it is not guaranteed that the sign of zero is that predicted by the IEEE model.

`fast` Some value altering optimizations, where accuracy is sacrificed to fast execution. This is not IEEE compatible, and is not standard C.

Note

This does not cause any changes to the code that you write.

2.28.2 Example

```
armasm --cpu=8-A.32 --fpmode ieee_full inputfile.s
```

2.28.3 See also

Reference

- *--fpu* on page 2-33.

Other information

- *IEEE Standards Association* http://standards.ieee.org/.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.29 --fpu

`--fpu=list` lists the supported FPU names that can be used with the `--fpu=name` option.

`--fpu=name` specifies the target FPU architecture.

2.29.1 Syntax

`--fpu=list`

or

`--fpu=name`

Where `name` is one of:

`none`	Selects no floating-point architecture. No floating-point code is to be used. This produces an error if your code contains floating-point instructions.
`softvfp`	Selects software floating-point linkage.
`fp-armv8`	Selects the integral ARMv8 floating-point hardware. This is the default in AArch32 and AArch64 states.
`softvfp+fp-armv8`	Selects the integral ARMv8 floating-point hardware with software floating-point linkage.

Note

Software floating-point linkage is not supported for AArch64 state.

2.29.2 Usage

If you specify the `--fpu=name` option it overrides any implicit FPU set by the `--cpu` option. `armasm` produces an error if the FPU you specify explicitly is incompatible with the CPU. Floating-point instructions also produce either errors or warnings if assembled for the wrong target FPU.

`armasm` sets a build attribute corresponding to *name* in the object file. The linker determines compatibility between object files, and selection of libraries, accordingly.

2.29.3 See also

- *--fpmode* on page 2-32.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.30 -g

This option is a synonym for `--debug`.

2.30.1 See also

- *--debug* on page 2-17.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.31 --help

This option instructs `armasm` to show a summary of the available command-line options.

2.31.1 See also

- *--version_number* on page 2-69.
- *--vsn* on page 2-71.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.32 -i

This option adds directories to the source file include path. Any directories added using this option have to be fully qualified.

2.32.1 Syntax

`-i`*dir*`{,`*dir*`, ...}`

2.32.2 See also

- *GET or INCLUDE* on page 10-56.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.33 --keep

This option instructs `armasm` to keep named local labels in the symbol table of the object file, for use by the debugger.

2.33.1 See also

- *KEEP* on page 10-63.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.34 --length

This option sets the listing page length to *n*. Length zero means an unpaged listing. The default is 66 lines.

2.34.1 Syntax

`--length=n`

2.34.2 See also

- *--list* on page 2-41.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.35 --li

This option is a synonym for `--littleend`.

2.35.1 See also

- *--littleend* on page 2-42.
- *--bigend* on page 2-12.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.36 --library_type

This option enables the selected library to be used at link time.

2.36.1 Syntax

`--library_type=lib`

Where *lib* is one of:

`standardlib` Specifies that the full ARM runtime libraries are selected at link time. This is the default.

`microlib` Specifies that the C micro-library (microlib) is selected at link time.

Note

- This option can be used with the assembler or linker when use of the libraries require more specialized optimizations.
- This option can be overridden at link time by providing it to the linker.
- microlib is not supported for AArch64 state.

2.36.2 See also

- *Building an application with microlib* on page 3-7 in the *ARM C and C++ Libraries and Floating Point Support User Guide.*

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.37 --list

This option instructs `armasm` to output a detailed listing of the assembly language it produces to a file.

2.37.1 Syntax

`--list={`*`file`*`}`

If `-` is given as *`file`*, the listing is sent to `stdout`.

If you omit *`file`*, the listing is sent to *`inputfile`*`.lst`.

Note

If you omit both *`file`* and the equals sign, output is sent to *`inputfile`*`.lst`. However, this syntax is deprecated and `armasm` issues a warning. This syntax is to be removed in a later release. Use `--list=` instead.

2.37.2 Usage

Use the following command-line options to control the behavior of `--list`:

- `--no_terse`.
- `--width`.
- `--length`.
- `--xref`.

2.37.3 See also

- *--no_terse* on page 2-54.
- *--width* on page 2-72.
- *--length* on page 2-38.
- *--xref* on page 2-73.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.38 --littleend

This option instructs `armasm` to assemble code suitable for a little-endian ARM processor.

2.38.1 See also

- *--bigend* on page 2-12.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.39 -m

This option instructs `armasm` to write source file dependency lists to `stdout`.

2.39.1 See also

- *--md* on page 2-45.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.40 --maxcache

This option sets the maximum source cache size to *n* bytes. The default is 8MB. `armasm` gives a warning if the size is less than 8MB.

2.40.1 Syntax

`--maxcache=`*n*

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.41 --md

This option instructs `armasm` to write source file dependency lists to *`inputfile`*`.d`.

2.41.1 See also

- *-m* on page 2-43.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.42 --no_code_gen

This option instructs `armasm` to exit after pass 1. No object file is generated. This option is useful if you only want to check the syntax of the source code or directives.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.43 --no_esc

This option instructs `armasm` to ignore C-style escaped special characters, such as `\n` and `\t`.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.44 --no_execstack

This option generates a .note.GNU-stack section marking the stack as non-executable.

You can also use the AREA directive to generate a non executable .note.GNU-stack section:

```
AREA    |.note.GNU-stack|,ALIGN=0,READONLY,NOALLOC
```

In the absence of --execstack and --no_execstack, the .note.GNU-stack section is not generated unless it is specified by the AREA directive.

If both the command-line option and source directive are used and are different, then the stack is marked as executable.

Table 2-3 Specifying a command-line option and an AREA directive for GNU-stack sections

	--execstack **command-line option**	--no_execstack **command-line option**
execstack AREA directive	execstack	execstack
no_execstack AREA directive	execstack	no_execstack

2.44.1 See also

- *--execstack* on page 2-29.
- *AREA* on page 10-13.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.45 --no_exceptions

This option instructs `armasm` to switch off exception table generation. No tables are generated. This is the default.

2.45.1 See also

- *--exceptions* on page 2-30.
- *--exceptions_unwind* on page 2-31.
- *--no_exceptions_unwind* on page 2-50.
- *FRAME UNWIND ON* on page 10-50.
- *FRAME UNWIND OFF* on page 10-51.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.46 --no_exceptions_unwind

This option instructs `armasm` to produce no unwind tables.

2.46.1 See also

- *--exceptions* on page 2-30.
- *--no_exceptions* on page 2-49.
- *--exceptions_unwind* on page 2-31.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.47 --no_hide_all

This option gives all exported and imported global symbols `STV_DEFAULT` visibility in ELF rather than `STV_HIDDEN`, unless overridden by source directives.

You can use the following directives to specify an attribute that overrides the implicit symbol visibility:

- `EXPORT`.
- `EXTERN`.
- `GLOBAL`.
- `IMPORT`.

2.47.1 See also

- *EXPORT or GLOBAL* on page 10-36.
- *IMPORT and EXTERN* on page 10-57.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.48 --no_reduce_paths

This option disables the elimination of redundant pathname information in file paths. This is the default setting.

Note

This option is valid for Windows systems only.

2.48.1 See also

- *--reduce_paths* on page 2-60.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.49 --no_regs

This option instructs `armasm` not to predefine register names.

Note

This option is deprecated. In AArch32 state, use `--regnames=none` instead.

2.49.1 See also

- *--regnames* on page 2-61.
- *Predeclared core register names in AArch32 state* on page 4-8 in *armasm User Guide*.
- *Predeclared extension register names in AArch32 state* on page 4-9 in *armasm User Guide*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.50 --no_terse

This option instructs `armasm` to show the lines of assembly code that have been skipped because of conditional assembly in the list file. When this option is not specified on the command line, `armasm` does not output the skipped assembly code to the list file.

This option turns off the terse flag. By default the terse flag is on.

2.50.1 See also

- *--list* on page 2-41.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.51 --no_unaligned_access

This option instructs `armasm` to set an attribute in the object file to disable the use of unaligned accesses.

2.51.1 See also

- *--unaligned_access* on page 2-66.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.52 --no_warn

This option turns off warning messages.

2.52.1 See also

- *--diag_warning* on page 2-24.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.53 -o

This option names the output object file. If this option is not specified, `armasm` creates an object filename of the form *`inputfilename`*`.o`. This option is case-sensitive.

2.53.1 Syntax

`-o` *`filename`*

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.54 --pd

This option is a synonym for `--predefine`.

2.54.1 See also

- *--predefine* on page 2-59.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.55 --predefine

This option instructs armasm to pre-execute one of the SETA, SETL, or SETS directives.

2.55.1 Syntax

```
--predefine "directive"
```

You must enclose *directive* in quotes, for example:

```
armasm --cpu=8-A.64 --predefine "VariableName SETA 20" inputfile.s
```

2.55.2 Usage

armasm also executes a corresponding GBLL, GBLS, or GBLA directive to define the variable before setting its value.

The variable name is case-sensitive. Variables defined using the command line are global to armasm source files specified on the command line.

Be aware of the following:

- The command-line interface of your system might require you to enter special character combinations, such as \", to include strings in *directive*. Alternatively, you can use --via *file* to include a --predefine argument. The command-line interface does not alter arguments from --via files.
- --predefine is not equivalent to the compiler option -D*name*. --predefine defines a global variable whereas -D*name* defines a macro that the C preprocessor expands.

2.55.3 See also

- *--pd* on page 2-58.
- *GBLA, GBLL, and GBLS* on page 10-54.
- *SETA, SETL, and SETS* on page 10-83.
- *IF, ELSE, ENDIF, and ELIF* on page 10-60.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.56 --reduce_paths

This option enables the elimination of redundant pathname information in file paths.

Windows systems impose a 260 character limit on file paths. Where relative pathnames exist whose absolute names expand to longer than 260 characters, you can use the `--reduce_paths` option to reduce absolute pathname length by matching up directories with corresponding instances of `..` and eliminating the directory/`..` sequences in pairs.

Note

ARM recommends that you avoid using long and deeply nested file paths, in preference to minimizing path lengths using the `--reduce_paths` option.

Note

This option is valid for 32-bit Windows systems only.

2.56.1 See also

- *--no_reduce_paths* on page 2-52.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.57 --regnames

This option controls the predefinition of register names.

2.57.1 Syntax

`--regnames=option`

Where *option* is one of the following:

none Instructs `armasm` not to predefine register names.

callstd Defines additional register names based on the AAPCS variant that you are using as specified by the `--apcs` option.

all Defines all AAPCS registers regardless of the value of `--apcs`.

Note

These options are not supported for AArch64 state.

2.57.2 See also

- *--apcs* on page 2-7.
- *--no_regs* on page 2-53.
- *Predeclared core register names in AArch32 state* on page 4-8 in *armasm User Guide*.
- *Predeclared extension register names in AArch32 state* on page 4-9 in *armasm User Guide*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.58 --report-if-not-wysiwyg

This option instructs armasm to report when it outputs an encoding that was not directly requested in the source code. This can happen when armasm:

- Uses a pseudo-instruction that is not available in other assemblers, for example MOV32.
- Outputs an encoding that does not directly match the instruction mnemonic, for example if armasm outputs the MVN encoding when assembling the MOV instruction.
- Inserts additional instructions where necessary for instruction syntax semantics, for example armasm can insert a missing IT instruction before a conditional T32 instruction.

Note

This option is not supported for AArch64 state.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.59 --show_cmdline

This option outputs the command line used by `armasm`. It shows the command line after processing by `armasm`, and can be useful to check:

- The command line a build system is using.
- How `armasm` is interpreting the supplied command line, for example, the ordering of command-line options.

The commands are shown normalized, and the contents of any via files are expanded.

The output is sent to the standard output stream, `stdout`.

2.59.1 See also

- --*via* on page 2-70.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.60 --thumb

This option instructs `armasm` to interpret instructions as T32 instructions, using UAL syntax. This is equivalent to a `THUMB` directive at the start of the source file.

Note

This option is not supported for AArch64 state.

2.60.1 See also

- *--arm* on page 2-9.
- *ARM, THUMB, CODE16 and CODE32* on page 10-16.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.61 --tool_variant

Specifies the type of DS-5 license to use when running outside of the DS-5 environment.

2.61.1 Syntax

`--tool_variant=`*`toolkit`*

Where:

`toolkit` is the type of license to use. This corresponds to the prefix used by the license features in your license file up to the first '_' character. For example, if your license file contains a feature named `ds5eval_compiler6` then specify the `--tool_variant=ds5eval` option.

2.61.2 Usage

When you run `armasm` within Eclipse, the provided DS-5 Command Prompt on Windows, or the `suite_exec` shell on Linux then all license managed components pick up the license and toolkit settings correctly.

If you use a different console then you must configure the tools manually. To do this:

- Set the environment variable `ARMLMD_LICENSE_FILE` to point at your license file.
- Pass the `--tool_variant=`*`toolkit`* option on each invocation.

Note

You can set the environment variable `ARMCOMPILER6_ASMOPT` to pass this option automatically. For example, `ARMCOMPILER6_ASMOPT=--tool_variant=ds5eval`.

Failure to set either the license location or the `--tool_variant` option can result in an error of the following form:

`error: License checkout for feature `*`feature`*` was denied...`

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.62 --unaligned_access

This option instructs `armasm` to set an attribute in the object file to enable the use of unaligned accesses.

2.62.1 See also

- *--no_unaligned_access* on page 2-55.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.63 --unsafe

This option enables instructions from differing architectures to be assembled without error. It changes error messages to corresponding warning messages. It also suppresses warnings about operator precedence.

Note

This option is not supported for AArch64 state.

2.63.1 See also

- *--diag_error* on page 2-20.
- *--diag_warning* on page 2-24.
- *Binary operators* on page 10-22 in *armasm User Guide*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.64 --untyped_local_labels

This option causes armasm not to set the T32 bit for the address of a numeric local label referenced in an LDR pseudo-instruction.

When this option is not used, if you reference a numeric local label in an LDR pseudo-instruction, and the label is in T32 code, then armasm sets the T32 bit (bit 0) of the address. You can then use the address as the target for a BX or BLX instruction.

If you require the actual address of the numeric local label, without the T32 bit set, then use this option.

Note

When using this option, if you use the address in a branch (register) instruction, armasm treats it as an A32 code address, causing the branch to arrive in A32 state, meaning it would interpret this code as A32 instructions.

Note

This option is not supported for AArch64 state.

2.64.1 Example

```
    THUMB
    ...
1
    ...
    LDR r0,=%B1 ; r0 contains the address of numeric local label "1",
                ; T32 bit is not set if --untyped_local_labels was used
    ...
```

2.64.2 See also

- *LDR pseudo-instruction* on page 3-86.
- *B, BL, BX, and BLX* on page 3-44.
- *Numeric local labels* on page 10-12 in *armasm User Guide*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.65 --version_number

This option displays the version of `armasm` you are using. `armasm` displays the version number in the format *nnbbbb*, where:

nn is the version number

bbbb is the build number.

For example, version 6.0 build 256 is displayed as `600256`.

2.65.1 See also

- --*vsn* on page 2-71.
- --*help* on page 2-35.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.66 --via

This option instructs `armasm` to open *`file`* and read in command-line arguments to `armasm`.

2.66.1 Syntax

`--via=`*`file`*

2.66.2 See also

Reference

- Appendix A *Via File Syntax*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.67 --vsn

This option displays the version information and license details. For example:

```
>armasm --vsn
ARM Assembler, 6.0 [Build 256]
Software supplied by: ARM Limited
```

2.67.1 See also

- *--version_number* on page 2-69.
- *--help* on page 2-35.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.68 --width

This option sets the listing page width to *n*. The default is 79 characters.

2.68.1 Syntax

`--width=`*n*

2.68.2 See also

- *--list* on page 2-41.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

2.69 --xref

This option instructs `armasm` to list cross-referencing information on symbols, including where they were defined and where they were used, both inside and outside macros. The default is off.

2.69.1 See also

- *--list* on page 2-41.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Chapter 3
A32 and T32 Instructions

The following topics describe the A32 and T32 instructions supported in AArch32 state by the ARM assembler:

- *A32 and T32 instruction summary* on page 3-2.
- *Instruction width specifiers* on page 3-8.
- *Memory access instructions* on page 3-9.
- *General data processing instructions* on page 3-11.
- *Multiply instructions* on page 3-18.
- *Saturating instructions* on page 3-19.
- *Parallel instructions* on page 3-20.
- *Packing and unpacking instructions* on page 3-21.
- *Branch and control instructions* on page 3-22.
- *Coprocessor instructions* on page 3-23.
- *Miscellaneous instructions* on page 3-24.
- *Pseudo-instructions* on page 3-25.
- *Condition codes* on page 3-26.

Note

Detailed information about the ARMv8 architecture is available under license. Contact your ARM Account Representative for details.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.1 A32 and T32 instruction summary

Table 3-1 gives an overview of the instructions available in the A32 and T32 instruction sets.

Table 3-1 Summary of instructions

Mnemonic	Brief description	See
ADC, ADD	Add with Carry, Add	page 3-27
ADR	Load program or register-relative address (short range)	page 3-32
ADRL pseudo-instruction	Load program or register-relative address (medium range)	page 3-36
AND	Logical AND	page 3-38
ASR	Arithmetic Shift Right	page 3-41
B	Branch	page 3-44
BFC, BFI	Bit Field Clear and Insert	page 3-47
BIC	Bit Clear	page 3-38
BKPT	Software breakpoint	page 3-48
BL	Branch with Link	page 3-44
BLX	Branch with Link, change instruction set	page 3-44
BX	Branch, change instruction set	page 3-44
CBZ, CBNZ	Compare and Branch if {Non}Zero	page 3-49
CLREX	Clear Exclusive	page 3-50
CLZ	Count leading zeros	page 3-51
CMN, CMP	Compare Negative, Compare	page 3-52
CPS	Change Processor State	page 3-54
CPY pseudo-instruction	Copy	page 3-55
DBG	Debug	page 3-56
DCPS1	Debug switch to exception level 1	page 3-56
DCPS2	Debug switch to exception level 2	page 3-56
DCPS3	Debug switch to exception level 3	page 3-56
DMB, DSB	Data Memory Barrier, Data Synchronization Barrier	page 3-60
EOR	Exclusive OR	page 3-38
ERET	Exception Return	page 3-63
HLT	Halting breakpoint	page 3-64
HVC	Hypervisor Call	page 3-65
ISB	Instruction Synchronization Barrier	page 3-60
IT	If-Then	page 3-66
LDC	Load Coprocessor	page 3-69

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 3-1 Summary of instructions (continued)

Mnemonic	Brief description	See
LDM	Load Multiple registers	page 3-71
LDR	Load Register with word	page 3-9
LDR pseudo-instruction	Load Register pseudo-instruction	page 3-86
LDA, LDAB, LDAH	Load-Acquire Register Word, Byte, Halfword	page 3-89
LDAEX, LDAEXB, LDAEXH, LDAEXD	Load-Acquire Register Exclusive Word, Byte, Halfword, Doubleword	page 3-91
LDRB	Load Register with Byte	page 3-9
LDRBT	Load Register with Byte, user mode	page 3-9
LDRD	Load Registers with two words	page 3-9
LDREX	Load Register Exclusive	page 3-94
LDREXB, LDREXH	Load Register Exclusive Byte, Halfword	page 3-94
LDREXD	Load Register Exclusive Doubleword	page 3-94
LDRH	Load Register with Halfword	page 3-9
LDRHT	Load Register with Halfword, user mode	page 3-9
LDRSB	Load Register with Signed Byte	page 3-9
LDRSBT	Load Register with Signed Byte, user mode	page 3-9
LDRSH	Load Register with Signed Halfword	page 3-9
LDRSHT	Load Register with Signed Halfword, user mode	page 3-9
LDRT	Load Register with word, user mode	page 3-9
LSL, LSR	Logical Shift Left, Logical Shift Right	page 3-41
MCR	Move from Register to Coprocessor	page 3-97
MCRR	Move from Registers to Coprocessor	page 3-97
MLA	Multiply Accumulate	page 3-110
MLS	Multiply and Subtract	page 3-110
MOV	Move	page 3-98
MOVT	Move Top	page 3-101
MOV32 pseudo-instruction	Move 32-bit immediate to register	page 3-102
MRC	Move from Coprocessor to Register	page 3-103
MRRC	Move from Coprocessor to Registers	page 3-103
MRS	Move from PSR to Register	page 3-105
MRS pseudo-instruction	Move from system Coprocessor to Register	page 3-104
MSR	Move from Register to PSR	page 3-108
MSR pseudo-instruction	Move from Register to system Coprocessor	page 3-107

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 3-1 Summary of instructions (continued)

Mnemonic	Brief description	See
MUL	Multiply	page 3-110
MVN	Move Not	page 3-98
NEG pseudo-instruction	Negate	page 3-112
NOP	No Operation	page 3-113
ORN	Logical OR NOT	page 3-38
ORR	Logical OR	page 3-38
PKHBT, PKHTB	Pack Halfwords	page 3-117
PLD	Preload Data	page 3-119
PLDW	Preload Data with intent to Write	page 3-119
PLI	Preload Instruction	page 3-119
PUSH, POP	PUSH registers to stack, POP registers from stack	page 3-121
QADD, QDADD, QDSUB, QSUB	Saturating arithmetic	page 3-123
QADD8, QADD16, QASX, QSUB8, QSUB16, QSAX	Parallel signed saturating arithmetic	page 3-114
RBIT	Reverse Bits	page 3-125
REV, REV16, REVSH	Reverse byte order	page 3-125
RFE	Return From Exception	page 3-127
ROR	Rotate Right Register	page 3-41
RRX	Rotate Right with Extend	page 3-41
RSB	Reverse Subtract	page 3-27
RSC	Reverse Subtract with Carry	page 3-27
SADD8, SADD16, SASX	Parallel Signed arithmetic	page 3-114
SBC	Subtract with Carry	page 3-27
SBFX, UBFX	Signed, Unsigned Bit Field eXtract	page 3-129
SDIV	Signed Divide	page 3-130
SEL	Select bytes according to APSR GE flags	page 3-131
SETEND	Set Endianness for memory accesses	page 3-133
SEV	Set Event	page 3-134
SEVL	Set Event Locally	page 3-134
SHADD8, SHADD16, SHASX, SHSUB8, SHSUB16, SHSAX	Parallel Signed Halving arithmetic	page 3-114
SMC	Secure Monitor Call	page 3-136
SMLAxy	Signed Multiply with Accumulate (32 <= 16 x 16 + 32)	page 3-146

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 3-1 Summary of instructions (continued)

Mnemonic	Brief description	See
SMLAD	Dual Signed Multiply Accumulate (32 <= 32 + 16 x 16 + 16 x 16)	page 3-137
SMLAL	Signed Multiply Accumulate (64 <= 64 + 32 x 32)	page 3-164
SMLALxy	Signed Multiply Accumulate (64 <= 64 + 16 x 16)	page 3-139
SMLALD	Dual Signed Multiply Accumulate Long (64 <= 64 + 16 x 16 + 16 x 16)	page 3-140
SMLAWy	Signed Multiply with Accumulate (32 <= 32 x 16 + 32)	page 3-145
SMLSD	Dual Signed Multiply Subtract Accumulate (32 <= 32 + 16 x 16 – 16 x 16)	page 3-137
SMLSLD	Dual Signed Multiply Subtract Accumulate Long (64 <= 64 + 16 x 16 – 16 x 16)	page 3-140
SMMLA	Signed top word Multiply with Accumulate (32 <= TopWord(32 x 32 + 32))	page 3-142
SMMLS	Signed top word Multiply with Subtract (32 <= TopWord(32 - 32 x 32))	page 3-142
SMMUL	Signed top word Multiply (32 <= TopWord(32 x 32))	page 3-142
SMUAD, SMUSD	Dual Signed Multiply, and Add or Subtract products	page 3-144
SMULxy	Signed Multiply (32 <= 16 x 16)	page 3-146
SMULL	Signed Multiply (64 <= 32 x 32)	page 3-164
SMULWy	Signed Multiply (32 <= 32 x 16)	page 3-145
SRS	Store Return State	page 3-148
SSAT	Signed Saturate	page 3-150
SSAT16	Signed Saturate, parallel halfwords	page 3-152
SSUB8, SSUB16, SSAX	Parallel Signed arithmetic	page 3-114
STC	Store Coprocessor	page 3-69
STM	Store Multiple registers	page 3-71
STR	Store Register with word	page 3-9
STRB	Store Register with Byte	page 3-9
STRBT	Store Register with Byte, user mode	page 3-9
STRD	Store Registers with two words	page 3-9
STREX	Store Register Exclusive	page 3-94
STREXB, STREXH	Store Register Exclusive Byte, Halfword	page 3-94
STREXD	Store Register Exclusive Doubleword	page 3-94

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 3-1 Summary of instructions (continued)

Mnemonic	Brief description	See
STRH	Store Register with Halfword	page 3-9
STRHT	Store Register with Halfword, user mode	page 3-9
STL, STLB, STLH	Store-Release Word, Byte, Halfword	page 3-89
STLEX, STLEXB, STLEXH, STLEXD	Store-Release Exclusive Word, Byte, Halfword, Doubleword	page 3-91
STRT	Store Register with word, user mode	page 3-9
SUB	Subtract	page 3-27
SUBS pc, lr	Exception return, no stack	page 3-154
SVC (formerly SWI)	Supervisor Call	page 3-156
SXTAB, SXTAB16, SXTAH	Signed extend, with Addition	page 3-157
SXTB, SXTH	Signed extend	page 3-157
SXTB16	Signed extend	page 3-157
SYS	Execute System coprocessor instruction	page 3-159
TBB, TBH	Table Branch Byte, Halfword	page 3-160
TEQ	Test Equivalence	page 3-161
TST	Test	page 3-161
UADD8, UADD16, UASX	Parallel Unsigned arithmetic	page 3-114
UDIV	Unsigned Divide	page 3-130
UHADD8, UHADD16, UHASX, UHSUB8, UHSUB16, UHSAX	Parallel Unsigned Halving arithmetic	page 3-114
UMAAL	Unsigned Multiply Accumulate Accumulate Long (64 <= 32 + 32 + 32 x 32)	page 3-163
UMLAL, UMULL	Unsigned Multiply Accumulate, Unsigned Multiply (64 <= 32 x 32 + 64), (64 <= 32 x 32)	page 3-164
UQADD8, UQADD16, UQASX, UQSUB8, UQSUB16, UQSAX	Parallel Unsigned Saturating arithmetic	page 3-114
USAD8	Unsigned Sum of Absolute Differences	page 3-167
USADA8	Accumulate Unsigned Sum of Absolute Differences	page 3-167
USAT	Unsigned Saturate	page 3-150
USAT16	Unsigned Saturate, parallel halfwords	page 3-152
USUB8, USUB16, USAX	Parallel Unsigned arithmetic	page 3-114
UXTAB, UXTAB16, UXTAH	Unsigned extend with Addition	page 3-157
UXTB, UXTH	Unsigned extend	page 3-157

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 3-1 Summary of instructions (continued)

Mnemonic	Brief description	See
UXTB16	Unsigned extend	page 3-157
V*	See Chapter 4 *Advanced SIMD and Floating-point Programming (32-bit)*	
WFE, WFI, YIELD	Wait For Event, Wait For Interrupt, Yield	page 3-134

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.2 Instruction width specifiers

The instruction width specifiers `.W` and `.N` control the size of T32 instruction encodings.

In T32 code the `.W` width specifier forces the assembler to generate a 32-bit encoding, even if a 16-bit encoding is available. The `.W` specifier has no effect when assembling to A32 code.

In T32 code the `.N` width specifier forces the assembler to generate a 16-bit encoding. In this case, if the instruction cannot be encoded in 16 bits or if `.N` is used in A32 code, the assembler generates an error.

If you use an instruction width specifier, you must place it immediately after the instruction mnemonic and any condition code, for example:

```
BCS.W   label   ; forces 32-bit instruction even for a short branch
B.N     label   ; faults if label out of range for 16-bit instruction
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.3 Memory access instructions

The following topics describe the memory access instructions:

- *LDR and STR (immediate offset)* on page 3-74

  Load and Store with immediate offset, pre-indexed immediate offset, or post-indexed immediate offset.

- *LDR and STR (register offset)* on page 3-77

  Load and Store with register offset, pre-indexed register offset, or post-indexed register offset.

- *LDR and STR, unprivileged* on page 3-80

  Load and Store, with User mode privilege.

- *LDR (PC-relative)* on page 3-82

  Load register. The address is an offset from the PC.

- *LDR (register-relative)* on page 3-84

  Load register. The address is an offset from a base register.

- *ADR (PC-relative)* on page 3-32

  Load a PC-relative address.

- *ADR (register-relative)* on page 3-34

  Load a register-relative address.

- *PLD, PLDW, and PLI* on page 3-119

  Preload an address for the future.

- *LDM and STM* on page 3-71

  Load and Store Multiple Registers.

- *PUSH and POP* on page 3-121

  Push low registers, and optionally the LR, onto the stack.

  Pop low registers, and optionally the PC, off the stack.

- *RFE* on page 3-127

  Return From Exception.

- *ERET* on page 3-63

  Exception Return.

- *SRS* on page 3-148

  Store Return State.

- *LDA and STL* on page 3-89

  Load-Acquire and Store-Release.

- *LDREX and STREX* on page 3-94

  Load and Store Register Exclusive.

- *LDAEX and STLEX* on page 3-91

  Load-Acquire and Store-Release Exclusive.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

- *CLREX* on page 3-50

  Clear Exclusive.

Note

There is also an `LDR` pseudo-instruction. This either assembles to an `LDR` instruction, or to a `MOV` or `MVN` instruction.

3.3.1 See also

Concepts

armasm User Guide:

- *Memory accesses* on page 7-28.

Reference

- *LDR pseudo-instruction* on page 3-86.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.4 General data processing instructions

The following topics describe the general data processing instructions:

- *Flexible second operand (Operand2)* on page 3-12.
- *Operand2 as a constant* on page 3-13.
- *Operand2 as a register with optional shift* on page 3-14.
- *Shift operations* on page 3-15.
- *ADD, SUB, RSB, ADC, SBC, and RSC* on page 3-27
  Add, Subtract, and Reverse Subtract, each with or without Carry.
- *SUBS pc, lr* on page 3-154
  Return from exception without popping the stack.
- *AND, ORR, EOR, BIC, and ORN* on page 3-38
  Logical AND, OR, Exclusive OR, OR NOT, and Bit Clear.
- *CLZ* on page 3-51
  Count Leading Zeros.
- *CMP and CMN* on page 3-52
  Compare and Compare Negative.
- *MOV and MVN* on page 3-98
  Move and Move Not.
- *MOVT* on page 3-101
  Move Top, Wide.
- *TST and TEQ* on page 3-161
  Test and Test Equivalence.
- *SEL* on page 3-131
  Select bytes from each operand according to the state of the APSR GE flags.
- *REV, REV16, REVSH, and RBIT* on page 3-125
  Reverse bytes or Bits.
- *ASR, LSL, LSR, ROR, and RRX* on page 3-41
  Arithmetic Shift Right.
- *SDIV and UDIV* on page 3-130
  Signed Divide and Unsigned Divide.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.5 Flexible second operand (Operand2)

Many A32 and T32 general data processing instructions have a flexible second operand. This is shown as *Operand2* in the descriptions of the syntax of each instruction.

Operand2 can be a:

- Constant.
- Register with optional shift.

3.5.1 See also

Reference

- *Operand2 as a constant* on page 3-13.
- *Operand2 as a register with optional shift* on page 3-14.
- *Shift operations* on page 3-15.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.6 Operand2 as a constant

You specify an Operand2 constant in the form:

`#constant`

where *constant* is an expression evaluating to a numeric value.

In A32 instructions, *constant* can have any value that can be produced by rotating an 8-bit value right by any even number of bits within a 32-bit word.

In T32 instructions, *constant* can be:

- Any constant that can be produced by shifting an 8-bit value left by any number of bits within a 32-bit word.
- Any constant of the form `0x00XY00XY`.
- Any constant of the form `0xXY00XY00`.
- Any constant of the form `0xXYXYXYXY`.

Note

In these constants, `X` and `Y` are hexadecimal digits.

In addition, in a small number of instructions, *constant* can take a wider range of values. These are described in the individual instruction descriptions.

When an Operand2 constant is used with the instructions `MOVS`, `MVNS`, `ANDS`, `ORRS`, `ORNS`, `EORS`, `BICS`, `TEQ` or `TST`, the carry flag is updated to bit[31] of the constant, if the constant is greater than 255 and can be produced by shifting an 8-bit value. These instructions do not affect the carry flag if Operand2 is any other constant.

3.6.1 Instruction substitution

If a value of *constant* is not available, but its logical inverse or negation is available, then the assembler produces an equivalent instruction and inverts or negates *constant*.

For example, an assembler might assemble the instruction `CMP` *Rd*, `#0xFFFFFFFE` as the equivalent instruction `CMN` *Rd*, `#0x2`.

Be aware of this when comparing disassembly listings with source code.

You can use the `--diag_warning 1645` assembler command line option to check when an instruction substitution occurs.

3.6.2 See also

Concepts

- *Flexible second operand (Operand2)* on page 3-12.

Reference

- *Operand2 as a register with optional shift* on page 3-14.
- *Shift operations* on page 3-15.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.7 Operand2 as a register with optional shift

You specify an Operand2 register in the form:

Rm {, *shift*}

where:

Rm is the register holding the data for the second operand.

shift is an optional constant or register-controlled shift to be applied to *Rm*. It can be one of:

- ASR #*n* — arithmetic shift right *n* bits, $1 \le n \le 32$.
- LSL #*n* — logical shift left *n* bits, $1 \le n \le 31$.
- LSR #*n* — logical shift right *n* bits, $1 \le n \le 32$.
- ROR #*n* — rotate right *n* bits, $1 \le n \le 31$.
- RRX — rotate right one bit, with extend.
- *type Rs* — register-controlled shift is available in A32 code only, where:
  - *type* is one of ASR, LSL, LSR, ROR.
  - *Rs* is a register supplying the shift amount, and only the least significant byte is used.
- \- if omitted, no shift occurs, equivalent to LSL #0.

If you omit the shift, or specify LSL #0, the instruction uses the value in *Rm*.

If you specify a shift, the shift is applied to the value in *Rm*, and the resulting 32-bit value is used by the instruction. However, the contents in the register *Rm* remains unchanged. Specifying a register with shift also updates the carry flag when used with certain instructions.

3.7.1 See also

Concepts

- *Flexible second operand (Operand2)* on page 3-12.

Reference

- *Operand2 as a constant* on page 3-13.
- *Shift operations* on page 3-15.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.8 Shift operations

Register shift operations move the bits in a register left or right by a specified number of bits, the *shift length*. Register shift can be performed:

- Directly by the instructions ASR, LSR, LSL, ROR, and RRX, and the result is written to a destination register.
- During the calculation of *Operand2* by the instructions that specify the second operand as a register with shift. The result is used by the instruction.

The permitted shift lengths depend on the shift type and the instruction, see the individual instruction description or the flexible second operand description. If the shift length is 0, no shift occurs. Register shift operations update the carry flag except when the specified shift length is 0. The following descriptions give information about the various shift operations and how they affect the carry flag. In these descriptions, *Rm* is the register containing the value to be shifted, and *n* is the shift length.

3.8.1 ASR

Arithmetic shift right by *n* bits moves the left-hand 32-*n* bits of the register *Rm*, to the right by *n* places, into the right-hand 32-*n* bits of the result. And it copies the original bit[31] of the register into the left-hand *n* bits of the result. See Figure 3-1.

You can use the ASR *#n* operation to divide the value in the register *Rm* by 2^n, with the result being rounded towards negative-infinity.

When the instruction is ASRS or when ASR *#n* is used in *Operand2* with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to the last bit shifted out, bit[*n*-1], of the register *Rm*.

Note

- If *n* is 32 or more, then all the bits in the result are set to the value of bit[31] of *Rm*.
- If *n* is 32 or more and the carry flag is updated, it is updated to the value of bit[31] of *Rm*.

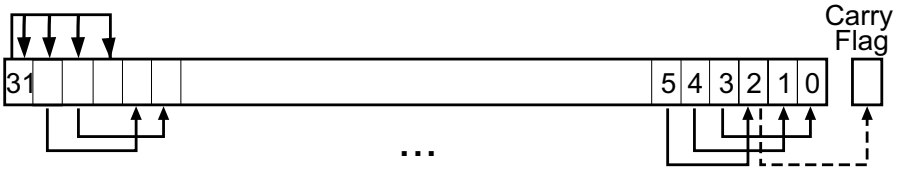


Figure 3-1 ASR #3

3.8.2 LSR

Logical shift right by *n* bits moves the left-hand 32-*n* bits of the register *Rm*, to the right by *n* places, into the right-hand 32-*n* bits of the result. And it sets the left-hand *n* bits of the result to 0. See Figure 3-2 on page 3-16.

You can use the LSR *#n* operation to divide the value in the register *Rm* by 2^n, if the value is regarded as an unsigned integer.

When the instruction is LSRS or when LSR *#n* is used in *Operand2* with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to the last bit shifted out, bit[*n*-1], of the register *Rm*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Note

- If *n* is 32 or more, then all the bits in the result are cleared to 0.
- If *n* is 33 or more and the carry flag is updated, it is updated to 0.

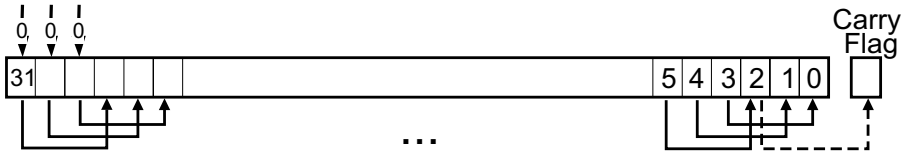


Figure 3-2 LSR #3

3.8.3 LSL

Logical shift left by *n* bits moves the right-hand 32-*n* bits of the register *Rm*, to the left by *n* places, into the left-hand 32-*n* bits of the result. And it sets the right-hand *n* bits of the result to 0. See Figure 3-3.

You can use the `LSL #n` operation to multiply the value in the register *Rm* by 2^n, if the value is regarded as an unsigned integer or a two's complement signed integer. Overflow can occur without warning.

When the instruction is `LSLS` or when `LSL #n`, with non-zero *n*, is used in *Operand2* with the instructions `MOVS`, `MVNS`, `ANDS`, `ORRS`, `ORNS`, `EORS`, `BICS`, `TEQ` or `TST`, the carry flag is updated to the last bit shifted out, bit[32-*n*], of the register *Rm*. These instructions do not affect the carry flag when used with `LSL #0`.

Note

- If *n* is 32 or more, then all the bits in the result are cleared to 0.
- If *n* is 33 or more and the carry flag is updated, it is updated to 0.

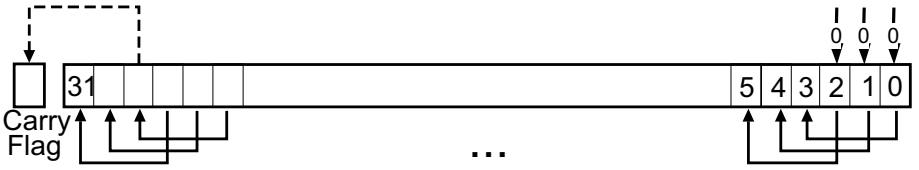


Figure 3-3 LSL #3

3.8.4 ROR

Rotate right by *n* bits moves the left-hand 32-*n* bits of the register *Rm*, to the right by *n* places, into the right-hand 32-*n* bits of the result. And it moves the right-hand *n* bits of the register into the left-hand *n* bits of the result. See Figure 3-4 on page 3-17.

When the instruction is `RORS` or when `ROR #n` is used in *Operand2* with the instructions `MOVS`, `MVNS`, `ANDS`, `ORRS`, `ORNS`, `EORS`, `BICS`, `TEQ` or `TST`, the carry flag is updated to the last bit rotation, bit[*n*-1], of the register *Rm*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Note

- If *n* is 32, then the value of the result is same as the value in *Rm*, and if the carry flag is updated, it is updated to bit[31] of *Rm*.
- ROR with shift length, *n*, more than 32 is the same as ROR with shift length *n*-32.

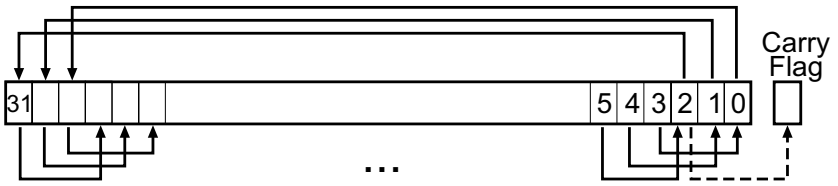


Figure 3-4 ROR #3

3.8.5 RRX

Rotate right with extend moves the bits of the register *Rm* to the right by one bit. And it copies the carry flag into bit[31] of the result. See Figure 3-5.

When the instruction is RRXS or when RRX is used in *Operand2* with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to bit[0] of the register *Rm*.

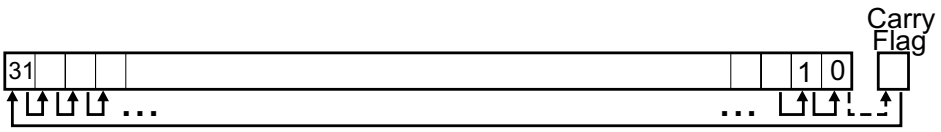


Figure 3-5 RRX

3.8.6 See also

Concepts

- *Flexible second operand (Operand2)* on page 3-12.

Reference

- *Operand2 as a constant* on page 3-13.
- *Operand2 as a register with optional shift* on page 3-14.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.9 Multiply instructions

The following topics describe the multiply instructions:

- *MUL, MLA, and MLS* on page 3-110

  Multiply, Multiply Accumulate, and Multiply Subtract (32-bit by 32-bit, bottom 32-bit result).

- *UMULL, UMLAL, SMULL, and SMLAL* on page 3-164

  Unsigned and signed Long Multiply and Multiply Accumulate (32-bit by 32-bit, 64-bit result or 64-bit accumulator).

- *SMULxy and SMLAxy* on page 3-146

  Signed Multiply and Signed Multiply Accumulate (16-bit by 16-bit, 32-bit result).

- *SMULWy and SMLAWy* on page 3-145

  Signed Multiply and Signed Multiply Accumulate (32-bit by 16-bit, top 32-bit result).

- *SMLALxy* on page 3-139

  Signed Multiply Accumulate (16-bit by 16-bit, 64-bit accumulate).

- *SMUAD{X} and SMUSD{X}* on page 3-144

  Dual 16-bit Signed Multiply with Addition or Subtraction of products.

- *SMMUL, SMMLA, and SMMLS* on page 3-142

  Multiply, Multiply Accumulate, and Multiply Subtract (32-bit by 32-bit, top 32-bit result).

- *SMLAD and SMLSD* on page 3-137

  Dual 16-bit Signed Multiply, 32-bit Accumulation of Sum or Difference of 32-bit products.

- *SMLALD and SMLSLD* on page 3-140

  Dual 16-bit Signed Multiply, 64-bit Accumulation of Sum or Difference of 32-bit products.

- *UMAAL* on page 3-163

  Unsigned Multiply Accumulate Accumulate Long.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.10 Saturating instructions

The saturating instructions are:

- QADD.
- QDADD.
- QDSUB.
- QSUB.
- SSAT.
- USAT.

Some of the parallel instructions are also saturating.

3.10.1 Saturating arithmetic

These operations are *saturating* (SAT). This means that, for some value of 2^n that depends on the instruction:

- For a signed saturating operation, if the full result would be less than -2^n, the result returned is -2^n.
- For an unsigned saturating operation, if the full result would be negative, the result returned is zero.
- If the full result would be greater than $2^n - 1$, the result returned is $2^n - 1$.

When any of these things occurs, it is called *saturation*. Some instructions set the Q flag when saturation occurs.

Note

Saturating instructions do not clear the Q flag when saturation does not occur. To clear the Q flag, use an MSR instruction.

The Q flag can also be set by two other instructions, but these instructions do not saturate.

3.10.2 See also

Reference

- *MSR (general-purpose register to PSR)* on page 3-108.
- *QADD, QSUB, QDADD, and QDSUB* on page 3-123.
- *SMULxy and SMLAxy* on page 3-146.
- *SMULWy and SMLAWy* on page 3-145.
- *SSAT and USAT* on page 3-150.
- *Parallel instructions* on page 3-20.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.11 Parallel instructions

The following topics describe the parallel instructions:

- *Parallel add and subtract* on page 3-114

  Various byte-wise and halfword-wise additions and subtractions.

- *USAD8 and USADA8* on page 3-167

  Unsigned sum of absolute differences, and accumulate unsigned sum of absolute differences.

- *SSAT16 and USAT16* on page 3-152

  Parallel halfword saturating instructions.

There are also parallel unpacking instructions such as `SXT`, `SXTA`, `UXT`, and `UXTA`.

3.11.1 See also

Reference

- *SXT, SXTA, UXT, and UXTA* on page 3-157.
- *Packing and unpacking instructions* on page 3-21.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.12 Packing and unpacking instructions

The following topics describe the packing and unpacking instructions:

- *BFC and BFI* on page 3-47
  Bit Field Clear and Bit Field Insert.
- *SBFX and UBFX* on page 3-129
  Signed or Unsigned Bit Field extract.
- *SXT, SXTA, UXT, and UXTA* on page 3-157
  Sign Extend or Zero Extend instructions, with optional Add.
- *PKHBT and PKHTB* on page 3-117
  Halfword Packing instructions.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.13 Branch and control instructions

The following topics describe the branch and control instructions:

- *B, BL, BX, and BLX* on page 3-44

  Branch, Branch with Link, Branch and exchange instruction set, Branch with Link and exchange instruction set.

- *IT* on page 3-66

  If-Then. `IT` makes a single subsequent 16-bit instruction from a restricted set conditional. `IT` can also make between two and four subsequent instructions conditional, with either the same condition, or some with one condition and others with the inverse condition, but this is deprecated.

- *CBZ and CBNZ* on page 3-49

  Compare against zero and branch.

- *TBB and TBH* on page 3-160

  Table Branch Byte or Halfword.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.14 Coprocessor instructions

The following topics describe the coprocessor instructions:

- *MCR and MCRR* on page 3-97

  Move to Coprocessor from ARM Register or Registers, possibly with coprocessor operations.

- *MRC and MRRC* on page 3-103

  Move to ARM Register or Registers from Coprocessor, possibly with coprocessor operations.

- *MSR (ARM register to system coprocessor register)* on page 3-107

  Move to system coprocessor from ARM register.

- *MRS (system coprocessor register to ARM register)* on page 3-104

  Move to ARM register from system coprocessor.

- *SYS* on page 3-159

  Execute system coprocessor instruction.

- *LDC and STC* on page 3-69

  Transfer Data between memory and Coprocessor.

Note

A coprocessor instruction causes an Undefined Instruction exception if the specified coprocessor is not present, or if it is not enabled.

3.14.1 See also

Reference

- Chapter 4 *Advanced SIMD and Floating-point Programming (32-bit)*.
- *Miscellaneous instructions* on page 3-24.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.15 Miscellaneous instructions

The following topics describe miscellaneous instructions:

- *BKPT* on page 3-48

  Software breakpoint.

- *HLT* on page 3-64

  Halting debug-mode breakpoint.

- *SVC* on page 3-156

  Supervisor Call (formerly `SWI`).

- *HVC* on page 3-65

  Hypervisor Call.

- *MRS (PSR to general-purpose register)* on page 3-105

  Move the contents of the CPSR or SPSR to a general-purpose register.

- *MSR (general-purpose register to PSR)* on page 3-108

  Load specified fields of the CPSR or SPSR with an immediate value, or from the contents of a general-purpose register.

- *CPS* on page 3-54

  Change Processor State.

- *SMC* on page 3-136

  Secure Monitor Call (formerly `SMI`).

- *SETEND* on page 3-133

  Set the Endianness bit in the CPSR.

- *NOP* on page 3-113

  No Operation.

- *SEV, SEVL, WFE, WFI, and YIELD* on page 3-134

  Set Event, Wait For Event, Wait for Interrupt, and Yield hint instructions.

- *DBG* on page 3-56

  Debug.

- *DCPS1 (T32 instruction)* on page 3-57

  Debug switch to exception level 1.

- *DCPS2 (T32 instruction)* on page 3-58

  Debug switch to exception level 2.

- *DCPS3 (T32 instruction)* on page 3-59

  Debug switch to exception level 3.

- *DMB, DSB, and ISB* on page 3-60

  Data Memory Barrier, Data Synchronization Barrier, and Instruction Synchronization Barrier hint instructions.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.16 Pseudo-instructions

The ARM assembler supports a number of pseudo-instructions that are translated into the appropriate combination of A32 or T32 instructions at assembly time.

The following topics describe the pseudo-instructions:

- *ADRL pseudo-instruction* on page 3-36

  Load a PC-relative or register-relative address into a register (medium range, position independent).

- *CPY pseudo-instruction* on page 3-55

  Copy a value from one register to another.

- *LDR pseudo-instruction* on page 3-86

  Load a register with a 32-bit immediate value or an address (unlimited range, but not position independent).

- *MOV32 pseudo-instruction* on page 3-102

  Load a register with a 32-bit immediate value or an address (unlimited range, but not position independent).

- *NEG pseudo-instruction* on page 3-112

  Negate a value in a register.

- *UND pseudo-instruction* on page 3-166

  Generate an architecturally undefined instruction.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.17 Condition codes

The instructions that can be conditional have an optional condition code, shown in syntax descriptions as {*cond*}. Table 3-2 shows the condition codes that you can use.

Table 3-2 Condition code suffixes

Suffix	Meaning
EQ	Equal
NE	Not equal
CS	Carry set (identical to HS)
HS	Unsigned higher or same (identical to CS)
CC	Carry clear (identical to LO)
LO	Unsigned lower (identical to CC)
MI	Minus or negative result
PL	Positive or zero result
VS	Overflow
VC	No overflow
HI	Unsigned higher
LS	Unsigned lower or same
GE	Signed greater than or equal
LT	Signed less than
GT	Signed greater than
LE	Signed less than or equal
AL	Always (this is the default)

Note

The precise meanings of the condition codes depend on whether the condition flags were set by a floating-point instruction or by an A32/T32 data processing instruction.

3.17.1 See also

Concepts

armasm User Guide:

- *Comparison of condition code meanings in integer and floating-point code* on page 8-14.
- *Conditional execution of A32/T32 Advanced SIMD and floating-point instructions* on page 11-15.

Reference

- *IT* on page 3-66.
- *VMRS and VMSR* on page 4-75.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.18 ADD, SUB, RSB, ADC, SBC, and RSC

Add, Subtract, and Reverse Subtract, each with or without Carry.

3.18.1 Syntax

```
op{S}{cond} {Rd}, Rn, Operand2
op{cond} {Rd}, Rn, #imm12              ; T32, 32-bit encodings of ADD and SUB only
```

where:

op is one of:

- ADD — Add.
- ADC — Add with Carry.
- SUB — Subtract.
- RSB — Reverse Subtract.
- SBC — Subtract with Carry.
- RSC — Reverse Subtract with Carry (A32 only).

S — is an optional suffix. If S is specified, the condition flags are updated on the result of the operation.

cond — is an optional condition code.

Rd — is the destination register.

Rn — is the register holding the first operand.

Operand2 — is a flexible second operand.

imm12 — is any value in the range 0-4095.

3.18.2 Usage

The ADD instruction adds the values in *Rn* and *Operand2* or *imm12*.

The SUB instruction subtracts the value of *Operand2* or *imm12* from the value in *Rn*.

The RSB (Reverse Subtract) instruction subtracts the value in *Rn* from the value of *Operand2*. This is useful because of the wide range of options for *Operand2*.

You can use ADC, SBC, and RSC to synthesize multiword arithmetic.

The ADC (Add with Carry) instruction adds the values in *Rn* and *Operand2*, together with the carry flag.

The SBC (Subtract with Carry) instruction subtracts the value of *Operand2* from the value in *Rn*. If the carry flag is clear, the result is reduced by one.

The RSC (Reverse Subtract with Carry) instruction subtracts the value in *Rn* from the value of *Operand2*. If the carry flag is clear, the result is reduced by one.

In certain circumstances, the assembler can substitute one instruction for another. Be aware of this when reading disassembly listings.

3.18.3 Use of PC and SP in T32 instructions

In most of these instructions, you cannot use PC (R15) for *Rd*, or any operand.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

The exceptions are:

- You can use PC for *Rn* in 32-bit encodings of T32 ADD and SUB instructions, with a constant *Operand2* value in the range 0-4095, and no S suffix. These instructions are useful for generating PC-relative addresses. Bit[1] of the PC value reads as 0 in this case, so that the base address for the calculation is always word-aligned.
- You can use PC in 16-bit encodings of T32 ADD{*cond*} Rd, Rd, Rm instructions, where both registers cannot be PC. However, the following 16-bit T32 instructions are deprecated:
  - ADD{*cond*} PC, SP, PC.
  - ADD{*cond*} SP, SP, PC.

In most of these instructions, you cannot use SP (R13) for *Rd*, or any operand. Except that:

- You can use SP for Rn in ADD and SUB instructions.
- ADD{*cond*} SP, SP, SP is permitted but is deprecated.
- ADD{S}{cond} SP, SP, Rm{,*shift*} and SUB{S}{cond} SP, SP, Rm{,*shift*} are permitted if *shift* is omitted or LSL #1, LSL #2, or LSL #3.

3.18.4 Use of PC and SP in A32 instructions

You cannot use PC for *Rd* or any operand in any data processing instruction that has a register-controlled shift.

With the exception of ADD and SUB, use of PC for any operand, in instructions without register-controlled shift, is deprecated.

In SUB instructions without register-controlled shift, use of PC is deprecated except for the following cases:

- Use of PC for R*d* .
- Use of PC for R*n* in the instruction SUB{*cond*} R*d*, R*n*, #Constant.

In ADD instructions without register-controlled shift, use of PC is deprecated except for the following cases:

- Use of PC for R*d* in instructions that do not add SP to a register.
- Use of PC for R*n* and use of PC for R*m* in instructions that add two registers other than SP.
- Use of PC for R*n* in the instruction ADD{*cond*} R*d*, R*n*, #Constant.

If you use PC (R15) as *Rn* or *Rm*, the value used is the address of the instruction plus 8.

If you use PC as *Rd*:

- Execution branches to the address corresponding to the result.
- If you use the S suffix, see the SUBS pc,lr instruction.

You can use SP for *Rn* in ADD and SUB instructions, however, ADDS PC, SP, #Constant and SUBS PC, SP, #Constant are deprecated.

You can use SP in ADD (register) and SUB (register) if *Rn* is SP and *shift* is omitted or LSL #1, LSL #2, or LSL #3.

Other uses of SP in these A32 instructions are deprecated.

3.18.5 Condition flags

If S is specified, these instructions update the N, Z, C and V flags according to the result.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.18.6 16-bit instructions

The following forms of these instructions are available in T32 code, and are 16-bit instructions:

`ADDS Rd, Rn, #imm`

imm range 0-7. *Rd* and *Rn* must both be Lo registers. This form can only be used outside an IT block.

`ADD{cond} Rd, Rn, #imm`

imm range 0-7. *Rd* and *Rn* must both be Lo registers. This form can only be used inside an IT block.

`ADDS Rd, Rn, Rm`

Rd, *Rn* and *Rm* must all be Lo registers. This form can only be used outside an IT block.

`ADD{cond} Rd, Rn, Rm`

Rd, *Rn* and *Rm* must all be Lo registers. This form can only be used inside an IT block.

`ADD Rd, Rd, Rm`

Rd and *Rm* can be Hi or Lo registers.

`ADDS Rd, Rd, #imm`

imm range 0-255. *Rd* must be a Lo register. This form can only be used outside an IT block.

`ADD{cond} Rd, Rd, #imm`

imm range 0-255. *Rd* must be a Lo register. This form can only be used inside an IT block.

`ADCS Rd, Rd, Rm`

Rd and *Rm* must both be Lo registers. This form can only be used outside an IT block.

`ADC{cond} Rd, Rd, Rm`

Rd and *Rm* must both be Lo registers. This form can only be used inside an IT block.

`ADD SP, SP, #imm`

imm range 0-508, word aligned.

`ADD Rd, SP, #imm`

imm range 0-1020, word aligned. *Rd* must be a Lo register.

`ADD Rd, pc, #imm`

imm range 0-1020, word aligned. *Rd* must be a Lo register. Bits[1:0] of the PC are read as 0 in this instruction.

`SUBS Rd, Rn, Rm`

Rd, *Rn* and *Rm* must all be Lo registers. This form can only be used outside an IT block.

`SUB{cond} Rd, Rn, Rm`

Rd, *Rn* and *Rm* must all be Lo registers. This form can only be used inside an IT block.

`SUBS Rd, Rn, #imm`

imm range 0-7. *Rd* and *Rn* must both be both Lo registers. This form can only be used outside an IT block.

`SUB{cond} Rd, Rn, #imm`

imm range 0-7. *Rd* and *Rn* must both be Lo registers. This form can only be used inside an IT block.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

`SUBS Rd, Rd, #imm`

imm range 0-255. *Rd* must be a Lo register. This form can only be used outside an IT block.

`SUB{cond} Rd, Rd, #imm`

imm range 0-255. *Rd* must be a Lo register. This form can only be used inside an IT block.

`SUB{cond} SP, SP, #imm`

imm range 0-508, word aligned.

`SBCS Rd, Rd, Rm`

Rd and *Rm* must both be Lo registers. This form can only be used outside an IT block.

`SBC{cond} Rd, Rd, Rm`

Rd and *Rm* must both be Lo registers. This form can only be used inside an IT block.

`RSBS Rd, Rn, #0`

Rd and *Rn* must both be both Lo registers. This form can only be used outside an IT block.

`RSB{cond} Rd, Rn, #0`

Rd and *Rn* must both be Lo registers. This form can only be used inside an IT block.

3.18.7 Examples

```
ADD     r2, r1, r3
SUBS    r8, r6, #240        ; sets the flags on the result
RSB     r4, r4, #1280       ; subtracts contents of R4 from 1280
ADCHI   r11, r0, r3         ; only executed if C flag set and Z
                            ; flag clear
RSCSLE  r0,r5,r0,LSL r4     ; conditional, flags set
```

3.18.8 Incorrect example

```
RSCSLE  r0,pc,r0,LSL r4    ; PC not permitted with register
                            ; controlled shift
```

3.18.9 Multiword arithmetic examples

These two instructions add a 64-bit integer contained in R2 and R3 to another 64-bit integer contained in R0 and R1, and place the result in R4 and R5.

```
ADDS    r4, r0, r2    ; adding the least significant words
ADC     r5, r1, r3    ; adding the most significant words
```

These instructions subtract one 96-bit integer from another:

```
SUBS    r3, r6, r9
SBCS    r4, r7, r10
SBC     r5, r8, r11
```

For clarity, these examples use consecutive registers for multiword values. There is no requirement to do this. The following, for example, is perfectly valid:

```
SUBS    r6, r6, r9
SBCS    r9, r2, r1
SBC     r2, r8, r11
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.18.10 See also

Concepts

- *Flexible second operand (Operand2)* on page 3-12.
- *Instruction substitution* on page 3-13.

Reference

- *Parallel add and subtract* on page 3-114.
- *ADR (PC-relative)* on page 3-32.
- *ADR (register-relative)* on page 3-34.
- *ADRL pseudo-instruction* on page 3-36.
- *SUBS pc, lr* on page 3-154.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.19 ADR (PC-relative)

ADR generates a PC-relative address in the destination register, for a label in the current area.

3.19.1 Syntax

ADR{*cond*}{.W} *Rd*,*label*

where:

cond is an optional condition code.

.W is an optional instruction width specifier.

Rd is the destination register to load.

label is a PC-relative expression.

label must be within a limited distance of the current instruction.

3.19.2 Usage

ADR produces position-independent code, because the assembler generates an instruction that adds or subtracts a value to the PC.

Use the ADRL pseudo-instruction to assemble a wider range of effective addresses.

label must evaluate to an address in the same assembler area as the ADR instruction.

If you use ADR to generate a target for a BX or BLX instruction, it is your responsibility to set the T32 bit (bit 0) of the address if the target contains T32 instructions.

3.19.3 Offset range and architectures

The assembler calculates the offset from the PC for you. The assembler generates an error if *label* is out of range.

Table 3-3 shows the possible offsets between the label and the current instruction.

Table 3-3 PC-relative offsets

Instruction	Offset range
A32 ADR	See *Operand2 as a constant* on page 3-13
32-bit T32 encoding ADR	+/– 4095
16-bit T32 encoding ADR [a]	0-1020 [b]

a. Rd must be in the range R0-R7.

b. Must be a multiple of 4.

3.19.4 ADR in T32

You can use the .W width specifier to force ADR to generate a 32-bit instruction in T32 code. ADR with .W always generates a 32-bit instruction, even if the address can be generated in a 16-bit instruction.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

For forward references, ADR without .W always generates a 16-bit instruction in T32 code, even if that results in failure for an address that could be generated in a 32-bit T32 ADD instruction.

3.19.5 Restrictions

In T32 code, *Rd* cannot be PC or SP.

In A32 code, *Rd* can be PC or SP but use of SP is deprecated.

3.19.6 See also

Concepts

armasm User Guide:

- *Register-relative and PC-relative expressions* on page 10-7.

Reference

- *Memory access instructions* on page 3-9.
- *ADRL pseudo-instruction* on page 3-36.
- *AREA* on page 10-13.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.20 ADR (register-relative)

ADR generates a register-relative address in the destination register, for a label defined in a storage map.

3.20.1 Syntax

ADR{*cond*}{.W} *Rd*,*label*

where:

cond is an optional condition code.

.W is an optional instruction width specifier.

Rd is the destination register to load.

label is a symbol defined by the FIELD directive. *label* specifies an offset from the base register which is defined using the MAP directive.

label must be within a limited distance from the base register.

3.20.2 Usage

ADR generates code to easily access named fields inside a storage map.

Use the ADRL pseudo-instruction to assemble a wider range of effective addresses.

3.20.3 Restrictions

In T32 code:

- R*d* cannot be PC.
- R*d* can be SP only if the base register is SP.

3.20.4 Offset range and architectures

The assembler calculates the offset from the base register for you. The assembler generates an error if *label* is out of range.

Table 3-4 shows the possible offsets between label and the current instruction.

Table 3-4 register-relative offsets

Instruction	Offset range
A32 ADR	See *Operand2 as a constant* on page 3-13
32-bit T32 encoding ADR	+/– 4095
16-bit T32 encoding ADR, base register is SP [a]	0-1020 [b]

a. *Rd* must be in the range R0-R7 or SP. If *Rd* is SP, the offset range is –508 to 508 and must be a multiple of 4

b. Must be a multiple of 4.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.20.5 ADR in T32

You can use the `.W` width specifier to force `ADR` to generate a 32-bit instruction in T32 code. `ADR` with `.W` always generates a 32-bit instruction, even if the address can be generated in a 16-bit instruction.

For forward references, `ADR` without `.W`, with base register SP, always generates a 16-bit instruction in T32 code, even if that results in failure for an address that could be generated in a 32-bit T32 `ADD` instruction.

3.20.6 See also

Concepts

armasm User Guide:

- *Register-relative and PC-relative expressions* on page 10-7.

Reference

- *Memory access instructions* on page 3-9.
- *MAP* on page 10-69.
- *FIELD* on page 10-39.
- *ADRL pseudo-instruction* on page 3-36.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.21 ADRL pseudo-instruction

Load a PC-relative or register-relative address into a register. It is similar to the ADR instruction. ADRL can load a wider range of addresses than ADR because it generates two data processing instructions.

3.21.1 Syntax

ADRL{*cond*} *Rd*,*label*

where:

cond is an optional condition code.

Rd is the register to load.

label is a PC-relative or register-relative expression.

3.21.2 Usage

ADRL always assembles to two 32-bit instructions. Even if the address can be reached in a single instruction, a second, redundant instruction is produced.

If the assembler cannot construct the address in two instructions, it generates an error message and the assembly fails. You can use the LDR pseudo-instruction for loading a wider range of addresses.

ADRL produces position-independent code, because the address is PC-relative or register-relative.

If *label* is PC-relative, it must evaluate to an address in the same assembler area as the ADRL pseudo-instruction.

If you use ADRL to generate a target for a BX or BLX instruction, it is your responsibility to set the T32 bit (bit 0) of the address if the target contains T32 instructions.

3.21.3 Availability and range

The available range depends on the instruction set in use:

A32 The range of the instruction is any value that can be generated by two ADD or two SUB instructions. That is, any value that can be produced by the addition of two values, each of which is 8 bits rotated right by any even number of bits within a 32-bit word. See *Operand2 as a constant* on page 3-13 for more information.

32-bit T32 encoding

±1MB bytes to a byte, halfword, or word-aligned address.

16-bit T32 encoding

ADRL is not available.

The given range is relative to a point four bytes (in T32 code) or two words (in A32 code) after the address of the current instruction.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.21.4 See also

Concepts

armasm User Guide:

- *Register-relative and PC-relative expressions* on page 10-7.
- *Load immediates into registers* on page 7-6.

Reference

- *LDR pseudo-instruction* on page 3-86.
- *AREA* on page 10-13.
- *ADD, SUB, RSB, ADC, SBC, and RSC* on page 3-27.
- *Condition codes* on page 3-26.

Other information

- *ARM Architecture Reference Manual*
  `http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/`.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.22 AND, ORR, EOR, BIC, and ORN

Logical AND, OR, Exclusive OR, Bit Clear, and OR NOT.

3.22.1 Syntax

op{S}{*cond*} *Rd*, *Rn*, *Operand2*

where:

op is one of:

- AND logical AND.
- ORR logical OR.
- EOR logical Exclusive OR.
- BIC logical AND NOT.
- ORN logical OR NOT (T32 only).

S is an optional suffix. If S is specified, the condition flags are updated on the result of the operation.

cond is an optional condition code.

Rd is the destination register.

Rn is the register holding the first operand.

Operand2 is a flexible second operand.

3.22.2 Usage

The AND, EOR, and ORR instructions perform bitwise AND, Exclusive OR, and OR operations on the values in *Rn* and *Operand2*.

The BIC (Bit Clear) instruction performs an AND operation on the bits in *Rn* with the complements of the corresponding bits in the value of *Operand2*.

The ORN T32 instruction performs an OR operation on the bits in *Rn* with the complements of the corresponding bits in the value of *Operand2*.

In certain circumstances, the assembler can substitute BIC for AND, AND for BIC, ORN for ORR, or ORR for ORN. Be aware of this when reading disassembly listings.

3.22.3 Use of PC in T32 instructions

You cannot use PC (R15) for *Rd* or any operand in any of these instructions.

3.22.4 Use of PC and SP in A32 instructions

Using PC and SP in these A32 instructions is deprecated.

If you use PC as *Rn*, the value used is the address of the instruction plus 8.

If you use PC as *Rd*:

- Execution branches to the address corresponding to the result.
- If you use the S suffix, see the SUBS pc,lr instruction.

You cannot use PC for any operand in any data processing instruction that has a register-controlled shift.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.22.5 Condition flags

If `S` is specified, these instructions:

- Update the N and Z flags according to the result.
- Can update the C flag during the calculation of *Operand2*.
- Do not affect the V flag.

3.22.6 16-bit instructions

The following forms of these instructions are available in T32 code, and are 16-bit instructions:

`ANDS` *Rd*, *Rd*, *Rm*

Rd and *Rm* must both be Lo registers. This form can only be used outside an IT block.

`AND{`*cond*`}` *Rd*, *Rd*, *Rm*

Rd and *Rm* must both be Lo registers. This form can only be used inside an IT block.

`EORS` *Rd*, *Rd*, *Rm*

Rd and *Rm* must both be Lo registers. This form can only be used outside an IT block.

`EOR{`*cond*`}` *Rd*, *Rd*, *Rm*

Rd and *Rm* must both be Lo registers. This form can only be used inside an IT block.

`ORRS` *Rd*, *Rd*, *Rm*

Rd and *Rm* must both be Lo registers. This form can only be used outside an IT block.

`ORR{`*cond*`}` *Rd*, *Rd*, *Rm*

Rd and *Rm* must both be Lo registers. This form can only be used inside an IT block.

`BICS` *Rd*, *Rd*, *Rm*

Rd and *Rm* must both be Lo registers. This form can only be used outside an IT block.

`BIC{`*cond*`}` *Rd*, *Rd*, *Rm*

Rd and *Rm* must both be Lo registers. This form can only be used inside an IT block.

Apart from `BIC{S}`, it does not matter if you specify *OP*`{S}` *Rd*, *Rm*, *Rd*. The instruction is the same.

3.22.7 Examples

```
AND     r9,r2,#0xFF00
ORREQ   r2,r0,r5
EORS    r0,r0,r3,ROR r6
ANDS    r9, r8, #0x19
EORS    r7, r11, #0x18181818
BIC     r0, r1, #0xab
ORN     r7, r11, lr, ROR #4
ORNS    r7, r11, lr, ASR #32
```

3.22.8 Incorrect example

```
EORS    r0,pc,r3,ROR r6     ; PC not permitted with register
                            ; controlled shift
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.22.9 See also

Concepts

- *Flexible second operand (Operand2)* on page 3-12.
- *Instruction substitution* on page 3-13.

Reference

- *SUBS pc, lr* on page 3-154.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.23 ASR, LSL, LSR, ROR, and RRX

Arithmetic Shift Right, Logical Shift Left, Logical Shift Right, Rotate Right, and Rotate Right with Extend.

These instructions are the preferred synonyms for `MOV` instructions with shifted register operands.

3.23.1 Syntax

op{S}{*cond*} *Rd*, *Rm*, *Rs*

op{S}{*cond*} *Rd*, *Rm*, *#sh*

RRX{S}{*cond*} *Rd*, *Rm*

where:

op is one of `ASR`, `LSL`, `LSR`, or `ROR`.

S is an optional suffix. If `S` is specified, the condition flags are updated on the result of the operation.

Rd is the destination register.

Rm is the register holding the first operand. This operand is shifted right or left.

Rs is a register holding a shift value to apply to the value in *Rm*. Only the least significant byte is used.

sh is a constant shift. The range of values permitted depends on the instruction:

- `ASR` permitted shifts 1-32
- `LSL` permitted shifts 0-31
- `LSR` permitted shifts 1-32
- `ROR` permitted shifts 1-31.

3.23.2 Usage

`ASR` provides the signed value of the contents of a register divided by a power of two. It copies the sign bit into vacated bit positions on the left.

`LSL` provides the value of a register multiplied by a power of two. `LSR` provides the unsigned value of a register divided by a variable power of two. Both instructions insert zeros into the vacated bit positions.

`ROR` provides the value of the contents of a register rotated by a value. The bits that are rotated off the right end are inserted into the vacated bit positions on the left.

`RRX` provides the value of the contents of a register shifted right one bit. The old carry flag is shifted into bit[31]. If the `S` suffix is present, the old bit[0] is placed in the carry flag.

3.23.3 Restrictions in T32 code

T32 instructions must not use PC or SP.

You cannot specify zero for the *sh* value in an `LSL` instruction in an IT block.

3.23.4 Use of SP and PC in A32 ASR, LSL, LSR, ROR, and RRX instructions

Using SP in these A32 instructions is deprecated.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

You cannot use PC in instructions with the `op{S}{cond} Rd, Rm, Rs` syntax. Using PC for `Rd` and `Rm` in the other syntaxes is deprecated.

If you use PC as *Rm*, the value used is the address of the instruction plus 8.

If you use PC as *Rd*:

- Execution branches to the address corresponding to the result.
- If you use the `S` suffix, the SPSR of the current mode is copied to the CPSR. You can use this to return from exceptions.

Note

The A32 instructions *op*`S{`*cond*`} pc,Rm,#sh` and `RRXS{cond} pc,Rm` always disassemble to the preferred form `MOVS{`*cond*`} pc,Rm{,shift}`.

Caution

Do not use the `S` suffix when using PC as *Rd* in User mode or System mode. The assembler cannot warn you about this because it has no information about what the processor mode is likely to be at execution time.

You cannot use PC for *Rd* or any operand in any of these instructions if they have a register-controlled shift.

3.23.5 Condition flags

If `S` is specified, these instructions update the N and Z flags according to the result.

The C flag is unaffected if the shift value is 0. Otherwise, the C flag is updated to the last bit shifted out.

3.23.6 16-bit instructions

The following forms of these instructions are available in T32 code, and are 16-bit instructions:

`ASRS` *Rd*, *Rm*, *#sh*

Rd and *Rm* must both be Lo registers. This form can only be used outside an IT block.

`ASR{`*cond*`}` *Rd*, *Rm*, *#sh*

Rd and *Rm* must both be Lo registers. This form can only be used inside an IT block.

`ASRS` *Rd*, *Rd*, *Rs*

Rd and *Rs* must both be Lo registers. This form can only be used outside an IT block.

`ASR{`*cond*`}` *Rd*, *Rd*, *Rs*

Rd and *Rs* must both be Lo registers. This form can only be used inside an IT block.

`LSLS` *Rd*, *Rm*, *#sh*

Rd and *Rm* must both be Lo registers. This form can only be used outside an IT block.

`LSL{`*cond*`}` *Rd*, *Rm*, *#sh*

Rd and *Rm* must both be Lo registers. This form can only be used inside an IT block.

`LSLS` *Rd*, *Rd*, *Rs*

Rd and *Rs* must both be Lo registers. This form can only be used outside an IT block.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

`LSL{cond} Rd, Rd, Rs`

Rd and *Rs* must both be Lo registers. This form can only be used inside an IT block.

`LSRS Rd, Rm, #sh`

Rd and *Rm* must both be Lo registers. This form can only be used outside an IT block.

`LSR{cond} Rd, Rm, #sh`

Rd and *Rm* must both be Lo registers. This form can only be used inside an IT block.

`LSRS Rd, Rd, Rs`

Rd and *Rs* must both be Lo registers. This form can only be used outside an IT block.

`LSR{cond} Rd, Rd, Rs`

Rd and *Rs* must both be Lo registers. This form can only be used inside an IT block.

`RORS Rd, Rd, Rs`

Rd and *Rs* must both be Lo registers. This form can only be used outside an IT block.

`ROR{cond} Rd, Rd, Rs`

Rd and *Rs* must both be Lo registers. This form can only be used inside an IT block.

3.23.7 Availability

These instructions are available in A32 and T32.

In T32, these instructions are available in 16-bit and 32-bit encodings.

There is no 16-bit `RRX` instruction in T32.

3.23.8 Examples

```
ASR     r7, r8, r9
LSLS    r1, r2, r3
LSR     r4, r5, r6
ROR     r4, r5, r6
```

3.23.9 See also

Reference

- *MOV and MVN* on page 3-98.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.24 B, BL, BX, and BLX

Branch, Branch with Link, Branch and exchange instruction set, Branch with Link and exchange instruction set.

3.24.1 Syntax

op1{*cond*}{.W} *label*

op2{*cond*} *Rm*

where:

op1 is one of:

- B — Branch.
- BL — Branch with link.
- BLX — Branch with link, and exchange instruction set.

op2 is one of:

- BX — Branch and exchange instruction set.
- BLX — Branch with link, and exchange instruction set.

cond is an optional condition code. *cond* is not available on all forms of this instruction.

.W is an optional instruction width specifier to force the use of a 32-bit B instruction in T32.

label is a PC-relative expression.

Rm is a register containing an address to branch to.

3.24.2 Operation

All these instructions cause a branch to *label*, or to the address contained in *Rm*. In addition:

- The BL and BLX instructions copy the address of the next instruction into LR (R14, the link register).
- The BX and BLX instructions can change the instruction set.

  BLX *label* always changes the instruction set. It changes a processor in A32 state to T32 state, or a processor in T32 state to A32 state.

  BX *Rm* and BLX *Rm* derive the target instruction set from bit[0] of *Rm*:
  - If bit[0] of *Rm* is 0, the processor changes to, or remains in, A32 state.
  - If bit[0] of *Rm* is 1, the processor changes to, or remains in, T32 state.

Note

There are no equivalent instructions to BX and BLX to change between AArch32 and AArch64 state. The only way to change execution state is by a change of exception level.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.24.3 Instruction availability and branch ranges

Table 3-5 shows the instructions that are available in A32 and T32 state. Instructions that are not shown in this table are not available.

Table 3-5 Branch instruction availability and range

Instruction	A32	16-bit T32 encoding	32-bit T32 encoding
`B label`	±32MB	±2KB	±16MB[a]
`B{cond} label`	±32MB	–252 to +258	±1MB[a]
`BL label`	±32MB	±4MB [b]	±16MB
`BL{cond} label`	±32MB	-	-
`BX Rm`	Available	Available	Use 16-bit
`BX{cond} Rm`	Available	-	-
`BLX label`	±32MB	±4MB [b]	±16MB
`BLX Rm`	Available	Available	Use 16-bit
`BLX{cond} Rm`	Available	-	-

a. Use `.W` to instruct the assembler to use this 32-bit instruction.
b. This is an instruction pair.

3.24.4 Extending branch ranges

Machine-level `B` and `BL` instructions have restricted ranges from the address of the current instruction. However, you can use these instructions even if *`label`* is out of range. Often you do not know where the linker places *`label`*. When necessary, the linker adds code to enable longer branches. The added code is called a *veneer*.

3.24.5 B in T32

You can use the `.W` width specifier to force `B` to generate a 32-bit instruction in T32 code.

`B.W` always generates a 32-bit instruction, even if the target could be reached using a 16-bit instruction.

For forward references, `B` without `.W` always generates a 16-bit instruction in T32 code, even if that results in failure for a target that could be reached using a 32-bit T32 instruction.

3.24.6 Register restrictions

Using PC for `Rm` in the A32 `BX` instruction is deprecated. You cannot use PC in other A32 instructions.

You can use PC for `Rm` in the T32 `BX` instruction. You cannot use PC in other T32 instructions.

Using SP for `Rm` in these A32 instructions is deprecated.

Using SP for `Rm` in the T32 `BX` and `BLX` instructions is deprecated. You cannot use SP in the other T32 instructions.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.24.7 Condition flags

These instructions do not change the flags.

3.24.8 Availability

See Table 3-5 on page 3-45 for details of availability of these instructions in both instruction sets.

3.24.9 Examples

```
B       loopA
BLE     ng+8
BL      subC
BLLT    rtX
BEQ     {PC}+4  ; #0x8004
```

3.24.10 See also

Concepts

armasm User Guide:

- *Register-relative and PC-relative expressions* on page 10-7.
- *Changing between AArch64 and AArch32 states* on page 3-4.
- *Exception levels* on page 5-3.

armlink User Guide:

- Chapter 4 *Image structure and generation*.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.25 BFC and BFI

Bit Field Clear and Bit Field Insert. Clear adjacent bits in a register, or Insert adjacent bits from one register into another.

3.25.1 Syntax

`BFC{cond} Rd, #lsb, #width`

`BFI{cond} Rd, Rn, #lsb, #width`

where:

cond is an optional condition code.

Rd is the destination register.

Rn is the source register.

lsb is the least significant bit that is to be cleared or copied.

width is the number of bits to be cleared or copied. *width* must not be 0, and (*width*+*lsb*) must be less than or equal to 32.

3.25.2 BFC

width bits in *Rd* are cleared, starting at *lsb*. Other bits in *Rd* are unchanged.

3.25.3 BFI

width bits in *Rd*, starting at *lsb*, are replaced by *width* bits from *Rn*, starting at bit[0]. Other bits in *Rd* are unchanged.

3.25.4 Register restrictions

You cannot use PC for any register.

You can use SP in A32 instructions but this is deprecated. You cannot use SP in T32 instructions.

3.25.5 Condition flags

These instructions do not change the flags.

3.25.6 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.25.7 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.26 BKPT

Software breakpoint.

3.26.1 Syntax

`BKPT #imm`

where:

imm is an expression evaluating to an integer in the range:

- 0-65535 (a 16-bit value) in an A32 instruction.
- 0-255 (an 8-bit value) in a 16-bit T32 instruction.

3.26.2 Usage

The `BKPT` instruction causes a `BKPT` instruction debug event, which generates a debug exception. The exception is routed to a debug monitor executing in EL1 or EL2.

In both A32 state and T32 state, *imm* is ignored by the ARM hardware. However, a debugger can use it to store additional information about the breakpoint.

`BKPT` is an unconditional instruction. It must not have a condition code in A32 code. In T32 code, the `BKPT` instruction does not require a condition code suffix because it always executes irrespective of its condition code suffix.

3.26.3 Availability

This instruction is available in A32 and T32.

In T32, it is only available as a 16-bit instruction.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.27 CBZ and CBNZ

Compare and Branch on Zero, Compare and Branch on Non-Zero.

3.27.1 Syntax

`CBZ Rn, label`

`CBNZ Rn, label`

where:

Rn is the register holding the operand.

label is the branch destination.

3.27.2 Usage

You can use the `CBZ` or `CBNZ` instructions to avoid changing the condition flags and to reduce the number of instructions.

Except that it does not change the condition flags, `CBZ Rn, label` is equivalent to:

```
CMP     Rn, #0
BEQ     label
```

Except that it does not change the condition flags, `CBNZ Rn, label` is equivalent to:

```
CMP     Rn, #0
BNE     label
```

3.27.3 Restrictions

The branch destination must be within 4 to 130 bytes after the instruction and in the same execution state.

These instructions must not be used inside an IT block.

3.27.4 Condition flags

These instructions do not change the flags.

3.27.5 Availability

These 16-bit instructions are available in T32 only.

There are no A32 or 32-bit T32 versions of these instructions.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.28 CLREX

Clear Exclusive. Clears the local record of the executing processor that an address has had a request for an exclusive access.

3.28.1 Syntax

`CLREX{`*cond*`}`

where:

cond is an optional condition code.

Note

cond is permitted only in T32 code, using a preceding `IT` instruction, but this is deprecated. This is an unconditional instruction in A32.

3.28.2 Usage

Use the `CLREX` instruction to return a closely-coupled exclusive access monitor to its open-access state. This removes the requirement for a dummy store to memory.

It is implementation defined whether `CLREX` also clears the global record of the executing processor that an address has had a request for an exclusive access.

3.28.3 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit `CLREX` instruction in T32.

3.28.4 See also

Reference

- *Memory access instructions* on page 3-9.
- *Condition codes* on page 3-26.

Other information

- *ARM Architecture Reference Manual* `http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/`.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.29 CLZ

Count Leading Zeros.

3.29.1 Syntax

`CLZ{cond} Rd, Rm`

where:

cond	is an optional condition code.
Rd	is the destination register.
Rm	is the operand register.

3.29.2 Usage

The `CLZ` instruction counts the number of leading zeros in the value in *Rm* and returns the result in *Rd*. The result value is 32 if no bits are set in the source register, and zero if bit 31 is set.

3.29.3 Register restrictions

You cannot use PC for any operand.

Using SP in these A32 instructions is deprecated.

You cannot use SP in T32 instructions.

3.29.4 Condition flags

This instruction does not change the flags.

3.29.5 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit version of this instruction in T32.

3.29.6 Examples

```
CLZ     r4,r9
CLZNE   r2,r3
```

Use the `CLZ` T32 instruction followed by a left shift of *Rm* by the resulting *Rd* value to normalize the value of register *Rm*. Use `MOVS`, rather than `MOV`, to flag the case where *Rm* is zero:

```
CLZ r5, r9
MOVS r9, r9, LSL r5
```

3.29.7 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.30 CMP and CMN

Compare and Compare Negative.

3.30.1 Syntax

`CMP{cond} Rn, Operand2`

`CMN{cond} Rn, Operand2`

where:

cond is an optional condition code.

Rn is the ARM register holding the first operand.

Operand2 is a flexible second operand.

3.30.2 Usage

These instructions compare the value in a register with *Operand2*. They update the condition flags on the result, but do not place the result in any register.

The `CMP` instruction subtracts the value of *Operand2* from the value in *Rn*. This is the same as a `SUBS` instruction, except that the result is discarded.

The `CMN` instruction adds the value of *Operand2* to the value in *Rn*. This is the same as an `ADDS` instruction, except that the result is discarded.

In certain circumstances, the assembler can substitute `CMN` for `CMP`, or `CMP` for `CMN`. Be aware of this when reading disassembly listings.

3.30.3 Use of PC in A32 and T32 instructions

You cannot use PC for any operand in any data processing instruction that has a register-controlled shift.

Using PC (`R15`) in these A32 instructions without register controlled shift is deprecated.

If you use PC as *Rn* in A32 instructions, the value used is the address of the instruction plus 8.

You cannot use PC for any operand in these T32 instructions.

3.30.4 Use of SP in A32 and T32 instructions

You can use SP for *Rn* in A32 and T32 instructions.

Using SP for *Rm* in A32 instructions is deprecated.

Using SP for *Rm* in a 16-bit T32 `CMP Rn, Rm` instruction but this is deprecated. Other use of SP for *Rm* is not permitted in T32.

3.30.5 Condition flags

These instructions update the N, Z, C and V flags according to the result.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.30.6 16-bit instructions

The following forms of these instructions are available in T32 code, and are 16-bit instructions:

`CMP Rn, Rm` Lo register restriction does not apply.

`CMN Rn, Rm` *Rn* and *Rm* must both be Lo registers.

`CMP Rn, #imm` *Rn* must be a Lo register. *imm* range 0-255.

3.30.7 Examples

```
CMP     r2, r9
CMN     r0, #6400
CMPGT   sp, r7, LSL #2
```

3.30.8 Incorrect example

```
CMP     r2, pc, ASR r0 ; PC not permitted with register-controlled shift
```

3.30.9 See also

Concepts

- *Flexible second operand (Operand2)* on page 3-12.
- *Instruction substitution* on page 3-13.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.31 CPS

CPS (Change Processor State) changes one or more of the mode, A, I, and F bits in the CPSR, without changing the other CPSR bits.

CPS is only permitted in privileged software execution, and has no effect in User mode.

CPS cannot be conditional, and is not permitted in an IT block.

3.31.1 Syntax

CPS*effect iflags*{, *#mode*}

CPS *#mode*

where:

effect is one of:

- IE — Interrupt or abort enable.
- ID — Interrupt or abort disable.

iflags is a sequence of one or more of:

- a — Enables or disables imprecise aborts.
- i — Enables or disables IRQ interrupts.
- f — Enables or disables FIQ interrupts.

mode specifies the number of the mode to change to.

3.31.2 Condition flags

This instruction does not change the condition flags.

3.31.3 16-bit instructions

The following forms of these instructions are available in T32 code, and are 16-bit instructions:

- CPSIE *iflags*
- CPSID *iflags*

You cannot specify a mode change in a 16-bit T32 instruction.

3.31.4 Availability

This instruction is available in A32 and T32.

In T32, 16-bit and 32-bit versions of this instruction are available.

3.31.5 Examples

```
CPSIE if      ; enable interrupts and fast interrupts
CPSID A       ; disable imprecise aborts
CPSID ai, #17 ; disable imprecise aborts and interrupts, and enter FIQ mode
CPS #16       ; enter User mode
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.32 CPY pseudo-instruction

Copy a value from one register to another.

3.32.1 Syntax

`CPY{cond} Rd, Rm`

where:

cond	is an optional condition code.
Rd	is the destination register.
Rm	is the register holding the value to be copied.

3.32.2 Usage

The `CPY` pseudo-instruction copies a value from one register to another, without changing the condition flags.

`CPY Rd, Rm` assembles to `MOV Rd, Rm`.

3.32.3 Availability

This pseudo-instruction is available in A32 code and in T32 code.

3.32.4 Register restrictions

Using SP or PC for both *Rd* and *Rm* is deprecated.

3.32.5 Condition flags

This instruction does not change the condition flags.

3.32.6 See also

Reference

- *MOV and MVN* on page 3-98.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.33 DBG

Debug.

3.33.1 Syntax

`DBG{cond} {option}`

where:

cond is an optional condition code.

option is an optional limitation on the operation of the hint. The range is 0-15.

3.33.2 Usage

`DBG` is a hint instruction. It is optional whether it is implemented or not. If it is not implemented, it behaves as a `NOP`. The assembler produces a diagnostic message if the instruction executes as `NOP` on the target.

Debug hint provides a hint to a debugger and related tools. See your debugger and related tools documentation to determine the use, if any, of this instruction.

3.33.3 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit version of this instruction in T32.

3.33.4 See also

Reference

- *NOP* on page 3-113.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.34 DCPS1 (T32 instruction)

Debug switch to exception level 1 (EL1).

Note

This instruction is supported only in the ARMv8.

3.34.1 Syntax

```
DCPS1
```

3.34.2 Usage

This instruction is valid in Debug state only, and is always UNDEFINED in Non-debug state.

DCPS1 targets EL1 and:

- If EL1 is using AArch32, the processing element (PE) enters SVC mode. If EL3 is using AArch32, Secure SVC is an EL3 mode. This means DCPS1 causes the PE to enter EL3.
- If EL1 is using AArch64, the PE enters EL1h, and executes future instructions as A64 instructions.

In Non-debug state, use the `SVC` instruction to generate a trap to EL1.

3.34.3 Availability

This 32-bit instruction is available in T32 only.

There is no 16-bit version of this instruction in T32.

3.34.4 See also

Reference

- *SVC* on page 3-156.

Other information

- *ARM Architecture Reference Manual* http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.35 DCPS2 (T32 instruction)

Debug switch to exception level 2.

Note

This instruction is supported only in the ARMv8.

3.35.1 Syntax

```
DCPS2
```

3.35.2 Usage

This instruction is valid in Debug state only, and is always UNDEFINED in Non-debug state.

DCPS2 targets EL2 and:

- If EL2 is using AArch32, the PE enters Hyp mode.
- If EL2 is using AArch64, the PE enters EL2h, and executes future instructions as A64 instructions.

In Non-debug state, use the `HVC` instruction to generate a trap to EL2.

3.35.3 Availability

This 32-bit instruction is available in T32 only.

There is no 16-bit version of this instruction in T32.

3.35.4 See also

Reference

- *HVC* on page 3-65.

Other information

- *ARM Architecture Reference Manual* http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.36 DCPS3 (T32 instruction)

Debug switch to exception level 3.

Note

This instruction is supported only in the ARMv8.

3.36.1 Syntax

```
DCPS3
```

3.36.2 Usage

This instruction is valid in Debug state only, and is always UNDEFINED in Non-debug state.

DCPS3 targets EL3 and:

- If EL3 is using AArch32, the PE enters Monitor mode.
- If EL3 is using AArch64, the PE enters EL3h, and executes future instructions as A64 instructions.

In Non-debug state, use the `SMC` instruction to generate a trap to EL3.

3.36.3 Availability

This 32-bit instruction is available in T32 only.

There is no 16-bit version of this instruction in T32.

3.36.4 See also

Reference

- *SMC* on page 3-136.

Other information

- *ARM Architecture Reference Manual* http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.37 DMB, DSB, and ISB

Data Memory Barrier, Data Synchronization Barrier, and Instruction Synchronization Barrier.

3.37.1 Syntax

`DMB{cond} {option}`

`DSB{cond} {option}`

`ISB{cond} {option}`

where:

cond is an optional condition code.

Note

cond is permitted only in T32 code. These are unconditional instructions in A32.

option is an optional limitation on the operation of the hint.

3.37.2 DMB

Data Memory Barrier acts as a memory barrier. It ensures that all explicit memory accesses that appear in program order before the `DMB` instruction are observed before any explicit memory accesses that appear in program order after the `DMB` instruction. It does not affect the ordering of any other instructions executing on the processor.

Permitted values of *option* are:

`SY` Full system DMB operation. This is the default and can be omitted.

`LD` DMB operation that waits only for loads to complete.

`ST` DMB operation that waits only for stores to complete.

`ISH` DMB operation only applies to the inner shareable domain.

`ISHLD` DMB operation that waits only for loads to complete, and only applies to the inner shareable domain.

`ISHST` DMB operation that waits only for stores to complete, and only applies to the inner shareable domain.

`NSH` DMB operation only applies out to the point of unification.

`NSHLD` DMB operation that waits only for loads to complete and only applies out to the point of unification.

`NSHST` DMB operation that waits only for stores to complete and only out to the point of unification.

`OSH` DMB operation only applies to the outer shareable domain.

`OSHLD` DMB operation that waits only for loads to complete, and only applies to the outer shareable domain.

`OSHST` DMB operation that waits only for stores to complete, and only applies to the outer shareable domain.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Note

The options LD, ISHLD, NSHLD, and OSHLD are supported only in ARMv8.

3.37.3 DSB

Data Synchronization Barrier acts as a special kind of memory barrier. No instruction in program order after this instruction executes until this instruction completes. This instruction completes when:

- All explicit memory accesses before this instruction complete.
- All Cache, Branch predictor and TLB maintenance operations before this instruction complete.

Permitted values of *option* are:

SY	Full system DSB operation. This is the default and can be omitted.
LD	DSB operation that waits only for loads to complete.
ST	DSB operation that waits only for stores to complete.
ISH	DSB operation only applies to the inner shareable domain.
ISHLD	DSB operation that waits only for loads to complete, and only applies to the inner shareable domain.
ISHST	DSB operation that waits only for stores to complete, and only applies to the inner shareable domain.
NSH	DSB operation only applies out to the point of unification.
NSHLD	DSB operation that waits only for loads to complete and only applies out to the point of unification.
NSHST	DSB operation that waits only for stores to complete and only applies out to the point of unification.
OSH	DSB operation only applies to the outer shareable domain.
OSHLD	DSB operation that waits only for loads to complete, and only applies to the outer shareable domain.
OSHST	DSB operation that waits only for stores to complete, and only applies to the outer shareable domain.

Note

The options LD, ISHLD, NSHLD, and OSHLD are supported only in ARMv8.

3.37.4 ISB

Instruction Synchronization Barrier flushes the pipeline in the processor, so that all instructions following the ISB are fetched from cache or memory, after the instruction has been completed. It ensures that the effects of context altering operations, such as changing the ASID, or completed TLB maintenance operations, or branch predictor maintenance operations, in addition to all changes to the CP15 registers, executed before the ISB instruction are visible to the instructions fetched after the ISB.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

In addition, the ISB instruction ensures that any branches that appear in program order after it are always written into the branch prediction logic with the context that is visible after the ISB instruction. This is required to ensure correct execution of the instruction stream.

Permitted values of *option* are:

SY Full system ISB operation. This is the default, and can be omitted.

3.37.5 Aliases

The following alternative values of *option* are supported for DMB and DSB, but ARM recommends that you do not use them:

- SH is an alias for ISH.
- SHST is an alias for ISHST.
- UN is an alias for NSH.
- UNST is an alias for NSHST.

3.37.6 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.37.7 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.38 ERET

Exception Return.

3.38.1 Syntax

`ERET{`*cond*`}`

where:

cond is an optional condition code.

3.38.2 Usage

In a processor that implements the Virtualization Extensions, you can use `ERET` to perform a return from an exception taken to Hyp mode.

3.38.3 Operation

When executed in Hyp mode, `ERET` loads the PC from ELR_hyp and loads the CPSR from SPSR_hyp. When executed in any other mode, apart from User or System, it behaves as:

- `MOVS PC, LR` in the A32 instruction set.
- `SUBS PC, LR, #0` in the T32 instruction set.

3.38.4 Notes

You must not use `ERET` in User or System mode. The assembler cannot warn you about this, because it has no information about the processor mode at execution time.

`ERET` is the preferred synonym for `SUBS PC, LR, #0` in the T32 instruction set.

3.38.5 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit version of this instruction in T32.

3.38.6 See also

Concepts

armasm User Guide:

- *Processor modes, and privileged and unprivileged software execution* on page 4-3.

Reference

- *MOV and MVN* on page 3-98.
- *SUBS pc, lr* on page 3-154.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.39 HLT

Halting breakpoint.

3.39.1 Syntax

`HLT{Q} #imm`

where:

`Q` is an optional suffix. It only has an effect when Halting debug-mode is disabled. In this case, if `Q` is specified, the instruction behaves as a `NOP`. If `Q` is not specified, the instruction is UNDEFINED.

imm is an expression evaluating to an integer in the range:

- 0-65535 (a 16-bit value) in an A32 instruction.
- 0-63 (a 6-bit value) in a 16-bit T32 instruction.

3.39.2 Usage

The `HLT` instruction causes the processor to enter Debug state if Halting debug-mode is enabled.

In both A32 state and T32 state, *imm* is ignored by the ARM hardware. However, a debugger can use it to store additional information about the breakpoint.

`HLT` is an unconditional instruction. It must not have a condition code in A32 code. In T32 code, the `HLT` instruction does not require a condition code suffix because it always executes irrespective of its condition code suffix.

3.39.3 Availability

This instruction is available in A32 and T32.

In T32, it is only available as a 16-bit instruction.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.40 HVC

Hypervisor Call.

3.40.1 Syntax

`HVC #imm`

where:

imm is any value in the range 0-65535.

3.40.2 Usage

In a processor that implements the Virtualization Extensions, the HVC instruction causes a Hypervisor Call exception. This means that the processor mode changes to Hyp, the CPSR is saved to the SPSR_hyp, and execution branches to the HVC vector.

HVC is UNDEFINED if the processor is in Secure state, or in User mode in Non-secure state.

imm is ignored by the processor. However, it can be retrieved by the exception handler to determine what service is being requested.

HVC cannot be conditional, and is not permitted in an IT block.

3.40.3 Notes

When in Hyp mode, the ERET instruction performs the exception return.

In an implementation that includes the Virtualization Extensions, an SVC instruction executed in Hyp mode is treated as an HVC instruction.

3.40.4 Availability

This 32-bit instruction is available in A32 and T32. It is available in ARMv7 architectures that include the Virtualization Extensions.

There is no 16-bit version of this instruction in T32.

3.40.5 See also

Reference

- *ERET* on page 3-63.
- *SVC* on page 3-156.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.41 IT

The IT (If-Then) instruction makes a single following instruction (the *IT block*) conditional. The conditional instruction must be from a restricted set of 16-bit instructions.

The *IT block* can contain between two and four conditional instructions, where the conditions can be all the same, or some of them can be the logical inverse of the others, but this is deprecated.

3.41.1 Syntax

```
IT cond
```

where:

cond specifies the condition for the following instruction.

3.41.2 Deprecated syntax

```
IT{x{y{z}}} {cond}
```

where:

x specifies the condition switch for the second instruction in the IT block.

y specifies the condition switch for the third instruction in the IT block.

z specifies the condition switch for the fourth instruction in the IT block.

cond specifies the condition for the first instruction in the IT block.

The condition switch for the second, third and fourth instruction in the IT block can be either:

T Then. Applies the condition *cond* to the instruction.

E Else. Applies the inverse condition of *cond* to the instruction.

3.41.3 Usage

The conditional instruction (including branches, but excluding the BKPT instruction) must specify the condition in the {*cond*} part of its syntax.

You are not required to write IT instructions in your code, because the assembler generates them for you automatically according to the conditions specified on the following instructions. However, if you do write IT instructions, the assembler validates the conditions specified in the IT instructions against the conditions specified in the following instructions.

Writing the IT instructions ensures that you consider the placing of conditional instructions, and the choice of conditions, in the design of your code.

When assembling to A32 code, the assembler performs the same checks, but does not generate any IT instructions.

With the exception of CMP, CMN, and TST, the 16-bit instructions that normally affect the condition flags, do not affect them when used inside an IT block.

A BKPT instruction in an IT block is always executed, so it does not require a condition in the {*cond*} part of its syntax. The IT block continues from the next instruction. Using a BKPT or HLT instruction inside an IT block is deprecated.

Note

You can use an IT block for unconditional instructions by using the AL condition.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Conditional branches inside an IT block have a longer branch range than those outside the IT block.

3.41.4 Restrictions

The following instructions are not permitted in an IT block:

- IT.
- CBZ and CBNZ.
- TBB and TBH.
- CPS, CPSID and CPSIE.
- SETEND.

Other restrictions when using an IT block are:

- A branch or any instruction that modifies the PC is only permitted in an IT block if it is the last instruction in the block.
- You cannot branch to any instruction in an IT block, unless when returning from an exception handler.
- You cannot use any assembler directives in an IT block.

Note

The assembler shows a diagnostic message when any of these instructions are used in an IT block.

Using any instruction not listed in Table 3-6 in an IT block is deprecated. Also, any explicit reference to R15 (the PC) in the IT block is deprecated.

Table 3-6 Permitted instructions inside an IT block

16-bit instruction	When deprecated
MOV, MVN	When R*m* or R*d* is the PC
LDR, LDRB, LDRH, LDRSB, LDRSH	For PC-relative forms
STR, STRB, STRH	-
ADD, ADC, RSB, SBC, SUB	ADD SP, SP, #imm or SUB SP, SP, #imm or when R*m*, R*dn* or R*dm* is the PC
CMP, CMN	When R*m* or R*n* is the PC
MUL	-
ASR, LSL, LSR, ROR	-
AND, BIC, EOR, ORR, TST	-
BX, BLX	When R*m* is the PC

3.41.5 Condition flags

This instruction does not change the flags.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.41.6 Exceptions

Exceptions can occur between an IT instruction and the corresponding IT block, or within an IT block. This exception results in entry to the appropriate exception handler, with suitable return information in LR and SPSR.

Instructions designed for use as exception returns can be used as normal to return from the exception, and execution of the IT block resumes correctly. This is the only way that a PC-modifying instruction can branch to an instruction in an IT block.

3.41.7 Availability

This 16-bit instruction is available in T32 only.

In A32 code, IT is a pseudo-instruction that does not generate any code.

There is no 32-bit version of this instruction.

3.41.8 Examples

```
IT     GT
LDRGT  r0, [r1,#4]

IT     EQ
ADDEQ  r0, r1, r2
```

3.41.9 Incorrect examples

```
IT     NE
ADD    r0,r0,r1  ; syntax error: no condition code used in IT block

ITT    EQ
MOVEQ  r0,r1
ADDEQ  r0,r0,#1  ; IT block covering more than one instruction is deprecated

IT     GT
LDRGT  r0,label  ; LDR (PC-relative) is deprecated in an IT block

IT     EQ
ADDEQ  PC,r0     ; ADD is deprecated when Rdn is the PC
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.42 LDC and STC

Transfer data between memory and coprocessor.

3.42.1 Syntax

```
op{L}{cond} coproc, CRd, [Rn]
op{L}{cond} coproc, CRd, [Rn, #{-}offset]   ; offset addressing
op{L}{cond} coproc, CRd, [Rn, #{-}offset]!  ; pre-index addressing
op{L}{cond} coproc, CRd, [Rn], #{-}offset    ; post-index addressing
op{L}{cond} coproc, CRd, label
```

where:

op is either LDC or STC.

cond is an optional condition code.

L is an optional suffix specifying a long transfer.

coproc is the name of the coprocessor the instruction is for. The standard name is p*n*, where *n* is either 14 or 15.

CRd is the coprocessor register to load or store.

Rn is the register on which the memory address is based. If PC is specified, the value used is the address of the current instruction plus eight.

\- is an optional minus sign. If - is present, the offset is subtracted from *Rn*. Otherwise, the offset is added to *Rn*.

offset is an expression evaluating to a multiple of 4, in the range 0 to 1020.

! is an optional suffix. If ! is present, the address including the offset is written back into *Rn*.

label is a word-aligned PC-relative expression.

label must be within 1020 bytes of the current instruction.

3.42.2 Usage

The use of these instructions depends on the coprocessor. See the coprocessor documentation for details.

3.42.3 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.42.4 Register restrictions

You cannot use PC for R*n* in the pre-index and post-index instructions. These are the forms that write back to R*n*.

You cannot use PC for R*n* in the T32 STC instruction.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

The A32 STC instruction that uses the label syntax, or where R*n* is PC, is deprecated.

3.42.5 See also

Concepts

armasm User Guide:

- *Register-relative and PC-relative expressions* on page 10-7.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.43 LDM and STM

Load and Store Multiple registers. Any combination of registers R0 to R15 (PC) can be transferred in A32 state, but there are some restrictions in T32 state.

3.43.1 Syntax

op{*addr_mode*}{*cond*} *Rn*{!}, *reglist*{^}

where:

op can be either:

- LDM Load Multiple registers
- STM Store Multiple registers.

addr_mode is any one of the following:

- IA Increment address After each transfer. This is the default, and can be omitted.
- IB Increment address Before each transfer (A32 only).
- DA Decrement address After each transfer (A32 only).
- DB Decrement address Before each transfer.

You can also use the stack oriented addressing mode suffixes, for example, when implementing stacks.

cond is an optional condition code.

Rn is the *base register*, the ARM register holding the initial address for the transfer. *Rn* must not be PC.

! is an optional suffix. If ! is present, the final address is written back into *Rn*.

reglist is a list of one or more registers to be loaded or stored, enclosed in braces. It can contain register ranges. It must be comma-separated if it contains more than one register or register range.

^ is an optional suffix, available in A32 state only. You must not use it in User mode or System mode. It has the following purposes:

- If the instruction is LDM (with any addressing mode) and *reglist* contains the PC (R15), in addition to the normal multiple register transfer, the SPSR is copied into the CPSR. This is for returning from exception handlers. Use this only from exception modes.
- Otherwise, data is transferred into or out of the User mode registers instead of the current mode registers.

3.43.2 Restrictions on reglist in 32-bit T32 instructions

In 32-bit T32 instructions:

- The SP cannot be in the list.
- The PC cannot be in the list in an STM instruction.
- The PC and LR cannot both be in the list in an LDM instruction.
- There must be two or more registers in the list.

If you write an STM or LDM instruction with only one register in reglist, the assembler automatically substitutes the equivalent STR or LDR instruction. Be aware of this when comparing disassembly listings with source code.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

You can use the `--diag_warning 1645` assembler command line option to check when an instruction substitution occurs.

3.43.3 Restrictions on reglist in A32 instructions

A32 store instructions can have SP and PC in the *reglist* but these instructions that include SP or PC in the *reglist* are deprecated.

A32 load instructions can have SP and PC in the *reglist* but these instructions that include SP in the *reglist* or both PC and LR in the *reglist* are deprecated.

3.43.4 16-bit instructions

16-bit versions of a subset of these instructions are available in T32 code.

The following restrictions apply to the 16-bit instructions:

- All registers in *reglist* must be Lo registers.
- *Rn* must be a Lo register.
- *addr_mode* must be omitted (or IA), meaning increment address after each transfer.
- Writeback must be specified for STM instructions.
- Writeback must be specified for LDM instructions where *Rn* is not in the *reglist*.

Note

16-bit T32 STM instructions with writeback that specify *Rn* as the lowest register in the *reglist* are deprecated.

In addition, the PUSH and POP instructions are subsets of the STM and LDM instructions and can therefore be expressed using the STM and LDM instructions. Some forms of PUSH and POP are also 16-bit instructions.

3.43.5 Loading to the PC

A load to the PC causes a branch to the instruction at the address loaded.

Bits[1:0] of the address loaded must not be 0b10.

If bit[0] is 1, execution continues in T32 state. If bit[0] is 0, execution continues in A32 state.

3.43.6 Loading or storing the base register, with writeback

In A32 or 16-bit T32 instructions, if *Rn* is in *reglist*, and writeback is specified with the ! suffix:

- If the instruction is STM{*addr_mode*}{*cond*} and *Rn* is the lowest-numbered register in *reglist*, the initial value of *Rn* is stored. These instructions are deprecated.
- Otherwise, the loaded or stored value of R*n* cannot be relied on, so these instructions are not permitted.

32-bit T32 instructions are not permitted if *Rn* is in *reglist*, and writeback is specified with the ! suffix.

3.43.7 Examples

```
LDM     r8,{r0,r2,r9}      ; LDMIA is a synonym for LDM
STMDB   r1!,{r3-r6,r11,r12}
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.43.8 Incorrect examples

```
STM     r5!,{r5,r4,r9} ; value stored for R5 UNKNOWN
LDMDA   r2, {}         ; must be at least one register in list
```

3.43.9 See also

Concepts

armasm User Guide:

- *Stack implementation using LDM and STM* on page 7-23.

Reference

- *Memory access instructions* on page 3-9.
- *PUSH and POP* on page 3-121.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.44 LDR and STR (immediate offset)

Load and Store with immediate offset, pre-indexed immediate offset, or post-indexed immediate offset.

3.44.1 Syntax

```
op{type}{cond} Rt, [Rn {, #offset}]        ; immediate offset
op{type}{cond} Rt, [Rn, #offset]!          ; pre-indexed
op{type}{cond} Rt, [Rn], #offset           ; post-indexed
opD{cond} Rt, Rt2, [Rn {, #offset}]        ; immediate offset, doubleword
opD{cond} Rt, Rt2, [Rn, #offset]!          ; pre-indexed, doubleword
opD{cond} Rt, Rt2, [Rn], #offset           ; post-indexed, doubleword
```

where:

op can be either:
- LDR Load Register
- STR Store Register.

type can be any one of:
- B unsigned Byte (Zero extend to 32 bits on loads.)
- SB signed Byte (LDR only. Sign extend to 32 bits.)
- H unsigned Halfword (Zero extend to 32 bits on loads.)
- SH signed Halfword (LDR only. Sign extend to 32 bits.)
- - omitted, for Word.

cond is an optional condition code.

Rt is the register to load or store.

Rn is the register on which the memory address is based.

offset is an offset. If *offset* is omitted, the address is the contents of *Rn*.

Rt2 is the additional register to load or store for doubleword operations.

Not all options are available in every instruction set.

3.44.2 Offset ranges and availability

Table 3-7 shows the ranges of offsets and availability of these instructions.

Table 3-7 Offsets and availability, LDR/STR, word, halfword, and byte

Instruction	Immediate offset	Pre-indexed	Post-indexed
A32, word or byte [a]	–4095 to 4095	–4095 to 4095	–4095 to 4095
A32, signed byte, halfword, or signed halfword	–255 to 255	–255 to 255	–255 to 255
A32, doubleword	–255 to 255	–255 to 255	–255 to 255

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 3-7 Offsets and availability, LDR/STR, word, halfword, and byte (continued)

Instruction	Immediate offset	Pre-indexed	Post-indexed
32-bit, T32, word, halfword, signed halfword, byte, or signed byte [a]	–255 to 4095	–255 to 255	–255 to 255
32-bit, T32, doubleword	–1020 to 1020 [c]	–1020 to 1020 [c]	–1020 to 1020 [c]
16-bit, T32, word [b]	0 to 124 [c]	Not available	Not available
16-bit, T32, unsigned halfword [b]	0 to 62 [d]	Not available	Not available
16-bit, T32, unsigned byte [b]	0 to 31	Not available	Not available
16-bit, T32, word, Rn is SP [e]	0 to 1020 [c]	Not available	Not available

a. For word loads, Rt can be the PC. A load to the PC causes a branch to the address loaded. Bits[1:0] must not be 0b10, and if bit[0] is 1, execution continues in T32 state, otherwise execution continues in A32 state.
b. Rt and Rn must be in the range R0-R7.
c. Must be divisible by 4.
d. Must be divisible by 2.
e. Rt must be in the range R0-R7.

3.44.3 Register restrictions

R*n* must be different from R*t* in the pre-index and post-index forms.

3.44.4 Doubleword register restrictions

R*n* must be different from R*t2* in the pre-index and post-index forms.

For T32 instructions, you must not specify SP or PC for either *Rt* or *Rt2*.

For A32 instructions:

- *Rt* must be an even-numbered register.
- *Rt* must not be LR.
- ARM strongly recommends that you do not use R12 for *Rt*.
- *Rt2* must be R(*t* + 1).

3.44.5 Use of PC

In A32 instructions:

- You can use PC for R*t* in LDR word instructions and PC for R*n* in LDR instructions.
- You can use PC for R*t* in STR word instructions and PC for R*n* in STR instructions with immediate offset syntax (that is the forms that do not writeback to the R*n*). However, this is deprecated.

Other uses of PC are not permitted in these A32 instructions.

In T32 instructions you can use PC for R*t* in LDR word instructions and PC for R*n* in LDR instructions. Other uses of PC in these T32 instructions are not permitted.

3.44.6 Use of SP

You can use SP for R*n*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

In A32 code, you can use SP for `Rt` in word instructions. Using SP for `Rt` in non-word instructions in A32 code is deprecated.

In T32 code, you can use SP for `Rt` in word instructions only. All other use of SP for `Rt` in these instructions are not permitted in T32 code.

3.44.7 Examples

```
LDR     r8,[r10]            ; loads R8 from the address in R10.
LDRNE   r2,[r5,#960]!       ; (conditionally) loads R2 from a word
                            ; 960 bytes above the address in R5, and
                            ; increments R5 by 960.
STR     r2,[r9,#consta-struc]   ; consta-struc is an expression evaluating
                                ; to a constant in the range 0-4095.
```

3.44.8 See also

Reference

- *Memory access instructions* on page 3-9.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.45 LDR and STR (register offset)

Load and Store with register offset, pre-indexed register offset, or post-indexed register offset.

3.45.1 Syntax

```
op{type}{cond} Rt, [Rn, +/-Rm {, shift}]   ; register offset
op{type}{cond} Rt, [Rn, +/-Rm {, shift}]!  ; pre-indexed       ; A32 only
op{type}{cond} Rt, [Rn], +/-Rm {, shift}   ; post-indexed      ; A32 only
opD{cond} Rt, Rt2, [Rn, +/-Rm]   ; register offset, doubleword ; A32 only
opD{cond} Rt, Rt2, [Rn, +/-Rm]!  ; pre-indexed, doubleword     ; A32 only
opD{cond} Rt, Rt2, [Rn], +/-Rm   ; post-indexed, doubleword    ; A32 only
```

where:

op can be either:
- LDR Load Register
- STR Store Register.

type can be any one of:
- B unsigned Byte (Zero extend to 32 bits on loads.)
- SB signed Byte (LDR only. Sign extend to 32 bits.)
- H unsigned Halfword (Zero extend to 32 bits on loads.)
- SH signed Halfword (LDR only. Sign extend to 32 bits.)
- - omitted, for Word.

cond is an optional condition code.

Rt is the register to load or store.

Rn is the register on which the memory address is based.

Rm is a register containing a value to be used as the offset. *-Rm* is not permitted in T32 code.

shift is an optional shift.

Rt2 is the additional register to load or store for doubleword operations.

Not all options are available in every instruction set.

3.45.2 Offset register and shift options

Table 3-8 shows the ranges of offsets and availability of these instructions.

Table 3-8 Options and availability, LDR/STR (register offsets)

Instruction	+/–*Rm* [a]	Shift		
A32, word or byte [b]	+/–*Rm*	LSL #0-31	LSR #1-32	
		ASR #1-32	ROR #1-31	RRX
A32, signed byte, halfword, or signed halfword	+/–*Rm*	Not available		

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 3-8 Options and availability, LDR/STR (register offsets) (continued)

Instruction	+/–*Rm* [a]	Shift
A32, doubleword	+/–*Rm*	Not available
32-bit, T32, word, halfword, signed halfword, byte, or signed byte [b]	+*Rm*	`LSL #0-3`
16-bit, T32, all except doubleword[c]	+*Rm*	Not available

a. Where +/–Rm is shown, you can use –Rm, +Rm, or Rm. Where +Rm is shown, you cannot use –Rm.

b. For word loads, Rt can be the PC. A load to the PC causes a branch to the address loaded. Bits[1:0] must not be 0b10, and if bit[0] is 1, execution continues in T32 state, otherwise execution continues in A32 state.

c. Rt, Rn, and Rm must all be in the range R0-R7.

3.45.3 Register restrictions

In the pre-index and post-index forms, R*n* must be different from R*t*.

3.45.4 Doubleword register restrictions

For A32 instructions:

- *Rt* must be an even-numbered register.
- *Rt* must not be LR.
- ARM strongly recommends that you do not use R12 for *Rt*.
- *Rt2* must be R(*t* + 1).
- R*m* must be different from R*t* and R*t2* in LDRD instructions.
- R*n* must be different from R*t2* in the pre-index and post-index forms.

3.45.5 Use of PC

In A32 instructions:

- You can use PC for R*t* in LDR word instructions, and you can use PC for R*n* in LDR instructions with register offset syntax (that is the forms that do not writeback to the R*n*).
- You can use PC for R*t* in STR word instructions, and you can use PC for R*n* in STR instructions with register offset syntax (that is the forms that do not writeback to the R*n*) but this is deprecated.

Other uses of PC are not permitted in A32 instructions.

In T32 instructions you can use PC for R*t* in LDR word instructions. Other uses of PC in these T32 instructions are not permitted.

3.45.6 Use of SP

You can use SP for R*n*.

In A32, you can use SP for R*t* in word instructions. You can use SP for R*t* in non-word A32 instructions but this is deprecated.

You can use SP for R*m* in A32 instructions but this is deprecated.

In T32, you can use SP for R*t* in word instructions only. All other use of SP for R*t* in these instructions are not permitted in T32 code.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Use of SP for `Rm` is not permitted in T32 state.

3.45.7 See also

Reference

- *Memory access instructions* on page 3-9.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.46 LDR and STR, unprivileged

Unprivileged Load and Store, byte, halfword, or word.

When these instructions are executed by privileged software, they access memory with the same restrictions as they would have if they were executed by unprivileged software.

When executed by unprivileged software these instructions behave in exactly the same way as the corresponding load or store instruction, for example `LDRSBT` behaves in the same way as `LDRSB`.

3.46.1 Syntax

```
op{type}T{cond} Rt, [Rn {, #offset}]       ; immediate offset (32-bit T32 encoding only)
op{type}T{cond} Rt, [Rn] {, #offset}       ; post-indexed (A32 only)
op{type}T{cond} Rt, [Rn], +/-Rm {, shift}  ; post-indexed (register) (A32 only)
```

where:

op can be either:
- `LDR` Load Register
- `STR` Store Register.

type can be any one of:
- `B` unsigned Byte (Zero extend to 32 bits on loads.)
- `SB` signed Byte (`LDR` only. Sign extend to 32 bits.)
- `H` unsigned Halfword (Zero extend to 32 bits on loads.)
- `SH` signed Halfword (`LDR` only. Sign extend to 32 bits.)
- `-` omitted, for Word.

cond is an optional condition code.

Rt is the register to load or store.

Rn is the register on which the memory address is based.

offset is an offset. If offset is omitted, the address is the value in *Rn*.

Rm is a register containing a value to be used as the offset. *Rm* must not be PC.

shift is an optional shift.

3.46.2 Offset ranges and availability

Table 3-9 shows the ranges of offsets and availability of these instructions.

Table 3-9 Offsets and availability, LDR/STR (User mode)

Instruction	Immediate offset	Post-indexed	+/–*Rm* [a]	Shift
A32, word or byte	Not available	–4095 to 4095	+/–*Rm*	`LSL #0-31`
				`LSR #1-32`
				`ASR #1-32`
				`ROR #1-31`

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 3-9 Offsets and availability, LDR/STR (User mode) (continued)

Instruction	Immediate offset	Post-indexed	+/–*Rm* [a]	Shift
				RRX
A32, signed byte, halfword, or signed halfword	Not available	–255 to 255	+/–*Rm*	Not available
32-bit, T32, word, halfword, signed halfword, byte, or signed byte	0 to 255	Not available	Not available	

a. You can use –Rm, +Rm, or Rm.

3.46.3 See also

Reference

- *Memory access instructions* on page 3-9.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.47 LDR (PC-relative)

Load register. The address is an offset from the PC.

3.47.1 Syntax

```
LDR{type}{cond}{.W} Rt, label
LDRD{cond} Rt, Rt2, label        ; Doubleword
```

where:

type can be any one of:

- `B` unsigned Byte (Zero extend to 32 bits on loads.)
- `SB` signed Byte (`LDR` only. Sign extend to 32 bits.)
- `H` unsigned Halfword (Zero extend to 32 bits on loads.)
- `SH` signed Halfword (`LDR` only. Sign extend to 32 bits.)
- `-` omitted, for Word.

cond is an optional condition code.

`.W` is an optional instruction width specifier.

Rt is the register to load or store.

Rt2 is the second register to load or store.

label is a PC-relative expression.

label must be within a limited distance of the current instruction.

Note

Equivalent syntaxes are available for the `STR` instruction in A32 code but they are deprecated.

3.47.2 Offset ranges

The assembler calculates the offset from the PC for you. The assembler generates an error if *label* is out of range.

Table 3-10 shows the possible offsets between the label and the current instruction.

Table 3-10 PC-relative offsets

Instruction	Offset range
A32 `LDR`, `LDRB`, `LDRSB`, `LDRH`, `LDRSH` [a]	+/– 4095
A32 `LDRD`	+/– 255
32-bit T32 `LDR`, `LDRB`, `LDRSB`, `LDRH`, `LDRSH` [a]	+/– 4095
32-bit T32 `LDRD`	+/– 1020 [b]
16-bit T32 `LDR` [c]	0-1020 [b]

a. For word loads, Rt can be the PC. A load to the PC causes a branch to the address loaded. Bits[1:0] must not be 0b10, and if bit[0] is 1, execution continues in T32 state, otherwise execution continues in A32 state.

b. Must be a multiple of 4.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

c. Rt must be in the range R0-R7. There are no byte, halfword, or doubleword 16-bit instructions.

3.47.3 LDR (PC-relative) in T32

You can use the `.W` width specifier to force `LDR` to generate a 32-bit instruction in T32 code. `LDR.W` always generates a 32-bit instruction, even if the target could be reached using a 16-bit `LDR`.

For forward references, `LDR` without `.W` always generates a 16-bit instruction in T32 code, even if that results in failure for a target that could be reached using a 32-bit T32 `LDR` instruction.

3.47.4 Doubleword register restrictions

For 32-bit T32 instructions, you must not specify SP or PC for either *`Rt`* or *`Rt2`*.

For A32 instructions:

- *`Rt`* must be an even-numbered register.
- *`Rt`* must not be LR.
- ARM strongly recommends that you do not use `R12` for *`Rt`*.
- *`Rt2`* must be `R(t + 1)`.

3.47.5 Use of SP

In A32, you can use SP for `Rt` in `LDR` word instructions. You can use SP for `Rt` in `LDR` non-word A32 instructions but this is deprecated.

In T32, you can use SP for `Rt` in `LDR` word instructions only. All other uses of SP in these instructions are not permitted in T32 code.

3.47.6 See also

Concepts

armasm User Guide:

- *Register-relative and PC-relative expressions* on page 10-7.

Reference

- *Pseudo-instructions* on page 3-25.
- *LDR (PC-relative) in T32*.
- *Memory access instructions* on page 3-9.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.48 LDR (register-relative)

Load register. The address is an offset from a base register.

3.48.1 Syntax

```
LDR{type}{cond}{.W} Rt, label
LDRD{cond} Rt, Rt2, label        ; Doubleword
```

where:

type can be any one of:

- `B` unsigned Byte (Zero extend to 32 bits on loads.)
- `SB` signed Byte (`LDR` only. Sign extend to 32 bits.)
- `H` unsigned Halfword (Zero extend to 32 bits on loads.)
- `SH` signed Halfword (`LDR` only. Sign extend to 32 bits.)
- `-` omitted, for Word.

cond is an optional condition code.

`.W` is an optional instruction width specifier.

Rt is the register to load or store.

Rt2 is the second register to load or store.

label is a symbol defined by the `FIELD` directive. *label* specifies an offset from the base register which is defined using the `MAP` directive.

label must be within a limited distance of the value in the base register.

3.48.2 Offset ranges

The assembler calculates the offset from the base register for you. The assembler generates an error if *label* is out of range.

Table 3-11 shows the possible offsets between *label* and the current instruction.

Table 3-11 Register-relative offsets

Instruction	Offset range
A32 `LDR`, `LDRB`[a]	+/– 4095
A32 `LDRSB`, `LDRH`, `LDRSH`	+/– 255
A32 `LDRD`	+/– 255
32-bit T32 `LDR`, `LDRB`, `LDRSB`, `LDRH`, `LDRSH` [a]	–255 to 4095
32-bit T32 `LDRD`	+/– 1020 [b]
16-bit T32 `LDR` [c]	0 to 124 [b]
16-bit T32 `LDRH` [c]	0 to 62 [d]
16-bit T32 `LDRB` [c]	0 to 31
16-bit T32 `LDR`, base register is SP[e]	0 to 1020 [b]

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

a. For word loads, Rt can be the PC. A load to the PC causes a branch to the address loaded. Bits[1:0] must not be 0b10, and if bit[0] is 1, execution continues in T32 state, otherwise execution continues in A32 state.
b. Must be a multiple of 4.
c. Rt and base register must be in the range R0-R7.
d. Must be a multiple of 2.
e. Rt must be in the range R0-R7.

3.48.3 LDR (register-relative) in T32

You can use the `.W` width specifier to force `LDR` to generate a 32-bit instruction in T32 code. `LDR.W` always generates a 32-bit instruction, even if the target could be reached using a 16-bit `LDR`.

For forward references, `LDR` without `.W` always generates a 16-bit instruction in T32 code, even if that results in failure for a target that could be reached using a 32-bit T32 `LDR` instruction.

3.48.4 Doubleword register restrictions

For 32-bit T32 instructions, you must not specify SP or PC for either *Rt* or *Rt2*.

For A32 instructions:

- *Rt* must be an even-numbered register.
- *Rt* must not be LR.
- ARM strongly recommends that you do not use R12 for *Rt*.
- *Rt2* must be R(*t* + 1).

3.48.5 Use of PC

You can use PC for `Rt` in word instructions. Other uses of PC are not permitted in these instructions.

3.48.6 Use of SP

In A32, you can use SP for `Rt` in word instructions. You can use SP for `Rt` in non-word A32 instructions but this is deprecated.

In T32, you can use SP for `Rt` in word instructions only. All other use of SP for `Rt` in these instructions are not permitted in T32 code.

3.48.7 See also

Concepts

armasm User Guide:

- *Register-relative and PC-relative expressions* on page 10-7.

Reference

- *Memory access instructions* on page 3-9.
- *Pseudo-instructions* on page 3-25.
- *LDR (register-relative) in T32*.
- *FIELD* on page 10-39.
- *MAP* on page 10-69.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.49 LDR pseudo-instruction

Load a register with either:

- A 32-bit immediate value.
- An address.

Note

This description is for the LDR pseudo-instruction only, and not the LDR instruction.

3.49.1 Syntax

`LDR{cond}{.W} Rt, =expr`

`LDR{cond}{.W} Rt, =label_expr`

where:

cond is an optional condition code.

.W is an optional instruction width specifier.

Rt is the register to be loaded.

expr evaluates to a numeric value.

label_expr is a PC-relative or external expression of an address in the form of a label plus or minus a numeric value.

3.49.2 Usage

When using the LDR pseudo-instruction:

- If the value of *expr* can be loaded with a valid MOV or MVN instruction, the assembler uses that instruction.
- If a valid MOV or MVN instruction cannot be used, or if the label_expr syntax is used, the assembler places the constant in a literal pool and generates a PC-relative LDR instruction that reads the constant from the literal pool.

  Note

  — An address loaded in this way is fixed at link time, so the code is not position-independent.

  — The address holding the constant remains valid regardless of where the linker places the ELF section containing the LDR instruction.

The assembler places the value of *label_expr* in a literal pool and generates a PC-relative LDR instruction that loads the value from the literal pool.

If *label_expr* is an external expression, or is not contained in the current section, the assembler places a linker relocation directive in the object file. The linker generates the address at link time.

If *label_expr* is either a named or numeric local label, the assembler places a linker relocation directive in the object file and generates a symbol for that local label. The address is generated at link time. If the local label references T32 code, the T32 bit (bit 0) of the address is set.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

The offset from the PC to the value in the literal pool must be less than ±4KB (in A32 and in the 32-bit T32 encoding) or in the range 0 to +1KB (in the 16-bit T32 encoding). You are responsible for ensuring that there is a literal pool within range.

If the label referenced is in T32 code, the LDR pseudo-instruction sets the T32 bit (bit 0) of *label_expr*.

Note

In *RealView® Compilation Tools* (RVCT) v2.2, the T32 bit of the address was not set. If you have code that relies on this behavior, use the command line option `--untyped_local_labels` to force the assembler not to set the T32 bit when referencing labels in T32 code.

3.49.3 LDR in T32 code

You can use the `.W` width specifier to force LDR to generate a 32-bit instruction in T32 code. `LDR.W` always generates a 32-bit instruction, even if the immediate value could be loaded in a 16-bit MOV, or there is a literal pool within reach of a 16-bit PC-relative load.

If the value to be loaded is not known in the first pass of the assembler, LDR without `.W` generates a 16-bit instruction in T32 code, even if that results in a 16-bit PC-relative load for a value that could be generated in a 32-bit MOV or MVN instruction. However, if the value is known in the first pass, and it can be generated using a 32-bit MOV or MVN instruction, the MOV or MVN instruction is used.

The LDR pseudo-instruction never generates a 16-bit flag-setting MOV instruction. Use the `--diag_warning 1727` assembler command line option to check when a 16-bit instruction could have been used.

You can use the MOV32 pseudo-instruction for generating immediate values or addresses without loading from a literal pool.

3.49.4 Examples

```
LDR     r3,=0xff0    ; loads 0xff0 into R3
                     ; =>  MOV.W r3,#0xff0
LDR     r1,=0xfff    ; loads 0xfff into R1
                     ; =>  LDR r1,[pc,offset_to_litpool]
                     ;     ...
                     ;     litpool DCD 0xfff
LDR     r2,=place    ; loads the address of
                     ; place into R2
                     ; =>  LDR r2,[pc,offset_to_litpool]
                     ;     ...
                     ;     litpool DCD place
```

3.49.5 See also

Concepts

armasm User Guide:

- *Numeric constants* on page 10-5.
- *Register-relative and PC-relative expressions* on page 10-7.
- *Numeric local labels* on page 10-12.
- *Load immediates into registers* on page 7-6.
- *Load immediate 32-bit values to a register using LDR Rd, =const* on page 7-11.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Reference

- *Memory access instructions* on page 3-9.
- *LTORG* on page 10-65.
- *MOV32 pseudo-instruction* on page 3-102.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.50 LDA and STL

Load-Acquire and Store-Release Register.

Note

These instructions are supported only in ARMv8.

3.50.1 Syntax

`LDA{cond} Rt, [Rn]`

`STL{cond} Rt, [Rn]`

`LDAB{cond} Rt, [Rn]`

`STLB{cond} Rt, [Rn]`

`LDAH{cond} Rt, [Rn]`

`STLH{cond} Rt, [Rn]`

where:

cond is an optional condition code.

Rt is the register to load or store.

Rn is the register on which the memory address is based.

3.50.2 Operation

`LDA` loads data from memory. If any loads or stores appear after a load-acquire in program order, then all observers are guaranteed to observe the load-acquire before observing the loads and stores. Loads and stores appearing before a load-acquire are unaffected.

`STL` stores data to memory. If any loads or stores appear before a store-release in program order, then all observers are guaranteed to observe the loads and stores before observing the store-release. Loads and stores appearing after a store-release are unaffected.

In addition, if a store-release is followed by a load-acquire, each observer is guaranteed to observe them in program order.

There is no requirement that a load-acquire and store-release be paired.

All store-release operations are multi-copy atomic, meaning that in a multiprocessing system, if one observer observes a write to memory because of a store-release operation, then all observers observe it. Also, all observers observe all such writes to the same location in the same order.

3.50.3 Restrictions

The address specified must be naturally aligned, or an alignment fault is generated.

The PC must not be used for *Rt* or *Rn*.

3.50.4 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.50.5 See also

Reference

- *LDAEX and STLEX* on page 3-91.
- *Memory access instructions* on page 3-9.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.51 LDAEX and STLEX

Load-Acquire and Store-Release Register Exclusive.

Note

These instructions are supported only in ARMv8.

3.51.1 Syntax

`LDAEX{cond} Rt, [Rn]`

`STLEX{cond} Rd, Rt, [Rn]`

`LDAEXB{cond} Rt, [Rn]`

`STLEXB{cond} Rd, Rt, [Rn]`

`LDAEXH{cond} Rt, [Rn]`

`STLEXH{cond} Rd, Rt, [Rn]`

`LDAEXD{cond} Rt, Rt2, [Rn]`

`STLEXD{cond} Rd, Rt, Rt2, [Rn]`

where:

cond is an optional condition code.

Rd is the destination register for the returned status.

Rt is the register to load or store.

Rt2 is the second register for doubleword loads or stores.

Rn is the register on which the memory address is based.

3.51.2 LDAEX

`LDAEX` loads data from memory.

- If the physical address has the Shared TLB attribute, `LDAEX` tags the physical address as exclusive access for the current processor, and clears any exclusive access tag for this processor for any other physical address.
- Otherwise, it tags the fact that the executing processor has an outstanding tagged physical address.
- If any loads or stores appear after `LDAEX` in program order, then all observers are guaranteed to observe the `LDAEX` before observing the loads and stores. Loads and stores appearing before `LDAEX` are unaffected.

3.51.3 STLEX

`STLEX` performs a conditional store to memory. The conditions are as follows:

- If the physical address does not have the Shared TLB attribute, and the executing processor has an outstanding tagged physical address, the store takes place, the tag is cleared, and the value 0 is returned in *Rd*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

- If the physical address does not have the Shared TLB attribute, and the executing processor does not have an outstanding tagged physical address, the store does not take place, and the value 1 is returned in *Rd*.
- If the physical address has the Shared TLB attribute, and the physical address is tagged as exclusive access for the executing processor, the store takes place, the tag is cleared, and the value 0 is returned in *Rd*.
- If the physical address has the Shared TLB attribute, and the physical address is not tagged as exclusive access for the executing processor, the store does not take place, and the value 1 is returned in *Rd*.

If any loads or stores appear before STLEX in program order, then all observers are guaranteed to observe the loads and stores before observing the store-release. Loads and stores appearing after STLEX are unaffected.

All store-release operations are multi-copy atomic.

3.51.4 Restrictions

The PC must not be used for any of *Rd*, *Rt*, *Rt2*, or *Rn*.

For STLEX, *Rd* must not be the same register as *Rt*, *Rt2*, or *Rn*.

For A32 instructions:

- SP can be used but use of SP for any of *Rd*, *Rt*, or *Rt2* is deprecated.
- For LDAEXD and STLEXD, *Rt* must be an even numbered register, and not LR.
- *Rt2* must be *R(t+1)*.

For T32 instructions:

- SP can be used for *Rn*, but must not be used for any of *Rd*, *Rt*, or *Rt2*.
- For LDAEXD, *Rt* and *Rt2* must not be the same register.

3.51.5 Usage

Use LDAEX and STLEX to implement interprocess communication in multiple-processor and shared-memory systems.

For reasons of performance, keep the number of instructions between corresponding LDAEX and STLEX instructions to a minimum.

Note

The address used in a STLEX instruction must be the same as the address in the most recently executed LDAEX instruction.

3.51.6 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions.

3.51.7 See also

Reference

- *Memory access instructions* on page 3-9.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

- *LDA and STL* on page 3-89.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.52 LDREX and STREX

Load and Store Register Exclusive.

3.52.1 Syntax

`LDREX{cond} Rt, [Rn {, #offset}]`

`STREX{cond} Rd, Rt, [Rn {, #offset}]`

`LDREXB{cond} Rt, [Rn]`

`STREXB{cond} Rd, Rt, [Rn]`

`LDREXH{cond} Rt, [Rn]`

`STREXH{cond} Rd, Rt, [Rn]`

`LDREXD{cond} Rt, Rt2, [Rn]`

`STREXD{cond} Rd, Rt, Rt2, [Rn]`

where:

cond is an optional condition code.

Rd is the destination register for the returned status.

Rt is the register to load or store.

Rt2 is the second register for doubleword loads or stores.

Rn is the register on which the memory address is based.

offset is an optional offset applied to the value in *Rn*. *offset* is permitted only in T32 instructions. If *offset* is omitted, an offset of 0 is assumed.

3.52.2 LDREX

`LDREX` loads data from memory.

- If the physical address has the Shared TLB attribute, `LDREX` tags the physical address as exclusive access for the current processor, and clears any exclusive access tag for this processor for any other physical address.
- Otherwise, it tags the fact that the executing processor has an outstanding tagged physical address.

`LDREXB` and `LDREXH` zero extend the value loaded.

3.52.3 STREX

`STREX` performs a conditional store to memory. The conditions are as follows:

- If the physical address does not have the Shared TLB attribute, and the executing processor has an outstanding tagged physical address, the store takes place, the tag is cleared, and the value 0 is returned in *Rd*.
- If the physical address does not have the Shared TLB attribute, and the executing processor does not have an outstanding tagged physical address, the store does not take place, and the value 1 is returned in *Rd*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

- If the physical address has the Shared TLB attribute, and the physical address is tagged as exclusive access for the executing processor, the store takes place, the tag is cleared, and the value 0 is returned in *Rd*.
- If the physical address has the Shared TLB attribute, and the physical address is not tagged as exclusive access for the executing processor, the store does not take place, and the value 1 is returned in *Rd*.

3.52.4 Restrictions

The PC must not be used for any of *Rd*, *Rt*, *Rt2*, or *Rn*.

For STREX, *Rd* must not be the same register as *Rt*, *Rt2*, or *Rn*.

For A32 instructions:

- SP can be used but use of SP for any of *Rd*, *Rt*, or *Rt2* is deprecated.
- For LDREXD and STREXD, *Rt* must be an even numbered register, and not LR.
- *Rt2* must be *R(t+1)*.
- *offset* is not permitted.

For T32 instructions:

- SP can be used for R*n*, but must not be used for any of *Rd*, *Rt*, or *Rt2*.
- For LDREXD, *Rt* and *Rt2* must not be the same register.
- The value of *offset* can be any multiple of four in the range 0-1020.

3.52.5 Usage

Use LDREX and STREX to implement interprocess communication in multiple-processor and shared-memory systems.

For reasons of performance, keep the number of instructions between corresponding LDREX and STREX instructions to a minimum.

Note

The address used in a STREX instruction must be the same as the address in the most recently executed LDREX instruction.

3.52.6 Availability

All these 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions.

3.52.7 Examples

```
    MOV r1, #0x1               ; load the 'lock taken' value
try
    LDREX r0, [LockAddr]       ; load the lock value
    CMP r0, #0                 ; is the lock free?
    STREXEQ r0, r1, [LockAddr] ; try and claim the lock
    CMPEQ r0, #0               ; did this succeed?
    BNE try                    ; no - try again
    ....                       ; yes - we have the lock
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.52.8 See also

Reference

- *Memory access instructions* on page 3-9.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.53 MCR and MCRR

Move to Coprocessor from ARM Register or Registers. Depending on the coprocessor, you might be able to specify various operations in addition.

3.53.1 Syntax

`MCR{cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}`

`MCRR{cond} coproc, #opcode3, Rt, Rt2, CRm`

where:

cond is an optional condition code.

coproc is the name of the coprocessor the instruction is for. The standard name is `pn`, where *n* must be either 14 or 15.

opcode1 is a 3-bit coprocessor-specific opcode.

opcode2 is an optional 3-bit coprocessor-specific opcode.

opcode3 is a 4-bit coprocessor-specific opcode.

Rt, *Rt2* are ARM source registers. `Rt` and `Rt2` must not be PC.

CRn, *CRm* are coprocessor registers.

3.53.2 Usage

The use of these instructions depends on the coprocessor. See the coprocessor documentation for details.

3.53.3 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.53.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.54 MOV and MVN

Move and Move Not.

3.54.1 Syntax

`MOV{S}{cond} Rd, Operand2`

`MOV{cond} Rd, #imm16`

`MVN{S}{cond} Rd, Operand2`

where:

`S` is an optional suffix. If `S` is specified, the condition flags are updated on the result of the operation.

`cond` is an optional condition code.

`Rd` is the destination register.

`Operand2` is a flexible second operand.

`imm16` is any value in the range 0-65535.

3.54.2 Usage

The `MOV` instruction copies the value of *`Operand2`* into *`Rd`*.

The `MVN` instruction takes the value of *`Operand2`*, performs a bitwise logical NOT operation on the value, and places the result into *`Rd`*.

In certain circumstances, the assembler can substitute `MVN` for `MOV`, or `MOV` for `MVN`. Be aware of this when reading disassembly listings.

3.54.3 Use of PC and SP in 32-bit T32 instructions

You cannot use PC (`R15`) for *`Rd`*, or in *`Operand2`*, in 32-bit T32 `MOV` or `MVN` instructions. With the following exceptions, you cannot use SP (`R13`) for *`Rd`*, or in *`Operand2`*:

- `MOV{cond}.W Rd, SP`, where *`Rd`* is not SP.
- `MOV{cond}.W SP, Rm`, where *`Rm`* is not SP.

3.54.4 Use of PC and SP in 16-bit T32 instructions

You can use PC or SP in 16-bit T32 `MOV{cond} Rd, Rm` instructions but these instructions in which both *`Rd`* and *`Rm`* are SP or PC are deprecated.

You cannot use PC or SP in any other `MOV{S}` or `MVN{S}` 16-bit T32 instructions.

3.54.5 Use of PC and SP in A32 MOV and MVN

You cannot use PC for *`Rd`* or any operand in any data processing instruction that has a register-controlled shift.

In instructions without register-controlled shift, the use of PC is deprecated except for the following cases:

- `MOVS PC, LR`.
- `MOV PC, Rm` when `Rm` is not PC or SP.
- `MOV Rd, PC` when `Rd` is not PC or SP.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

You can use SP for R*d* or R*m*. But this is deprecated except for the following cases:

- `MOV SP, Rm` when R*m* is not PC or SP.
- `MOV Rd, SP` when R*d* is not PC or SP.

Note

- You cannot use PC for R*d* in `MOV Rd, #imm16` if the `#imm16` value is not a permitted `Operand2` value. You can use PC in forms with `Operand2` without register-controlled shift.

If you use PC as *Rm*, the value used is the address of the instruction plus 8.

If you use PC as *Rd*:

- Execution branches to the address corresponding to the result.
- If you use the `S` suffix, see the `SUBS pc,lr` instruction.

3.54.6 Condition flags

If `S` is specified, these instructions:

- Update the N and Z flags according to the result.
- Can update the C flag during the calculation of *Operand2*.
- Do not affect the V flag.

3.54.7 16-bit instructions

The following forms of these instructions are available in T32 code, and are 16-bit instructions:

`MOVS Rd, #imm`

Rd must be a Lo register. *imm* range 0-255. This form can only be used outside an IT block.

`MOV{cond} Rd, #imm`

Rd must be a Lo register. *imm* range 0-255. This form can only be used inside an IT block.

`MOVS Rd, Rm`

Rd and *Rm* must both be Lo registers. This form can only be used outside an IT block.

`MOV{cond} Rd, Rm`

Rd and *Rm* can be Lo or Hi registers.

`MVNS Rd, Rm`

Rd and *Rm* must both be Lo registers. This form can only be used outside an IT block.

`MVN{cond} Rd, Rm`

Rd and *Rm* must both be Lo registers. This form can only be used inside an IT block.

3.54.8 Availability

These instructions are available in A32 and T32.

In T32, 16-bit and 32-bit versions of these instructions are available.

3.54.9 Example

```
MVNNE   r11, #0xF000000B ; A32 only. This immediate value is not
                         ; available in T32.
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.54.10 Incorrect example

```
MVN     pc,r3,ASR r0     ; PC not permitted with register-controlled shift
```

3.54.11 See also

Concepts

- *Flexible second operand (Operand2)* on page 3-12.
- *Instruction substitution* on page 3-13.

Reference

- *Condition codes* on page 3-26.
- *SUBS pc, lr* on page 3-154.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.55 MOVT

Move Top. Writes a 16-bit immediate value to the top halfword of a register, without affecting the bottom halfword.

3.55.1 Syntax

`MOVT{cond} Rd, #imm16`

where:

cond	is an optional condition code.
Rd	is the destination register.
imm16	is a 16-bit immediate value.

3.55.2 Usage

`MOVT` writes *imm16* to *Rd*[31:16]. The write does not affect *Rd*[15:0].

You can generate any 32-bit immediate with a `MOV`, `MOVT` instruction pair. The assembler implements the `MOV32` pseudo-instruction for convenient generation of this instruction pair.

3.55.3 Register restrictions

You cannot use PC in A32 or T32 instructions.

You can use SP for *Rd* in A32 instructions but this is deprecated.

You cannot use SP in T32 instructions.

3.55.4 Condition flags

This instruction does not change the flags.

3.55.5 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit version of this instruction in T32.

3.55.6 See also

Reference

- *MOV32 pseudo-instruction* on page 3-102.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.56 MOV32 pseudo-instruction

Load a register with either:

- A 32-bit immediate value.
- Any address.

`MOV32` always generates two 32-bit instructions, a `MOV`, `MOVT` pair. This enables you to load any 32-bit immediate, or to access the whole 32-bit address space.

3.56.1 Syntax

`MOV32{`*`cond`*`} `*`Rd`*`, `*`expr`*

where:

cond is an optional condition code.

Rd is the register to be loaded. *Rd* must not be SP or PC.

expr can be any one of the following:

symbol	A label in this or another program area.
#constant	Any 32-bit immediate value.
symbol + constant	A label plus a 32-bit immediate value.

3.56.2 Usage

The main purposes of the `MOV32` pseudo-instruction are:

- To generate literal constants when an immediate value cannot be generated in a single instruction.
- To load a PC-relative or external address into a register. The address remains valid regardless of where the linker places the ELF section containing the `MOV32`.

  Note

  An address loaded in this way is fixed at link time, so the code is not position-independent.

`MOV32` sets the T32 bit (bit 0) of the address if the label referenced is in T32 code.

3.56.3 Availability

This pseudo-instruction is available in both A32 and T32.

3.56.4 Examples

```
MOV32 r3, #0xABCDEF12   ; loads 0xABCDEF12 into R3
MOV32 r1, Trigger+12    ; loads the address that is 12 bytes higher than
                        ; the address Trigger into R1
```

3.56.5 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.57 MRC and MRRC

Move to ARM Register or Registers from Coprocessor.

Depending on the coprocessor, you might be able to specify various operations in addition.

3.57.1 Syntax

`MRC{cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}`

`MRRC{cond} coproc, #opcode3, Rt, Rt2, CRm`

where:

cond is an optional condition code.

coproc is the name of the coprocessor the instruction is for. The standard name is `pn`, where *n* must be either 14 or 15.

opcode1 is a 3-bit coprocessor-specific opcode.

opcode2 is an optional 3-bit coprocessor-specific opcode.

opcode3 is a 4-bit coprocessor-specific opcode.

Rt, *Rt2* are ARM destination registers. `Rt` and `Rt2` must not be PC.

In `MRC`, *Rt* can be `APSR_nzcv`. This means that the coprocessor executes an instruction that changes the value of the condition flags in the APSR.

CRn, *CRm* are coprocessor registers.

3.57.2 Usage

The use of these instructions depends on the coprocessor. See the coprocessor documentation for details.

3.57.3 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.57.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.58 MRS (system coprocessor register to ARM register)

Move to ARM register from system coprocessor register.

3.58.1 Syntax

`MRS{cond} Rn, coproc_register`

`MRS{cond} APSR_nzcv, special_register`

where:

cond is an optional condition code.

coproc_register
is the name of the coprocessor register.

special_register
is the name of the coprocessor register that can be written to APSR_nzcv. This is only possible for the coprocessor register DBGDSCRint.

Rn is the ARM destination register. *Rn* must not be PC.

3.58.2 Usage

You can use this pseudo-instruction to read CP14 or CP15 coprocessor registers, with the exception of write-only registers. A complete list of the applicable coprocessor register names is in the *ARM Architecture Reference Manual*. For example:

```
MRS R1, SCTLR ; writes the contents of the CP15 coprocessor register SCTLR
              ; into R1
```

3.58.3 Availability

This 32-bit pseudo-instruction is available in A32 and T32.

There is no 16-bit version of this pseudo-instruction in T32.

3.58.4 See also

Reference

- *Condition codes* on page 3-26.
- *MSR (ARM register to system coprocessor register)* on page 3-107.
- *MSR (general-purpose register to PSR)* on page 3-108.
- *MRS (PSR to general-purpose register)* on page 3-105.

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.59 MRS (PSR to general-purpose register)

Move the contents of a PSR to a general-purpose register.

3.59.1 Syntax

`MRS{cond} Rd, psr`

where:

cond is an optional condition code.

Rd is the destination register.

psr is one of:

- `APSR` in any processor mode.
- `CPSR` deprecated synonym for APSR and for use in Debug state.
- `SPSR` in privileged software execution only.

3.59.2 Usage

Use `MRS` in combination with `MSR` as part of a read-modify-write sequence for updating a PSR, for example to change processor mode, or to clear the Q flag.

In process swap code, the programmers' model state of the process being swapped out must be saved, including relevant PSR contents. Similarly, the state of the process being swapped in must also be restored. These operations make use of `MRS`/store and load/`MSR` instruction sequences.

3.59.3 SPSR

You must not attempt to access the SPSR when the processor is in User or System mode. This is your responsibility. The assembler cannot warn you about this, because it has no information about the processor mode at execution time.

3.59.4 CPSR

ARM deprecates reading the CPSR endianness bit (E) with an `MRS` instruction.

The CPSR execution state bits, other than the E bit, can only be read when the processor is in Debug state, halting debug-mode. Otherwise, the execution state bits in the CPSR read as zero.

The condition flags can be read in any mode on any processor. Use APSR if you are only interested in accessing the condition flags in User mode.

3.59.5 Register restrictions

You cannot use PC for `Rd` in A32 instructions. You can use SP for `Rd` in A32 instructions but this is deprecated.

You cannot use PC or SP for `Rd` in T32 instructions.

3.59.6 Condition flags

This instruction does not change the flags.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.59.7 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit version of this instruction in T32.

3.59.8 See also

Concepts

armasm User Guide:

- *Current Program Status Register in AArch32* on page 4-13.

Reference

- *MSR (general-purpose register to PSR)* on page 3-108.
- *MSR (ARM register to system coprocessor register)* on page 3-107.
- *MRS (system coprocessor register to ARM register)* on page 3-104.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.60 MSR (ARM register to system coprocessor register)

Move to system coprocessor register from ARM register.

3.60.1 Syntax

`MSR{cond} coproc_register, Rn`

where:

cond is an optional condition code.

coproc_register
is the name of the coprocessor register.

Rn is the ARM source register. `Rn` must not be PC.

3.60.2 Usage

You can use this pseudo-instruction to write to any CP14 or CP15 coprocessor writable register. A complete list of the applicable coprocessor register names is in the *ARM Architecture Reference Manual*. For example:

```
MSR SCTLR, R1 ; writes the contents of R1 into the CP15 coprocessor register
              ; SCTLR
```

3.60.3 Availability

This 32-bit pseudo-instruction is available in A32 and T32.

There is no 16-bit T32 version of this pseudo-instruction.

3.60.4 See also

Reference

- *SYS* on page 3-159.
- *MRS (system coprocessor register to ARM register)* on page 3-104.
- *MRS (PSR to general-purpose register)* on page 3-105.
- *MSR (general-purpose register to PSR)* on page 3-108.
- *Condition codes* on page 3-26.

Other information

- *ARM Architecture Reference Manual*
  `http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/`.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.61 MSR (general-purpose register to PSR)

Load an immediate value, or the contents of a general-purpose register, into specified fields of a *Program Status Register* (PSR).

3.61.1 Syntax

`MSR{cond} APSR_flags, Rm`

`MSR{cond} APSR_flags, #constant`

`MSR{cond} psr_fields, #constant`

`MSR{cond} psr_fields, Rm`

where:

cond is an optional condition code.

flags specifies the APSR flags to be moved. *flags* can be one or more of:

- `nzcvq` ALU flags field mask, PSR[31:27] (User mode)
- `g` SIMD GE flags field mask, PSR[19:16] (User mode).

Rm is the source register. *Rm* must not be PC.

constant is an expression evaluating to a numeric value. The value must correspond to an 8-bit pattern rotated by an even number of bits within a 32-bit word. Not available in T32.

psr is one of:

- `CPSR` for use in Debug state, also deprecated synonym for APSR
- `SPSR` on any processor, in privileged software execution only.

fields specifies the SPSR or CPSR fields to be moved. *fields* can be one or more of:

- `c` control field mask byte, PSR[7:0] (privileged software execution)
- `x` extension field mask byte, PSR[15:8] (privileged software execution)
- `s` status field mask byte, PSR[23:16] (privileged software execution)
- `f` flags field mask byte, PSR[31:24] (privileged software execution).

3.61.2 Usage

In User mode:

- Use APSR to access condition flags, Q, or GE bits.
- Writes to unallocated, privileged or execution state bits in the CPSR are ignored. This ensures that User mode programs cannot change to privileged software execution.

ARM deprecates using `MSR` to change the endianness bit (E) of the CPSR, in any mode.

You must not attempt to access the SPSR when the processor is in User or System mode.

3.61.3 Register restrictions

You cannot use PC for *Rm* in A32 instructions. You can use SP for *Rm* in A32 instructions but this is deprecated.

You cannot use PC or SP for *Rm* in T32 instructions.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.61.4 Condition flags

This instruction updates the flags explicitly if the APSR_nzcvq or CPSR_f field is specified.

3.61.5 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit version of this instruction in T32.

3.61.6 See also

Reference

- *MRS (PSR to general-purpose register)* on page 3-105.
- *MRS (system coprocessor register to ARM register)* on page 3-104.
- *MSR (ARM register to system coprocessor register)* on page 3-107.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.62 MUL, MLA, and MLS

Multiply, Multiply-Accumulate, and Multiply-Subtract, with signed or unsigned 32-bit operands, giving the least significant 32 bits of the result.

3.62.1 Syntax

`MUL{S}{cond} {Rd}, Rn, Rm`

`MLA{S}{cond} Rd, Rn, Rm, Ra`

`MLS{cond} Rd, Rn, Rm, Ra`

where:

cond is an optional condition code.

S is an optional suffix. If S is specified, the condition flags are updated on the result of the operation.

Rd is the destination register.

Rn, Rm are registers holding the values to be multiplied.

Ra is a register holding the value to be added or subtracted from.

3.62.2 Usage

The MUL instruction multiplies the values from *Rn* and *Rm*, and places the least significant 32 bits of the result in *Rd*.

The MLA instruction multiplies the values from *Rn* and *Rm*, adds the value from *Ra*, and places the least significant 32 bits of the result in *Rd*.

The MLS instruction multiplies the values from *Rn* and *Rm*, subtracts the result from the value from *Ra*, and places the least significant 32 bits of the final result in *Rd*.

3.62.3 Register restrictions

You cannot use PC for any register.

You can use SP in A32 instructions but this is deprecated. You cannot use SP in T32 instructions.

3.62.4 Condition flags

If S is specified, the MUL and MLA instructions:

- Update the N and Z flags according to the result.
- Do not affect the C or V flag.

3.62.5 16-bit instructions

The following forms of the MUL instruction are available in T32 code, and are 16-bit instructions:

`MULS Rd, Rn, Rd`

Rd and *Rn* must both be Lo registers. This form can only be used outside an IT block.

`MUL{cond} Rd, Rn, Rd`

Rd and *Rn* must both be Lo registers. This form can only be used inside an IT block.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

There are no other T32 multiply instructions that can update the condition flags.

3.62.6 Availability

These 32-bit instructions are available in A32 and T32.

The `MUL{S}` instruction is available in T32 in a 16-bit encoding.

3.62.7 Examples

```
MUL     r10, r2, r5
MLA     r10, r2, r1, r5
MULS    r0, r2, r2
MULLT   r2, r3, r2
MLS     r4, r5, r6, r7
```

3.62.8 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.63 NEG pseudo-instruction

Negate the value in a register.

3.63.1 Syntax

`NEG{cond} Rd, Rm`

where:

cond	is an optional condition code.
Rd	is the destination register.
Rm	is the register containing the value that is subtracted from zero.

3.63.2 Usage

The `NEG` pseudo-instruction negates the value in one register and stores the result in a second register.

`NEG{cond} Rd, Rm` assembles to `RSBS{cond} Rd, Rm, #0`.

3.63.3 Availability

This pseudo-instruction is available in A32 and T32.

3.63.4 Register restrictions

In A32 instructions, using SP or PC for *Rd* or *Rm* is deprecated. In T32 instructions, you cannot use SP or PC for *Rd* or *Rm*.

3.63.5 Condition flags

This pseudo-instruction updates the condition flags, based on the result.

3.63.6 See also

Reference

- *ADD, SUB, RSB, ADC, SBC, and RSC* on page 3-27.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.64 NOP

No Operation.

3.64.1 Syntax

`NOP{`*cond*`}`

where:

cond is an optional condition code.

3.64.2 Usage

`NOP` does nothing. If `NOP` is not implemented as a specific instruction on your target architecture, the assembler treats it as a pseudo-instruction and generates an alternative instruction that does nothing, such as `MOV r0, r0` (A32) or `MOV r8, r8` (T32).

`NOP` is not necessarily a time-consuming `NOP`. The processor might remove it from the pipeline before it reaches the execution stage.

You can use `NOP` for padding, for example to place the following instruction on a 64-bit boundary in A32, or a 32-bit boundary in T32.

3.64.3 Availability

This instruction is available in A32 and T32.

In T32, this instruction is available in 16-bit and 32-bit encodings.

3.64.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.65 Parallel add and subtract

Various byte-wise and halfword-wise additions and subtractions.

3.65.1 Syntax

`<prefix>op{cond} {Rd}, Rn, Rm`

where:

`<prefix>` is one of:

`S`	Signed arithmetic modulo 2^8 or 2^{16}. Sets the APSR GE flags.
`Q`	Signed saturating arithmetic.
`SH`	Signed arithmetic, halving the results.
`U`	Unsigned arithmetic modulo 2^8 or 2^{16}. Sets the APSR GE flags.
`UQ`	Unsigned saturating arithmetic.
`UH`	Unsigned arithmetic, halving the results.

op is one of:

`ADD8`	Byte-wise Addition
`ADD16`	Halfword-wise Addition.
`SUB8`	Byte-wise Subtraction.
`SUB16`	Halfword-wise Subtraction.
`ASX`	Exchange halfwords of *Rm*, then Add top halfwords and Subtract bottom halfwords.
`SAX`	Exchange halfwords of *Rm*, then Subtract top halfwords and Add bottom halfwords.

cond is an optional condition code.

Rd is the destination register.

Rm, Rn are the registers holding the operands.

3.65.2 Operation

These instructions perform arithmetic operations separately on the bytes or halfwords of the operands. They perform two or four additions or subtractions, or one addition and one subtraction.

You can choose various kinds of arithmetic:

- Signed or unsigned arithmetic modulo 2^8 or 2^{16}. This sets the APSR GE flags.
- Signed saturating arithmetic to one of the signed ranges $-2^{15} \le x \le 2^{15}-1$ or $-2^7 \le x \le 2^7-1$. The Q flag is not affected even if these operations saturate.
- Unsigned saturating arithmetic to one of the unsigned ranges $0 \le x \le 2^{16}-1$ or $0 \le x \le 2^8-1$. The Q flag is not affected even if these operations saturate.
- Signed or unsigned arithmetic, halving the results. This cannot cause overflow.

3.65.3 Register restrictions

You cannot use PC for any register.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.65.4 GE flags

These instructions do not affect the N, Z, C, V, or Q flags.

The `Q`, `SH`, `UQ` and `UH` prefix variants of these instructions do not change the flags.

The `S` and `U` prefix variants of these instructions set the GE flags in the APSR as follows:

- The `S` prefix variants set the flags to indicate whether the result is greater than or equal to zero. This is equivalent to an `ADDS` or `SUBS` instruction setting the N and V condition flags to the same value, so that the GE condition passes.
- The `U` prefix variants set the flags to indicate the following:
  - For an addition, whether the result overflowed, generating a carry. This is equivalent to an `ADDS` instruction setting the C condition flag to 1.
  - For a subtraction, whether the result is greater than or equal to zero, meaning a borrow did not occur. This is equivalent to a `SUBS` instruction setting the C condition flag to 1.

For byte-wise operations, the GE flags are set as follows:

`GE[0]` for bits[7:0] of the result

`GE[1]` for bits[15:8] of the result

`GE[2]` for bits[23:16] of the result

`GE[3]` for bits[31:24] of the result.

For halfword-wise operations, the GE flags are set as follows:

`GE[1:0]` for bits[15:0] of the result

`GE[3:2]` for bits[31:16] of the result.

You can use these flags to control a following `SEL` instruction.

Note

For halfword-wise operations, GE[1:0] are set or cleared together, and GE[3:2] are set or cleared together.

3.65.5 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.65.6 Examples

```
SHADD8      r4, r3, r9
USAXNE      r0, r0, r2
```

3.65.7 Incorrect examples

```
QHADD8      r2, r9, r3    ; No such instruction
SAX         r10, r8, r5   ; Must have a prefix.
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.65.8 See also

Reference

- *SEL* on page 3-131.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.66 PKHBT and PKHTB

Halfword Packing instructions.

Combine a halfword from one register with a halfword from another register. One of the operands can be shifted before extraction of the halfword.

3.66.1 Syntax

`PKHBT{cond} {Rd}, Rn, Rm{, LSL #leftshift}`

`PKHTB{cond} {Rd}, Rn, Rm{, ASR #rightshift}`

where:

`PKHBT` combines bits[15:0] of *Rn* with bits[31:16] of the shifted value from *Rm*.

`PKHTB` combines bits[31:16] of *Rn* with bits[15:0] of the shifted value from *Rm*.

cond is an optional condition code.

Rd is the destination register.

Rn is the register holding the first operand.

Rm is the register holding the first operand.

leftshift is in the range 0 to 31.

rightshift is in the range 1 to 32.

3.66.2 Register restrictions

You cannot use PC for any register.

You can use SP in A32 instructions but this is deprecated. You cannot use SP in T32 instructions.

3.66.3 Condition flags

These instructions do not change the flags.

3.66.4 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.66.5 Examples

```
PKHBT   r0, r3, r5          ; combine the bottom halfword of R3 with
                            ; the top halfword of R5
PKHBT   r0, r3, r5, LSL #16 ; combine the bottom halfword of R3 with
                            ; the bottom halfword of R5
PKHTB   r0, r3, r5, ASR #16 ; combine the top halfword of R3 with
                            ; the top halfword of R5
```

You can also scale the second operand by using different values of shift.

3.66.6 Incorrect examples

```
PKHBTEQ r4, r5, r1, ASR #8  ; ASR not permitted with PKHBT
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.66.7 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.67 PLD, PLDW, and PLI

Preload Data and Preload Instruction. The processor can signal the memory system that a data or instruction load from an address is likely in the near future.

3.67.1 Syntax

`PL`*type*`{`*cond*`} [`*Rn* `{, #`*offset*`}]`

`PL`*type*`{`*cond*`} [`*Rn*`, +/-`*Rm* `{,` *shift*`}]`

`PL`*type*`{`*cond*`}` *label*

where:

type can be one of:

- `D` Data address
- `DW` Data address with intention to write
- `I` Instruction address.

type cannot be `DW` if the syntax specifies *label*.

cond is an optional condition code.

Note

cond is permitted only in T32 code, using a preceding `IT` instruction, but this is deprecated. This is an unconditional instruction in A32 code and you must not use *cond*.

Rn is the register on which the memory address is based.

offset is an immediate offset. If offset is omitted, the address is the value in *Rn*.

Rm is a register containing a value to be used as the offset.

shift is an optional shift.

label is a PC-relative expression.

3.67.2 Range of offset

The offset is applied to the value in *Rn* before the preload takes place. The result is used as the memory address for the preload. The range of offsets permitted is:

- –4095 to +4095 for A32 instructions.
- –255 to +4095 for T32 instructions, when *Rn* is not PC.
- –4095 to +4095 for T32 instructions, when *Rn* is PC.

The assembler calculates the offset from the PC for you. The assembler generates an error if *label* is out of range.

3.67.3 Register or shifted register offset

In A32 code, the value in *Rm* is added to or subtracted from the value in *Rn*. In T32 code, the value in *Rm* can only be added to the value in *Rn*. The result is used as the memory address for the preload.

The range of shifts permitted is:

- `LSL` #0 to #3 for T32 instructions.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

- Any one of the following for A32 instructions:
  - LSL #0 to #31.
  - LSR #1 to #32.
  - ASR #1 to #32.
  - ROR #1 to #31.
  - RRX.

3.67.4 Address alignment for preloads

No alignment checking is performed for preload instructions.

3.67.5 Register restrictions

Rm must not be PC. For T32 instructions *Rm* must also not be SP.

Rn must not be PC for T32 instructions of the syntax PL*type*{*cond*} [R*n*, +/-R*m*{, #*shift*}].

3.67.6 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit PLD, PLDW, or PLI instructions in T32.

These are hint instructions, and their implementation is optional. If they are not implemented, they execute as NOPs.

3.67.7 See also

Concepts

armasm User Guide:

- *Register-relative and PC-relative expressions* on page 10-7.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.68 PUSH and POP

Push registers onto, and pop registers off a full descending stack.

3.68.1 Syntax

`PUSH{`*`cond`*`} `*`reglist`*

`POP{`*`cond`*`} `*`reglist`*

where:

`cond` is an optional condition code.

`reglist` is a non-empty list of registers, enclosed in braces.It can contain register ranges. It must be comma separated if it contains more than one register or register range.

3.68.2 Usage

`PUSH` is a synonym for `STMDB sp!, reglist` and `POP` is a synonym for `LDMIA sp! reglist`. `PUSH` and `POP` are the preferred mnemonics in these cases.

Note

`LDM` and `LDMFD` are synonyms of `LDMIA`. `STMFD` is a synonym of `STMDB`.

Registers are stored on the stack in numerical order, with the lowest numbered register at the lowest address.

3.68.3 POP, with reglist including the PC

This instruction causes a branch to the address popped off the stack into the PC. This is usually a return from a subroutine, where the LR was pushed onto the stack at the start of the subroutine.

Bits[1:0] of the address loaded must not be 0b10. If bit[0] is 1, execution continues in T32 state. If bit[0] is 0, execution continues in A32 state.

3.68.4 T32 instructions

A subset of these instructions are available in the T32 instruction set.

The following restrictions apply to the 16-bit instructions:

- For `PUSH`, *`reglist`* can only include the Lo registers and the LR.
- For `POP`, *`reglist`* can only include the Lo registers and the PC.

The following restrictions apply to the 32-bit instructions:

- *`reglist`* must not include the SP.
- For `PUSH`, *`reglist`* must not include the PC.
- For `POP`, *`reglist`* can include either the LR or the PC, but not both.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.68.5 Restrictions on reglist in A32 instructions

A32 `PUSH` instructions can have SP and PC in the *reglist* but these instructions that include SP or PC in the *reglist* are deprecated.

A32 `POP` instructions cannot have SP but can have PC in the *reglist*. These instructions that include both PC and LR in the *reglist* are deprecated.

3.68.6 Examples

```
PUSH    {r0,r4-r7}
PUSH    {r2,lr}
POP     {r0,r10,pc} ; no 16-bit version available
```

3.68.7 See also

Reference

- *Memory access instructions* on page 3-9.
- *LDM and STM* on page 3-71.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.69 QADD, QSUB, QDADD, and QDSUB

Signed Add, Subtract, Double and Add, Double and Subtract, saturating the result to the signed range $-2^{31} \leq x \leq 2^{31}-1$.

3.69.1 Syntax

op{*cond*} {*Rd*}, *Rm*, *Rn*

where:

op is one of QADD, QSUB, QDADD, or QDSUB.

cond is an optional condition code.

Rd is the destination register.

Rm, Rn are the registers holding the operands.

3.69.2 Usage

The QADD instruction adds the values in *Rm* and *Rn*.

The QSUB instruction subtracts the value in *Rn* from the value in *Rm*.

The QDADD instruction calculates SAT(*Rm* + SAT(*Rn* * 2)). Saturation can occur on the doubling operation, on the addition, or on both. If saturation occurs on the doubling but not on the addition, the Q flag is set but the final result is unsaturated.

The QDSUB instruction calculates SAT(*Rm* - SAT(*Rn* * 2)). Saturation can occur on the doubling operation, on the subtraction, or on both. If saturation occurs on the doubling but not on the subtraction, the Q flag is set but the final result is unsaturated.

Note

All values are treated as two's complement signed integers by these instructions.

3.69.3 Register restrictions

You cannot use PC for any operand.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.69.4 Q flag

If saturation occurs, these instructions set the Q flag. To read the state of the Q flag, use an MRS instruction.

3.69.5 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit T32 versions of these instructions.

3.69.6 Examples

```
QADD    r0, r1, r9
QDSUBLT r9, r0, r1
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.69.7 See also

Reference

- *Parallel add and subtract* on page 3-114.
- *MRS (PSR to general-purpose register)* on page 3-105.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.70 REV, REV16, REVSH, and RBIT

Reverse bytes or bits within words or halfwords.

3.70.1 Syntax

op{*cond*} *Rd*, *Rn*

where:

op is any one of the following:

- REV — Reverse byte order in a word.
- REV16 — Reverse byte order in each halfword independently.
- REVSH — Reverse byte order in the bottom halfword, and sign extend to 32 bits.
- RBIT — Reverse the bit order in a 32-bit word.

cond is an optional condition code.

Rd is the destination register.

Rn is the register holding the operand.

3.70.2 Usage

You can use these instructions to change endianness:

REV converts 32-bit big-endian data into little-endian data or 32-bit little-endian data into big-endian data.

REV16 converts 16-bit big-endian data into little-endian data or 16-bit little-endian data into big-endian data.

REVSH converts either:

- 16-bit signed big-endian data into 32-bit signed little-endian data
- 16-bit signed little-endian data into 32-bit signed big-endian data.

3.70.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.70.4 Condition flags

These instructions do not change the flags.

3.70.5 16-bit instructions

The following forms of these instructions are available in T32 code, and are 16-bit instructions:

REV *Rd*, *Rm* *Rd* and *Rm* must both be Lo registers.

REV16 *Rd*, *Rm* *Rd* and *Rm* must both be Lo registers.

REVSH *Rd*, *Rm* *Rd* and *Rm* must both be Lo registers.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.70.6 Availability

These instructions are available in A32 and T32.

In T32, all these instructions are available in a 32-bit encoding, and all except `RBIT` are available in a 16-bit encoding.

3.70.7 Examples

```
REV     r3, r7
REV16   r0, r0
REVSH   r0, r5       ; Reverse Signed Halfword
REVHS   r3, r7       ; Reverse with Higher or Same condition
RBIT    r7, r8
```

3.70.8 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.71 RFE

Return From Exception.

3.71.1 Syntax

`RFE{addr_mode}{cond} Rn{!}`

where:

addr_mode is any one of the following:

- `IA` Increment address After each transfer (Full Descending stack)
- `IB` Increment address Before each transfer (A32 only)
- `DA` Decrement address After each transfer (A32 only)
- `DB` Decrement address Before each transfer.

If *addr_mode* is omitted, it defaults to Increment After.

cond is an optional condition code.

Note

cond is permitted only in T32 code, using a preceding `IT` instruction, but this is deprecated. This is an unconditional instruction in A32.

Rn specifies the base register. *Rn* must not be PC.

`!` is an optional suffix. If `!` is present, the final address is written back into *Rn*.

3.71.2 Usage

You can use `RFE` to return from an exception if you previously saved the return state using the `SRS` instruction. *Rn* is usually the `SP` where the return state information was saved.

3.71.3 Operation

Loads the PC and the CPSR from the address contained in *Rn*, and the following address. Optionally updates *Rn*.

3.71.4 Notes

`RFE` writes an address to the PC. The alignment of this address must be correct for the instruction set in use after the exception return:

- For a return to A32, the address written to the PC must be word-aligned.
- For a return to T32, the address written to the PC must be halfword-aligned.

No special precautions are required in software to follow these rules, if you use the instruction to return after a valid exception entry mechanism.

Where addresses are not word-aligned, `RFE` ignores the least significant two bits of *Rn*.

The time order of the accesses to individual words of memory generated by `RFE` is not architecturally defined. Do not use this instruction on memory-mapped I/O locations where access order matters.

Do not use `RFE` in unprivileged software execution.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.71.5 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit version of this instruction.

3.71.6 Example

```
RFE sp!
```

3.71.7 See also

Concepts

armasm User Guide:

- *Processor modes, and privileged and unprivileged software execution* on page 4-3.

Reference

- *SRS* on page 3-148.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.72 SBFX and UBFX

Signed and Unsigned Bit Field Extract. Copies adjacent bits from one register into the least significant bits of a second register, and sign extends or zero extends to 32 bits.

3.72.1 Syntax

`op{cond} Rd, Rn, #lsb, #width`

where:

op is either SBFX or UBFX.

cond is an optional condition code.

Rd is the destination register.

Rn is the source register.

lsb is the bit number of least significant bit in the bitfield, in the range 0 to 31.

width is the width of the bitfield, in the range 1 to (32–*lsb*).

3.72.2 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.72.3 Condition flags

These instructions do not alter any flags.

3.72.4 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.72.5 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.73 SDIV and UDIV

Signed and Unsigned Divide.

3.73.1 Syntax

`SDIV{cond} {Rd}, Rn, Rm`

`UDIV{cond} {Rd}, Rn, Rm`

where:

cond is an optional condition code.

Rd is the destination register.

Rn is the register holding the value to be divided.

Rm is a register holding the divisor.

3.73.2 Register restrictions

PC or SP cannot be used for `Rd`, `Rn` or `Rm`.

3.73.3 Availability

For ARMv7-A, these 32-bit instructions are available in A32 and T32 if Virtualization Extensions are implemented, and optional if not.

For ARMv8 AArch32, these 32-bit instructions are always available in A32 and T32.

There are no 16-bit T32 `SDIV` and `UDIV` instructions.

3.73.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.74 SEL

Select bytes from each operand according to the state of the APSR GE flags.

3.74.1 Syntax

`SEL{cond} {Rd}, Rn, Rm`

where:

cond	is an optional condition code.
Rd	is the destination register.
Rn	is the register holding the first operand.
Rm	is the register holding the second operand.

3.74.2 Operation

The `SEL` instruction selects bytes from *Rn* or *Rm* according to the APSR GE flags:

- If GE[0] is set, `Rd[7:0]` come from `Rn[7:0]`, otherwise from `Rm[7:0]`.
- If GE[1] is set, `Rd[15:8]` come from `Rn[15:8]`, otherwise from `Rm[15:8]`.
- If GE[2] is set, `Rd[23:16]` come from `Rn[23:16]`, otherwise from `Rm[23:16]`.
- If GE[3] is set, `Rd[31:24]` come from `Rn[31:24]`, otherwise from `Rm[31:24]`.

3.74.3 Usage

Use the `SEL` instruction after one of the signed parallel instructions. You can use this to select maximum or minimum values in multiple byte or halfword data.

3.74.4 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.74.5 Condition flags

This instruction does not change the flags.

3.74.6 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit version of this instruction in T32.

3.74.7 Examples

```
SEL     r0, r4, r5
SELLT   r4, r0, r4
```

The following instruction sequence sets each byte in `R4` equal to the unsigned minimum of the corresponding bytes of `R1` and `R2`:

```
USUB8    r4, r1, r2
SEL      r4, r2, r1
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.74.8 See also

Reference

- *Parallel add and subtract* on page 3-114.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.75 SETEND

Set the endianness bit in the CPSR, without affecting any other bits in the CPSR.

SETEND cannot be conditional, and is not permitted in an IT block.

Note

SETEND is deprecated in ARMv8.

3.75.1 Syntax

SETEND *specifier*

where:

specifier is one of:

BE	Big-endian.
LE	Little-endian.

3.75.2 Usage

Use SETEND to access data of different endianness, for example, to access several big-endian DMA-formatted data fields from an otherwise little-endian application.

3.75.3 Availability

This instruction is available in A32 and T32.

In T32, this instruction is available in a 16-bit encoding only.

3.75.4 Example

```
SETEND  BE          ; Set the CPSR E bit for big-endian accesses
LDR     r0, [r2, #header]
LDR     r1, [r2, #CRC32]
SETEND  le          ; Set the CPSR E bit for little-endian accesses for the
                    ; rest of the application
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.76 SEV, SEVL, WFE, WFI, and YIELD

Set Event, Set Event Locally, Wait For Event, Wait for Interrupt, and Yield.

3.76.1 Syntax

`SEV{`*cond*`}`

`SEVL{`*cond*`}`

`WFE{`*cond*`}`

`WFI{`*cond*`}`

`YIELD{`*cond*`}`

where:

cond is an optional condition code.

3.76.2 Usage

These are hint instructions. It is optional whether they are implemented or not. If any one of them is not implemented, it executes as a `NOP`. The assembler produces a diagnostic message if the instruction executes as a `NOP` on the target.

SEV `SEV` causes an event to be signaled to all cores within a multiprocessor system.

SEVL `SEVL` causes an event to be signaled to the current processor. `SEVL` is not required to affect other processors although it is permitted to do so.

Note

This instruction is supported only in ARMv8.

WFE If the Event Register is not set, `WFE` suspends execution until one of the following events occurs:

- An IRQ interrupt, unless masked by the CPSR I-bit.
- An FIQ interrupt, unless masked by the CPSR F-bit.
- An Imprecise Data abort, unless masked by the CPSR A-bit.
- A Debug Entry request, if Debug is enabled.
- An Event signaled by another processor using the `SEV` instruction, or by the current processor using the `SEVL` instruction.

If the Event Register is set, `WFE` clears it and returns immediately.

WFI `WFI` suspends execution until one of the following events occurs:

- An IRQ interrupt, regardless of the CPSR I-bit.
- An FIQ interrupt, regardless of the CPSR F-bit.
- An Imprecise Data abort, unless masked by the CPSR A-bit.
- A Debug Entry request, regardless of whether Debug is enabled.

YIELD `YIELD` indicates to the hardware that the current thread is performing a task, for example a spinlock, that can be swapped out. Hardware can use this hint to suspend and resume threads in a multithreading system.

3.76.3 Availability

These instructions are available in A32 and T32.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

In T32, they are available in 16-bit and 32-bit encodings.

3.76.4 See also

Reference

- *NOP* on page 3-113.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.77 SMC

Secure Monitor Call.

3.77.1 Syntax

`SMC{cond} #imm4`

where:

cond is an optional condition code.

imm4 is a 4-bit immediate value. This is ignored by the ARM processor, but can be used by the SMC exception handler to determine what service is being requested.

3.77.2 Note

`SMC` was called `SMI` in earlier versions of the ARM assembly language. `SMI` instructions disassemble to `SMC`, with a comment to say that this was formerly `SMI`.

3.77.3 Availability

This 32-bit instruction is available in A32 and T32. It is available in the ARMv7-A architecture with the Security Extensions.

There is no 16-bit version of this instruction in T32.

3.77.4 See also

Reference

- *--cpu* on page 2-15.
- *Condition codes* on page 3-26.

Other information

- *ARM Architecture Reference Manual* `http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/`.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.78 SMLAD and SMLSD

Dual 16-bit Signed Multiply with Addition or Subtraction of products and 32-bit accumulation.

3.78.1 Syntax

op{X}{*cond*} *Rd*, *Rn*, *Rm*, *Ra*

where:

op is one of:

- SMLAD — Dual multiply, accumulate sum of products.
- SMLSD — Dual multiply, accumulate difference of products.

cond is an optional condition code.

X is an optional parameter. If X is present, the most and least significant halfwords of the second operand are exchanged before the multiplications occur.

Rd is the destination register.

Rn, Rm are the registers holding the operands.

Ra is the register holding the accumulate operand.

3.78.2 Operation

SMLAD multiplies the bottom halfword of *Rn* with the bottom halfword of *Rm*, and the top halfword of *Rn* with the top halfword of *Rm*. It then adds both products to the value in *Ra* and stores the sum to *Rd*.

SMLSD multiplies the bottom halfword of *Rn* with the bottom halfword of *Rm*, and the top halfword of *Rn* with the top halfword of *Rm*. It then subtracts the second product from the first, adds the difference to the value in *Ra*, and stores the result to *Rd*.

3.78.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.78.4 Condition flags

These instructions do not change the flags.

3.78.5 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.78.6 Examples

```
SMLSD       r1, r2, r0, r7
SMLSDX      r11, r10, r2, r3
SMLADLT     r1, r2, r4, r1
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.78.7 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.79 SMLAL*xy*

Signed Multiply-Accumulate with 16-bit operands and a 64-bit accumulator.

3.79.1 Syntax

`SMLAL<x><y>{cond} RdLo, RdHi, Rn, Rm`

where:

`<x>` is either `B` or `T`. `B` means use the bottom half (bits [15:0]) of `Rn`, `T` means use the top half (bits [31:16]) of `Rn`.

`<y>` is either `B` or `T`. `B` means use the bottom half (bits [15:0]) of `Rm`, `T` means use the top half (bits [31:16]) of `Rm`.

`cond` is an optional condition code.

`RdLo, RdHi` are the destination registers. They also hold the accumulate value. `RdHi` and `RdLo` must be different registers.

`Rn, Rm` are the registers holding the values to be multiplied.

3.79.2 Usage

`SMLALxy` multiplies the signed integer from the selected half of `Rm` by the signed integer from the selected half of `Rn`, and adds the 32-bit result to the 64-bit value in `RdHi` and `RdLo`.

3.79.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.79.4 Condition flags

This instruction does not change the flags.

Note

`SMLALxy` cannot raise an exception. If overflow occurs on this instruction, the result wraps round without any warning.

3.79.5 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit version of this instruction in T32.

3.79.6 Examples

```
SMLALTB     r2, r3, r7, r1
SMLALBTVS   r0, r1, r9, r2
```

3.79.7 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.80 SMLALD and SMLSLD

Dual 16-bit Signed Multiply with Addition or Subtraction of products and 64-bit Accumulation.

3.80.1 Syntax

op{X}{*cond*} *RdLo*, *RdHi*, *Rn*, *Rm*

where:

op is one of:

- SMLALD — Dual multiply, accumulate sum of products.
- SMLSLD — Dual multiply, accumulate difference of products.

X is an optional parameter. If X is present, the most and least significant halfwords of the second operand are exchanged before the multiplications occur.

cond is an optional condition code.

RdLo, *RdHi* are the destination registers for the 64-bit result. They also hold the 64-bit accumulate operand. *RdHi* and *RdLo* must be different registers.

Rn, *Rm* are the registers holding the operands.

3.80.2 Operation

SMLALD multiplies the bottom halfword of *Rn* with the bottom halfword of *Rm*, and the top halfword of *Rn* with the top halfword of *Rm*. It then adds both products to the value in *RdLo*, *RdHi* and stores the sum to *RdLo*, *RdHi*.

SMLSLD multiplies the bottom halfword of *Rn* with the bottom halfword of *Rm*, and the top halfword of *Rn* with the top halfword of *Rm*. It then subtracts the second product from the first, adds the difference to the value in *RdLo*, *RdHi*, and stores the result to *RdLo*, *RdHi*.

3.80.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.80.4 Condition flags

These instructions do not change the flags.

3.80.5 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.80.6 Examples

```
SMLALD      r10, r11, r5, r1
SMLSLD      r3, r0, r5, r1
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.80.7 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.81 SMMUL, SMMLA, and SMMLS

Signed Most significant word Multiply, Signed Most significant word Multiply with Accumulation, and Signed Most significant word Multiply with Subtraction. These instructions have 32-bit operands and produce only the most significant 32-bits of the result.

3.81.1 Syntax

`SMMUL{R}{`*cond*`} {`*Rd*`}, `*Rn*`, `*Rm*

`SMMLA{R}{`*cond*`} `*Rd*`, `*Rn*`, `*Rm*`, `*Ra*

`SMMLS{R}{`*cond*`} `*Rd*`, `*Rn*`, `*Rm*`, `*Ra*

where:

`R` is an optional parameter. If `R` is present, the result is rounded, otherwise it is truncated.

cond is an optional condition code.

Rd is the destination register.

Rn, Rm are the registers holding the operands.

Ra is a register holding the value to be added or subtracted from.

3.81.2 Operation

`SMMUL` multiplies the values from *Rn* and *Rm*, and stores the most significant 32 bits of the 64-bit result to *Rd*.

`SMMLA` multiplies the values from *Rn* and *Rm*, adds the value in *Ra* to the most significant 32 bits of the product, and stores the result in *Rd*.

`SMMLS` multiplies the values from *Rn* and *Rm*, subtracts the product from the value in *Ra* shifted left by 32 bits, and stores the most significant 32 bits of the result in *Rd*.

If the optional `R` parameter is specified, `0x80000000` is added before extracting the most significant 32 bits. This has the effect of rounding the result.

3.81.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.81.4 Condition flags

These instructions do not change the flags.

3.81.5 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.81.6 Examples

```
SMMULGE     r6, r4, r3
SMMULR      r2, r2, r2
```

3.81.7 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.82 SMUAD{X} and SMUSD{X}

Dual 16-bit Signed Multiply with Addition or Subtraction of products, and optional exchange of operand halves.

3.82.1 Syntax

op{X}{*cond*} {*Rd*}, *Rn*, *Rm*

where:

op is one of:

- SMUAD Dual multiply, add products.
- SMUSD Dual multiply, subtract products.

X is an optional parameter. If X is present, the most and least significant halfwords of the second operand are exchanged before the multiplications occur.

cond is an optional condition code.

Rd is the destination register.

Rn, *Rm* are the registers holding the operands.

3.82.2 Usage

SMUAD multiplies the bottom halfword of *Rn* with the bottom halfword of *Rm*, and the top halfword of *Rn* with the top halfword of *Rm*. It then adds the products and stores the sum to *Rd*.

SMUSD multiplies the bottom halfword of *Rn* with the bottom halfword of *Rm*, and the top halfword of *Rn* with the top halfword of *Rm*. It then subtracts the second product from the first, and stores the difference to *Rd*.

3.82.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.82.4 Q flag

The SMUAD instruction sets the Q flag if the addition overflows.

3.82.5 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.82.6 Examples

```
SMUAD       r2, r3, r2
SMUSDXNE    r0, r1, r2
```

3.82.7 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.83 SMULW*y* and SMLAW*y*

Signed Multiply Wide and Signed Multiply-Accumulate Wide, with one 32-bit and one 16-bit operand, providing the top 32-bits of the result.

3.83.1 Syntax

`SMULW<y>{cond} {Rd}, Rn, Rm`

`SMLAW<y>{cond} Rd, Rn, Rm, Ra`

where:

`<y>` is either `B` or `T`. `B` means use the bottom half (bits [15:0]) of `Rm`, `T` means use the top half (bits [31:16]) of `Rm`.

`cond` is an optional condition code.

`Rd` is the destination register.

`Rn, Rm` are the registers holding the values to be multiplied.

`Ra` is the register holding the value to be added.

3.83.2 Usage

`SMULWy` multiplies the signed integer from the selected half of `Rm` by the signed integer from `Rn`, and places the upper 32-bits of the 48-bit result in `Rd`.

`SMLAWy` multiplies the signed integer from the selected half of `Rm` by the signed integer from `Rn`, adds the 32-bit result to the 32-bit value in `Ra`, and places the result in `Rd`.

3.83.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.83.4 Condition flags

These instructions do not affect the N, Z, C, or V flags.

If overflow occurs in the accumulation, `SMLAWy` sets the Q flag.

3.83.5 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.83.6 See also

Reference

- *MRS (PSR to general-purpose register)* on page 3-105.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.84 SMUL*xy* and SMLA*xy*

Signed Multiply and Multiply Accumulate, with 16-bit operands and a 32-bit result and accumulator.

3.84.1 Syntax

`SMUL<x><y>{cond} {Rd}, Rn, Rm`

`SMLA<x><y>{cond} Rd, Rn, Rm, Ra`

where:

`<x>` is either `B` or `T`. `B` means use the bottom half (bits [15:0]) of `Rn`, `T` means use the top half (bits [31:16]) of `Rn`.

`<y>` is either `B` or `T`. `B` means use the bottom half (bits [15:0]) of `Rm`, `T` means use the top half (bits [31:16]) of `Rm`.

`cond` is an optional condition code.

`Rd` is the destination register.

`Rn, Rm` are the registers holding the values to be multiplied.

`Ra` is the register holding the value to be added.

3.84.2 Usage

`SMULxy` multiplies the 16-bit signed integers from the selected halves of `Rn` and `Rm`, and places the 32-bit result in `Rd`.

`SMLAxy` multiplies the 16-bit signed integers from the selected halves of `Rn` and `Rm`, adds the 32-bit result to the 32-bit value in `Ra`, and places the result in `Rd`.

3.84.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.84.4 Condition flags

These instructions do not affect the N, Z, C, or V flags.

If overflow occurs in the accumulation, `SMLAxy` sets the Q flag. To read the state of the Q flag, use an `MRS` instruction.

Note

`SMLAxy` never clears the Q flag. To clear the Q flag, use an `MSR` instruction.

3.84.5 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.84.6 Examples

```
SMULTBEQ    r8, r7, r9
SMLABBNE    r0, r2, r1, r10
SMLABT      r0, r0, r3, r5
```

3.84.7 See also

Reference

- *MRS (PSR to general-purpose register)* on page 3-105.
- *MSR (general-purpose register to PSR)* on page 3-108.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.85 SRS

Store Return State onto a stack.

3.85.1 Syntax

```
SRS{addr_mode}{cond} sp{!}, #modenum

SRS{addr_mode}{cond} #modenum{!}        ; This is pre-UAL syntax
```

where:

addr_mode is any one of the following:

- `IA` Increment address After each transfer
- `IB` Increment address Before each transfer (A32 only)
- `DA` Decrement address After each transfer (A32 only)
- `DB` Decrement address Before each transfer (Full Descending stack).

If *addr_mode* is omitted, it defaults to Increment After. You can also use stack oriented addressing mode suffixes, for example, when implementing stacks.

cond is an optional condition code.

Note

cond is permitted only in T32 code, using a preceding `IT` instruction, but this is deprecated. This is an unconditional instruction in A32.

`!` is an optional suffix. If `!` is present, the final address is written back into the SP of the mode specified by *modenum*.

modenum specifies the number of the mode whose banked SP is used as the base register. You must use only the defined mode numbers.

3.85.2 Operation

`SRS` stores the LR and the SPSR of the current mode, at the address contained in SP of the mode specified by *modenum*, and the following word respectively. Optionally updates SP of the mode specified by *modenum*. This is compatible with the normal use of the `STM` instruction for stack accesses.

Note

For full descending stack, you must use `SRSFD` or `SRSDB`.

3.85.3 Usage

You can use `SRS` to store return state for an exception handler on a different stack from the one automatically selected.

3.85.4 Notes

Where addresses are not word-aligned, `SRS` ignores the least significant two bits of the specified address.

The time order of the accesses to individual words of memory generated by `SRS` is not architecturally defined. Do not use this instruction on memory-mapped I/O locations where access order matters.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Do not use SRS in User and System modes because these modes do not have a SPSR.

SRS is not permitted in a non-secure state if *modenum* specifies monitor mode.

3.85.5 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit version of this instruction.

3.85.6 Example

```
R13_usr  EQU     16
         SRSFD   sp,#R13_usr
```

3.85.7 See also

Concepts

armasm User Guide:

- *Stack implementation using LDM and STM* on page 7-23.
- *Processor modes, and privileged and unprivileged software execution* on page 4-3.

Reference

- *LDM and STM* on page 3-71.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.86 SSAT and USAT

Signed Saturate and Unsigned Saturate to any bit position, with optional shift before saturating.

SSAT saturates a signed value to a signed range.

USAT saturates a signed value to an unsigned range.

3.86.1 Syntax

op{*cond*} *Rd*, #*sat*, *Rm*{, *shift*}

where:

op is either SSAT or USAT.

cond is an optional condition code.

Rd is the destination register.

sat specifies the bit position to saturate to, in the range 1 to 32 for SSAT, and 0 to 31 for USAT.

Rm is the register containing the operand.

shift is an optional shift. It must be one of the following:

- ASR #*n* where *n* is in the range 1-32 (A32) or 1-31 (T32)
- LSL #*n* where *n* is in the range 0-31.

3.86.2 Operation

The SSAT instruction applies the specified shift, then saturates to the signed range $-2^{sat-1} \le x \le 2^{sat-1} - 1$.

The USAT instruction applies the specified shift, then saturates to the unsigned range $0 \le x \le 2^{sat} - 1$.

3.86.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.86.4 Q flag

If saturation occurs, these instructions set the Q flag. To read the state of the Q flag, use an MRS instruction.

3.86.5 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.86.6 Examples

```
SSAT    r7, #16, r7, LSL #4
USATNE  r0, #7, r5
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.86.7 See also

Reference

- *SSAT16 and USAT16* on page 3-152.
- *MRS (PSR to general-purpose register)* on page 3-105.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.87 SSAT16 and USAT16

Parallel halfword Saturating instructions.

SSAT16 saturates a signed value to a signed range.

USAT16 saturates a signed value to an unsigned range.

3.87.1 Syntax

op{*cond*} *Rd*, *#sat*, *Rn*

where:

op is one of:

- SSAT16 Signed saturation.
- USAT16 Unsigned saturation.

cond is an optional condition code.

Rd is the destination register.

sat specifies the bit position to saturate to, and is in the range 1 to 16 for SSAT16, or 0 to 15 for USAT16.

Rn is the register holding the operand.

3.87.2 Operation

Halfword-wise signed and unsigned saturation to any bit position.

The SSAT16 instruction saturates each signed halfword to the signed range $-2^{sat-1} \leq x \leq 2^{sat-1} - 1$.

The USAT16 instruction saturates each signed halfword to the unsigned range $0 \leq x \leq 2^{sat} - 1$.

3.87.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.87.4 Q flag

If saturation occurs on either halfword, these instructions set the Q flag. To read the state of the Q flag, use an MRS instruction.

3.87.5 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.87.6 Examples

```
SSAT16  r7, #12, r7
USAT16  r0, #7, r5
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.87.7 Incorrect examples

```
SSAT16  r1, #16, r2, LSL #4 ; shifts not permitted with halfword saturations
```

3.87.8 See also

Reference

- *MRS (PSR to general-purpose register)* on page 3-105.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.88 SUBS pc, lr

Exception return, without popping anything from the stack.

3.88.1 Syntax

```
SUBS{cond} pc, lr, #imm              ; A32 and T32 code
MOVS{cond} pc, lr                    ; A32 and T32 code
op1S{cond} pc, Rn, #imm              ; A32 code only and is deprecated
op1S{cond} pc, Rn, Rm {, shift}      ; A32 code only and is deprecated
op2S{cond} pc, #imm                  ; A32 code only and is deprecated
op2S{cond} pc, Rm {, shift}          ; A32 code only and is deprecated
```

where:

op1 is one of ADC, ADD, AND, BIC, EOR, ORN, ORR, RSB, RSC, SBC, and SUB.

op2 is one of MOV and MVN.

cond is an optional condition code.

imm is an immediate value. In T32 code, it is limited to the range 0-255. In A32 code, it is a flexible second operand.

R*n* is the first operand register. ARM deprecates the use of any register except LR.

R*m* is the optionally shifted second or only operand register.

shift is an optional condition code.

3.88.2 Usage

`SUBS pc, lr, #imm` subtracts a value from the link register and loads the PC with the result, then copies the SPSR to the CPSR.

You can use `SUBS pc, lr, #imm` to return from an exception if there is no return state on the stack. The value of `#imm` depends on the exception to return from.

3.88.3 Notes

`SUBS pc, lr, #imm` writes an address to the PC. The alignment of this address must be correct for the instruction set in use after the exception return:

- For a return to A32, the address written to the PC must be word-aligned.
- For a return to T32, the address written to the PC must be halfword-aligned.

No special precautions are required in software to follow these rules, if you use the instruction to return after a valid exception entry mechanism.

In T32, only `SUBS{cond} pc, lr, #imm` is a valid instruction. `MOVS pc, lr` is a synonym of `SUBS pc, lr, #0`. Other instructions are undefined.

In A32, only `SUBS{cond} pc, lr, #imm` and `MOVS{cond} pc, lr` are valid instructions. Other instructions are deprecated.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Caution

Do not use these instructions in User mode or System mode. The assembler cannot warn you about this.

3.88.4 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit version of this instruction in T32.

3.88.5 See also

Concepts

- *Flexible second operand (Operand2)* on page 3-12.

Reference

- *ADD, SUB, RSB, ADC, SBC, and RSC* on page 3-27.
- *AND, ORR, EOR, BIC, and ORN* on page 3-38.
- *MOV and MVN* on page 3-98.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.89 SVC

Supervisor Call.

3.89.1 Syntax

`SVC{cond} #imm`

where:

cond is an optional condition code.

imm is an expression evaluating to an integer in the range:

- 0 to 2^{24}–1 (a 24-bit value) in an A32 instruction
- 0-255 (an 8-bit value) in a T32 instruction.

3.89.2 Usage

The `SVC` instruction causes an exception. This means that the processor mode changes to Supervisor, the CPSR is saved to the Supervisor mode SPSR, and execution branches to the SVC vector.

imm is ignored by the processor. However, it can be retrieved by the exception handler to determine what service is being requested.

Note

`SVC` was called `SWI` in earlier versions of the ARM assembly language. `SWI` instructions disassemble to `SVC`, with a comment to say that this was formerly `SWI`.

3.89.3 Condition flags

This instruction does not change the flags.

3.89.4 Availability

This instruction is available in A32 and T32 and in the ARMv7 architectures.

In T32, it is only available as a 16-bit instruction.

3.89.5 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.90 SXT, SXTA, UXT, and UXTA

Sign extend, Sign extend with Add, Zero extend, and Zero extend with Add.

3.90.1 Syntax

```
SXT<extend>{cond}  {Rd}, Rm  {,rotation}
SXTA<extend>{cond} {Rd}, Rn, Rm {,rotation}
UXT<extend>{cond}  {Rd}, Rm  {,rotation}
UXTA<extend>{cond} {Rd}, Rn, Rm {,rotation}
```

where:

<extend> is one of:

- `B16` Extends two 8-bit values to two 16-bit values.
- `B` Extends an 8-bit value to a 32-bit value.
- `H` Extends a 16-bit value to a 32-bit value.

cond is an optional condition code.

Rd is the destination register.

Rn is the register holding the number to add (`SXTA` and `UXTA` only).

Rm is the register holding the value to extend.

rotation is one of:

- `ROR #8` Value from *Rm* is rotated right 8 bits.
- `ROR #16` Value from *Rm* is rotated right 16 bits.
- `ROR #24` Value from *Rm* is rotated right 24 bits.

If *rotation* is omitted, no rotation is performed.

3.90.2 Operation

These instructions do the following:

1. Rotate the value from *Rm* right by 0, 8, 16 or 24 bits.
2. Do one of the following to the value obtained:
   - Extract bits[7:0], sign or zero extend to 32 bits. If the instruction is extend and add, add the value from *Rn*.
   - Extract bits[15:0], sign or zero extend to 32 bits. If the instruction is extend and add, add the value from *Rn*.
   - Extract bits[23:16] and bits[7:0] and sign or zero extend them to 16 bits. If the instruction is extend and add, add them to bits[31:16] and bits[15:0] respectively of *Rn* to form bits[31:16] and bits[15:0] of the result.

3.90.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.90.4 Condition flags

These instructions do not change the flags.

3.90.5 16-bit instructions

The following forms of these instructions are available in T32 code, and are 16-bit instructions:

`SXTB Rd, Rm` *Rd* and *Rm* must both be Lo registers.

`SXTH Rd, Rm` *Rd* and *Rm* must both be Lo registers.

`UXTB Rd, Rm` *Rd* and *Rm* must both be Lo registers.

`UXTH Rd, Rm` *Rd* and *Rm* must both be Lo registers.

3.90.6 Availability

These instructions are available in A32 and T32.

In T32, they are available in 16-bit and 32-bit encodings.

3.90.7 Examples

```
SXTH         r3, r9, r4
UXTAB16EQ    r0, r0, r4, ROR #16
```

3.90.8 Incorrect examples

```
SXTH     r9, r3, r2, ROR #12 ; rotation must be by 0, 8, 16, or 24.
```

3.90.9 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.91 SYS

Execute system coprocessor instruction.

3.91.1 Syntax

`SYS{cond} instruction{, Rn}`

where:

cond is an optional condition code.

instruction
is the coprocessor instruction to execute.

Rn is an operand to the instruction. For instructions that take an argument, *Rn* is compulsory. For instructions that do not take an argument, *Rn* is optional and if it is not specified, `R0` is used. `Rn` must not be PC.

3.91.2 Usage

You can use this instruction to execute special coprocessor instructions such as cache, branch predictor, and TLB operations. The instructions operate by writing to special write-only coprocessor registers. The instruction names are the same as the write-only coprocessor register names and are listed in the *ARM Architecture Reference Manual*. For example:

```
SYS ICIALLUIS ; invalidates all instruction caches Inner Shareable to Point
              ; of Unification and also flushes branch target cache.
```

3.91.3 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit version of this instruction in T32.

3.91.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.92 TBB and TBH

Table Branch Byte and Table Branch Halfword.

3.92.1 Syntax

```
TBB [Rn, Rm]
TBH [Rn, Rm, LSL #1]
```

where:

Rn is the base register. This contains the address of the table of branch lengths. *Rn* must not be SP.

If PC is specified for *Rn*, the value used is the address of the instruction plus 4.

Rm is the index register. This contains an index into the table.

Rm must not be PC or SP.

3.92.2 Operation

These instructions cause a PC-relative forward branch using a table of single byte offsets (TBB) or halfword offsets (TBH). *Rn* provides a pointer to the table, and *Rm* supplies an index into the table. The branch length is twice the value of the byte (TBB) or the halfword (TBH) returned from the table. The target of the branch table must be in the same execution state.

3.92.3 Availability

These 32-bit instructions are available in T32 only.

There are no 16-bit versions of these instructions in T32.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.93 TST and TEQ

Test bits and Test Equivalence.

3.93.1 Syntax

`TST{cond} Rn, Operand2`

`TEQ{cond} Rn, Operand2`

where:

cond is an optional condition code.

Rn is the ARM register holding the first operand.

Operand2 is a flexible second operand.

3.93.2 Usage

These instructions test the value in a register against *Operand2*. They update the condition flags on the result, but do not place the result in any register.

The `TST` instruction performs a bitwise AND operation on the value in *Rn* and the value of *Operand2*. This is the same as an `ANDS` instruction, except that the result is discarded.

The `TEQ` instruction performs a bitwise Exclusive OR operation on the value in *Rn* and the value of *Operand2*. This is the same as a `EORS` instruction, except that the result is discarded.

Use the `TEQ` instruction to test if two values are equal, without affecting the V or C flags (as `CMP` does).

`TEQ` is also useful for testing the sign of a value. After the comparison, the N flag is the logical Exclusive OR of the sign bits of the two operands.

3.93.3 Register restrictions

In these T32 instructions, you cannot use SP or PC for *Rn* or *Operand2*.

In these A32 instructions, use of SP or PC is deprecated.

For A32 instructions:

- If you use PC (`R15`) as *Rn*, the value used is the address of the instruction plus 8.
- You cannot use PC for any operand in any data processing instruction that has a register-controlled shift.

3.93.4 Condition flags

These instructions:

- Update the N and Z flags according to the result.
- Can update the C flag during the calculation of *Operand2*.
- Do not affect the V flag.

3.93.5 Availability

These 32-bit instructions are available in A32 and T32.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

In T32, only the following form of the TST instruction is available as a 16-bit instruction:

`TST Rn, Rm` *Rn* and *Rm* must both be Lo registers.

3.93.6 Examples

```
TST     r0, #0x3F8
TEQEQ   r10, r9
TSTNE   r1, r5, ASR r1
```

3.93.7 Incorrect example

```
TEQ     pc, r1, ROR r0      ; PC not permitted with register
                            ; controlled shift
```

3.93.8 See also

Concepts

- *Flexible second operand (Operand2)* on page 3-12.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.94 UMAAL

Unsigned Multiply Accumulate Accumulate Long.

3.94.1 Syntax

`UMAAL{cond} RdLo, RdHi, Rn, Rm`

where:

cond is an optional condition code.

RdLo, RdHi are the destination registers for the 64-bit result. They also hold the two 32-bit accumulate operands. *RdLo* and *RdHi* must be different registers.

Rn, Rm are the registers holding the multiply operands.

3.94.2 Operation

The `UMAAL` instruction multiplies the 32-bit values in *Rn* and *Rm*, adds the two 32-bit values in *RdHi* and *RdLo*, and stores the 64-bit result to *RdLo, RdHi*.

3.94.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.94.4 Condition flags

This instruction does not change the flags.

3.94.5 Availability

This 32-bit instruction is available in A32 and T32.

There is no 16-bit version of this instruction in T32.

3.94.6 Examples

```
UMAAL       r8, r9, r2, r3
UMAALGE     r2, r0, r5, r3
```

3.94.7 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.95 UMULL, UMLAL, SMULL, and SMLAL

Signed and Unsigned Long Multiply, with optional Accumulate, with 32-bit operands, and 64-bit result and accumulator.

3.95.1 Syntax

Op{S}{*cond*} *RdLo*, *RdHi*, *Rn*, *Rm*

where:

Op is one of UMULL, UMLAL, SMULL, or SMLAL.

S is an optional suffix available in A32 state only. If S is specified, the condition flags are updated on the result of the operation.

cond is an optional condition code.

RdLo, *RdHi* are the destination registers. For UMLAL and SMLAL they also hold the accumulating value. *RdLo* and *RdHi* must be different registers

Rn, *Rm* are ARM registers holding the operands.

3.95.2 Usage

The UMULL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The UMLAL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers, and adds the 64-bit result to the 64-bit unsigned integer contained in *RdHi* and *RdLo*.

The SMULL instruction interprets the values from *Rn* and *Rm* as two's complement signed integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The SMLAL instruction interprets the values from *Rn* and *Rm* as two's complement signed integers. It multiplies these integers, and adds the 64-bit result to the 64-bit signed integer contained in *RdHi* and *RdLo*.

3.95.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.95.4 Condition flags

If S is specified, these instructions:

- Update the N and Z flags according to the result.
- Do not affect the C or V flags.

3.95.5 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.95.6 Examples

```
UMULL       r0, r4, r5, r6
UMLALS      r4, r5, r3, r8
```

3.95.7 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.96 UND pseudo-instruction

Generate an architecturally undefined instruction. An attempt to execute an undefined instruction causes the Undefined Instruction exception. Architecturally undefined instructions are expected to remain undefined.

3.96.1 Syntax

`UND{cond}{.W} {#expr}`

where:

cond is an optional condition code.

`.W` is an optional instruction width specifier.

expr evaluates to a numeric value. Table 3-12 shows the range and encoding of *expr* in the instruction, where Y shows the locations of the bits that encode for *expr* and V is the 4 bits that encode for the condition code.

If *expr* is omitted, the value 0 is used.

Table 3-12 Range and encoding of *expr*

Instruction	Encoding	Number of bits for *expr*	Range
A32	0xV7FYYYFY	16	0-65535
T32, 32-bit encoding	0xF7FYAYFY	12	0-4095
T32, 16-bit encoding	0xDEYY	8	0-255

3.96.2 UND in T32 code

You can use the `.W` width specifier to force `UND` to generate a 32-bit instruction in T32 code. `UND.W` always generates a 32-bit instruction, even if *expr* is in the range 0-255.

3.96.3 Disassembly

The encodings that this pseudo-instruction produces disassemble to `DCI`.

3.96.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.97 USAD8 and USADA8

Unsigned Sum of Absolute Differences, and Accumulate unsigned sum of absolute differences.

3.97.1 Syntax

`USAD8{cond} {Rd}, Rn, Rm`

`USADA8{cond} Rd, Rn, Rm, Ra`

where:

cond is an optional condition code.

Rd is the destination register.

Rn is the register holding the first operand.

Rm is the register holding the second operand.

Ra is the register holding the accumulate operand.

3.97.2 Operation

The `USAD8` instruction finds the four differences between the unsigned values in corresponding bytes of *Rn* and *Rm*. It adds the absolute values of the four differences, and saves the result to *Rd*.

The `USADA8` instruction adds the absolute values of the four differences to the value in *Ra*, and saves the result to *Rd*.

3.97.3 Register restrictions

You cannot use PC for any register.

Using SP in A32 instructions is deprecated. You cannot use SP in T32 instructions.

3.97.4 Condition flags

These instructions do not alter any flags.

3.97.5 Availability

These 32-bit instructions are available in A32 and T32.

There are no 16-bit versions of these instructions in T32.

3.97.6 Examples

```
USAD8       r2, r4, r6
USADA8      r0, r3, r5, r2
USADA8VS    r0, r4, r0, r1
```

3.97.7 Incorrect examples

```
USADA8      r2, r4, r6      ; USADA8 requires four registers
USADA16     r0, r4, r0, r1  ; no such instruction
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

3.97.8 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Chapter 4
Advanced SIMD and Floating-point Programming (32-bit)

The following topics describe Advanced SIMD and floating-point assembly language programming:

- *Advanced SIMD and floating-point instruction summary* on page 4-2.
- *Shared Advanced SIMD and floating-point instructions* on page 4-7.
- *Advanced SIMD logical and compare operations* on page 4-8.
- *Advanced SIMD general data processing instructions* on page 4-9.
- *Advanced SIMD shift instructions* on page 4-10.
- *Advanced SIMD general arithmetic instructions* on page 4-11.
- *Advanced SIMD multiply instructions* on page 4-12.
- *Advanced SIMD load and store element and structure instructions* on page 4-13.
- *Interleaving provided by load and store, element and structure instructions* on page 4-14.
- *Alignment restrictions in load and store, element and structure instructions* on page 4-15.
- *Advanced SIMD and floating-point pseudo-instructions* on page 4-16.
- *Floating-point instructions* on page 4-17.
- *Cryptographic instructions* on page 4-18.

Note

Detailed information about the ARMv8 architecture is available under license. Contact your ARM Account Representative for details.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.1 Advanced SIMD and floating-point instruction summary

The following topics provide a summary of the Advanced SIMD and floating-point instructions:

- *Summary of advanced SIMD instructions*.
- *Summary of shared Advanced SIMD and floating-point instructions* on page 4-4.
- *Summary of floating-point instructions* on page 4-5.

4.1.1 Summary of advanced SIMD instructions

Table 4-1 shows a summary of Advanced SIMD instructions. These are not available as floating-point instructions.

Table 4-1 Summary of Advanced SIMD instructions

Mnemonic	Brief description	See
VABA, VABD	Absolute difference and Accumulate, Absolute Difference	page 4-28
VABS	Absolute value	page 4-20
VACGE, VACGT	Absolute Compare Greater than or Equal, Greater Than	page 4-30
VACLE, VACLT	Absolute Compare Less than or Equal, Less Than (pseudo-instructions)	page 4-31
VADD	Add	page 4-21
VADDHN	Add, select High half	page 4-23
VAND	Bitwise AND	page 4-33
VAND	Bitwise AND (pseudo-instruction)	page 4-34
VBIC	Bitwise Bit Clear (register)	page 4-33
VBIC	Bitwise Bit Clear (immediate)	page 4-35
VBIF, VBIT, VBSL	Bitwise Insert if False, Insert if True, Select	page 4-36
VCEQ, VCLE, VCLT	Compare Equal, Less than or Equal, Compare Less Than	page 4-37
VCGE, VCGT	Compare Greater than or Equal, Greater Than	page 4-37
VCLE, VCLT	Compare Less than or Equal, Compare Less Than (pseudo-instruction)	page 4-38
VCLS, VCLZ, VCNT	Count Leading Sign bits, Count Leading Zeros, and Count set bits	page 4-39
VCVT	Convert fixed-point or integer to floating-point, floating-point to integer or fixed-point	page 4-41
VCVT	Convert floating-point to integer with directed rounding modes	page 4-42
VCVT	Convert between half-precision and single-precision floating-point numbers	page 4-43
VDUP	Duplicate scalar to all lanes of vector	page 4-50
VEOR	Bitwise Exclusive OR	page 4-33
VEXT	Extract	page 4-51
VFMA, VFMS	Fused Multiply Accumulate, Fused Multiply Subtract (vector)	page 4-52
VHADD, VHSUB	Halving Add, Halving Subtract	page 4-24
VLD	Vector Load	page 4-13
VMAX, VMIN	Maximum, Minimum	page 4-64

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 4-1 Summary of Advanced SIMD instructions (continued)

Mnemonic	Brief description	See
VMAXNM, VMINNM	Maximum, Minimum, consistent with IEEE 754-2008	page 4-65
VMLA, VMLS	Multiply Accumulate, Multiply Subtract (vector)	page 4-77
VMLA, VMLS	Multiply Accumulate, Multiply Subtract (by scalar)	page 4-78
VMOV	Move (immediate)	page 4-68
VMOV	Move (register)	page 4-69
VMOVL, VMOV{U}N	Move Long, Move Narrow (register)	page 4-74
VMUL	Multiply (vector)	page 4-77
VMUL	Multiply (by scalar)	page 4-78
VMVN	Move Negative (immediate)	page 4-68
VNEG	Negate	page 4-20
VORN	Bitwise OR NOT	page 4-33
VORN	Bitwise OR NOT (pseudo-instruction)	page 4-34
VORR	Bitwise OR (register)	page 4-33
VORR	Bitwise OR (immediate)	page 4-35
VPADD, VPADAL	Pairwise Add, Pairwise Add and Accumulate	page 4-79
VPMAX, VPMIN	Pairwise Maximum, Pairwise Minimum	page 4-64
VQABS	Absolute value, saturate	page 4-20
VQADD	Add, saturate	page 4-21
VQDMLAL, VQDMLSL	Saturating Doubling Multiply Accumulate, and Multiply Subtract	page 4-83
VQDMULL	Saturating Doubling Multiply	page 4-83
VQDMULH	Saturating Doubling Multiply returning High half	page 4-81
VQMOV{U}N	Saturating Move (register)	page 4-74
VQNEG	Negate, saturate	page 4-20
VQRDMULH	Saturating Doubling Multiply returning High half	page 4-81
VQRSHL	Shift Left, Round, saturate (by signed variable)	page 4-19
VQRSHR{U}N	Shift Right, Round, saturate (by immediate)	page 4-82
VQSHL	Shift Left, saturate (by immediate)	page 4-91
VQSHL	Shift Left, saturate (by signed variable)	page 4-19
VQSHR{U}N	Shift Right, saturate (by immediate)	page 4-82
VQSUB	Subtract, saturate	page 4-21
VRADDHN	Add, select High half, Round	page 4-23
VRECPE	Reciprocal Estimate	page 4-84
VRECPS	Reciprocal Step	page 4-85

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 4-1 Summary of Advanced SIMD instructions (continued)

Mnemonic	Brief description	See
VREV	Reverse elements	page 4-87
VRHADD	Halving Add, Round	page 4-24
VRINT	Round to integer	page 4-88
VRSHR	Shift Right and Round (by immediate)	page 4-25
VRSHRN	Shift Right, Round, Narrow (by immediate)	page 4-26
VRSQRTE	Reciprocal Square Root Estimate	page 4-84
VRSQRTS	Reciprocal Square Root Step	page 4-85
VRSRA	Shift Right, Round, and Accumulate (by immediate)	page 4-27
VRSUBHN	Subtract, select High half, Round	page 4-23
VSHL	Shift Left (by immediate)	page 4-91
VSHR	Shift Right (by immediate)	page 4-25
VSHRN	Shift Right, Narrow (by immediate)	page 4-26
VSLI	Shift Left and Insert	page 4-93
VSRA	Shift Right, Accumulate (by immediate)	page 4-27
VSRI	Shift Right and Insert	page 4-93
VST	Vector Store	page 4-13
VSUB	Subtract	page 4-21
VSUBHN	Subtract, select High half	page 4-23
VSWP	Swap vectors	page 4-95
VTBL, VTBX	Vector table look-up	page 4-96
VTRN	Vector transpose	page 4-97
VTST	Test bits	page 4-98
VUZP, VZIP	Vector interleave and de-interleave	page 4-99

4.1.2 Summary of shared Advanced SIMD and floating-point instructions

Table 4-2 shows a summary of instructions that are common to the Advanced SIMD and floating-point instruction sets.

Table 4-2 Summary of shared Advanced SIMD and floating-point instructions

Mnemonic	Brief description	See
VLDM	Load multiple	page 4-54
VLDR	Load (see also *VLDR pseudo-instruction* on page 4-62)	page 4-55
	Load (post-increment and pre-decrement)	page 4-63
VMOV	Transfer from one ARM register to a scalar	page 4-71

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 4-2 Summary of shared Advanced SIMD and floating-point instructions (continued)

Mnemonic	Brief description	See
	Transfer from two ARM registers to either one double-precision or two single-precision registers	page 4-70
	Transfer from a scalar to an ARM register	page 4-71
	Transfer from either one double-precision or two single-precision registers to two ARM registers	page 4-70
	Transfer from single-precision to ARM register	page 4-72
	Transfer from ARM register to single-precision	page 4-72
VMRS	Transfer from SIMD and floating-point system register to ARM register	page 4-75
VMSR	Transfer from ARM register to SIMD and floating-point system register	page 4-75
VPOP	Pop floating-point or SIMD registers from full-descending stack	page 4-54
VPUSH	Push floating-point or SIMD registers to full-descending stack	page 4-54
VSTM	Store multiple	page 4-54
VSTR	Store	page 4-55
	Store (post-increment and pre-decrement)	page 4-63

4.1.3 Summary of floating-point instructions

Table 4-3 shows a summary of floating-point instructions that are not available in Advanced SIMD.

Note

Floating-point vector mode is not supported in ARMv8. Use Advanced SIMD instructions for vector floating-point.

Table 4-3 Summary of floating-point instructions

Mnemonic	Brief description	See
VABS	Absolute value	page 4-29
VADD	Add	page 4-32
VCMP, VCMPE	Compare	page 4-40
VCVT	Convert between single-precision and double-precision	page 4-44
	Convert between floating-point and integer	page 4-45
	Convert between floating-point and fixed-point	page 4-47
	Convert floating-point to integer with directed rounding modes	page 4-46
VCVTB, VCVTT	Convert between half-precision and single-precision floating-point	page 4-48

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 4-3 Summary of floating-point instructions (continued)

Mnemonic	Brief description	See
	Convert between half-precision and double-precision	page 4-49
VDIV	Divide	page 4-32
VFMA, VFMS	Fused multiply accumulate, Fused multiply subtract	page 4-52
VFNMA, VFNMS	Fused multiply accumulate with negation, Fused multiply subtract with negation	page 4-53
VMAXNM, VMINNM	Maximum, Minimum, consistent with IEEE 754-2008	page 4-66
VMLA	Multiply accumulate	page 4-76
VMLS	Multiply subtract	page 4-76
VMOV	Insert floating-point immediate in single-precision or double-precision register, or copy one FP register into another FP register of the same width (see also Table 4-2 on page 4-4)	page 4-67
VMUL	Multiply	page 4-76
VNEG	Negate	page 4-29
VNMLA	Negated multiply accumulate	page 4-76
VNMLS	Negated multiply subtract	page 4-76
VNMUL	Negated multiply	page 4-76
VRINT	Round to integer	page 4-89
VSEL	Select	page 4-90
VSQRT	Square Root	page 4-29
VSUB	Subtract	page 4-32

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.2 Shared Advanced SIMD and floating-point instructions

The following topics describe the instructions that are shared by the Advanced SIMD and floating-point instruction sets:

- *VLDR and VSTR* on page 4-55

  Extension register load and store.

- *VLDM, VSTM, VPOP, and VPUSH* on page 4-54

  Extension register load and store multiple.

- *VMOV (between two ARM registers and one or two extension registers)* on page 4-70

  Transfer contents between two ARM registers and a 64-bit extension register.

- *VMOV (between an ARM register and a scalar)* on page 4-71

  Transfer contents between an ARM register and a scalar.

- *VMOV (between one ARM register and single precision floating-point register)* on page 4-72

  Transfer contents between a 32-bit extension register and an ARM register.

- *VMRS and VMSR* on page 4-75

  Transfer contents between an ARM register and a SIMD and floating-point system register.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.3 Advanced SIMD logical and compare operations

The following topics describe the Advanced SIMD logical and compare operations:

- *VAND, VBIC, VEOR, VORN, and VORR (register)* on page 4-33
  Bitwise AND, Bit Clear, Exclusive OR, OR Not, and OR (register).
- *VBIC and VORR (immediate)* on page 4-35
  Bitwise Bit Clear and OR (immediate).
- *VBIF, VBIT, and VBSL* on page 4-36
  Bitwise Insert if False, Insert if True, and Select.
- *VMOV, VMVN (register)* on page 4-69
  Move, and Move NOT.
- *VACGE and VACGT* on page 4-30
  Compare Absolute.
- *VCEQ, VCGE, VCGT, VCLE, and VCLT* on page 4-37
  Compare.
- *VTST* on page 4-98
  Test bits.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.4 Advanced SIMD general data processing instructions

The following topics describe the Advanced SIMD general data processing instructions:

- *VCVT (between fixed-point or integer, and floating-point)* on page 4-41
  Vector convert between fixed-point or integer and floating-point.
- *VCVT (from floating-point to integer with directed rounding modes)* on page 4-42
  Vector convert from floating-point to integer with directed rounding modes.
- *VCVT (between half-precision and single-precision floating-point)* on page 4-43
  Vector convert between half-precision and single-precision floating-point.
- *VRINT (Advanced SIMD)* on page 4-88
  Vector round to integer.
- *VDUP* on page 4-50
  Duplicate scalar to all lanes of vector.
- *VEXT* on page 4-51
  Extract.
- *VMOV, VMVN (immediate)* on page 4-68
  Move and Move Negative (immediate).
- *VMOVL, V{Q}MOVN, VQMOVUN* on page 4-74
  Move (register).
- *VREV* on page 4-87
  Reverse elements within a vector.
- *VSWP* on page 4-95
  Swap vectors.
- *VTBL, VTBX* on page 4-96
  Vector table look-up.
- *VTRN* on page 4-97
  Vector transpose.
- *VUZP, VZIP* on page 4-99
  Vector interleave and de-interleave.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.5 Advanced SIMD shift instructions

The following topics describe the Advanced SIMD shift instructions:

- *VSHL, VQSHL, VQSHLU, and VSHLL (by immediate)* on page 4-91
  Shift Left by immediate value.
- *V{Q}{R}SHL (by signed variable)* on page 4-19
  Shift left by signed variable.
- *V{R}SHR (by immediate)* on page 4-25
  Shift Right by immediate value.
- *V{R}SHRN (by immediate)* on page 4-26
  Shift Right, Narrow, by immediate value.
- *V{R}SRA (by immediate)* on page 4-27
  Shift Right by immediate value and Accumulate.
- *VQ{R}SHR{U}N (by immediate)* on page 4-82
  Shift Right by immediate value, and saturate.
- *VSLI and VSRI* on page 4-93
  Shift Left and Insert, and Shift Right and Insert.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.6 Advanced SIMD general arithmetic instructions

The following topics describe the Advanced SIMD general arithmetic instructions:

- *VABA{L} and VABD{L}* on page 4-28
  Vector Absolute Difference and Accumulate, and Absolute Difference.
- *V{Q}ABS and V{Q}NEG* on page 4-20
  Vector Absolute value, and Negate.
- *V{Q}ADD, VADDL, VADDW, V{Q}SUB, VSUBL, and VSUBW* on page 4-21
  Vector Add and Subtract.
- *V{R}ADDHN and V{R}SUBHN* on page 4-23
  Vector Add selecting High half, and Subtract selecting High Half.
- *V{R}HADD and VHSUB* on page 4-24
  Vector Halving Add and Subtract.
- *VPADD{L}, VPADAL* on page 4-79
  Vector Pairwise Add, Add and Accumulate.
- *VMAX, VMIN, VPMAX, and VPMIN* on page 4-64
  Vector Maximum, Minimum, Pairwise Maximum, and Pairwise Minimum.
- *VMAXNM, VMINNM (Advanced SIMD)* on page 4-65
  Vector Maximum, Minimum, consistent with IEEE 754-2008.
- *VCLS, VCLZ, and VCNT* on page 4-39
  Vector Count Leading Sign bits, Count Leading Zeros, and Count set bits.
- *VRECPE and VRSQRTE* on page 4-84
  Vector Reciprocal Estimate and Reciprocal Square Root Estimate.
- *VRECPS and VRSQRTS* on page 4-85
  Vector Reciprocal Step and Reciprocal Square Root Step.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.7 Advanced SIMD multiply instructions

The following topics describe the Advanced SIMD multiply instructions:

- *VMUL{L}, VMLA{L}, and VMLS{L}* on page 4-77.

  Vector Multiply, Multiply Accumulate, and Multiply Subtract.

- *VMUL{L}, VMLA{L}, and VMLS{L} (by scalar)* on page 4-78.

  Vector Multiply, Multiply Accumulate, and Multiply Subtract (by scalar).

- *VFMA, VFMS* on page 4-52.

  Vector Fused Multiply Accumulate and Vector Fused Multiply Subtract.

- *VQDMULL, VQDMLAL, and VQDMLSL (by vector or by scalar)* on page 4-83

  Vector Saturating Doubling Multiply, Multiply Accumulate, and Multiply Subtract (by vector or scalar).

- *VQ{R}DMULH (by vector or by scalar)* on page 4-81

  Vector Saturating Doubling Multiply returning High half (by vector or scalar).

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.8 Advanced SIMD load and store element and structure instructions

The following topics describe the Advanced SIMD load and store element and structure instructions:

- *Interleaving provided by load and store, element and structure instructions* on page 4-14.
- *Alignment restrictions in load and store, element and structure instructions* on page 4-15.
- *VLDn and VSTn (single n-element structure to one lane)* on page 4-56.
  This is used for almost all data accesses. A normal vector can be loaded ($n = 1$).
- *VLDn (single n-element structure to all lanes)* on page 4-58.
- *VLDn and VSTn (multiple n-element structures)* on page 4-60.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.9 Interleaving provided by load and store, element and structure instructions

Many instructions in this group provide interleaving when structures are stored to memory, and de-interleaving when structures are loaded from memory. Figure 4-1 shows an example of de-interleaving. Interleaving is the inverse process.

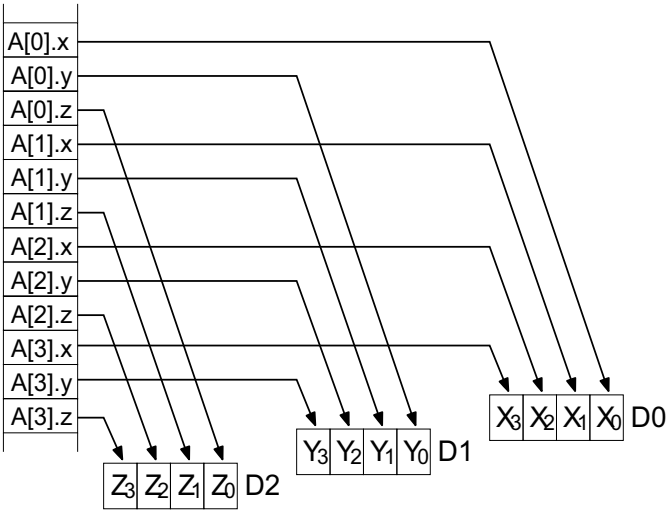


Figure 4-1 De-interleaving an array of 3-element structures

4.9.1 See also

Reference

- *Alignment restrictions in load and store, element and structure instructions* on page 4-15.
- *VLDn and VSTn (single n-element structure to one lane)* on page 4-56.
- *VLDn (single n-element structure to all lanes)* on page 4-58.
- *VLDn and VSTn (multiple n-element structures)* on page 4-60.

Other information

- *ARM Architecture Reference Manual*
  `http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/`.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.10 Alignment restrictions in load and store, element and structure instructions

Many of these instructions permit memory alignment restrictions to be specified. When the alignment is not specified in the instruction, the alignment restriction is controlled by the A bit (SCTLR bit[1]):

- If the A bit is 0, there are no alignment restrictions (except for strongly-ordered or device memory, where accesses must be element-aligned).
- If the A bit is 1, accesses must be element-aligned.

If an address is not correctly aligned, an alignment fault occurs.

4.10.1 See also

Reference

- *Interleaving provided by load and store, element and structure instructions* on page 4-14.
- *VLDn and VSTn (single n-element structure to one lane)* on page 4-56.
- *VLDn (single n-element structure to all lanes)* on page 4-58.
- *VLDn and VSTn (multiple n-element structures)* on page 4-60.

Other information

- *ARM Architecture Reference Manual* `http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/`.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.11 Advanced SIMD and floating-point pseudo-instructions

The following topics describe the Advanced SIMD and floating-point pseudo-instructions:

- *VLDR pseudo-instruction* on page 4-62 (Advanced SIMD and floating-point).
- *VLDR and VSTR (post-increment and pre-decrement)* on page 4-63 (Advanced SIMD and floating-point).
- *VMOV2* on page 4-73 (Advanced SIMD only).
- *VAND and VORN (immediate)* on page 4-34 (Advanced SIMD only).
- *VACLE and VACLT* on page 4-31 (Advanced SIMD only).
- *VCLE and VCLT* on page 4-38 (Advanced SIMD only).

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.12 Floating-point instructions

The following topics describe the floating-point instructions:

- *VABS, VNEG, and VSQRT* on page 4-29

  Floating-point absolute value, negate, and square root.

- *VADD, VSUB, and VDIV* on page 4-32

  Floating-point add, subtract, and divide.

- *VMUL, VMLA, VMLS, VNMUL, VNMLA, and VNMLS* on page 4-76

  Floating-point multiply and multiply accumulate, with optional negation.

- *VFNMA, VFNMS* on page 4-53

  Fused floating-point multiply accumulate and fused floating-point multiply subtract, with optional negation.

- *VCMP, VCMPE* on page 4-40

  Floating-point compare.

- *VSEL* on page 4-90

  Floating-point select.

- *VCVT (between single-precision and double-precision)* on page 4-44

  Convert between single-precision and double-precision.

- *VCVT (between floating-point and integer)* on page 4-45

  Convert between floating-point and integer.

- *VCVT (from floating-point to integer with directed rounding modes)* on page 4-46

  Convert from floating-point to integer with directed rounding modes.

- *VCVT (between floating-point and fixed-point)* on page 4-47

  Convert between floating-point and fixed-point.

- *VCVTB, VCVTT (half-precision extension)* on page 4-48

  Convert between half-precision and single-precision floating-point.

- *VCVTB, VCVTT (between half-precision and double-precision)* on page 4-49

  Convert between half-precision and double-precision.

- *VRINT (floating-point)* on page 4-89

  Floating-point round to integer.

- *VMAXNM, VMINNM (floating-point)* on page 4-66

  Floating-point maximum, minimum, consistent with IEEE 754-2008.

- *VMOV* on page 4-67

  Insert a floating-point immediate value in a single-precision or double-precision register.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.13 Cryptographic instructions

A set of cryptographic instructions is available in some implementations of the ARMv8 architecture. These instructions use the 128-bit Advanced SIMD registers and support the acceleration of the following cryptographic and hash algorithms:

- SHA1.
- SHA256.
- AES.

4.13.1 See also

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.14 V{Q}{R}SHL (by signed variable)

VSHL (Vector Shift Left by signed variable) takes each element in a vector, shifts them by a value from the least significant byte of the corresponding element of a second vector, and places the results in the destination vector. If the shift value is positive, the operation is a left shift. Otherwise, it is a right shift.

The results can be optionally saturated, rounded, or both. The sticky QC flag in the *Floating-Point Status and Control Register* (FPSCR) is set if saturation occurs.

4.14.1 Syntax

`V{Q}{R}SHL{cond}.datatype {Qd}, Qm, Qn`

`V{Q}{R}SHL{cond}.datatype {Dd}, Dm, Dn`

where:

`Q` if present, indicates that if any of the results overflow, they are saturated.

`R` if present, indicates that each result is rounded. Otherwise, each result is truncated.

cond is an optional condition code.

datatype must be one of `S8`, `S16`, `S32`, `S64`, `U8`, `U16`, `U32`, or `U64`.

Qd, *Qm*, *Qn* are the destination vector, the first operand vector, and the second operand vector, for a quadword operation.

Dd, *Dm*, *Dn* are the destination vector, the first operand vector, and the second operand vector, for a doubleword operation.

4.14.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.15 V{Q}ABS and V{Q}NEG

VABS (Vector Absolute) takes the absolute value of each element in a vector, and places the results in a second vector. (The floating-point version only clears the sign bit.)

VNEG (Vector Negate) negates each element in a vector, and places the results in a second vector. (The floating-point version only inverts the sign bit.)

Saturating versions of both instructions are available. The sticky QC flag in the FPSCR is set if saturation occurs.

4.15.1 Syntax

V{Q}*op*{*cond*}.*datatype Qd*, *Qm*

V{Q}*op*{*cond*}.*datatype Dd*, *Dm*

where:

Q	if present, indicates that if any of the results overflow, they are saturated.	
op	must be either ABS or NEG.	
cond	is an optional condition code.	
datatype	must be one of:	
	S8, S16, S32	for VABS, VNEG, VQABS, or VQNEG
	F32	for VABS and VNEG only.
Qd, *Qm*	are the destination vector and the operand vector, for a quadword operation.	
Dd, *Dm*	are the destination vector and the operand vector, for a doubleword operation.	

4.15.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.16 V{Q}ADD, VADDL, VADDW, V{Q}SUB, VSUBL, and VSUBW

VADD (Vector Add) adds corresponding elements in two vectors, and places the results in the destination vector.

VSUB (Vector Subtract) subtracts the elements of one vector from the corresponding elements of another vector, and places the results in the destination vector.

VADD and VSUB have these forms:

- Saturating. The sticky QC flag in the FPSCR is set if saturation occurs.
- Long.
- Wide.

4.16.1 Syntax

```
V{Q}op{cond}.datatype {Qd}, Qn, Qm    ; Saturating instruction
V{Q}op{cond}.datatype {Dd}, Dn, Dm    ; Saturating instruction
VopL{cond}.datatype Qd, Dn, Dm        ; Long instruction
VopW{cond}.datatype {Qd}, Qn, Dm      ; Wide instruction
```

where:

Q — if present, indicates that if any of the results overflow, they are saturated.

op — must be either ADD or SUB.

cond — is an optional condition code.

datatype — must be one of:

I8, I16, I32, I64, F32	for VADD or VSUB
S8, S16, S32	for VQADD, VQSUB, VADDL, VADDW, VSUBL, or VSUBW
U8, U16, U32	for VQADD, VQSUB, VADDL, VADDW, VSUBL, or VSUBW
S64, U64	for VQADD or VQSUB.

Qd, *Qn*, *Qm* — are the destination vector, the first operand vector, and the second operand vector, for a quadword operation.

Dd, *Dn*, *Dm* — are the destination vector, the first operand vector, and the second operand vector, for a doubleword operation.

Qd, *Dn*, *Dm* — are the destination vector, the first operand vector, and the second operand vector, for a long operation.

Qd, *Qn*, *Dm* — are the destination vector, the first operand vector, and the second operand vector, for a wide operation.

4.16.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.17 V{R}ADDHN and V{R}SUBHN

V{R}ADDHN (Vector Add and Narrow, selecting High half) adds corresponding elements in two vectors, selects the most significant halves of the results, and places the final results in the destination vector. Results can be either rounded or truncated.

V{R}SUBHN (Vector Subtract and Narrow, selecting High half) subtracts the elements of one vector from the corresponding elements of another vector, selects the most significant halves of the results, and places the final results in the destination vector. Results can be either rounded or truncated.

4.17.1 Syntax

V{R}*op*HN{*cond*}.*datatype Dd*, *Qn*, *Qm*

where:

R — if present, indicates that each result is rounded. Otherwise, each result is truncated.

op — must be either ADD or SUB.

cond — is an optional condition code.

datatype — must be one of I16, I32, or I64.

Dd, *Qn*, *Qm* — are the destination vector, the first operand vector, and the second operand vector.

4.17.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.18 V{R}HADD and VHSUB

VHADD (Vector Halving Add) adds corresponding elements in two vectors, shifts each result right one bit, and places the results in the destination vector. Results can be either rounded or truncated.

VHSUB (Vector Halving Subtract) subtracts the elements of one vector from the corresponding elements of another vector, shifts each result right one bit, and places the results in the destination vector. Results are always truncated.

4.18.1 Syntax

V{R}HADD{*cond*}.*datatype* {*Qd*}, *Qn*, *Qm*

V{R}HADD{*cond*}.*datatype* {*Dd*}, *Dn*, *Dm*

VHSUB{*cond*}.*datatype* {*Qd*}, *Qn*, *Qm*

VHSUB{*cond*}.*datatype* {*Dd*}, *Dn*, *Dm*

where:

R — if present, indicates that each result is rounded. Otherwise, each result is truncated.

cond — is an optional condition code.

datatype — must be one of S8, S16, S32, U8, U16, or U32.

Qd, *Qn*, *Qm* — are the destination vector, the first operand vector, and the second operand vector, for a quadword operation.

Dd, *Dn*, *Dm* — are the destination vector, the first operand vector, and the second operand vector, for a doubleword operation.

4.18.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.19 V{R}SHR (by immediate)

V{R}SHR (Vector Shift Right by immediate value) takes each element in a vector, right shifts them by an immediate value, and places the results in the destination vector. The results can optionally be rounded.

4.19.1 Syntax

V{R}SHR{*cond*}.*datatype* {*Qd*}, *Qm*, #*imm*

V{R}SHR{*cond*}.*datatype* {*Dd*}, *Dm*, #*imm*

where:

R
: if present, indicates that the results are rounded. Otherwise, the results are truncated.

cond
: is an optional condition code.

datatype
: must be one of S8, S16, S32, S64, U8, U16, U32, or U64.

Qd, *Qm*
: are the destination vector and the operand vector, for a quadword operation.

Dd, *Dm*
: are the destination vector and the operand vector, for a doubleword operation.

imm
: is the immediate value specifying the size of the shift, in the range 0 to (size in bits of *datatype*).

  V{R}SHR with an immediate value of zero is a pseudo-instruction for VMOV.

4.19.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *VMOV, VMVN (register)* on page 4-69.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.20 V{R}SHRN (by immediate)

V{R}SHRN (Vector Shift Right, Narrow, by immediate value) takes each element in a quadword vector, right shifts them by an immediate value, and places the results in a doubleword vector. The results can optionally be rounded.

4.20.1 Syntax

V{R}SHRN{*cond*}.*datatype* *Dd*, *Qm*, #*imm*

where:

R — if present, indicates that the results are rounded. Otherwise, the results are truncated.

cond — is an optional condition code.

datatype — must be one of I16, I32, or I64.

Dd, *Qm* — are the destination vector and the operand vector.

imm — is the immediate value specifying the size of the shift, in the range 0 to (size in bits of *datatype*)/2.

V{R}SHRN with an immediate value of zero is a pseudo-instruction for VMOVN.

4.20.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *VMOVL, V{Q}MOVN, VQMOVUN* on page 4-74.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.21 V{R}SRA (by immediate)

V{R}SRA (Vector Shift Right by immediate value and Accumulate) takes each element in a vector, right shifts them by an immediate value, and accumulates the results into the destination vector. The results can optionally be rounded.

4.21.1 Syntax

V{R}SRA{*cond*}.*datatype* {*Qd*}, *Qm*, #*imm*

V{R}SRA{*cond*}.*datatype* {*Dd*}, *Dm*, #*imm*

where:

R — if present, indicates that the results are rounded. Otherwise, the results are truncated.

cond — is an optional condition code.

datatype — must be one of S8, S16, S32, S64, U8, U16, U32, or U64.

Qd, *Qm* — are the destination vector and the operand vector, for a quadword operation.

Dd, *Dm* — are the destination vector and the operand vector, for a doubleword operation.

imm — is the immediate value specifying the size of the shift, in the range 1 to (size in bits of *datatype*).

4.21.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.22 VABA{L} and VABD{L}

VABA (Vector Absolute Difference and Accumulate) subtracts the elements of one vector from the corresponding elements of another vector, and accumulates the absolute values of the results into the elements of the destination vector.

VABD (Vector Absolute Difference) subtracts the elements of one vector from the corresponding elements of another vector, and places the absolute values of the results into the elements of the destination vector.

Long versions of both instructions are available.

4.22.1 Syntax

`V`*op*{*cond*}`.`*datatype* {*Qd*}, *Qn*, *Qm*

`V`*op*{*cond*}`.`*datatype* {*Dd*}, *Dn*, *Dm*

`V`*op*`L`{*cond*}`.`*datatype* *Qd*, *Dn*, *Dm*

where:

op must be either ABA or ABD.

cond is an optional condition code.

datatype must be one of:

- S8, S16, S32, U8, U16, or U32 for VABA, VABAL, or VABDL.
- S8, S16, S32, U8, U16, U32 or F32 for VABD.

Qd, *Qn*, *Qm* are the destination vector, the first operand vector, and the second operand vector, for a quadword operation.

Dd, *Dn*, *Dm* are the destination vector, the first operand vector, and the second operand vector, for a doubleword operation.

Qd, *Dn*, *Dm* are the destination vector, the first operand vector, and the second operand vector, for a long operation.

4.22.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.23 VABS, VNEG, and VSQRT

Floating-point absolute value, negate, and square root.

4.23.1 Syntax

`V`*op*`{`*cond*`}.F32` *Sd*, *Sm*

`V`*op*`{`*cond*`}.F64` *Dd*, *Dm*

where:

op is one of `ABS`, `NEG`, or `SQRT`.

cond is an optional condition code.

Sd, *Sm* are the single-precision registers for the result and operand.

Dd, *Dm* are the double-precision registers for the result and operand.

4.23.2 Usage

The `VABS` instruction takes the contents of *Sm* or *Dm*, clears the sign bit, and places the result in *Sd* or *Dd*. This gives the absolute value.

The `VNEG` instruction takes the contents of *Sm* or *Dm*, changes the sign bit, and places the result in *Sd* or *Dd*. This gives the negation of the value.

The `VSQRT` instruction takes the square root of the contents of *Sm* or *Dm*, and places the result in *Sd* or *Dd*.

In the case of a `VABS` and `VNEG` instruction, if the operand is a NaN, the sign bit is determined in each case as described, but no exception is produced.

4.23.3 Floating-point exceptions

`VABS` and `VNEG` instructions cannot produce any exceptions.

`VSQRT` instructions can produce Invalid Operation or Inexact exceptions.

4.23.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.24 VACGE and VACGT

Vector Absolute Compare takes the absolute value of each element in a vector, and compares it with the absolute value of the corresponding element of a second vector. If the condition is true, the corresponding element in the destination vector is set to all ones. Otherwise, it is set to all zeros.

4.24.1 Syntax

`VACop{cond}.F32 {Qd}, Qn, Qm`

`VACop{cond}.F32 {Dd}, Dn, Dm`

where:

op must be one of:

- `GE` Absolute Greater than or Equal
- `GT` Absolute Greater Than.

cond is an optional condition code.

Qd, Qn, Qm specifies the destination register, the first operand register, and the second operand register, for a quadword operation.

Dd, Dn, Dm specifies the destination register, the first operand register, and the second operand register, for a doubleword operation.

The result datatype is `I32`.

4.24.2 See also

Reference

- *VACLE and VACLT* on page 4-31.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.25 VACLE and VACLT

Vector Absolute Compare takes the absolute value of each element in a vector, and compares it with the absolute value of the corresponding element of a second vector. If the condition is true, the corresponding element in the destination vector is set to all ones. Otherwise, it is set to all zeros.

Note

On disassembly, these pseudo-instructions are disassembled to the corresponding `VACGE` and `VACGT` instructions, with the operands reversed.

4.25.1 Syntax

`VACop{cond}.datatype {Qd}, Qn, Qm`

`VACop{cond}.datatype {Dd}, Dn, Dm`

where:

op must be one of:
- `LE` Absolute Less than or Equal
- `LT` Absolute Less Than.

cond is an optional condition code.

datatype must be `F32`.

Qd or *Dd* is the Advanced SIMD register for the result.
The result datatype is `I32`.

Qn or *Dn* is the Advanced SIMD register holding the first operand.

Qm or *Dm* is the Advanced SIMD register holding the second operand.

4.25.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.
- *VACGE and VACGT* on page 4-30.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.26 VADD, VSUB, and VDIV

Floating-point add, subtract, and divide.

4.26.1 Syntax

`V`*op*`{`*cond*`}.F32 {`*Sd*`}, `*Sn*`, `*Sm*

`V`*op*`{`*cond*`}.F64 {`*Dd*`}, `*Dn*`, `*Dm*

where:

op is one of `ADD`, `SUB`, or `DIV`.

cond is an optional condition code.

Sd, *Sn*, *Sm* are the single-precision registers for the result and operands.

Dd, *Dn*, *Dm* are the double-precision registers for the result and operands.

4.26.2 Usage

The `VADD` instruction adds the values in the operand registers and places the result in the destination register.

The `VSUB` instruction subtracts the value in the second operand register from the value in the first operand register, and places the result in the destination register.

The `VDIV` instruction divides the value in the first operand register by the value in the second operand register, and places the result in the destination register.

4.26.3 Floating-point exceptions

`VADD` and `VSUB` instructions can produce Invalid Operation, Overflow, or Inexact exceptions.

`VDIV` operations can produce Division by Zero, Invalid Operation, Overflow, Underflow, or Inexact exceptions.

4.26.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.27 VAND, VBIC, VEOR, VORN, and VORR (register)

VAND (Bitwise AND), VBIC (Bit Clear), VEOR (Bitwise Exclusive OR), VORN (Bitwise OR NOT), and VORR (Bitwise OR) instructions perform bitwise logical operations between two registers, and place the results in the destination register.

4.27.1 Syntax

V*op*{*cond*}{.*datatype*} {*Qd*}, *Qn*, *Qm*

V*op*{*cond*}{.*datatype*} {*Dd*}, *Dn*, *Dm*

where:

op must be one of:

AND	Logical AND
ORR	Logical OR
EOR	Logical exclusive OR
BIC	Logical AND complement
ORN	Logical OR complement.

cond is an optional condition code.

datatype is an optional data type. The assembler ignores *datatype*.

Qd, Qn, Qm specifies the destination register, the first operand register, and the second operand register, for a quadword operation.

Dd, Dn, Dm specifies the destination register, the first operand register, and the second operand register, for a doubleword operation.

Note

VORR with the same register for both operands is a VMOV instruction. You can use VORR in this way, but disassembly of the resulting code produces the VMOV syntax.

4.27.2 See also

Reference

- *VMOV, VMVN (register)* on page 4-69.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.28 VAND and VORN (immediate)

VAND (Bitwise AND immediate) takes each element of the destination vector, performs a bitwise AND with an immediate value, and returns the result into the destination vector.

VORN (Bitwise OR NOT immediate) takes each element of the destination vector, performs a bitwise OR Complement with an immediate value, and returns the result into the destination vector.

Note

On disassembly, these pseudo-instructions are disassembled to the corresponding VBIC and VORR instructions, with the complementary immediate values.

4.28.1 Syntax

```
Vop{cond}.datatype Qd, #imm
Vop{cond}.datatype Dd, #imm
```

where:

op	must be either VAND or VORN.
cond	is an optional condition code.
datatype	must be either I8, I16, I32, or I64.
Qd or *Dd*	is the Advanced SIMD register for the result.
imm	is the immediate value.

4.28.2 Immediate values

If *datatype* is I16, the immediate value must have one of the following forms:

- 0xFFXY.
- 0xXYFF.

If *datatype* is I32, the immediate value must have one of the following forms:

- 0xFFFFFFXY.
- 0xFFFFXYFF.
- 0xFFXYFFFF.
- 0xXYFFFFFF.

4.28.3 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *VBIC and VORR (immediate)* on page 4-35.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.29 VBIC and VORR (immediate)

VBIC (Bit Clear immediate) takes each element of the destination vector, performs a bitwise AND Complement with an immediate value, and returns the result into the destination vector.

VORR (Bitwise OR immediate) takes each element of the destination vector, performs a bitwise OR with an immediate value, and returns the result into the destination vector.

4.29.1 Syntax

`V`*op*{*cond*}`.`*datatype* *Qd*`, #`*imm*

`V`*op*{*cond*}`.`*datatype* *Dd*`, #`*imm*

where:

op must be either BIC or ORR.

cond is an optional condition code.

datatype must be either I8, I16, I32, or I64.

Qd or *Dd* is the Advanced SIMD register for the source and result.

imm is the immediate value.

4.29.2 Immediate values

You can either specify *imm* as a pattern which the assembler repeats to fill the destination register, or you can directly specify the immediate value (that conforms to the pattern) in full. The pattern for *imm* depends on *datatype* as shown in Table 4-4:

Table 4-4 Patterns for immediate value

I16	I32
0x00XY	0x000000XY
0xXY00	0x0000XY00
	0x00XY0000
	0xXY000000

If you use the I8 or I64 datatypes, the assembler converts it to either the I16 or I32 instruction to match the pattern of *imm*. If the immediate value does not match any of the patterns in Table 4-4, the assembler generates an error.

4.29.3 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *VAND and VORN (immediate)* on page 4-34.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.30 VBIF, VBIT, and VBSL

VBIT (Bitwise Insert if True) inserts each bit from the first operand into the destination if the corresponding bit of the second operand is 1, otherwise leaves the destination bit unchanged.

VBIF (Bitwise Insert if False) inserts each bit from the first operand into the destination if the corresponding bit of the second operand is 0, otherwise leaves the destination bit unchanged.

VBSL (Bitwise Select) selects each bit for the destination from the first operand if the corresponding bit of the destination is 1, or from the second operand if the corresponding bit of the destination is 0.

4.30.1 Syntax

```
Vop{cond}{.datatype} {Qd}, Qn, Qm
Vop{cond}{.datatype} {Dd}, Dn, Dm
```

where:

op must be one of BIT, BIF, or BSL.

cond is an optional condition code.

datatype is an optional datatype. The assembler ignores *datatype*.

Qd, Qn, Qm specifies the destination register, the first operand register, and the second operand register, for a quadword operation.

Dd, Dn, Dm specifies the destination register, the first operand register, and the second operand register, for a doubleword operation.

4.30.2 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.31 VCEQ, VCGE, VCGT, VCLE, and VCLT

Vector Compare takes the value of each element in a vector, and compares it with the value of the corresponding element of a second vector, or zero. If the condition is true, the corresponding element in the destination vector is set to all ones. Otherwise, it is set to all zeros.

4.31.1 Syntax

```
VCop{cond}.datatype {Qd}, Qn, Qm
VCop{cond}.datatype {Dd}, Dn, Dm
VCop{cond}.datatype {Qd}, Qn, #0
VCop{cond}.datatype {Dd}, Dn, #0
```

where:

op
: must be one of:

  EQ — Equal

  GE — Greater than or Equal

  GT — Greater Than

  LE — Less than or Equal (only if the second operand is #0)

  LT — Less Than (only if the second operand is #0).

cond
: is an optional condition code.

datatype
: must be one of:

- I8, I16, I32, or F32 for EQ.
- S8, S16, S32, U8, U16, U32, or F32 for GE, GT, LE, or LT (except #0 form).
- S8, S16, S32, or F32 for GE, GT, LE, or LT (#0 form).

The result datatype is:

- I32 for operand datatypes I32, S32, U32, or F32.
- I16 for operand datatypes I16, S16, or U16.
- I8 for operand datatypes I8, S8, or U8.

Qd, Qn, Qm
: specifies the destination register, the first operand register, and the second operand register, for a quadword operation.

Dd, Dn, Dm
: specifies the destination register, the first operand register, and the second operand register, for a doubleword operation.

#0
: replaces *Qm* or *Dm* for comparisons with zero.

4.31.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *VCLE and VCLT* on page 4-38.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.32 VCLE and VCLT

Vector Compare takes the value of each element in a vector, and compares it with the value of the corresponding element of a second vector, or zero. If the condition is true, the corresponding element in the destination vector is set to all ones. Otherwise, it is set to all zeros.

Note

On disassembly, these pseudo-instructions are disassembled to the corresponding `VCGE` and `VCGT` instructions, with the operands reversed.

4.32.1 Syntax

`VCop{cond}.datatype {Qd}, Qn, Qm`

`VCop{cond}.datatype {Dd}, Dn, Dm`

where:

op
must be one of:
- `LE` Less than or Equal
- `LT` Less Than.

cond
is an optional condition code.

datatype
must be one of `S8`, `S16`, `S32`, `U8`, `U16`, `U32`, or `F32`.

Qd or *Dd*
is the Advanced SIMD register for the result.

The result datatype is:
- `I32` for operand datatypes `I32`, `S32`, `U32`, or `F32`.
- `I16` for operand datatypes `I16`, `S16`, or `U16`.
- `I8` for operand datatypes `I8`, `S8`, or `U8`.

Qn or *Dn*
is the Advanced SIMD register holding the first operand.

Qm or *Dm*
is the Advanced SIMD register holding the second operand.

4.32.2 See also

Concepts

armasm User Guide:
- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference
- *Condition codes* on page 3-26.
- *VCLE and VCLT*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.33 VCLS, VCLZ, and VCNT

VCLS (Vector Count Leading Sign bits) counts the number of consecutive bits following the topmost bit, that are the same as the topmost bit, in each element in a vector, and places the results in a second vector.

VCLZ (Vector Count Leading Zeros) counts the number of consecutive zeros, starting from the top bit, in each element in a vector, and places the results in a second vector.

VCNT (Vector Count set bits) counts the number of bits that are one in each element in a vector, and places the results in a second vector.

4.33.1 Syntax

`V`*op*{*cond*}`.`*datatype* *Qd*, *Qm*

`V`*op*{*cond*}`.`*datatype* *Dd*, *Dm*

where:

op must be one of CLS, CLZ, or CNT.

cond is an optional condition code.

datatype must be one of:

- S8, S16, or S32 for CLS.
- I8, I16, or I32 for CLZ.
- I8 for CNT.

Qd, *Qm* are the destination vector and the operand vector, for a quadword operation.

Dd, *Dm* are the destination vector and the operand vector, for a doubleword operation.

4.33.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.34 VCMP, VCMPE

Floating-point compare.

4.34.1 Syntax

```
VCMP{E}{cond}.F32 Sd, Sm
VCMP{E}{cond}.F32 Sd, #0
VCMP{E}{cond}.F64 Dd, Dm
VCMP{E}{cond}.F64 Dd, #0
```

where:

`E` if present, indicates that the instruction raises an Invalid Operation exception if either operand is a quiet or signaling NaN. Otherwise, it raises the exception only if either operand is a signaling NaN.

cond is an optional condition code.

Sd, *Sm* are the single-precision registers holding the operands.

Dd, *Dm* are the double-precision registers holding the operands.

4.34.2 Usage

The `VCMP{E}` instruction subtracts the value in the second operand register (or 0 if the second operand is #0) from the value in the first operand register, and sets the floating-point condition flags on the result.

4.34.3 Floating-point exceptions

`VCMP{E}` instructions can produce Invalid Operation exceptions.

4.34.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.35 VCVT (between fixed-point or integer, and floating-point)

VCVT (Vector Convert) converts each element in a vector in one of the following ways, and places the results in the destination vector:

- From floating-point to integer.
- From integer to floating-point.
- From floating-point to fixed-point.
- From fixed-point to floating-point.

4.35.1 Syntax

```
VCVT{cond}.type Qd, Qm {, #fbits}
VCVT{cond}.type Dd, Dm {, #fbits}
```

where:

cond is an optional condition code.

type specifies the data types for the elements of the vectors. It must be one of:

- `S32.F32` floating-point to signed integer or fixed-point
- `U32.F32` floating-point to unsigned integer or fixed-point
- `F32.S32` signed integer or fixed-point to floating-point
- `F32.U32` unsigned integer or fixed-point to floating-point.

Qd, Qm specifies the destination vector and the operand vector, for a quadword operation.

Dd, Dm specifies the destination vector and the operand vector, for a doubleword operation.

fbits if present, specifies the number of fraction bits in the fixed point number. Otherwise, the conversion is between floating-point and integer. *fbits* must lie in the range 0-32. If *fbits* is omitted, the number of fraction bits is 0.

4.35.2 Rounding

Integer or fixed-point to floating-point conversions use round to nearest.

Floating-point to integer or fixed-point conversions use round towards zero.

4.35.3 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.36 VCVT (from floating-point to integer with directed rounding modes)

VCVT (Vector Convert) converts each element in a vector from floating-point to signed or unsigned integer, and places the results in the destination vector.

Note

This instruction is supported only in ARMv8.

4.36.1 Syntax

VCVT*mode*.*type* *Qd*, *Qm*

VCVT*mode*.type *Dd*, *Dm*

where:

mode must be one of:

- A meaning round to nearest, ties away from zero
- N meaning round to nearest, ties to even
- P meaning round towards plus infinity
- M meaning round towards minus infinity.

type specifies the data types for the elements of the vectors. It must be one of:

- S32.F32 floating-point to signed integer
- U32.F32 floating-point to unsigned integer.

Qd, Qm specifies the destination and operand vectors, for a quadword operation.

Dd, Dm specifies the destination and operand vectors, for a doubleword operation.

4.36.2 Notes

You cannot use VCVT with a directed rounding mode inside an IT block.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.37 VCVT (between half-precision and single-precision floating-point)

VCVT (Vector Convert), with half-precision extension, converts each element in a vector in one of the following ways, and places the results in the destination vector:

- From half-precision floating-point to single-precision floating-point (F32.F16).
- From single-precision floating-point to half-precision floating-point (F16.F32).

4.37.1 Syntax

VCVT{*cond*}.F32.F16 *Qd*, *Dm*

VCVT{*cond*}.F16.F32 *Dd*, *Qm*

where:

cond is an optional condition code.

Qd, Dm specifies the destination vector for the single-precision results and the half-precision operand vector.

Dd, Qm specifies the destination vector for half-precision results and the single-precision operand vector.

4.37.2 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.38 VCVT (between single-precision and double-precision)

Convert between single-precision and double-precision numbers.

4.38.1 Syntax

```
VCVT{cond}.F64.F32 Dd, Sm
VCVT{cond}.F32.F64 Sd, Dm
```

where:

cond is an optional condition code.

Dd is a double-precision register for the result.

Sm is a single-precision register holding the operand.

Sd is a single-precision register for the result.

Dm is a double-precision register holding the operand.

4.38.2 Usage

These instructions convert the single-precision value in *Sm* to double-precision and places the result in *Dd*, or the double-precision value in *Dm* to single-precision and place the result in *Sd*.

4.38.3 Floating-point exceptions

These instructions can produce Invalid Operation, Input Denormal, Overflow, Underflow, or Inexact exceptions.

4.38.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.39 VCVT (between floating-point and integer)

Convert between floating-point numbers and integers.

4.39.1 Syntax

```
VCVT{R}{cond}.type.F64 Sd, Dm
VCVT{R}{cond}.type.F32 Sd, Sm
VCVT{cond}.F64.type Dd, Sm
VCVT{cond}.F32.type Sd, Sm
```

where:

`R`	makes the operation use the rounding mode specified by the FPSCR. Otherwise, the operation rounds towards zero.
cond	is an optional condition code.
type	can be either `U32` (unsigned 32-bit integer) or `S32` (signed 32-bit integer).
Sd	is a single-precision register for the result.
Dd	is a double-precision register for the result.
Sm	is a single-precision register holding the operand.
Dm	is a double-precision register holding the operand.

4.39.2 Usage

The first two forms of this instruction convert from floating-point to integer.

The third and fourth forms convert from integer to floating-point.

4.39.3 Floating-point exceptions

These instructions can produce Input Denormal, Invalid Operation, or Inexact exceptions.

4.39.4 See also

Reference

- *Condition codes* on page 3-26.

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.40 VCVT (from floating-point to integer with directed rounding modes)

Convert from floating-point to signed or unsigned integer with directed rounding modes.

Note

This instruction is supported only in ARMv8.

4.40.1 Syntax

`VCVT`*mode*`.`*S32.F64 Sd*`,` *Dm*

`VCVT`*mode*`.`*S32.F32 Sd*`,` *Sm*

`VCVT`*mode*`.`*U32.F64 Sd*`,` *Dm*

`VCVT`*mode*`.`*U32.F32 Sd*`,` *Sm*

where:

mode must be one of:

- `A` meaning round to nearest, ties away from zero
- `N` meaning round to nearest, ties to even
- `P` meaning round towards plus infinity
- `M` meaning round towards minus infinity.

Sd, Sm specifies the single-precision registers for the operand and result.

Sd, Dm specifies a single-precision register for the result and double-precision register holding the operand.

4.40.2 Notes

You cannot use `VCVT` with a directed rounding mode inside an IT block.

4.40.3 Floating-point exceptions

These instructions can produce Input Denormal, Invalid Operation, or Inexact exceptions.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.41 VCVT (between floating-point and fixed-point)

Convert between floating-point and fixed-point numbers.

4.41.1 Syntax

```
VCVT{cond}.type.F64 Dd, Dd, #fbits
VCVT{cond}.type.F32 Sd, Sd, #fbits
VCVT{cond}.F64.type Dd, Dd, #fbits
VCVT{cond}.F32.type Sd, Sd, #fbits
```

where:

cond is an optional condition code.

type can be any one of:

- S16 16-bit signed fixed-point number
- U16 16-bit unsigned fixed-point number
- S32 32-bit signed fixed-point number
- U32 32-bit unsigned fixed-point number.

Sd is a single-precision register for the operand and result.

Dd is a double-precision register for the operand and result.

fbits is the number of fraction bits in the fixed-point number, in the range 0-16 if *type* is S16 or U16, or in the range 1-32 if *type* is S32 or U32.

4.41.2 Usage

The first two forms of this instruction convert from floating-point to fixed-point.

The third and fourth forms convert from fixed-point to floating-point.

In all cases the fixed-point number is contained in the least significant 16 or 32 bits of the register.

4.41.3 Floating-point exceptions

These instructions can produce Input Denormal, Invalid Operation, or Inexact exceptions.

4.41.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.42 VCVTB, VCVTT (half-precision extension)

Converts between half-precision and single-precision floating-point numbers in the following ways:

- VCVTB uses the bottom half (bits[15:0]) of the single word register to obtain or store the half-precision value.
- VCVTT uses the top half (bits[31:16]) of the single word register to obtain or store the half-precision value.

4.42.1 Syntax

VCVTB{*cond*}.*type Sd, Sm*

VCVTT{*cond*}.*type Sd, Sm*

where:

cond is an optional condition code.

type can be any one of:

- F32.F16 convert from half-precision to single-precision
- F16.F32 convert from single-precision to half-precision.

Sd is a single word register for the result.

Sm is a single word register for the operand.

4.42.2 Floating-point exceptions

These instructions can produce Input Denormal, Invalid Operation, Overflow, Underflow, or Inexact exceptions.

4.42.3 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.43 VCVTB, VCVTT (between half-precision and double-precision)

Converts between half-precision and double-precision floating-point numbers in either of the following ways:

- From half-precision floating-point to double-precision floating-point (`F64.F16`).
- From double-precision floating-point to half-precision floating-point (`F16.F64`).

`VCVTB` uses the bottom half (bits[15:0]) of the single word register to obtain or store the half-precision value.

`VCVTT` uses the top half (bits[31:16]) of the single word register to obtain or store the half-precision value.

Note

These instructions are supported only in ARMv8.

4.43.1 Syntax

`VCVTB{`*cond*`}.F64.F16` *Dd*, *Sm*

`VCVTB{`*cond*`}.F16.F64` *Sd*, *Dm*

`VCVTT{`*cond*`}.F64.F16` *Dd*, *Sm*

`VCVTT{`*cond*`}.F16.F64` *Sd*, *Dm*

where:

cond is an optional condition code.

Dd is a double-precision register for the result.

Sm is a single word register holding the operand.

Sd is a single word register for the result.

Dm is a double-precision register holding the operand.

4.43.2 Usage

These instructions convert the half-precision value in *Sm* to double-precision and place the result in *Dd*, or the double-precision value in *Dm* to half-precision and place the result in *Sd*.

4.43.3 Floating-point exceptions

These instructions can produce Input Denormal, Invalid Operation, Overflow, Underflow, or Inexact exceptions.

4.43.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.44 VDUP

VDUP (Vector Duplicate) duplicates a scalar into every element of the destination vector. The source can be an Advanced SIMD scalar or an ARM register.

4.44.1 Syntax

VDUP{*cond*}.*size* *Qd*, *Dm*[*x*]

VDUP{*cond*}.*size* *Dd*, *Dm*[*x*]

VDUP{*cond*}.*size* *Qd*, *Rm*

VDUP{*cond*}.*size* *Dd*, *Rm*

where:

cond is an optional condition code.

size must be 8, 16, or 32.

Qd specifies the destination register for a quadword operation.

Dd specifies the destination register for a doubleword operation.

Dm[*x*] specifies the Advanced SIMD scalar.

Rm specifies the ARM register. *Rm* must not be PC.

4.44.2 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.45 VEXT

VEXT (Vector Extract) extracts 8-bit elements from the bottom end of the second operand vector and the top end of the first, concatenates them, and places the result in the destination vector. See Figure 4-2 for an example.

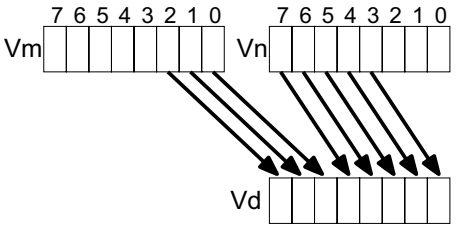


Figure 4-2 Operation of doubleword VEXT for imm = 3

4.45.1 Syntax

```
VEXT{cond}.8 {Qd}, Qn, Qm, #imm
VEXT{cond}.8 {Dd}, Dn, Dm, #imm
```

where:

cond is an optional condition code.

Qd, Qn, Qm specifies the destination register, the first operand register, and the second operand register, for a quadword operation.

Dd, Dn, Dm specifies the destination register, the first operand register, and the second operand register, for a doubleword operation.

imm is the number of 8-bit elements to extract from the bottom of the second operand vector, in the range 0-7 for doubleword operations, or 0-15 for quadword operations.

4.45.2 VEXT pseudo-instruction

You can specify a datatype of 16, 32, or 64 instead of 8. In this case, #imm refers to halfwords, words, or doublewords instead of referring to bytes, and the permitted ranges are correspondingly reduced.

4.45.3 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.46 VFMA, VFMS

VFMA (Vector Fused Multiply Accumulate) multiplies corresponding elements in the two operand vectors, and accumulates the results into the elements of the destination vector. The result of the multiply is not rounded before the accumulation.

VFMS (Vector Fused Multiply Subtract) multiplies corresponding elements in the two operand vectors, then subtracts the products from the corresponding elements of the destination vector, and places the final results in the destination vector. The result of the multiply is not rounded before the subtraction.

4.46.1 Syntax

```
Vop{cond}.F32 {Qd}, Qn, Qm
Vop{cond}.F32 {Dd}, Dn, Dm
Vop{cond}.F64 {Dd}, Dn, Dm
Vop{cond}.F32 {Sd}, Sn, Sm
```

where:

op is one of FMA or FMS.

cond is an optional condition code.

Sd, *Sn*, *Sm* are the destination and operand vectors for word operation.

Dd, *Dn*, *Dm* are the destination and operand vectors for doubleword operation.

Qd, *Qn*, *Qm* are the destination and operand vectors for quadword operation.

4.46.2 See also

Reference

- *Condition codes* on page 3-26.
- *VMUL{L}, VMLA{L}, and VMLS{L}* on page 4-77.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.47 VFNMA, VFNMS

Fused floating-point multiply accumulate and fused floating-point multiply subtract with optional negation.

4.47.1 Syntax

`VFN`*op*`{`*cond*`}.F64 {`*Dd*`}, `*Dn*`, `*Dm*

`VFN`*op*`{`*cond*`}.F32 {`*Sd*`}, `*Sn*`, `*Sm*

where:

op is one of `MA` or `MS`.

cond is an optional condition code.

Sd, *Sn*, *Sm* are the single-precision registers for the result and operands.

Dd, *Dn*, *Dm* are the double-precision registers for the result and operands.

Qd, *Qn*, *Qm* are the double-precision registers for the result and operands.

4.47.2 Usage

`VFNMA` multiplies the values in the operand registers, adds the value in the destination register, and places the final result in the destination register. The result of the multiply is not rounded before the accumulation. The final result is negated.

`VFNMS` multiplies the values in the operand registers, subtracts the product from the value in the destination register, and places the final result in the destination register. The result of the multiply is not rounded before the subtraction. The final result is negated.

4.47.3 Floating-point exceptions

These instructions can produce Input Denormal, Invalid Operation, Overflow, Underflow, or Inexact exceptions.

4.47.4 See also

Reference

- *Condition codes* on page 3-26.
- *VMUL, VMLA, VMLS, VNMUL, VNMLA, and VNMLS* on page 4-76.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.48 VLDM, VSTM, VPOP, and VPUSH

Extension register load multiple, store multiple, pop from stack, push onto stack.

4.48.1 Syntax

`VLDM`*mode*`{`*cond*`} `*Rn*`{!}, `*Registers*

`VSTM`*mode*`{`*cond*`} `*Rn*`{!}, `*Registers*

`VPOP{`*cond*`} `*Registers*

`VPUSH{`*cond*`} `*Registers*

where:

mode must be one of:

- `IA` meaning Increment address After each transfer. `IA` is the default, and can be omitted.
- `DB` meaning Decrement address Before each transfer.
- `EA` meaning Empty Ascending stack operation. This is the same as `DB` for loads, and the same as `IA` for saves.
- `FD` meaning Full Descending stack operation. This is the same as `IA` for loads, and the same as `DB` for saves.

cond is an optional condition code.

Rn is the ARM register holding the base address for the transfer.

`!` is optional. `!` specifies that the updated base address must be written back to *Rn*. If `!` is not specified, *mode* must be `IA`.

Registers is a list of consecutive extension registers enclosed in braces, `{` and `}`. The list can be comma-separated, or in range format. There must be at least one register in the list.

You can specify `S`, `D`, or `Q` registers, but they must not be mixed. The number of registers must not exceed 16 `D` registers, or 8 `Q` registers. If `Q` registers are specified, on disassembly they are shown as `D` registers.

Note

- `VPOP` *Registers* is equivalent to `VLDM sp!,` *Registers*.
- `VPUSH` *Registers* is equivalent to `VSTMDB sp!,` *Registers*.
- You can use either form of these instructions. They disassemble to `VPOP` and `VPUSH`.

4.48.2 See also

Concepts

armasm User Guide:

- *Stack implementation using LDM and STM* on page 7-23.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.49 VLDR and VSTR

Extension register load and store.

4.49.1 Syntax

```
VLDR{cond}{.size} Fd, [Rn{, #offset}]
VSTR{cond}{.size} Fd, [Rn{, #offset}]
VLDR{cond}{.size} Fd, label
```

where:

cond is an optional condition code.

size is an optional data size specifier. Must be 32 if *Fd* is an S register, or 64 otherwise.

Fd is the extension register to be loaded or saved. For an Advanced SIMD instruction, it must be a D register. For a floating-point instruction, it can be either a D or S register.

Rn is the ARM register holding the base address for the transfer.

offset is an optional numeric expression. It must evaluate to a numeric value at assembly time. The value must be a multiple of 4, and lie in the range –1020 to +1020. The value is added to the base address to form the address used for the transfer.

label is a PC-relative expression.

label must be aligned on a word boundary within ±1KB of the current instruction.

4.49.2 Usage

The VLDR instruction loads an extension register from memory. The VSTR instruction saves the contents of an extension register to memory.

One word is transferred if *Fd* is an S register (floating-point only). Two words are transferred otherwise.

There is also a VLDR pseudo-instruction.

4.49.3 See also

Concepts

armasm User Guide:

- *Register-relative and PC-relative expressions* on page 10-7.

Reference

- *Condition codes* on page 3-26.
- *VLDR pseudo-instruction* on page 4-62.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.50 VLD*n* and VST*n* (single *n*-element structure to one lane)

Vector Load single *n*-element structure to one lane. It loads one *n*-element structure from memory into one or more Advanced SIMD registers. Elements of the register that are not loaded are unaltered.

Vector Store single *n*-element structure to one lane. It stores one *n*-element structure into memory from one or more Advanced SIMD registers.

4.50.1 Syntax

`V`*op*{*cond*}`.`*datatype* *list*, [*Rn*{`@`*align*}]{`!`}

`V`*op*{*cond*}`.`*datatype* *list*, [*Rn*{`@`*align*}], *Rm*

where:

op must be either `LD` or `ST`.

n must be one of 1, 2, 3, or 4.

cond is an optional condition code.

datatype see Table 4-5.

list specifies the Advanced SIMD register list. See Table 4-5 for options.

Rn is the ARM register containing the base address. *Rn* cannot be PC.

align specifies an optional alignment. See Table 4-5 for options.

! if ! is present, *Rn* is updated to (*Rn* + the number of bytes transferred by the instruction). The update occurs after all the loads or stores have taken place.

Rm is an ARM register containing an offset from the base address. If *Rm* is present, the instruction updates *Rn* to (*Rn* + *Rm*) *after* using the address to access memory. *Rm* cannot be SP or PC.

Table 4-5 Permitted combinations of parameters

n	*datatype*	*list* [a]	*align* [b]	Alignment
1	8	`{Dd[x]}`	-	Standard only
	16	`{Dd[x]}`	`@16`	2-byte
	32	`{Dd[x]}`	`@32`	4-byte
2	8	`{Dd[x], D(d+1)[x]}`	`@16`	2-byte
	16	`{Dd[x], D(d+1)[x]}`	`@32`	4-byte
		`{Dd[x], D(d+2)[x]}`	`@32`	4-byte
	32	`{Dd[x], D(d+1)[x]}`	`@64`	8-byte
		`{Dd[x], D(d+2)[x]}`	`@64`	8-byte
3	8	`{Dd[x], D(d+1)[x], D(d+2)[x]}`	-	Standard only
	16 or 32	`{Dd[x], D(d+1)[x], D(d+2)[x]}`	-	Standard only
		`{Dd[x], D(d+2)[x], D(d+4)[x]}`	-	Standard only

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 4-5 Permitted combinations of parameters (continued)

n	*datatype*	*list* [a]	*align* [b]	Alignment
4	8	{Dd[x], D(d+1)[x], D(d+2)[x], D(d+3)[x]}	@32	4-byte
	16	{Dd[x], D(d+1)[x], D(d+2)[x], D(d+3)[x]}	@64	8-byte
		{Dd[x], D(d+2)[x], D(d+4)[x], D(d+6)[x]}	@64	8-byte
	32	{Dd[x], D(d+1)[x], D(d+2)[x], D(d+3)[x]}	@64 or @128	8-byte or 16-byte
		{Dd[x], D(d+2)[x], D(d+4)[x], D(d+6)[x]}	@64 or @128	8-byte or 16-byte

a. Every register in the list must be in the range D0-D31.

b. *align* can be omitted. In this case, standard alignment rules apply, see *Alignment restrictions in load and store, element and structure instructions* on page 4-15.

4.50.2 See also

Reference

- *Condition codes* on page 3-26.
- *Interleaving provided by load and store, element and structure instructions* on page 4-14.
- *Alignment restrictions in load and store, element and structure instructions* on page 4-15.
- *VLDn (single n-element structure to all lanes)* on page 4-58.
- *VLDn and VSTn (multiple n-element structures)* on page 4-60.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.51 VLD*n* (single *n*-element structure to all lanes)

Vector Load single *n*-element structure to all lanes. It loads multiple copies of one *n*-element structure from memory into one or more Advanced SIMD registers.

4.51.1 Syntax

`VLDn{cond}.datatype list, [Rn{@align}]{!}`

`VLDn{cond}.datatype list, [Rn{@align}], Rm`

where:

n must be one of 1, 2, 3, or 4.

cond is an optional condition code.

datatype see Table 4-6.

list specifies the Advanced SIMD register list. See Table 4-6 for options.

Rn is the ARM register containing the base address. *Rn* cannot be PC.

align specifies an optional alignment. See Table 4-6 for options.

! if ! is present, *Rn* is updated to (*Rn* + the number of bytes transferred by the instruction). The update occurs after all the loads or stores have taken place.

Rm is an ARM register containing an offset from the base address. If *Rm* is present, the instruction updates *Rn* to (*Rn* + *Rm*) *after* using the address to access memory. *Rm* cannot be SP or PC.

Table 4-6 Permitted combinations of parameters

n	*datatype*	*list* [a]	*align* [b]	Alignment
1	8	{Dd[]}	-	Standard only
		{Dd[],D(d+1)[]}	-	Standard only
	16	{Dd[]}	@16	2-byte
		{Dd[],D(d+1)[]}	@16	2-byte
	32	{Dd[]}	@32	4-byte
		{Dd[],D(d+1)[]}	@32	4-byte
2	8	{Dd[], D(d+1)[]}	@8	byte
		{Dd[], D(d+2)[]}	@8	byte
	16	{Dd[], D(d+1)[]}	@16	2-byte
		{Dd[], D(d+2)[]}	@16	2-byte
	32	{Dd[], D(d+1)[]}	@32	4-byte
		{Dd[], D(d+2)[]}	@32	4-byte
3	8, 16, or 32	{Dd[], D(d+1)[], D(d+2)[]}	-	Standard only
		{Dd[], D(d+2)[], D(d+4)[]}	-	Standard only
4	8	{Dd[], D(d+1)[], D(d+2)[], D(d+3)[]}	@32	4-byte

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 4-6 Permitted combinations of parameters (continued)

n	*datatype*	*list* [a]	*align* [b]	Alignment
		{Dd[], D(d+2)[], D(d+4)[], D(d+6)[]}	@32	4-byte
	16	{Dd[], D(d+1)[], D(d+2)[], D(d+3)[]}	@64	8-byte
		{Dd[], D(d+2)[], D(d+4)[], D(d+6)[]}	@64	8-byte
	32	{Dd[], D(d+1)[], D(d+2)[], D(d+3)[]}	@64 or @128	8-byte or 16-byte
		{Dd[], D(d+2)[], D(d+4)[], D(d+6)[]}	@64 or @128	8-byte or 16-byte

a. Every register in the list must be in the range D0-D31.

b. *align* can be omitted. In this case, standard alignment rules apply, see *Alignment restrictions in load and store, element and structure instructions* on page 4-15.

4.51.2 See also

Reference

- *Condition codes* on page 3-26.
- *Interleaving provided by load and store, element and structure instructions* on page 4-14.
- *Alignment restrictions in load and store, element and structure instructions* on page 4-15.
- *VLDn and VSTn (single n-element structure to one lane)* on page 4-56.
- *VLDn and VSTn (multiple n-element structures)* on page 4-60.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.52 VLD*n* and VST*n* (multiple *n*-element structures)

Vector Load multiple *n*-element structures. It loads multiple *n*-element structures from memory into one or more Advanced SIMD registers, with de-interleaving (unless *n* == 1). Every element of each register is loaded.

Vector Store multiple *n*-element structures. It stores multiple *n*-element structures to memory from one or more Advanced SIMD registers, with interleaving (unless *n* == 1). Every element of each register is stored.

4.52.1 Syntax

`V`*op*{*cond*}`.`*datatype list*, [*Rn*{`@`*align*}]{`!`}

`V`*op*{*cond*}`.`*datatype list*, [*Rn*{`@`*align*}], *Rm*

where:

op must be either `LD` or `ST`.

n must be one of 1, 2, 3, or 4.

cond is an optional condition code.

datatype see Table 4-7 for options.

list specifies the Advanced SIMD register list. See Table 4-7 for options.

Rn is the ARM register containing the base address. *Rn* cannot be PC.

align specifies an optional alignment. See Table 4-7 for options.

! if ! is present, *Rn* is updated to (*Rn* + the number of bytes transferred by the instruction). The update occurs after all the loads or stores have taken place.

Rm is an ARM register containing an offset from the base address. If *Rm* is present, the instruction updates *Rn* to (*Rn* + *Rm*) *after* using the address to access memory. *Rm* cannot be SP or PC.

Table 4-7 Permitted combinations of parameters

n	*datatype*	*list* [a]	*align* [b]	Alignment
1	8, 16, 32, or 64	{Dd}	@64	8-byte
		{Dd, D(d+1)}	@64 or @128	8-byte or 16-byte
		{Dd, D(d+1), D(d+2)}	@64	8-byte
		{Dd, D(d+1), D(d+2), D(d+3)}	@64, @128, or @256	8-byte, 16-byte, or 32-byte
2	8, 16, or 32	{Dd, D(d+1)}	@64, @128	8-byte or 16-byte
		{Dd, D(d+2)}	@64, @128	8-byte or 16-byte
		{Dd, D(d+1), D(d+2), D(d+3)}	@64, @128, or @256	8-byte, 16-byte, or 32-byte
3	8, 16, or 32	{Dd, D(d+1), D(d+2)}	@64	8-byte
		{Dd, D(d+2), D(d+4)}	@64	8-byte
4	8, 16, or 32	{Dd, D(d+1), D(d+2), D(d+3)}	@64, @128, or @256	8-byte, 16-byte, or 32-byte
		{Dd, D(d+2), D(d+4), D(d+6)}	@64, @128, or @256	8-byte, 16-byte, or 32-byte

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

a. Every register in the list must be in the range `D0-D31`.
b. *`align`* can be omitted. In this case, standard alignment rules apply, see *Alignment restrictions in load and store, element and structure instructions* on page 4-15.

4.52.2 See also

Reference

- *Condition codes* on page 3-26.
- *Interleaving provided by load and store, element and structure instructions* on page 4-14.
- *Alignment restrictions in load and store, element and structure instructions* on page 4-15.
- *VLDn and VSTn (single n-element structure to one lane)* on page 4-56.
- *VLDn (single n-element structure to all lanes)* on page 4-58.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.53 VLDR pseudo-instruction

The VLDR pseudo-instruction loads a constant value into every element of a 64-bit Advanced SIMD vector, or into a floating-point single-precision or double-precision register.

Note

This description is for the VLDR pseudo-instruction only.

4.53.1 Syntax

`VLDR{cond}.datatype Dd,=constant`

`VLDR{cond}.datatype Sd,=constant`

where:

datatype must be one of:

I*n*	Advanced SIMD only
S*n*	Advanced SIMD only
U*n*	Advanced SIMD only
F32	Advanced SIMD or floating-point
F64	Floating-point only.

n must be one of 8, 16, 32, or 64.

cond is an optional condition code.

Dd or *Sd* is the extension register to be loaded.

constant is an immediate value of the appropriate type for *datatype*.

4.53.2 Usage

If an instruction (for example, VMOV) is available that can generate the constant directly into the register, the assembler uses it. Otherwise, it generates a doubleword literal pool entry containing the constant and loads the constant using a VLDR instruction.

4.53.3 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *VLDR and VSTR* on page 4-55.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.54 VLDR and VSTR (post-increment and pre-decrement)

Pseudo-instructions that load or store extension registers with post-increment and pre-decrement.

Note

There are also VLDR and VSTR instructions without post-increment and pre-decrement.

4.54.1 Syntax

```
op{cond}{.size} Fd, [Rn], #offset          ; post-increment
op{cond}{.size} Fd, [Rn, #-offset]!        ; pre-decrement
```

where:

op can be:
- VLDR - load extension register from memory.
- VSTR - store contents of extension register to memory.

cond is an optional condition code.

size is an optional data size specifier. Must be 32 if *Fd* is an S register, or 64 if *Fd* is a D register.

Fd is the extension register to be loaded or saved. For an Advanced SIMD instruction, it must be a doubleword (*Dd*) register. For a floating-point instruction, it can be either a double precision (*Dd*) or a single precision (*Sd*) register.

Rn is the ARM register holding the base address for the transfer.

offset is a numeric expression that must evaluate to a numeric value at assembly time. The value must be 4 if *Fd* is an S register, or 8 if *Fd* is a D register.

4.54.2 Usage

The post-increment instruction increments the base address in the register by the offset value, after the transfer. The pre-decrement instruction decrements the base address in the register by the offset value, and then performs the transfer using the new address in the register. These pseudo-instructions assemble to VLDM or VSTM instructions.

4.54.3 See also

Reference

- *VLDR and VSTR* on page 4-55.
- *VLDM, VSTM, VPOP, and VPUSH* on page 4-54.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.55 VMAX, VMIN, VPMAX, and VPMIN

VMAX (Vector Maximum) compares corresponding elements in two vectors, and copies the larger of each pair into the corresponding element in the destination vector.

VMIN (Vector Minimum) compares corresponding elements in two vectors, and copies the smaller of each pair into the corresponding element in the destination vector.

VPMAX (Vector Pairwise Maximum) compares adjacent pairs of elements in two vectors, and copies the larger of each pair into the corresponding element in the destination vector. Operands and results must be doubleword vectors.

VPMIN (Vector Pairwise Minimum) compares adjacent pairs of elements in two vectors, and copies the smaller of each pair into the corresponding element in the destination vector. Operands and results must be doubleword vectors.

4.55.1 Syntax

V*op*{*cond*}.*datatype Qd*, *Qn*, *Qm*

V*op*{*cond*}.*datatype Dd*, *Dn*, *Dm*

VP*op*{*cond*}.*datatype Dd*, *Dn*, *Dm*

where:

op must be either MAX or MIN.

cond is an optional condition code.

datatype must be one of S8, S16, S32, U8, U16, U32, or F32.

Qd, *Qn*, *Qm* are the destination vector, the first operand vector, and the second operand vector, for a quadword operation.

Dd, *Dn*, *Dm* are the destination vector, the first operand vector, and the second operand vector, for a doubleword operation.

4.55.2 Floating-point maximum and minimum

max(+0.0, –0.0) = +0.0

min(+0.0, –0.0) = –0.0

If any input is a NaN, the corresponding result element is the default NaN.

4.55.3 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.
- *VPADD{L}, VPADAL* on page 4-79.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.56 VMAXNM, VMINNM (Advanced SIMD)

VMAXNM (Vector Maximum) compares corresponding elements in two vectors, and copies the larger of each pair into the corresponding element in the destination vector.

VMINNM (Vector Minimum) compares corresponding elements in two vectors, and copies the smaller of each pair into the corresponding element in the destination vector.

If one of the elements in a pair is a number and the other element is NaN, the corresponding result element is the number. This is consistent with the IEEE 754-2008 standard.

Note

These instructions are supported only in ARMv8.

4.56.1 Syntax

V*op*.F32 *Qd*, *Qn*, *Qm*

V*op*.F32 *Dd*, *Dn*, *Dm*

where:

op must be either MAXNM or MINNM.

Qd, *Qn*, *Qm* are the destination vector, the first operand vector, and the second operand vector, for a quadword operation.

Dd, *Dn*, *Dm* are the destination vector, the first operand vector, and the second operand vector, for a doubleword operation.

4.56.2 Notes

You cannot use VMAXNM or VMINNM inside an IT block.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.57 VMAXNM, VMINNM (floating-point)

VMAXNM compares the values in the operand registers, and copies the larger value into the destination operand register.

VMINNM compares the values in the operand registers, and copies the smaller value into the destination operand register.

If one of the values being compared is a number and the other value is NaN, the number is copied into the destination operand register. This is consistent with the IEEE 754-2008 standard.

Note

These instructions are supported only in ARMv8.

4.57.1 Syntax

V*op*.F32 *Sd*, *Sn*, *Sm*

V*op*.F64 *Dd*, *Dn*, *Dm*

where:

op must be either MAXNM or MINNM.

Sd, *Sn*, *Sm* are the single-precision destination register, first operand register, and second operand register.

Dd, *Dn*, *Dm* are the double-precision destination register, first operand register, and second operand register.

4.57.2 Notes

You cannot use VMAXNM or VMINNM inside an IT block.

4.57.3 Floating-point exceptions

These instructions can produce Input Denormal, Invalid Operation, Overflow, Underflow, or Inexact exceptions.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.58 VMOV

Insert a floating-point immediate value in a single-precision or double-precision register, or copy one register into another register.

4.58.1 Syntax

```
VMOV{cond}.F32 Sd, #imm
VMOV{cond}.F64 Dd, #imm
VMOV{cond}.F32 Sd, Sm
VMOV{cond}.F64 Dd, Dm
```

where:

cond is an optional condition code.

Sd is the single-precision destination register.

Dd is the double-precision destination register.

imm is the floating-point immediate value.

Sm is the single-precision source register.

Dm is the double-precision source register.

4.58.2 Immediate values

Any number that can be expressed as +/–*n* * 2^{-r},where *n* and *r* are integers, 16 <= *n* <= 31, 0 <= r <= 7.

4.58.3 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.59 VMOV, VMVN (immediate)

VMOV (Vector Move) and VMVN (Vector Move Negative) immediate generate an immediate value into the destination register.

4.59.1 Syntax

V*op*{*cond*}.*datatype Qd*, #*imm*

V*op*{*cond*}.*datatype Dd*, #*imm*

where:

op must be either MOV or MVN.

cond is an optional condition code.

datatype must be one of I8, I16, I32, I64, or F32.

Qd or *Dd* is the Advanced SIMD register for the result.

imm is an immediate value of the type specified by *datatype*. This is replicated to fill the destination register.

Table 4-8 Available immediate values

datatype	VMOV	VMVN
I8	0xXY	-
I16	0x00XY, 0xXY00	0xFFXY, 0xXYFF
I32	0x000000XY, 0x0000XY00, 0x00XY0000, 0xXY000000	0xFFFFFFXY, 0xFFFFXYFF, 0xFFXYFFFF, 0xXYFFFFFF
	0x0000XYFF, 0x00XYFFFF	0xFFFFXY00, 0xFFXY0000
I64	byte masks, 0xGGHHJJKKLLMMNNPP [a]	-
F32	floating-point numbers [b]	-

a. Each of 0xGG, 0xHH, 0xJJ, 0xKK, 0xLL, 0xMM, 0xNN, and 0xPP must be either 0x00 or 0xFF.

b. Any number that can be expressed as +/–n * 2^{-r}, where n and r are integers, 16 <= n <= 31, 0 <= r <= 7.

4.59.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.60 VMOV, VMVN (register)

Vector Move (register) copies a value from the source register into the destination register.

Vector Move Not (register) inverts the value of each bit from the source register and places the results into the destination register.

4.60.1 Syntax

```
VMOV{cond}{.datatype} Qd, Qm
VMOV{cond}{.datatype} Dd, Dm
VMVN{cond}{.datatype} Qd, Qm
VMVN{cond}{.datatype} Dd, Dm
```

where:

cond is an optional condition code.

datatype is an optional datatype. The assembler ignores *datatype*.

Qd, Qm specifies the destination vector and the source vector, for a quadword operation.

Dd, Dm specifies the destination vector and the source vector, for a doubleword operation.

4.60.2 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.61 VMOV (between two ARM registers and one or two extension registers)

Transfer contents between two ARM registers and a 64-bit extension register, or two consecutive 32-bit floating-point registers.

4.61.1 Syntax

`VMOV{cond} Dm, Rd, Rn`

`VMOV{cond} Rd, Rn, Dm`

`VMOV{cond} Sm, Sm1, Rd, Rn`

`VMOV{cond} Rd, Rn, Sm, Sm1`

where:

cond is an optional condition code.

Dm is a 64-bit extension register.

Sm is a floating-point 32-bit register.

Sm1 is the next consecutive floating-point 32-bit register after *Sm*.

Rd, *Rn* are the ARM registers. R*d* and R*n* must not be PC.

4.61.2 Usage

`VMOV Dm, Rd, Rn` transfers the contents of *Rd* into the low half of *Dm*, and the contents of *Rn* into the high half of *Dm*.

`VMOV Rd, Rn, Dm` transfers the contents of the low half of *Dm* into *Rd*, and the contents of the high half of *Dm* into *Rn*.

`VMOV Rd, Rn, Sm, Sm1` transfers the contents of *Sm* into *Rd*, and the contents of *Sm1* into *Rn*.

`VMOV Sm, Sm1, Rd, Rn` transfers the contents of *Rd* into *Sm*, and the contents of *Rn* into *Sm1*.

4.61.3 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.62 VMOV (between an ARM register and a scalar)

Transfer contents between an ARM register and a scalar.

4.62.1 Syntax

`VMOV{cond}{.size} Dn[x], Rd`

`VMOV{cond}{.datatype} Rd, Dn[x]`

where:

cond is an optional condition code.

size the data size. Can be `8`, `16`, or `32`. If omitted, *size* is `32`. For floating-point instructions, *size* must be `32` or omitted.

datatype the data type. Can be `U8`, `S8`, `U16`, `S16`, or `32`. If omitted, *datatype* is `32`. For floating-point instructions, *datatype* must be `32` or omitted.

Dn[x] can be:

- An Advanced SIMD scalar.
- The top half or the bottom half of a double precision floating point value.

Rd is the ARM register. *Rd* must not be PC.

4.62.2 Usage

`VMOV Rd, Dn[x]` transfers the contents of *Dn*`[x]` into the least significant byte, halfword, or word of *Rd*. The remaining bits of *Rd* are either zero or sign extended.

`VMOV Dn[x], Rd` transfers the contents of the least significant byte, halfword, or word of *Rd* into *Dn*`[x]`.

4.62.3 See also

Concepts

armasm User Guide:

- *Advanced SIMD scalars* on page 11-24.
- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.63 VMOV (between one ARM register and single precision floating-point register)

Transfer contents between a single-precision floating-point register and an ARM register.

4.63.1 Syntax

```
VMOV{cond} Rd, Sn
VMOV{cond} Sn, Rd
```

where:

cond is an optional condition code.

Sn is the floating-point single-precision register.

Rd is the ARM register. *Rd* must not be PC.

4.63.2 Usage

`VMOV Rd, Sn` transfers the contents of *Sn* into *Rd*.

`VMOV Sn, Rd` transfers the contents of *Rd* into *Sn*.

4.63.3 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.64 VMOV2

The VMOV2 pseudo-instruction generates an immediate value and places it in every element of an Advanced SIMD vector, without loading a value from a literal pool. It always assembles to exactly two instructions.

VMOV2 can generate any 16-bit immediate value, and a restricted range of 32-bit and 64-bit immediate values.

4.64.1 Syntax

`VMOV2{cond}.datatype Qd, #constant`

`VMOV2{cond}.datatype Dd, #constant`

where:

datatype must be one of:
- I8, I16, I32, or I64.
- S8, S16, S32, or S64.
- U8, U16, U32, or U64.
- F32.

cond is an optional condition code.

Qd or *Dd* is the extension register to be loaded.

constant is an immediate value of the appropriate type for *datatype*.

4.64.2 Usage

VMOV2 typically assembles to a VMOV or VMVN instruction, followed by a VBIC or VORR instruction.

4.64.3 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *VMOV, VMVN (immediate)* on page 4-68.
- *VBIC and VORR (immediate)* on page 4-35.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.65 VMOVL, V{Q}MOVN, VQMOVUN

VMOVL (Vector Move Long) takes each element in a doubleword vector, sign or zero extends them to twice their original length, and places the results in a quadword vector.

VMOVN (Vector Move and Narrow) copies the least significant half of each element of a quadword vector into the corresponding elements of a doubleword vector.

VQMOVN (Vector Saturating Move and Narrow) copies each element of the operand vector to the corresponding element of the destination vector. The result element is half the width of the operand element, and values are saturated to the result width.

VQMOVUN (Vector Saturating Move and Narrow, signed operand with Unsigned result) copies each element of the operand vector to the corresponding element of the destination vector. The result element is half the width of the operand element, and values are saturated to the result width.

4.65.1 Syntax

VMOVL{*cond*}.*datatype Qd*, *Dm*

V{Q}MOVN{*cond*}.*datatype Dd*, *Qm*

VQMOVUN{*cond*}.*datatype Dd*, *Qm*

where:

Q
: if present, specifies that the results are saturated.

cond
: is an optional condition code.

datatype
: must be one of:

S8, S16, S32	for VMOVL
U8, U16, U62	for VMOVL
I16, I32, I64	for VMOVN
S16, S32, S64	for VQMOVN or VQMOVUN
U16, U32, U64	for VQMOVN.

Qd, Dm
: specifies the destination vector and the operand vector for VMOVL.

Dd, Qm
: specifies the destination vector and the operand vector for V{Q}MOV{U}N.

4.65.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.66 VMRS and VMSR

Transfer contents between an ARM register and an Advanced SIMD and floating-point system register.

4.66.1 Syntax

`VMRS{cond} Rd, extsysreg`

`VMSR{cond} extsysreg, Rd`

where:

cond is an optional condition code.

extsysreg is the Advanced SIMD and floating-point system register, usually `FPSCR`, `FPSID`, or `FPEXC`.

Rd is the ARM register. *Rd* must not be PC.

It can be `APSR_nzcv`, if *extsysreg* is `FPSCR`. In this case, the floating-point status flags are transferred into the corresponding flags in the ARM APSR.

4.66.2 Usage

The `VMRS` instruction transfers the contents of *extsysreg* into *Rd*.

The `VMSR` instruction transfers the contents of *Rd* into *extsysreg*.

Note

These instructions stall the processor until all current Advanced SIMD or floating-point operations complete.

4.66.3 Examples

```
VMRS    r2,FPCID
VMRS    APSR_nzcv, FPSCR     ; transfer FP status register to ARM APSR
VMSR    FPSCR, r4
```

4.66.4 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point system registers in AArch32 state* on page 11-27.

Reference

- *Condition codes* on page 3-26.

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.67 VMUL, VMLA, VMLS, VNMUL, VNMLA, and VNMLS

Floating-point multiply and multiply accumulate, with optional negation.

4.67.1 Syntax

```
V{N}MUL{cond}.F32 {Sd,} Sn, Sm
V{N}MUL{cond}.F64 {Dd,} Dn, Dm
V{N}MLA{cond}.F32 Sd, Sn, Sm
V{N}MLA{cond}.F64 Dd, Dn, Dm
V{N}MLS{cond}.F32 Sd, Sn, Sm
V{N}MLS{cond}.F64 Dd, Dn, Dm
```

where:

`N` negates the final result.

cond is an optional condition code.

Sd, *Sn*, *Sm* are the single-precision registers for the result and operands.

Dd, *Dn*, *Dm* are the double-precision registers for the result and operands.

4.67.2 Usage

The `VMUL` operation multiplies the values in the operand registers and places the result in the destination register.

The `VMLA` operation multiplies the values in the operand registers, adds the value in the destination register, and places the final result in the destination register.

The `VMLS` operation multiplies the values in the operand registers, subtracts the result from the value in the destination register, and places the final result in the destination register.

In each case, the final result is negated if the `N` option is used.

4.67.3 Floating-point exceptions

These instructions can produce Invalid Operation, Overflow, Underflow, Inexact, or Input Denormal exceptions.

4.67.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.68 VMUL{L}, VMLA{L}, and VMLS{L}

VMUL (Vector Multiply) multiplies corresponding elements in two vectors, and places the results in the destination vector.

VMLA (Vector Multiply Accumulate) multiplies corresponding elements in two vectors, and accumulates the results into the elements of the destination vector.

VMLS (Vector Multiply Subtract) multiplies corresponding elements in two vectors, subtracts the results from corresponding elements of the destination vector, and places the final results in the destination vector.

4.68.1 Syntax

`V`*op*{*cond*}`.`*datatype* {*Qd*}, *Qn*, *Qm*

`V`*op*{*cond*}`.`*datatype* {*Dd*}, *Dn*, *Dm*

`V`*op*`L`{*cond*}`.`*datatype* *Qd*, *Dn*, *Dm*

where:

op must be one of:

MUL	Multiply
MLA	Multiply Accumulate
MLS	Multiply Subtract.

cond is an optional condition code.

datatype must be one of:

I8, I16, I32, F32	for VMUL, VMLA, or VMLS
S8, S16, S32	for VMULL, VMLAL, or VMLSL
U8, U16, U32	for VMULL, VMLAL, or VMLSL
P8	for VMUL or VMULL.

Qd, *Qn*, *Qm* are the destination vector, the first operand vector, and the second operand vector, for a quadword operation.

Dd, *Dn*, *Dm* are the destination vector, the first operand vector, and the second operand vector, for a doubleword operation.

Qd, *Dn*, *Dm* are the destination vector, the first operand vector, and the second operand vector, for a long operation.

4.68.2 See also

Concepts

armasm User Guide:

- *Polynomial arithmetic over {0,1}* on page 11-26.
- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.69 VMUL{L}, VMLA{L}, and VMLS{L} (by scalar)

VMUL (Vector Multiply by scalar) multiplies each element in a vector by a scalar, and places the results in the destination vector.

VMLA (Vector Multiply Accumulate) multiplies each element in a vector by a scalar, and accumulates the results into the corresponding elements of the destination vector.

VMLS (Vector Multiply Subtract) multiplies each element in a vector by a scalar, and subtracts the results from the corresponding elements of the destination vector, and places the final results in the destination vector.

4.69.1 Syntax

`V`*op*{*cond*}`.`*datatype* {*Qd*}, *Qn*, *Dm*[*x*]

`V`*op*{*cond*}`.`*datatype* {*Dd*}, *Dn*, *Dm*[*x*]

`V`*op*`L`{*cond*}`.`*datatype* *Qd*, *Dn*, *Dm*[*x*]

where:

op
: must be one of:

  `MUL` Multiply

  `MLA` Multiply Accumulate

  `MLS` Multiply Subtract.

cond
: is an optional condition code.

datatype
: must be one of:

  `I16`, `I32`, `F32` for `VMUL`, `VMLA`, or `VMLS`

  `S16`, `S32` for `VMULL`, `VMLAL`, or `VMLSL`

  `U16`, `U32` for `VMULL`, `VMLAL`, or `VMLSL`.

Qd, *Qn*
: are the destination vector and the first operand vector, for a quadword operation.

Dd, *Dn*
: are the destination vector and the first operand vector, for a doubleword operation.

Qd, *Dn*
: are the destination vector and the first operand vector, for a long operation.

Dm[*x*]
: is the scalar holding the second operand.

4.69.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.70 VPADD{L}, VPADAL

VPADD (Vector Pairwise Add) adds adjacent pairs of elements of two vectors, and places the results in the destination vector.

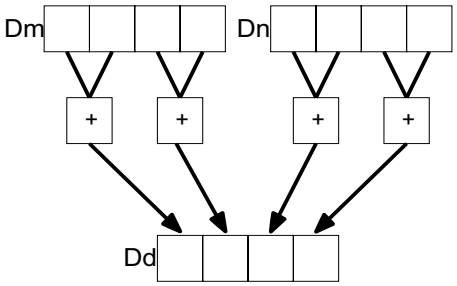


Figure 4-3 Example of operation of VPADD (in this case, for data type I16)

VPADDL (Vector Pairwise Add Long) adds adjacent pairs of elements of a vector, sign or zero extends the results to twice their original width, and places the final results in the destination vector.

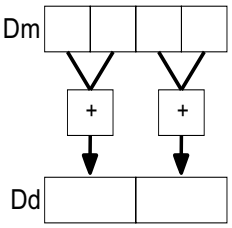


Figure 4-4 Example of operation of doubleword VPADDL (in this case, for data type S16)

VPADAL (Vector Pairwise Add and Accumulate Long) adds adjacent pairs of elements of a vector, and accumulates the absolute values of the results into the elements of the destination vector.

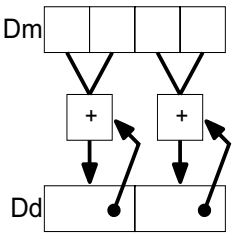


Figure 4-5 Example of operation of VPADAL (in this case for data type S16)

4.70.1 Syntax

```
VPADD{cond}.datatype {Dd}, Dn, Dm
VPopL{cond}.datatype Qd, Qm
VPopL{cond}.datatype Dd, Dm
```

where:

op must be either ADD or ADA.

cond is an optional condition code.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

datatype must be one of:

I8, I16, I32, F32	for VPADD
S8, S16, S32	for VPADDL or VPADAL
U8, U16, U32	for VPADDL or VPADAL.

Dd, *Dn*, *Dm* are the destination vector, the first operand vector, and the second operand vector, for a VPADD instruction.

Qd, *Qm* are the destination vector and the operand vector, for a quadword VPADDL or VPADAL.

Dd, *Dm* are the destination vector and the operand vector, for a doubleword VPADDL or VPADAL.

4.70.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.71 VQ{R}DMULH (by vector or by scalar)

Vector Saturating Doubling Multiply instructions multiply their operands and double the results. They return only the high half of the results.

If any of the results overflow, they are saturated. The sticky QC flag in the FPSCR is set if saturation occurs.

4.71.1 Syntax

`VQ{R}DMULH{cond}.datatype {Qd}, Qn, Qm`

`VQ{R}DMULH{cond}.datatype {Dd}, Dn, Dm`

`VQ{R}DMULH{cond}.datatype {Qd}, Qn, Dm[x]`

`VQ{R}DMULH{cond}.datatype {Dd}, Dn, Dm[x]`

where:

`R` if present, indicates that each result is rounded. Otherwise, each result is truncated.

cond is an optional condition code.

datatype must be either `S16` or `S32`.

Qd, *Qn* are the destination vector and the first operand vector, for a quadword operation.

Dd, *Dn* are the destination vector and the first operand vector, for a doubleword operation.

Qm or *Dm* is the vector holding the second operand, for a *by vector* operation.

Dm[*x*] is the scalar holding the second operand, for a *by scalar* operation.

4.71.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.72 VQ{R}SHR{U}N (by immediate)

VQ{R}SHR{U}N (Vector Saturating Shift Right, Narrow, by immediate value, with optional Rounding) takes each element in a quadword vector of integers, right shifts them by an immediate value, and places the results in a doubleword vector.

The sticky QC flag in the FPSCR is set if saturation occurs.

4.72.1 Syntax

VQ{R}SHR{U}N{*cond*}.*datatype* *Dd*, *Qm*, #*imm*

where:

R — if present, indicates that the results are rounded. Otherwise, the results are truncated.

U — if present, indicates that the results are unsigned, although the operands are signed. Otherwise, the results are the same type as the operands.

cond — is an optional condition code.

datatype — must be one of:

- I16, I32, I64 — for VQ{R}SHRN or VQ{R}SHRUN. Only a #0 immediate is permitted with these datatypes.
- S16, S32, S64 — for VQ{R}SHRN or VQ{R}SHRUN
- U16, U32, U64 — for VQ{R}SHRN only.

Dd, *Qm* — are the destination vector and the operand vector.

imm — is the immediate value specifying the size of the shift. The ranges are shown in Table 4-9.

Table 4-9

datatype	imm range
S16 or U16	0 to 8
S32 or U32	0 to 16
S64 or U64	0 to 32

4.72.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.73 VQDMULL, VQDMLAL, and VQDMLSL (by vector or by scalar)

Vector Saturating Doubling Multiply instructions multiply their operands and double the results. VQDMULL places the results in the destination register. VQDMLAL adds the results to the values in the destination register. VQDMLSL subtracts the results from the values in the destination register.

If any of the results overflow, they are saturated. The sticky QC flag in the FPSCR is set if saturation occurs.

4.73.1 Syntax

VQD*op*L{*cond*}.*datatype Qd*, *Dn*, *Dm*

VQD*op*L{*cond*}.*datatype Qd*, *Dn*, *Dm*[*x*]

where:

op must be one of:

MUL	Multiply
MLA	Multiply Accumulate
MLS	Multiply Subtract.

cond is an optional condition code.

datatype must be either S16 or S32.

Qd, *Dn* are the destination vector and the first operand vector.

Dm is the vector holding the second operand, for a *by vector* operation.

Dm[*x*] is the scalar holding the second operand, for a *by scalar* operation.

4.73.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.74 VRECPE and VRSQRTE

VRECPE (Vector Reciprocal Estimate) finds an approximate reciprocal of each element in a vector, and places the results in a second vector.

VRSQRTE (Vector Reciprocal Square Root Estimate) finds an approximate reciprocal square root of each element in a vector, and places the results in a second vector.

4.74.1 Syntax

`V`*op*{*cond*}`.`*datatype* *Qd*, *Qm*

`V`*op*{*cond*}`.`*datatype* *Dd*, *Dm*

where:

op must be either RECPE or RSQRTE.

cond is an optional condition code.

datatype must be either U32 or F32.

Qd, *Qm* are the destination vector and the operand vector, for a quadword operation.

Dd, *Dm* are the destination vector and the operand vector, for a doubleword operation.

4.74.2 Results for out-of-range inputs

Table 4-10 shows the results where input values are out of range.

Table 4-10 Results for out-of-range inputs

	Operand element (VRECPE)	Operand element (VRSQRTE)	Result element
Integer	<= 0x7FFFFFFF	<= 0x3FFFFFFF	0xFFFFFFFF
Floating-point	NaN	NaN, Negative Normal, Negative Infinity	Default NaN
	Negative 0, Negative Denormal	Negative 0, Negative Denormal	Negative Infinity [a]
	Positive 0, Positive Denormal	Positive 0, Positive Denormal	Positive Infinity [a]
	Positive infinity	Positive infinity	Positive 0
	Negative infinity		Negative 0

a. The Division by Zero exception bit in the FPSCR is set.

4.74.3 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.75 VRECPS and VRSQRTS

VRECPS (Vector Reciprocal Step) multiplies the elements of one vector by the corresponding elements of another vector, subtracts each of the results from 2, and places the final results into the elements of the destination vector.

VRSQRTS (Vector Reciprocal Square Root Step) multiplies the elements of one vector by the corresponding elements of another vector, subtracts each of the results from 3, divides these results by two, and places the final results into the elements of the destination vector.

4.75.1 Syntax

V*op*{*cond*}.F32 {*Qd*}, *Qn*, *Qm*

V*op*{*cond*}.F32 {*Dd*}, *Dn*, *Dm*

where:

op must be either RECPS or RSQRTS.

cond is an optional condition code.

Qd, *Qn*, *Qm* are the destination vector, the first operand vector, and the second operand vector, for a quadword operation.

Dd, *Dn*, *Dm* are the destination vector, the first operand vector, and the second operand vector, for a doubleword operation.

4.75.2 Results for out-of-range inputs

Table 4-11 shows the results where input values are out of range.

Table 4-11 Results for out-of-range inputs

1st operand element	2nd operand element	Result element (VRECPS)	Result element (VRSQRTS)
NaN	-	Default NaN	Default NaN
-	NaN	Default NaN	Default NaN
+/– 0.0 or denormal	+/– infinity	2.0	1.5
+/– infinity	+/– 0.0 or denormal	2.0	1.5

4.75.3 Usage

The Newton-Raphson iteration:

$x_{n+1} = x_n(2-dx_n)$

converges to $(1/d)$ if x_0 is the result of VRECPE applied to d.

The Newton-Raphson iteration:

$x_{n+1} = x_n(3-dx_n^2)/2$

converges to $(1/\sqrt{d})$if x_0 is the result of VRSQRTE applied to d.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.75.4 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.76 VREV

VREV16 (Vector Reverse within halfwords) reverses the order of 8-bit elements within each halfword of the vector, and places the result in the corresponding destination vector.

VREV32 (Vector Reverse within words) reverses the order of 8-bit or 16-bit elements within each word of the vector, and places the result in the corresponding destination vector.

VREV64 (Vector Reverse within doublewords) reverses the order of 8-bit, 16-bit, or 32-bit elements within each doubleword of the vector, and places the result in the corresponding destination vector.

4.76.1 Syntax

VREV*n*{*cond*}.*size Qd*, *Qm*

VREV*n*{*cond*}.*size Dd*, *Dm*

where:

n must be one of 16, 32, or 64.

cond is an optional condition code.

size must be one of 8, 16, or 32, and must be less than *n*.

Qd, Qm specifies the destination vector and the operand vector, for a quadword operation.

Dd, Dm specifies the destination vector and the operand vector, for a doubleword operation.

4.76.2 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.77 VRINT (Advanced SIMD)

VRINT (Vector Round to Integer) rounds each floating-point element in a vector to integer, and places the results in the destination vector.

The resulting integers are represented in floating-point format.

Note

These instructions are supported only in ARMv8.

4.77.1 Syntax

VRINT*mode*.F32.F32 *Qd*, *Qm*

VRINT*mode*.F32.F32 *Dd*, *Dm*

where:

mode must be one of:

- A meaning round to nearest, ties away from zero. This cannot generate an Inexact exception, even if the result is not exact.
- N meaning round to nearest, ties to even. This cannot generate an Inexact exception, even if the result is not exact.
- X meaning round to nearest, ties to even, generating an Inexact exception if the result is not exact.
- P meaning round towards plus infinity. This cannot generate an Inexact exception, even if the result is not exact.
- M meaning round towards minus infinity. This cannot generate an Inexact exception, even if the result is not exact.
- Z meaning round towards zero. This cannot generate an Inexact exception, even if the result is not exact.

Qd, Qm specifies the destination vector and the operand vector, for a quadword operation.

Dd, Dm specifies the destination and operand vectors, for a doubleword operation.

4.77.2 Notes

You cannot use VRINT inside an IT block.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.78 VRINT (floating-point)

Rounds a floating-point number to integer and places the result in the destination register. The resulting integer is represented in floating-point format.

Note

These instructions are supported only in ARMv8.

4.78.1 Syntax

`VRINT`*mode*`{`*cond*`}.F64.F64` *Dd*, *Dm*

`VRINT`*mode*`{`*cond*`}.F32.F32` *Sd*, *Sm*

where:

mode must be one of:

- `Z` meaning round towards zero.
- `R` meaning use the rounding mode specified in the FPSCR.
- `X` meaning use the rounding mode specified in the FPSCR, generating an Inexact exception if the result is not exact.
- `A` meaning round to nearest, ties away from zero.
- `N` meaning round to nearest, ties to even.
- `P` meaning round towards plus infinity.
- `M` meaning round towards minus infinity.

cond is an optional condition code. This can only be used when *mode* is `Z`, `R` or `X`.

Sd, *Sm* specifies the destination and operand registers, for a word operation.

Dd, *Dm* specifies the destination and operand registers, for a doubleword operation.

4.78.2 Notes

You cannot use `VRINT` with a rounding mode of `A`, `N`, `P` or `M` inside an IT block.

4.78.3 Floating-point exceptions

These instructions cannot produce any exceptions, except `VRINTX` which can generate an Inexact exception.

4.78.4 See also

Reference

- *Condition codes* on page 3-26.

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.79 VSEL

Floating-point select.

Note

This instruction is supported only in ARMv8.

4.79.1 Syntax

`VSEL`*`cond`*`.F32` *`Sd`*, *`Sn`*, *`Sm`*

`VSEL`*`cond`*`.F64` *`Dd`*, *`Dn`*, *`Dm`*

where:

`cond` must be one of `GE`, `GT`, `EQ`, `VS`.

`Sd`, *`Sn`*, *`Sm`* are the single-precision registers for the result and operands.

`Dd`, *`Dn`*, *`Dm`* are the double-precision registers for the result and operands.

4.79.2 Usage

The `VSEL` instruction compares the values in the operand registers. If the condition is true, it copies the value in the first operand register into the destination operand register. Otherwise, it copies the value in the second operand register.

You cannot use `VSEL` inside an IT block.

4.79.3 Floating-point exceptions

`VSEL` instructions cannot produce any exceptions.

4.79.4 See also

Concepts

armasm User Guide:

- *Comparison of condition code meanings in integer and floating-point code* on page 8-14.

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.80 VSHL, VQSHL, VQSHLU, and VSHLL (by immediate)

Vector Shift Left (by immediate) instructions take each element in a vector of integers, left shift them by an immediate value, and place the results in the destination vector.

For VSHL (Vector Shift Left), bits shifted out of the left of each element are lost.

For VQSHL (Vector Saturating Shift Left) and VQSHLU (Vector Saturating Shift Left Unsigned), the sticky QC flag in the FPSCR is set if saturation occurs.

For VSHLL (Vector Shift Left Long), values are sign or zero extended.

Figure 4-6 shows the operation of VSHL with two elements and a shift value of one. The least significant bit in each element in the destination vector is set to zero.

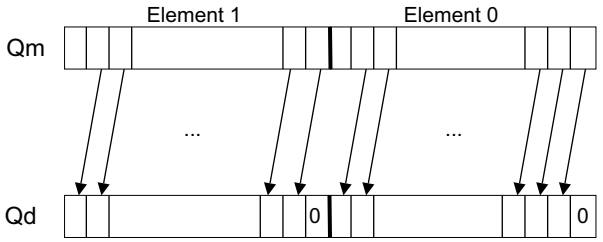


Figure 4-6 Operation of quadword VSHL.64 Qd, Qm, #1

4.80.1 Syntax

V{Q}SHL{U}{*cond*}.*datatype* {*Qd*}, *Qm*, *#imm*

V{Q}SHL{U}{*cond*}.*datatype* {*Dd*}, *Dm*, *#imm*

VSHLL{*cond*}.*datatype Qd*, *Dm*, *#imm*

where:

Q — if present, indicates that if any of the results overflow, they are saturated.

U — only permitted if Q is also present. Indicates that the results are unsigned even though the operands are signed.

cond — is an optional condition code.

datatype — must be one of:

I8, I16, I32, I64	for VSHL
S8, S16, S32	for VSHLL, VQSHL, or VQSHLU
U8, U16, U32	for VSHLL or VQSHL
S64	for VQSHL or VQSHLU
U64	for VQSHL.

Qd, Qm — are the destination and operand vectors, for a quadword operation.

Dd, Dm — are the destination and operand vectors, for a doubleword operation.

Qd, Dm — are the destination and operand vectors, for a long operation.

imm — is the immediate value specifying the size of the shift, in the range:

- 1 to (size in bits of *datatype*) for VSHLL.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

- 0 to (size in bits of *datatype* – 1) for VSHL, VQSHL, or VQSHLU.

For VSHLL, 0 is permitted, but the resulting code disassembles to VMOVL.

4.80.2 See also

Concepts

armasm User Guide:

- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Reference

- *Condition codes* on page 3-26.

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.81 VSLI and VSRI

VSLI (Vector Shift Left and Insert) takes each element in a vector, left shifts them by an immediate value, and inserts the results in the destination vector. Bits shifted out of the left of each element are lost. Figure 4-7 shows the operation of VSLI with two elements and a shift value of one. The least significant bit in each element in the destination vector is unchanged.

VSRI (Vector Shift Right and Insert) takes each element in a vector, right shifts them by an immediate value, and inserts the results in the destination vector. Bits shifted out of the right of each element are lost. Figure 4-8 shows the operation of VSRI with a single element and a shift value of two. The two most significant bits in the destination vector are unchanged.

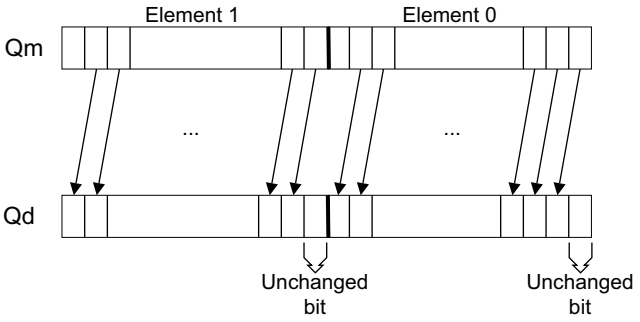


Figure 4-7 Operation of quadword VSLI.64 Qd, Qm, #1

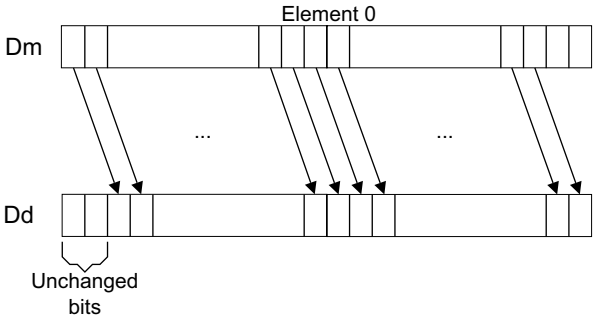


Figure 4-8 Operation of doubleword VSRI.64 Dd, Dm, #2

4.81.1 Syntax

`V`*op*{*cond*}`.`*size* {*Qd*}, *Qm*, *#imm*

`V`*op*{*cond*}`.`*size* {*Dd*}, *Dm*, *#imm*

where:

op must be either SLI or SRI.

cond is an optional condition code.

size must be one of 8, 16, 32, or 64.

Qd, *Qm* are the destination vector and the operand vector, for a quadword operation.

Dd, *Dm* are the destination vector and the operand vector, for a doubleword operation.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

imm is the immediate value specifying the size of the shift, in the range:

- 0 to (*size* – 1) for `VSLI`.
- 1 to *size* for `VSRI`.

4.81.2 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.82 VSWP

VSWP (Vector Swap) exchanges the contents of two vectors. The vectors can be either doubleword or quadword. There is no distinction between data types.

4.82.1 Syntax

VSWP{*cond*}{*.datatype*} *Qd*, *Qm*

VSWP{*cond*}{*.datatype*} *Dd*, *Dm*

where:

cond is an optional condition code.

datatype is an optional datatype. The assembler ignores *datatype*.

Qd, Qm specifies the vectors for a quadword operation.

Dd, Dm specifies the vectors for a doubleword operation.

4.82.2 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.83 VTBL, VTBX

VTBL (Vector Table Lookup) uses byte indexes in a control vector to look up byte values in a table and generate a new vector. Indexes out of range return 0.

VTBX (Vector Table Extension) works in the same way, except that indexes out of range leave the destination element unchanged.

4.83.1 Syntax

`V`*op*{*cond*}`.8` *Dd*, *list*, *Dm*

where:

op must be either TBL or TBX.

cond is an optional condition code.

Dd specifies the destination vector.

list Specifies the vectors containing the table. It must be one of:

- {*Dn*}.
- {*Dn*,*D(n+1)*}.
- {*Dn*,*D(n+1)*,*D(n+2)*}.
- {*Dn*,*D(n+1)*,*D(n+2)*,*D(n+3)*}.
- {*Qn*,*Q(n+1)*}.

All the registers in *list* must be in the range D0-D31 or Q0-Q15 and must not wraparound the end of the register bank. For example {D31,D0,D1} is not permitted. If *list* contains Q registers, they disassemble to the equivalent D registers.

Dm specifies the index vector.

4.83.2 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.84 VTRN

VTRN (Vector Transpose) treats the elements of its operand vectors as elements of 2 x 2 matrices, and transposes the matrices. Figure 4-9 and Figure 4-10 show examples of the operation of VTRN.

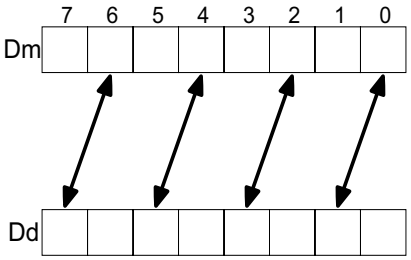


Figure 4-9 Operation of doubleword VTRN.8

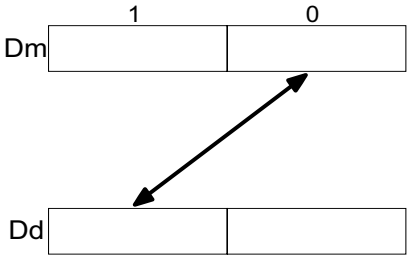


Figure 4-10 Operation of doubleword VTRN.32

4.84.1 Syntax

VTRN{*cond*}.*size Qd*, *Qm*

VTRN{*cond*}.*size Dd*, *Dm*

where:

cond is an optional condition code.

size must be one of 8, 16, or 32.

Qd, Qm specifies the vectors, for a quadword operation.

Dd, Dm specifies the vectors, for a doubleword operation.

4.84.2 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.85 VTST

VTST (Vector Test Bits) takes each element in a vector, and bitwise logical ANDs them with the corresponding element of a second vector. If the result is not zero, the corresponding element in the destination vector is set to all ones. Otherwise, it is set to all zeros.

4.85.1 Syntax

VTST{*cond*}.*size* {*Qd*}, *Qn*, *Qm*

VTST{*cond*}.*size* {*Dd*}, *Dn*, *Dm*

where:

cond is an optional condition code.

size must be one of 8, 16, or 32.

Qd, Qn, Qm specifies the destination register, the first operand register, and the second operand register, for a quadword operation.

Dd, Dn, Dm specifies the destination register, the first operand register, and the second operand register, for a doubleword operation.

4.85.2 See also

Reference

- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

4.86 VUZP, VZIP

VZIP (Vector Zip) interleaves the elements of two vectors.

VUZP (Vector Unzip) de-interleaves the elements of two vectors.

De-interleaving is the inverse process of interleaving.

Table 4-12 Operation of doubleword VZIP.8

	Register state before operation								Register state after operation							
Dd	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_3	A_3	B_2	A_2	B_1	A_1	B_0	A_0
Dm	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	B_7	A_7	B_6	A_6	B_5	A_5	B_4	A_4

Table 4-13 Operation of quadword VZIP.32

	Register state before operation				Register state after operation			
Qd	A_3	A_2	A_1	A_0	B_1	A_1	B_0	A_0
Qm	B_3	B_2	B_1	B_0	B_3	A_3	B_2	A_2

Table 4-14 Operation of doubleword VUZP.8

	Register state before operation								Register state after operation							
Dd	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	B_6	B_4	B_2	B_0	A_6	A_4	A_2	A_0
Dm	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	B_7	B_5	B_3	B_1	A_7	A_5	A_3	A_1

Table 4-15 Operation of quadword VUZP.32

	Register state before operation				Register state after operation			
Qd	A_3	A_2	A_1	A_0	B_2	B_0	A_2	A_0
Qm	B_3	B_2	B_1	B_0	B_3	B_1	A_3	A_1

4.86.1 Syntax

V*op*{*cond*}.*size Qd*, *Qm*

V*op*{*cond*}.*size Dd*, *Dm*

where:

op must be either UZP or ZIP.

cond is an optional condition code.

size must be one of 8, 16, or 32.

Qd, Qm specifies the vectors, for a quadword operation.

Dd, Dm specifies the vectors, for a doubleword operation.

Note

The following are all the same instruction:

- VZIP.32 *Dd*, *Dm*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

- `VUZP.32 Dd, Dm`.
- `VTRN.32 Dd, Dm`.

The instruction is disassembled as `VTRN.32 Dd, Dm`.

4.86.2 See also

Reference

- *De-interleaving an array of 3-element structures* on page 4-14.
- *VTRN* on page 4-97.
- *Condition codes* on page 3-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Chapter 5
A64 General Instructions

The following topic gives a summary of the A64 general instructions and pseudo-instructions supported by the ARM assembler:

- *A64 general instructions in alphabetical order* on page 5-2.
- *Register restrictions for A64 instructions* on page 5-8.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.1 A64 general instructions in alphabetical order

The following A64 general instructions and pseudo-instructions are supported:

Table 5-1 Location of general instructions

Mnemonic	Brief description	See
ADC	Add with carry	page 5-9
ADCS	Add with carry, setting the condition flags	page 5-10
ADD (extended register)	Add (extended register)	page 5-11
ADD (immediate)	Add (immediate)	page 5-13
ADD (shifted register)	Add (shifted register)	page 5-14
ADDS (extended register)	Add (extended register), setting the condition flags	page 5-15
ADDS (immediate)	Add (immediate), setting the condition flags	page 5-17
ADDS (shifted register)	Add (shifted register), setting the condition flags	page 5-18
ADR	Address of label at a PC-relative offset	page 5-19
ADRL pseudo-instruction	Load a PC-relative address into a register	page 5-20
ADRP	Address of 4KB page at a PC-relative offset	page 5-21
AND (immediate)	Bitwise AND (immediate)	page 5-22
AND (shifted register)	Bitwise AND (shifted register)	page 5-23
ANDS (immediate)	Bitwise AND (immediate), setting the condition flags	page 5-24
ANDS (shifted register)	Bitwise AND (shifted register), setting the condition flags	page 5-25
ASR (register)	Arithmetic shift right (register)	page 5-26
ASR (immediate)	Arithmetic shift right (immediate)	page 5-27
ASRV	Arithmetic shift right variable	page 5-28
AT	Address translate	page 5-29
B.*cond*	Branch conditionally to a label at a PC-relative offset, with a hint that this is not a subroutine call or return	page 5-30
B	Branch unconditionally to a label at a PC-relative offset, with a hint that this is not a subroutine call or return	page 5-31
BFI	Bitfield insert, leaving other bits unchanged	page 5-32
BFM	Bitfield move, leaving other bits unchanged	page 5-33
BFXIL	Bitfield extract and insert at low end, leaving other bits unchanged	page 5-34
BIC (shifted register)	Bitwise bit clear (shifted register)	page 5-35
BICS (shifted register)	Bitwise bit clear (shifted register), setting the condition flags	page 5-36
BL	Branch with link, calls a subroutine at a PC-relative offset, setting register X30 to PC + 4	page 5-37

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 5-1 Location of general instructions (continued)

Mnemonic	Brief description	See
BLR	Branch with link to register, calls a subroutine at an address in a register, setting register X30 to PC + 4	page 5-38
BR	Branch to register, branches unconditionally to an address in a register, with a hint that this is not a subroutine return	page 5-39
BRK	Self-hosted debug breakpoint	page 5-40
CBNZ	Compare and branch if nonzero to a label at a PC-relative offset, without affecting the condition flags, and with a hint that this is not a subroutine call or return	page 5-41
CBZ	Compare and branch if zero to a label at a PC-relative offset, without affecting the condition flags, and with a hint that this is not a subroutine call or return	page 5-42
CCMN (immediate)	Conditional compare negative (immediate), setting condition flags to result of comparison or an immediate value	page 5-43
CCMN (register)	Conditional compare negative (register), setting condition flags to result of comparison or an immediate value	page 5-44
CCMP (immediate)	Conditional compare (immediate), setting condition flags to result of comparison or an immediate value	page 5-45
CCMP (register)	Conditional compare (register), setting condition flags to result of comparison or an immediate value	page 5-46
CINC	Conditional increment	page 5-47
CINV	Conditional invert	page 5-48
CLREX	Clear exclusive monitor	page 5-49
CLS	Count leading sign bits	page 5-50
CLZ	Count leading zero bits	page 5-51
CMN (extended register)	Compare negative (extended register), setting the condition flags and discarding the result	page 5-52
CMN (immediate)	Compare negative (immediate), setting the condition flags and discarding the result	page 5-54
CMN (shifted register)	Compare negative (shifted register), setting the condition flags and discarding the result	page 5-55
CMP (extended register)	Compare (extended register), setting the condition flags and discarding the result	page 5-56
CMP (immediate)	Compare (immediate), setting the condition flags and discarding the result	page 5-58
CMP (shifted register)	Compare (shifted register), setting the condition flags and discarding the result	page 5-59
CNEG	Conditional negate	page 5-60
CRC32B, CRC32H, CRC32W, CRC32X	CRC-32 checksum from byte, halfword, word or doubleword	page 5-61

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 5-1 Location of general instructions (continued)

Mnemonic	Brief description	See
CRC32CB, CRC32CH, CRC32CW, CRC32CX	CRC-32C checksum from byte, halfword, word, or doubleword	page 5-62
CSEL	Conditional select, returning the first or second input	page 5-63
CSET	Conditional set	page 5-64
CSETM	Conditional set mask	page 5-65
CSINC	Conditional select increment, returning the first input or incremented second input	page 5-66
CSINV	Conditional select inversion, returning the first input or inverted second input	page 5-67
CSNEG	Conditional select negation, returning the first input or negated second input	page 5-68
DC	Data cache operation	page 5-69
DCPS1	Debug switch to exception level 1	page 5-70
DCPS2	Debug switch to exception level 2	page 5-71
DCPS3	Debug switch to exception level 3	page 5-72
DMB	Data memory barrier	page 5-73
DRPS	Debug restore process state	page 5-74
DSB	Data synchronization barrier	page 5-75
EON (shifted register)	Bitwise exclusive OR NOT (shifted register)	page 5-76
EOR (immediate)	Bitwise exclusive OR (immediate)	page 5-77
EOR (shifted register)	Bitwise exclusive OR (shifted register)	page 5-78
ERET	Returns from an exception	page 5-79
EXTR	Extract register from pair of registers	page 5-80
HINT	Hint instruction	page 5-81
HLT	External debug breakpoint	page 5-82
HVC	Hypervisor call to allow OS code to call the Hypervisor	page 5-83
IC	Instruction cache operation	page 5-84
ISB	Instruction synchronization barrier	page 5-85
LSL (register)	Logical shift left (register)	page 5-86
LSL (immediate)	Logical shift left (immediate)	page 5-87
LSLV	Logical shift left variable	page 5-88
LSR (register)	Logical shift right (register)	page 5-89
LSR (immediate)	Logical shift right (immediate)	page 5-90
LSRV	Logical shift right variable	page 5-91

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 5-1 Location of general instructions (continued)

Mnemonic	Brief description	See
MADD	Multiply-add	page 5-92
MNEG	Multiply-negate	page 5-93
MOV (to or from SP)	Move between register and stack pointer	page 5-94
MOV (inverted wide immediate)	Move inverted 16-bit immediate to register	page 5-95
MOV (wide immediate)	Move 16-bit immediate to register	page 5-96
MOV (bitmask immediate)	Move bitmask immediate to register	page 5-97
MOV (register)	Move register to register	page 5-98
MOVK	Move 16-bit immediate into register, keeping other bits unchanged	page 5-99
MOVL pseudo-instruction	Load a register	page 5-100
MOVN	Move inverse of shifted 16-bit immediate to register	page 5-102
MOVZ	Move shifted 16-bit immediate to register	page 5-103
MRS	Move from system register	page 5-104
MSR (immediate)	Move immediate to process state field	page 5-105
MSR (register)	Move to system register	page 5-106
MSUB	Multiply-subtract	page 5-107
MUL	Multiply	page 5-108
MVN	Bitwise NOT (shifted register)	page 5-109
NEG (shifted register)	Negate	page 5-110
NEGS	Negate, setting the condition flags	page 5-111
NGC	Negate with carry	page 5-112
NGCS	Negate with carry, setting the condition flags	page 5-113
NOP	No operation	page 5-114
ORN (shifted register)	Bitwise inclusive OR NOT (shifted register)	page 5-115
ORR (immediate)	Bitwise inclusive OR (immediate)	page 5-116
ORR (shifted register)	Bitwise inclusive OR (shifted register)	page 5-117
RBIT	Reverse bit order	page 5-118
RET	Return from subroutine, branches unconditionally to an address in a register, with a hint that this is a subroutine return	page 5-119
REV	Reverse bytes	page 5-120
REV16	Reverse bytes in 16-bit halfwords	page 5-121
REV32	Reverse bytes in 32-bit words	page 5-122
ROR (immediate)	Rotate right (immediate)	page 5-123

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 5-1 Location of general instructions (continued)

Mnemonic	Brief description	See
ROR (register)	Rotate right (register)	page 5-124
RORV	Rotate right variable	page 5-125
SBC	Subtract with carry	page 5-126
SBCS	Subtract with carry, setting the condition flags	page 5-127
SBFIZ	Signed bitfield insert in zero, with sign replication to left and zeros to right	page 5-128
SBFM	Signed bitfield move, with sign replication to left and zeros to right	page 5-129
SBFX	Signed bitfield extract	page 5-131
SDIV	Signed divide	page 5-132
SEV	Send event	page 5-133
SEVL	Send event locally	page 5-134
SMADDL	Signed multiply-add long	page 5-135
SMC	Supervisor call to allow OS or Hypervisor code to call the Secure Monitor	page 5-136
SMNEGL	Signed multiply-negate long	page 5-137
SMSUBL	Signed multiply-subtract long	page 5-138
SMULH	Signed multiply high	page 5-139
SMULL	Signed multiply long	page 5-140
SUB (extended register)	Subtract (extended register)	page 5-141
SUB (immediate)	Subtract (immediate)	page 5-143
SUB (shifted register)	Subtract (shifted register)	page 5-144
SUBS (extended register)	Subtract (extended register), setting the condition flags	page 5-145
SUBS (immediate)	Subtract (immediate), setting the condition flags	page 5-147
SUBS (shifted register)	Subtract (shifted register), setting the condition flags	page 5-148
SVC	Supervisor call to allow application code to call the OS	page 5-149
SXTB	Signed extend byte	page 5-150
SXTH	Signed extend halfword	page 5-151
SXTW	Signed extend word	page 5-152
SYS	System instruction	page 5-153
SYSL	System instruction with result	page 5-154
TBNZ	Test bit and branch if nonzero to a label at a PC-relative offset, without affecting the condition flags, and with a hint that this is not a subroutine call or return	page 5-155

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 5-1 Location of general instructions (continued)

Mnemonic	Brief description	See
TBZ	Test bit and branch if zero to a label at a PC-relative offset, without affecting the condition flags, and with a hint that this is not a subroutine call or return	page 5-156
TLBI	TLB invalidate operation	page 5-157
TST (immediate)	Test bits (immediate), setting the condition flags and discarding the result	page 5-158
TST (shifted register)	Test bits (shifted register), setting the condition flags and discarding the result	page 5-159
UBFIZ	Unsigned bitfield insert in zero, with zeros to left and right	page 5-160
UBFM	Unsigned bitfield move, with zeros to left and right	page 5-161
UBFX	Unsigned bitfield extract	page 5-163
UDIV	Unsigned divide	page 5-164
UMADDL	Unsigned multiply-add long	page 5-165
UMNEGL	Unsigned multiply-negate long	page 5-166
UMSUBL	Unsigned multiply-subtract long	page 5-167
UMULH	Unsigned multiply high	page 5-168
UMULL	Unsigned multiply long	page 5-169
UXTB	Unsigned extend byte	page 5-170
UXTH	Unsigned extend halfword	page 5-171
WFE	Wait for event	page 5-172
WFI	Wait for interrupt	page 5-173
YIELD	Yield hint	page 5-174

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.2 Register restrictions for A64 instructions

In A64 instructions, the range of general-purpose integer registers is as follows:

- W0-W30 for 32-bit registers.
- X0-X30 for 64-bit registers.

You cannot refer to register 31 by number. In a few instructions, you can refer to it using one of the following names:

`WSP` the current stack pointer in a 32-bit context.

`SP` the current stack pointer in a 64-bit context.

`WZR` the zero register in a 32-bit context.

`XZR` the zero register in a 64-bit context.

You can only use one of these names if it is mentioned in the Syntax section for the instruction.

You cannot refer to the Program Counter (PC) explicitly by name or by number.

5.2.1 See also

Concepts

Using the Assembler:

- *Registers in AArch64 state* on page 5-2
- *Program Counter in AArch64 state* on page 5-8.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.3 ADC

Add with carry.

5.3.1 Syntax

```
ADC  Wd, Wn, Wm    ; 32-bit general registers
ADC  Xd, Xn, Xm    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

5.3.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.4 ADCS

Add with carry, setting the condition flags.

5.4.1 Syntax

```
ADCS  Wd, Wn, Wm    ; 32-bit general registers
ADCS  Xd, Xn, Xm    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

5.4.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.5 ADD (extended register)

Add (extended register).

5.5.1 Syntax

```
ADD  Wd|WSP, Wn|WSP, Wm{, extend {#amount}}    ; 32-bit general registers
ADD  Xd|SP, Xn|SP, Rm{, extend {#amount}}    ; 64-bit general registers
```

Where:

Wd|WSP Is the 32-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Wn|WSP Is the 32-bit name of the first source general-purpose register or stack pointer, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

extend Is the extension to be applied to the second source operand:

32-bit general registers

Can be one of UXTB, UXTH, LSL|UXTW, UXTX, SXTB, SXTH, SXTW or SXTX.

If *Rd* or *Rn* is WSP then LSL is preferred rather than UXTW, and can be omitted when *amount* is 0. In all other cases *extend* is required and must be UXTW rather than LSL.

64-bit general registers

Can be one of UXTB, UXTH, UXTW, LSL|UXTX, SXTB, SXTH, SXTW or SXTX.

If *Rd* or *Rn* is SP then LSL is preferred rather than UXTX, and can be omitted when *amount* is 0. In all other cases *extend* is required and must be UXTX rather than LSL.

Xd|SP Is the 64-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Xn|SP Is the 64-bit name of the first source general-purpose register or stack pointer, in the range 0 to 31.

R Is a width specifier, and can be either W or X.

m Is the number of the second general-purpose source register, in the range 0 to 30, or the name ZR (31).

amount Is the left shift amount to be applied after extension in the range 0 to 4, defaulting to 0. It must be absent when *extend* is absent, is required when *extend* is LSL, and is optional when *extend* is present but not LSL.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.5.2 Usage

The following table shows valid specifier combinations:

Table 5-2 ADD (64-bit general registers) specifier combinations

R	*extend*
W	SXTB
W	SXTH
W	SXTW
W	UXTB
W	UXTH
W	UXTW
X	LSL|UXTX
X	SXTX

5.5.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.6 ADD (immediate)

Add (immediate).

This instruction is used by the alias `MOV` (to or from SP).

5.6.1 Syntax

```
ADD  Wd|WSP, Wn|WSP, #imm{, shift}    ; 32-bit general registers
ADD  Xd|SP, Xn|SP, #imm{, shift}    ; 64-bit general registers
```

Where:

Wd|WSP Is the 32-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Wn|WSP Is the 32-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

Xd|SP Is the 64-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Xn|SP Is the 64-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

imm Is an unsigned immediate, in the range 0 to 4095.

shift Is the optional left shift to apply to the immediate, defaulting to `LSL #0`, and can be either `LSL #0` or `LSL #12`.

5.6.2 See also

Reference

- *MOV (to or from SP)* on page 5-94.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.7 ADD (shifted register)

Add (shifted register).

5.7.1 Syntax

```
ADD  Wd, Wn, Wm{, shift #amount}    ; 32-bit general registers
ADD  Xd, Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers

Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers

Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift type to be applied to the second source operand, defaulting to LSL, and can be one of LSL, LSR, or ASR.

5.7.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.8 ADDS (extended register)

Add (extended register), setting the condition flags.

This instruction is used by the alias CMN (extended register).

5.8.1 Syntax

```
ADDS  Wd, Wn|WSP, Wm{, extend {#amount}}    ; 32-bit general registers
ADDS  Xd, Xn|SP, Rm{, extend {#amount}}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn|WSP Is the 32-bit name of the first source general-purpose register or stack pointer, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

extend Is the extension to be applied to the second source operand:

32-bit general registers

Can be one of UXTB, UXTH, LSL|UXTW, UXTX, SXTB, SXTH, SXTW or SXTX.

If *Rn* is WSP then LSL is preferred rather than UXTW, and can be omitted when *amount* is 0. In all other cases *extend* is required and must be UXTW rather than LSL.

64-bit general registers

Can be one of UXTB, UXTH, UXTW, LSL|UXTX, SXTB, SXTH, SXTW or SXTX.

If *Rn* is SP then LSL is preferred rather than UXTX, and can be omitted when *amount* is 0. In all other cases *extend* is required and must be UXTX rather than LSL.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn|SP Is the 64-bit name of the first source general-purpose register or stack pointer, in the range 0 to 31.

R Is a width specifier, and can be either W or X.

m Is the number of the second general-purpose source register, in the range 0 to 30, or the name ZR (31).

amount Is the left shift amount to be applied after extension in the range 0 to 4, defaulting to 0. It must be absent when *extend* is absent, is required when *extend* is LSL, and is optional when *extend* is present but not LSL.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.8.2 Usage

The following table shows valid specifier combinations:

Table 5-3 ADDS (64-bit general registers) specifier combinations

R	*extend*
W	SXTB
W	SXTH
W	SXTW
W	UXTB
W	UXTH
W	UXTW
X	LSL|UXTX
X	SXTX

5.8.3 See also

Reference

- *CMN (extended register)* on page 5-52.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.9 ADDS (immediate)

Add (immediate), setting the condition flags.

This instruction is used by the alias `CMN` (immediate).

5.9.1 Syntax

```
ADDS  Wd, Wn|WSP, #imm{, shift}    ; 32-bit general registers
ADDS  Xd, Xn|SP, #imm{, shift}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn|WSP Is the 32-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn|SP Is the 64-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

imm Is an unsigned immediate, in the range 0 to 4095.

shift Is the optional left shift to apply to the immediate, defaulting to `LSL #0`, and can be either `LSL #0` or `LSL #12`.

5.9.2 See also

Reference

- *CMN (immediate)* on page 5-54.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.10 ADDS (shifted register)

Add (shifted register), setting the condition flags.

This instruction is used by the alias `CMN` (shifted register).

5.10.1 Syntax

```
ADDS  Wd, Wn, Wm{, shift #amount}    ; 32-bit general registers
ADDS  Xd, Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers
Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift type to be applied to the second source operand, defaulting to `LSL`, and can be one of `LSL`, `LSR`, or `ASR`.

5.10.2 See also

Reference

- *CMN (shifted register)* on page 5-55.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.11 ADR

Address of label at a PC-relative offset.

5.11.1 Syntax

```
ADR  Xd, label
```

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

label Is the program label whose address is to be calculated. It is an offset from the address of this instruction, in the range ±1MB.

5.11.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.12 ADRL pseudo-instruction

Load a PC-relative address into a register. It is similar to the ADR instruction. ADRL can load a wider range of addresses than ADR because it generates two data processing instructions.

5.12.1 Syntax

ADRL *Wd*,*label*

ADRL *Xd*,*label*

where:

Wd Is the register to load with a 32-bit address.

Xd Is the register to load with a 64-bit address.

label Is a PC-relative expression.

5.12.2 Usage

ADRL assembles to two instructions, an ADRP followed by ADD.

If the assembler cannot construct the address in two instructions, it generates a relocation. The linker then generates the correct offsets.

ADRL produces position-independent code, because the address is calculated relative to PC.

5.12.3 Example

```
ADRL  x0, mylabel   ; loads address of mylabel into x0
```

5.12.4 See also

Concepts

Using the Assembler

- *Register-relative and PC-relative expressions* on page 10-7

Reference

- *ADD (immediate)* on page 5-13.
- *ADR* on page 5-19.
- *ADRP* on page 5-21.
- *RELOC* on page 10-76.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.13 ADRP

Address of 4KB page at a PC-relative offset.

5.13.1 Syntax

```
ADRP  Xd, label
```

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

label Is the program label whose 4KB page address is to be calculated. An offset from the page address of this instruction, in the range ±4GB.

5.13.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.14 AND (immediate)

Bitwise AND (immediate).

5.14.1 Syntax

```
AND  Wd|WSP, Wn, #imm    ; 32-bit general registers
AND  Xd|SP, Xn, #imm     ; 64-bit general registers
```

Where:

Wd|WSP Is the 32-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Xd|SP Is the 64-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

imm Is the bitmask immediate. Such an immediate is a 32-bit or 64-bit pattern viewed as a vector of identical elements of size *e* = 2, 4, 8, 16, 32, or 64 bits. Each element contains the same sub-pattern: a single run of 1 to *e*-1 non-zero bits, rotated by 0 to *e*-1 bits. This mechanism can generate 5,334 unique 64-bit patterns (as 2,667 pairs of pattern and their bitwise inverse). Because the all-zeros and all-ones values cannot be described in this way, the assembler generates an error message.

Note

Logical immediate instructions do not set the condition flags, but interesting results can usually directly control a `CBZ`, `CBNZ`, `TBZ`, or `TBNZ` conditional branch.

5.14.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.15 AND (shifted register)

Bitwise AND (shifted register).

5.15.1 Syntax

```
AND  Wd, Wn, Wm{, shift #amount}    ; 32-bit general registers
AND  Xd, Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers

Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers

Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift to be applied to the final source, defaulting to `LSL`, and can be one of `LSL`, `LSR`, `ASR`, or `ROR`.

5.15.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.16 ANDS (immediate)

Bitwise AND (immediate), setting the condition flags.

This instruction is used by the alias `TST` (immediate).

5.16.1 Syntax

```
ANDS  Wd, Wn, #imm    ; 32-bit general registers
ANDS  Xd, Xn, #imm    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

imm Is the bitmask immediate. Such an immediate is a 32-bit or 64-bit pattern viewed as a vector of identical elements of size *e* = 2, 4, 8, 16, 32, or 64 bits. Each element contains the same sub-pattern: a single run of 1 to *e*-1 non-zero bits, rotated by 0 to *e*-1 bits. This mechanism can generate 5,334 unique 64-bit patterns (as 2,667 pairs of pattern and their bitwise inverse). Because the all-zeros and all-ones values cannot be described in this way, the assembler generates an error message.

5.16.2 See also

Reference

- *TST (immediate)* on page 5-158.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.17 ANDS (shifted register)

Bitwise AND (shifted register), setting the condition flags.

This instruction is used by the alias TST (shifted register).

5.17.1 Syntax

```
ANDS  Wd, Wn, Wm{, shift #amount}    ; 32-bit general registers
ANDS  Xd, Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers
Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift to be applied to the final source, defaulting to LSL, and can be one of LSL, LSR, ASR, or ROR.

5.17.2 See also

Reference

- *TST (shifted register)* on page 5-159.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.18 ASR (register)

Arithmetic shift right (register).

This instruction is an alias of ASRV.

5.18.1 Syntax

```
ASR  Wd, Wn, Wm    ; 32-bit general registers
```

Equivalent to `ASRV Wd, Wn, Wm`

```
ASR  Xd, Xn, Xm    ; 64-bit general registers
```

Equivalent to `ASRV Xd, Xn, Xm`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 31 in its bottom 5 bits.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 63 in its bottom 6 bits.

5.18.2 See also

Reference

- *ASRV* on page 5-28.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.19 ASR (immediate)

Arithmetic shift right (immediate).

This instruction is an alias of SBFM.

5.19.1 Syntax

```
ASR  Wd, Wn, #shift    ; 32-bit general registers
```

Equivalent to `SBFM Wd, Wn, #shift, #31`

```
ASR  Xd, Xn, #shift    ; 64-bit general registers
```

Equivalent to `SBFM Xd, Xn, #shift, #63`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

shift The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31.

64-bit general registers
Is the shift amount, in the range 0 to 63.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.19.2 See also

Reference

- *SBFM* on page 5-129.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.20 ASRV

Arithmetic shift right variable.

This instruction is used by the alias ASR (register).

5.20.1 Syntax

```
ASRV  Wd, Wn, Wm    ; 32-bit general registers
ASRV  Xd, Xn, Xm    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 31 in its bottom 5 bits.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 63 in its bottom 6 bits.

5.20.2 See also

Reference

- *ASR (register)* on page 5-26.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.21 AT

Address translate.

This instruction is an alias of SYS.

5.21.1 Syntax

```
AT  at_op, Xt
```

Equivalent to `SYS #op1, Cn, Cm, #op2, Xt`

Where:

at_op Is an AT operation name, as listed for the AT system operation group specified by the parameters *op1*, *Cn*, *Cm*, and *op2*.

op1 Is a 3-bit unsigned immediate, in the range 0 to 7.

Cn Is a name C*n*, with *n* in the range 0 to 15.

Cm Is a name C*m*, with *m* in the range 0 to 15.

op2 Is a 3-bit unsigned immediate, in the range 0 to 7.

Xt Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.21.2 See also

Reference

- *SYS* on page 5-153.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.22 B.*cond*

Branch conditionally to a label at a PC-relative offset, with a hint that this is not a subroutine call or return.

5.22.1 Syntax

```
B.cond  label
```

Where:

cond Is one of the standard conditions.

label Is the program label to be conditionally branched to. It is an offset from the address of this instruction, in the range ±1MB.

5.22.2 See also

Reference

- *Condition codes* on page 3-26
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.23 B

Branch unconditionally to a label at a PC-relative offset, with a hint that this is not a subroutine call or return.

5.23.1 Syntax

```
B  label
```

Where:

label Is the program label to be unconditionally branched to. It is an offset from the address of this instruction, in the range ±128MB.

5.23.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.24 BFI

Bitfield insert, leaving other bits unchanged.

This instruction is an alias of BFM.

5.24.1 Syntax

```
BFI  Wd, Wn, #lsb, #width    ; 32-bit general registers
```

Equivalent to `BFM Wd, Wn, #(-lsb MOD 32), #(width-1)`

```
BFI  Xd, Xn, #lsb, #width    ; 64-bit general registers
```

Equivalent to `BFM Xd, Xn, #(-lsb MOD 64), #(width-1)`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

lsb The value depends on the instruction variant:

32-bit general registers
Is the bit number of the lsb of the source bitfield, in the range 0 to 31.

64-bit general registers
Is the bit number of the lsb of the source bitfield, in the range 0 to 63.

width The value depends on the instruction variant:

32-bit general registers
Is the width of the bitfield, in the range 1 to 32-*lsb*.

64-bit general registers
Is the width of the bitfield, in the range 1 to 64-*lsb*.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.24.2 See also

Reference

- *BFM* on page 5-33.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.25 BFM

Bitfield move, leaving other bits unchanged.

This instruction is used by the aliases:

- `BFI`.
- `BFXIL`.

5.25.1 Syntax

```
BFM  Wd, Wn, #immr, #imms    ; 32-bit general registers
BFM  Xd, Xn, #immr, #imms    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

immr The value depends on the instruction variant:

32-bit general registers
Is the right rotate amount, in the range 0 to 31.

64-bit general registers
Is the right rotate amount, in the range 0 to 63.

imms The value depends on the instruction variant:

32-bit general registers
Is the leftmost bit number to be moved from the source, in the range 0 to 31.

64-bit general registers
Is the leftmost bit number to be moved from the source, in the range 0 to 63.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.25.2 See also

Reference

- *BFI* on page 5-32.
- *BFXIL* on page 5-34.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.26 BFXIL

Bitfield extract and insert at low end, leaving other bits unchanged.

This instruction is an alias of `BFM`.

5.26.1 Syntax

```
BFXIL  Wd, Wn, #lsb, #width    ; 32-bit general registers
```

Equivalent to `BFM Wd, Wn, #lsb, #(lsb+width-1)`

```
BFXIL  Xd, Xn, #lsb, #width    ; 64-bit general registers
```

Equivalent to `BFM Xd, Xn, #lsb, #(lsb+width-1)`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

lsb The value depends on the instruction variant:

32-bit general registers
Is the bit number of the lsb of the source bitfield, in the range 0 to 31.

64-bit general registers
Is the bit number of the lsb of the source bitfield, in the range 0 to 63.

width The value depends on the instruction variant:

32-bit general registers
Is the width of the bitfield, in the range 1 to 32-*lsb*.

64-bit general registers
Is the width of the bitfield, in the range 1 to 64-*lsb*.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.26.2 See also

Reference

- *BFM* on page 5-33.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.27 BIC (shifted register)

Bitwise bit clear (shifted register).

5.27.1 Syntax

```
BIC  Wd, Wn, Wm{, shift #amount}    ; 32-bit general registers
BIC  Xd, Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers

Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers

Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift to be applied to the final source, defaulting to LSL, and can be one of LSL, LSR, ASR, or ROR.

5.27.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.28 BICS (shifted register)

Bitwise bit clear (shifted register), setting the condition flags.

5.28.1 Syntax

```
BICS  Wd, Wn, Wm{, shift #amount}    ; 32-bit general registers
BICS  Xd, Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers

Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers

Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift to be applied to the final source, defaulting to LSL, and can be one of LSL, LSR, ASR, or ROR.

5.28.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.29 BL

Branch with link, calls a subroutine at a PC-relative offset, setting register X30 to PC + 4.

5.29.1 Syntax

```
BL  label
```

Where:

label Is the program label to be unconditionally branched to. It is an offset from the address of this instruction, in the range ±128MB.

5.29.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.30 BLR

Branch with link to register, calls a subroutine at an address in a register, setting register X30 to PC + 4.

5.30.1 Syntax

`BLR` *Xn*

Where:

Xn Is the 64-bit name of the general-purpose register holding the address to be branched to, in the range 0 to 31.

5.30.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.31 BR

Branch to register, branches unconditionally to an address in a register, with a hint that this is not a subroutine return.

5.31.1 Syntax

`BR` *Xn*

Where:

Xn Is the 64-bit name of the general-purpose register holding the address to be branched to, in the range 0 to 31.

5.31.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.32 BRK

Self-hosted debug breakpoint.

5.32.1 Syntax

```
BRK  #imm
```

Where:

imm Is a 16-bit unsigned immediate, in the range 0 to 65535.

5.32.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.33 CBNZ

Compare and branch if nonzero to a label at a PC-relative offset, without affecting the condition flags, and with a hint that this is not a subroutine call or return.

5.33.1 Syntax

```
CBNZ  Wt, label    ; 32-bit general registers
CBNZ  Xt, label    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be tested, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be tested, in the range 0 to 31.

label Is the program label to be conditionally branched to. It is an offset from the address of this instruction, in the range ±1MB.

5.33.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.34 CBZ

Compare and branch if zero to a label at a PC-relative offset, without affecting the condition flags, and with a hint that this is not a subroutine call or return.

5.34.1 Syntax

```
CBZ  Wt, label    ; 32-bit general registers
CBZ  Xt, label    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be tested, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be tested, in the range 0 to 31.

label Is the program label to be conditionally branched to. It is an offset from the address of this instruction, in the range ±1MB.

5.34.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.35 CCMN (immediate)

Conditional compare negative (immediate), setting condition flags to result of comparison or an immediate value.

5.35.1 Syntax

```
CCMN  Wn, #imm, #nzcv, cond    ; 32-bit general registers
CCMN  Xn, #imm, #nzcv, cond    ; 64-bit general registers
```

Where:

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

imm Is a five bit unsigned immediate.

nzcv Is the flag bit specifier, an immediate in the range 0 to 15, giving the alternative state for the 4-bit NZCV condition flags. Bit 3 is the N flag, bit 2 is the Z flag, bit 1 is the C flag, and bit 0 is the V flag.

cond Is one of the standard conditions.

5.35.2 See also

Reference

- *Condition codes* on page 3-26
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.36 CCMN (register)

Conditional compare negative (register), setting condition flags to result of comparison or an immediate value.

5.36.1 Syntax

```
CCMN  Wn, Wm, #nzcv, cond    ; 32-bit general registers
CCMN  Xn, Xm, #nzcv, cond    ; 64-bit general registers
```

Where:

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

nzcv Is the flag bit specifier, an immediate in the range 0 to 15, giving the alternative state for the 4-bit NZCV condition flags. Bit 3 is the N flag, bit 2 is the Z flag, bit 1 is the C flag, and bit 0 is the V flag.

cond Is one of the standard conditions.

5.36.2 See also

Reference

- *Condition codes* on page 3-26
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.37 CCMP (immediate)

Conditional compare (immediate), setting condition flags to result of comparison or an immediate value.

5.37.1 Syntax

```
CCMP  Wn, #imm, #nzcv, cond    ; 32-bit general registers
CCMP  Xn, #imm, #nzcv, cond    ; 64-bit general registers
```

Where:

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

imm Is a five bit unsigned immediate.

nzcv Is the flag bit specifier, an immediate in the range 0 to 15, giving the alternative state for the 4-bit NZCV condition flags. Bit 3 is the N flag, bit 2 is the Z flag, bit 1 is the C flag, and bit 0 is the V flag.

cond Is one of the standard conditions.

5.37.2 See also

Reference

- *Condition codes* on page 3-26
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.38 CCMP (register)

Conditional compare (register), setting condition flags to result of comparison or an immediate value.

5.38.1 Syntax

```
CCMP  Wn, Wm, #nzcv, cond    ; 32-bit general registers
CCMP  Xn, Xm, #nzcv, cond    ; 64-bit general registers
```

Where:

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

nzcv Is the flag bit specifier, an immediate in the range 0 to 15, giving the alternative state for the 4-bit NZCV condition flags. Bit 3 is the N flag, bit 2 is the Z flag, bit 1 is the C flag, and bit 0 is the V flag.

cond Is one of the standard conditions.

5.38.2 See also

Reference

- *Condition codes* on page 3-26
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.39 CINC

Conditional increment.

This instruction is an alias of CSINC.

5.39.1 Syntax

```
CINC  Wd, Wn, cond    ; 32-bit general registers
```

Equivalent to `CSINC Wd, Wn, Wn, invert(cond)`

```
CINC  Xd, Xn, cond    ; 64-bit general registers
```

Equivalent to `CSINC Xd, Xn, Xn, invert(cond)`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register in the range 0 to 31.

cond Is one of the standard conditions, excluding AL and NV.

5.39.2 See also

Reference

- *CSINC* on page 5-66.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.40 CINV

Conditional invert.

This instruction is an alias of CSINV.

5.40.1 Syntax

```
CINV  Wd, Wn, cond    ; 32-bit general registers
```

Equivalent to `CSINV Wd, Wn, Wn, invert(cond)`

```
CINV  Xd, Xn, cond    ; 64-bit general registers
```

Equivalent to `CSINV Xd, Xn, Xn, invert(cond)`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register in the range 0 to 31.

cond Is one of the standard conditions, excluding AL and NV.

5.40.2 See also

Reference

- *CSINV* on page 5-67.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.41 CLREX

Clear exclusive monitor.

5.41.1 Syntax

```
CLREX  {#imm}
```

Where:

imm Is an optional 4-bit unsigned immediate, in the range 0 to 15, defaulting to 15.

5.41.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.42 CLS

Count leading sign bits.

5.42.1 Syntax

```
CLS  Wd, Wn    ; 32-bit general registers
CLS  Xd, Xn    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.42.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.43 CLZ

Count leading zero bits.

5.43.1 Syntax

```
CLZ  Wd, Wn    ; 32-bit general registers
CLZ  Xd, Xn    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.43.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.44 CMN (extended register)

Compare negative (extended register), setting the condition flags and discarding the result.

This instruction is an alias of ADDS (extended register).

5.44.1 Syntax

```
CMN  Wn|WSP, Wm{, extend {#amount}}    ; 32-bit general registers
```

Equivalent to `ADDS WZR, Wn|WSP, Wm{, extend {#amount}}`

```
CMN  Xn|SP, Rm{, extend {#amount}}    ; 64-bit general registers
```

Equivalent to `ADDS XZR, Xn|SP, Rm{, extend {#amount}}`

Where:

Wn|WSP Is the 32-bit name of the first source general-purpose register or stack pointer, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

extend Is the extension to be applied to the second source operand:

32-bit general registers

Can be one of UXTB, UXTH, LSL|UXTW, UXTX, SXTB, SXTH, SXTW or SXTX.

If *Rn* is WSP then LSL is preferred rather than UXTW, and can be omitted when *amount* is 0. In all other cases *extend* is required and must be UXTW rather than LSL.

64-bit general registers

Can be one of UXTB, UXTH, UXTW, LSL|UXTX, SXTB, SXTH, SXTW or SXTX.

If *Rn* is SP then LSL is preferred rather than UXTX, and can be omitted when *amount* is 0. In all other cases *extend* is required and must be UXTX rather than LSL.

Xn|SP Is the 64-bit name of the first source general-purpose register or stack pointer, in the range 0 to 31.

R Is a width specifier, and can be either W or X.

m Is the number of the second general-purpose source register, in the range 0 to 30, or the name ZR (31).

amount Is the left shift amount to be applied after extension in the range 0 to 4, defaulting to 0. It must be absent when *extend* is absent, is required when *extend* is LSL, and is optional when *extend* is present but not LSL.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.44.2 Usage

The following table shows valid specifier combinations:

Table 5-4 CMN (64-bit general registers) specifier combinations

R	*extend*
W	SXTB
W	SXTH
W	SXTW
W	UXTB
W	UXTH
W	UXTW
X	LSL|UXTX
X	SXTX

5.44.3 See also

Reference

- *ADDS (extended register)* on page 5-15.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.45 CMN (immediate)

Compare negative (immediate), setting the condition flags and discarding the result.

This instruction is an alias of ADDS (immediate).

5.45.1 Syntax

```
CMN  Wn|WSP, #imm{, shift}    ; 32-bit general registers
```

Equivalent to `ADDS WZR, Wn|WSP, #imm {, shift}`

```
CMN  Xn|SP, #imm{, shift}    ; 64-bit general registers
```

Equivalent to `ADDS XZR, Xn|SP, #imm {, shift}`

Where:

Wn|WSP Is the 32-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

Xn|SP Is the 64-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

imm Is an unsigned immediate, in the range 0 to 4095.

shift Is the optional left shift to apply to the immediate, defaulting to `LSL #0`, and can be either `LSL #0` or `LSL #12`.

5.45.2 See also

Reference

- *ADDS (immediate)* on page 5-17.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.46 CMN (shifted register)

Compare negative (shifted register), setting the condition flags and discarding the result.

This instruction is an alias of ADDS (shifted register).

5.46.1 Syntax

```
CMN  Wn, Wm{, shift #amount}    ; 32-bit general registers
```

Equivalent to `ADDS WZR, Wn, Wm {, shift #amount}`

```
CMN  Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Equivalent to `ADDS XZR, Xn, Xm {, shift #amount}`

Where:

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers
Is the shift amount, in the range 0 to 63, defaulting to 0.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift type to be applied to the second source operand, defaulting to LSL, and can be one of LSL, LSR, or ASR.

5.46.2 See also

Reference

- *ADDS (shifted register)* on page 5-18.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.47 CMP (extended register)

Compare (extended register), setting the condition flags and discarding the result.

This instruction is an alias of SUBS (extended register).

5.47.1 Syntax

```
CMP  Wn|WSP, Wm{, extend {#amount}}    ; 32-bit general registers
```

Equivalent to `SUBS WZR, Wn|WSP, Wm{, extend {#amount}}`

```
CMP  Xn|SP, Rm{, extend {#amount}}    ; 64-bit general registers
```

Equivalent to `SUBS XZR, Xn|SP, Rm{, extend {#amount}}`

Where:

Wn|WSP Is the 32-bit name of the first source general-purpose register or stack pointer, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

extend Is the extension to be applied to the second source operand:

32-bit general registers

Can be one of UXTB, UXTH, LSL|UXTW, UXTX, SXTB, SXTH, SXTW or SXTX.

If *Rn* is WSP then LSL is preferred rather than UXTW, and can be omitted when *amount* is 0. In all other cases *extend* is required and must be UXTW rather than LSL.

64-bit general registers

Can be one of UXTB, UXTH, UXTW, LSL|UXTX, SXTB, SXTH, SXTW or SXTX.

If *Rn* is SP then LSL is preferred rather than UXTX, and can be omitted when *amount* is 0. In all other cases *extend* is required and must be UXTX rather than LSL.

Xn|SP Is the 64-bit name of the first source general-purpose register or stack pointer, in the range 0 to 31.

R Is a width specifier, and can be either W or X.

m Is the number of the second general-purpose source register, in the range 0 to 30, or the name ZR (31).

amount Is the left shift amount to be applied after extension in the range 0 to 4, defaulting to 0. It must be absent when *extend* is absent, is required when *extend* is LSL, and is optional when *extend* is present but not LSL.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.47.2 Usage

The following table shows valid specifier combinations:

Table 5-5 CMP (64-bit general registers) specifier combinations

R	*extend*
W	SXTB
W	SXTH
W	SXTW
W	UXTB
W	UXTH
W	UXTW
X	LSL|UXTX
X	SXTX

5.47.3 See also

Reference

- *SUBS (extended register)* on page 5-145.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.48 CMP (immediate)

Compare (immediate), setting the condition flags and discarding the result.

This instruction is an alias of SUBS (immediate).

5.48.1 Syntax

```
CMP  Wn|WSP, #imm{, shift}    ; 32-bit general registers
```

Equivalent to `SUBS WZR, Wn|WSP, #imm {, shift}`

```
CMP  Xn|SP, #imm{, shift}    ; 64-bit general registers
```

Equivalent to `SUBS XZR, Xn|SP, #imm {, shift}`

Where:

Wn|WSP Is the 32-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

Xn|SP Is the 64-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

imm Is an unsigned immediate, in the range 0 to 4095.

shift Is the optional left shift to apply to the immediate, defaulting to `LSL #0`, and can be either `LSL #0` or `LSL #12`.

5.48.2 See also

Reference

- *SUBS (immediate)* on page 5-147.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.49 CMP (shifted register)

Compare (shifted register), setting the condition flags and discarding the result.

This instruction is an alias of SUBS (shifted register).

5.49.1 Syntax

```
CMP  Wn, Wm{, shift #amount}    ; 32-bit general registers
```

Equivalent to `SUBS WZR, Wn, Wm {, shift #amount}`

```
CMP  Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Equivalent to `SUBS XZR, Xn, Xm {, shift #amount}`

Where:

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers
Is the shift amount, in the range 0 to 63, defaulting to 0.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift type to be applied to the second source operand, defaulting to LSL, and can be one of LSL, LSR, or ASR.

5.49.2 See also

Reference

- *SUBS (shifted register)* on page 5-148.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.50 CNEG

Conditional negate.

This instruction is an alias of CSNEG.

5.50.1 Syntax

```
CNEG  Wd, Wn, cond    ; 32-bit general registers
```

Equivalent to `CSNEG Wd, Wn, Wn, invert(cond)`

```
CNEG  Xd, Xn, cond    ; 64-bit general registers
```

Equivalent to `CSNEG Xd, Xn, Xn, invert(cond)`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register in the range 0 to 31.

cond Is one of the standard conditions, excluding AL and NV.

5.50.2 See also

Reference

- *CSNEG* on page 5-68.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.51 CRC32B, CRC32H, CRC32W, CRC32X

CRC-32 checksum from byte, halfword, word or doubleword.

5.51.1 Syntax

```
CRC32B  Wd, Wn, Wm    ; Wd = CRC32(Wn, Rm[7:0])
CRC32H  Wd, Wn, Wm    ; Wd = CRC32(Wn, Rm[15:0])
CRC32W  Wd, Wn, Wm    ; Wd = CRC32(Wn, Rm[31:0])
CRC32X  Wd, Wn, Xm    ; Wd = CRC32(Wn, Rm[63:0])
```

Where:

Wm Is the 32-bit name of the general-purpose data source register, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose data source register, in the range 0 to 31.

Wd Is the 32-bit name of the general-purpose accumulator output register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose accumulator input register, in the range 0 to 31.

5.51.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.52 CRC32CB, CRC32CH, CRC32CW, CRC32CX

CRC-32C checksum from byte, halfword, word, or doubleword.

5.52.1 Syntax

```
CRC32CB  Wd, Wn, Wm    ; Wd = CRC32C(Wn, Rm[7:0])
CRC32CH  Wd, Wn, Wm    ; Wd = CRC32C(Wn, Rm[15:0])
CRC32CW  Wd, Wn, Wm    ; Wd = CRC32C(Wn, Rm[31:0])
CRC32CX  Wd, Wn, Xm    ; Wd = CRC32C(Wn, Rm[63:0])
```

Where:

Wm Is the 32-bit name of the general-purpose data source register, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose data source register, in the range 0 to 31.

Wd Is the 32-bit name of the general-purpose accumulator output register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose accumulator input register, in the range 0 to 31.

5.52.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.53 CSEL

Conditional select, returning the first or second input.

5.53.1 Syntax

```
CSEL  Wd, Wn, Wm, cond    ; 32-bit general registers
CSEL  Xd, Xn, Xm, cond    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

cond Is one of the standard conditions.

5.53.2 See also

Reference

- *Condition codes* on page 3-26
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.54 CSET

Conditional set.

This instruction is an alias of CSINC.

5.54.1 Syntax

```
CSET  Wd, cond    ; 32-bit general registers
```

Equivalent to `CSINC Wd, WZR, WZR, invert(cond)`

```
CSET  Xd, cond    ; 64-bit general registers
```

Equivalent to `CSINC Xd, XZR, XZR, invert(cond)`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

cond Is one of the standard conditions, excluding AL and NV.

5.54.2 See also

Reference

- *CSINC* on page 5-66.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.55 CSETM

Conditional set mask.

This instruction is an alias of `CSINV`.

5.55.1 Syntax

```
CSETM  Wd, cond    ; 32-bit general registers
```

Equivalent to `CSINV Wd, WZR, WZR, invert(cond)`

```
CSETM  Xd, cond    ; 64-bit general registers
```

Equivalent to `CSINV Xd, XZR, XZR, invert(cond)`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

cond Is one of the standard conditions, excluding `AL` and `NV`.

5.55.2 See also

Reference

- *CSINV* on page 5-67.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.56 CSINC

Conditional select increment, returning the first input or incremented second input.

This instruction is used by the aliases:

- CINC.
- CSET.

5.56.1 Syntax

```
CSINC  Wd, Wn, Wm, cond    ; 32-bit general registers
CSINC  Xd, Xn, Xm, cond    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

cond Is one of the standard conditions.

5.56.2 See also

Reference

- *CINC* on page 5-47.
- *CSET* on page 5-64.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.57 CSINV

Conditional select inversion, returning the first input or inverted second input.

This instruction is used by the aliases:

- `CINV`.
- `CSETM`.

5.57.1 Syntax

```
CSINV  Wd, Wn, Wm, cond    ; 32-bit general registers
CSINV  Xd, Xn, Xm, cond    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

cond Is one of the standard conditions.

5.57.2 See also

Reference

- *CINV* on page 5-48.
- *CSETM* on page 5-65.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.58 CSNEG

Conditional select negation, returning the first input or negated second input.

This instruction is used by the alias `CNEG`.

5.58.1 Syntax

```
CSNEG  Wd, Wn, Wm, cond    ; 32-bit general registers
CSNEG  Xd, Xn, Xm, cond    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

cond Is one of the standard conditions.

5.58.2 See also

Reference

- *CNEG* on page 5-60.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.59 DC

Data cache operation.

This instruction is an alias of SYS.

5.59.1 Syntax

`DC  dc_op, Xt`

Equivalent to `SYS #op1, Cn, Cm, #op2, Xt`

Where:

dc_op Is a DC operation name, as listed for the DC system operation group specified by the parameters *op1*, *Cn*, *Cm*, and *op2*.

op1 Is a 3-bit unsigned immediate, in the range 0 to 7.

Cn Is a name C*n*, with *n* in the range 0 to 15.

Cm Is a name C*m*, with *m* in the range 0 to 15.

op2 Is a 3-bit unsigned immediate, in the range 0 to 7.

Xt Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.59.2 See also

Reference

- *SYS* on page 5-153.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.60 DCPS1 (A64 instruction)

Debug switch to exception level 1.

5.60.1 Syntax

```
DCPS1  {#imm}
```

Where:

imm Is an optional 16-bit unsigned immediate, in the range 0 to 65535, defaulting to 0.

5.60.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.61 DCPS2 (A64 instruction)

Debug switch to exception level 2.

5.61.1 Syntax

```
DCPS2  {#imm}
```

Where:

imm Is an optional 16-bit unsigned immediate, in the range 0 to 65535, defaulting to 0.

5.61.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.62 DCPS3 (A64 instruction)

Debug switch to exception level 3.

5.62.1 Syntax

```
DCPS3  {#imm}
```

Where:

imm Is an optional 16-bit unsigned immediate, in the range 0 to 65535, defaulting to 0.

5.62.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.63 DMB

Data memory barrier.

5.63.1 Syntax

```
DMB  option|#imm
```

Where:

option Is a barrier option name. The options in the following table are ordered in decreasing scope of the shareability domain. ARM recommends that you use the option names in preference to the equivalent *#imm* values.

Table 5-6 Data memory barrier options

option	*imm*	Ordered Accesses (before-after)	Shareability domain
LD	13	Load-Load, Load-Store	
ST	14	Store-Store	Full system
SY	15	Any-Any	
OSHLD	1	Load-Load, Load-Store	
OSHST	2	Store-Store	Outer shareable
OSH	3	Any-Any	
ISHLD	9	Load-Load, Load-Store	
ISHST	10	Store-Store	Inner shareable
ISH	11	Any-Any	
NSHLD	5	Load-Load, Load-Store	
NSHST	6	Store-Store	Non-shareable
NSH	7	Any-Any	

imm Is a 4-bit unsigned immediate, in the range 0 to 15.

5.63.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.64 DRPS

Debug restore process state.

5.64.1 Syntax

```
DRPS
```

5.64.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.65 DSB

Data synchronization barrier.

5.65.1 Syntax

```
DSB  option|#imm
```

Where:

option Is a barrier option name. The options in the following table are ordered in decreasing scope of the shareability domain. ARM recommends that you use the option names in preference to the equivalent *#imm* values.

Table 5-7 Data synchronization barrier options

option	*imm*	Ordered Accesses (before-after)	Shareability domain
LD	13	Load-Load, Load-Store	
ST	14	Store-Store	Full system
SY	15	Any-Any	
OSHLD	1	Load-Load, Load-Store	
OSHST	2	Store-Store	Outer shareable
OSH	3	Any-Any	
ISHLD	9	Load-Load, Load-Store	
ISHST	10	Store-Store	Inner shareable
ISH	11	Any-Any	
NSHLD	5	Load-Load, Load-Store	
NSHST	6	Store-Store	Non-shareable
NSH	7	Any-Any	

imm Is a 4-bit unsigned immediate, in the range 0 to 15.

5.65.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.66 EON (shifted register)

Bitwise exclusive OR NOT (shifted register).

5.66.1 Syntax

```
EON  Wd, Wn, Wm{, shift #amount}    ; 32-bit general registers
EON  Xd, Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers

Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers

Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift to be applied to the final source, defaulting to LSL, and can be one of LSL, LSR, ASR, or ROR.

5.66.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.67 EOR (immediate)

Bitwise exclusive OR (immediate).

5.67.1 Syntax

```
EOR  Wd|WSP, Wn, #imm    ; 32-bit general registers
EOR  Xd|SP, Xn, #imm     ; 64-bit general registers
```

Where:

Wd|WSP Is the 32-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Xd|SP Is the 64-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

imm Is the bitmask immediate. Such an immediate is a 32-bit or 64-bit pattern viewed as a vector of identical elements of size *e* = 2, 4, 8, 16, 32, or 64 bits. Each element contains the same sub-pattern: a single run of 1 to *e*-1 non-zero bits, rotated by 0 to *e*-1 bits. This mechanism can generate 5,334 unique 64-bit patterns (as 2,667 pairs of pattern and their bitwise inverse). Because the all-zeros and all-ones values cannot be described in this way, the assembler generates an error message.

Note

Logical immediate instructions do not set the condition flags, but interesting results can usually directly control a `CBZ`, `CBNZ`, `TBZ`, or `TBNZ` conditional branch.

5.67.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.68 EOR (shifted register)

Bitwise exclusive OR (shifted register).

5.68.1 Syntax

```
EOR  Wd, Wn, Wm{, shift #amount}    ; 32-bit general registers
EOR  Xd, Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers

Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers

Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift to be applied to the final source, defaulting to LSL, and can be one of LSL, LSR, ASR, or ROR.

5.68.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.69 ERET

Returns from an exception. It restores the processor state based on SPSR_EL*n* and branches to ELR_EL*n*, where *n* is the current exception level.

5.69.1 Syntax

```
ERET
```

5.69.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.70 EXTR

Extract register from pair of registers.

This instruction is used by the alias ROR (immediate).

5.70.1 Syntax

```
EXTR  Wd, Wn, Wm, #lsb    ; 32-bit general registers
EXTR  Xd, Xn, Xm, #lsb    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

lsb The value depends on the instruction variant:

32-bit general registers

Is the least significant bit position from which to extract, in the range 0 to 31.

64-bit general registers

Is the least significant bit position from which to extract, in the range 0 to 63.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

5.70.2 See also

Reference

- *ROR (immediate)* on page 5-123.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.71 HINT

Hint instruction.

This instruction is used by the aliases:

- `NOP`.
- `SEVL`.
- `SEV`.
- `WFE`.
- `WFI`.
- `YIELD`.

5.71.1 Syntax

`HINT #imm`

Where:

imm Is a 7-bit unsigned immediate, in the range 0 to 127.

5.71.2 See also

Reference

- *NOP* on page 5-114.
- *SEVL* on page 5-134.
- *SEV* on page 5-133.
- *WFE* on page 5-172.
- *WFI* on page 5-173.
- *YIELD* on page 5-174.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.72 HLT

External debug breakpoint.

5.72.1 Syntax

```
HLT  #imm
```

Where:

imm Is a 16-bit unsigned immediate, in the range 0 to 65535.

5.72.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.73 HVC

Hypervisor call to allow OS code to call the Hypervisor. It generates an exception targeting exception level 2 (EL2).

5.73.1 Syntax

```
HVC  #imm
```

Where:

imm Is a 16-bit unsigned immediate, in the range 0 to 65535. This value is made available to the handler in the Exception Syndrome Register.

5.73.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.74 IC

Instruction cache operation.

This instruction is an alias of SYS.

5.74.1 Syntax

`IC  ic_op{, Xt}`

Equivalent to `SYS #op1, Cn, Cm, #op2{, Xt}`

Where:

ic_op Is an IC operation name, as listed for the IC system operation group specified by the parameters *op1*, *Cn*, *Cm*, and *op2*.

op1 Is a 3-bit unsigned immediate, in the range 0 to 7.

Cn Is a name C*n*, with *n* in the range 0 to 15.

Cm Is a name C*m*, with *m* in the range 0 to 15.

op2 Is a 3-bit unsigned immediate, in the range 0 to 7.

Xt Is the 64-bit name of the optional general-purpose source register, defaulting to 31.

5.74.2 See also

Reference

- *SYS* on page 5-153.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.75 ISB

Instruction synchronization barrier.

5.75.1 Syntax

```
ISB  {option|#imm}
```

Where:

option Is the barrier option name SY.

imm Is an optional 4-bit unsigned immediate, in the range 0 to 15, defaulting to 15.

5.75.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.76 LSL (register)

Logical shift left (register).

This instruction is an alias of LSLV.

5.76.1 Syntax

```
LSL  Wd, Wn, Wm    ; 32-bit general registers
```

Equivalent to `LSLV Wd, Wn, Wm`

```
LSL  Xd, Xn, Xm    ; 64-bit general registers
```

Equivalent to `LSLV Xd, Xn, Xm`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 31 in its bottom 5 bits.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 63 in its bottom 6 bits.

5.76.2 See also

Reference

- *LSLV* on page 5-88.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.77 LSL (immediate)

Logical shift left (immediate).

This instruction is an alias of UBFM.

5.77.1 Syntax

```
LSL  Wd, Wn, #shift    ; 32-bit general registers
```

Equivalent to `UBFM Wd, Wn, #(-shift MOD 32), #(31-shift)`

```
LSL  Xd, Xn, #shift    ; 64-bit general registers
```

Equivalent to `UBFM Xd, Xn, #(-shift MOD 64), #(63-shift)`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

shift The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31.

64-bit general registers
Is the shift amount, in the range 0 to 63.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.77.2 See also

Reference

- *UBFM* on page 5-161.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.78 LSLV

Logical shift left variable.

This instruction is used by the alias LSL (register).

5.78.1 Syntax

```
LSLV  Wd, Wn, Wm    ; 32-bit general registers
LSLV  Xd, Xn, Xm    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 31 in its bottom 5 bits.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 63 in its bottom 6 bits.

5.78.2 See also

Reference

- *LSL (register)* on page 5-86.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.79 LSR (register)

Logical shift right (register).

This instruction is an alias of LSRV.

5.79.1 Syntax

```
LSR  Wd, Wn, Wm    ; 32-bit general registers
```

Equivalent to `LSRV Wd, Wn, Wm`

```
LSR  Xd, Xn, Xm    ; 64-bit general registers
```

Equivalent to `LSRV Xd, Xn, Xm`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 31 in its bottom 5 bits.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 63 in its bottom 6 bits.

5.79.2 See also

Reference

- *LSRV* on page 5-91.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.80 LSR (immediate)

Logical shift right (immediate).

This instruction is an alias of UBFM.

5.80.1 Syntax

```
LSR  Wd, Wn, #shift    ; 32-bit general registers
```

Equivalent to `UBFM Wd, Wn, #shift, #31`

```
LSR  Xd, Xn, #shift    ; 64-bit general registers
```

Equivalent to `UBFM Xd, Xn, #shift, #63`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

shift The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31.

64-bit general registers
Is the shift amount, in the range 0 to 63.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.80.2 See also

Reference

- *UBFM* on page 5-161.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.81 LSRV

Logical shift right variable.

This instruction is used by the alias LSR (register).

5.81.1 Syntax

```
LSRV  Wd, Wn, Wm    ; 32-bit general registers
LSRV  Xd, Xn, Xm    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 31 in its bottom 5 bits.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 63 in its bottom 6 bits.

5.81.2 See also

Reference

- *LSR (register)* on page 5-89.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.82 MADD

Multiply-add.

This instruction is used by the alias `MUL`.

5.82.1 Syntax

```
MADD  Wd, Wn, Wm, Wa    ; 32-bit general registers
MADD  Xd, Xn, Xm, Xa    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

Wa Is the 32-bit name of the third general-purpose source register holding the addend.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

Xa Is the 64-bit name of the third general-purpose source register holding the addend.

5.82.2 See also

Reference

- *MUL* on page 5-108.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.83 MNEG

Multiply-negate.

This instruction is an alias of `MSUB`.

5.83.1 Syntax

```
MNEG  Wd, Wn, Wm    ; 32-bit general registers
```

Equivalent to `MSUB Wd, Wn, Wm, WZR`

```
MNEG  Xd, Xn, Xm    ; 64-bit general registers
```

Equivalent to `MSUB Xd, Xn, Xm, XZR`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

5.83.2 See also

Reference

- *MSUB* on page 5-107.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.84 MOV (to or from SP)

Move between register and stack pointer.

This instruction is an alias of ADD (immediate).

5.84.1 Syntax

```
MOV  Wd|WSP, Wn|WSP    ; 32-bit general registers
```

Equivalent to `ADD Wd|WSP, Wn|WSP, #0`

```
MOV  Xd|SP, Xn|SP    ; 64-bit general registers
```

Equivalent to `ADD Xd|SP, Xn|SP, #0`

Where:

Wd|WSP Is the 32-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Wn|WSP Is the 32-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

Xd|SP Is the 64-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Xn|SP Is the 64-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

5.84.2 See also

Reference

- *ADD (immediate)* on page 5-13.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.85 MOV (inverted wide immediate)

Move inverted 16-bit immediate to register.

This instruction is an alias of `MOVN`.

5.85.1 Syntax

```
MOV  Wd, #imm    ; 32-bit general registers
```

Equivalent to `MOVN Wd, #imm16, LSL #shift`

```
MOV  Xd, #imm    ; 64-bit general registers
```

Equivalent to `MOVN Xd, #imm16, LSL #shift`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

imm The value depends on the instruction variant:

32-bit general registers
Is a 32-bit immediate.

64-bit general registers
Is a 64-bit immediate.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

5.85.2 See also

Reference

- *MOVN* on page 5-102.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.86 MOV (wide immediate)

Move 16-bit immediate to register.

This instruction is an alias of MOVZ.

5.86.1 Syntax

```
MOV  Wd, #imm    ; 32-bit general registers
```

Equivalent to `MOVZ Wd, #imm16, LSL #shift`

```
MOV  Xd, #imm    ; 64-bit general registers
```

Equivalent to `MOVZ Xd, #imm16, LSL #shift`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

imm The value depends on the instruction variant:

32-bit general registers
Is a 32-bit immediate.

64-bit general registers
Is a 64-bit immediate.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

5.86.2 See also

Reference

- *MOVZ* on page 5-103.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.87 MOV (bitmask immediate)

Move bitmask immediate to register.

This instruction is an alias of ORR (immediate).

5.87.1 Syntax

```
MOV  Wd|WSP, #imm    ; 32-bit general registers
```

Equivalent to `ORR Wd|WSP, WZR, #imm`

```
MOV  Xd|SP, #imm    ; 64-bit general registers
```

Equivalent to `ORR Xd|SP, XZR, #imm`

Where:

Wd|WSP Is the 32-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Xd|SP Is the 64-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

imm Is the bitmask immediate. Such an immediate is a 32-bit or 64-bit pattern viewed as a vector of identical elements of size $e = 2, 4, 8, 16, 32$, or 64 bits. Each element contains the same sub-pattern: a single run of 1 to e-1 non-zero bits, rotated by 0 to e-1 bits. This mechanism can generate 5,334 unique 64-bit patterns (as 2,667 pairs of pattern and their bitwise inverse). Because the all-zeros and all-ones values cannot be described in this way, the assembler generates an error message.

Note

Logical immediate instructions do not set the condition flags, but interesting results can usually directly control a CBZ, CBNZ, TBZ, or TBNZ conditional branch.

5.87.2 See also

Reference

- *ORR (immediate)* on page 5-116.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.88 MOV (register)

Move register to register.

This instruction is an alias of ORR (shifted register).

5.88.1 Syntax

```
MOV  Wd, Wm    ; 32-bit general registers
```

Equivalent to `ORR Wd, WZR, Wm`

```
MOV  Xd, Xm    ; 64-bit general registers
```

Equivalent to `ORR Xd, XZR, Xm`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wm Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.88.2 See also

Reference

- *ORR (shifted register)* on page 5-117.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.89 MOVK

Move 16-bit immediate into register, keeping other bits unchanged.

5.89.1 Syntax

```
MOVK  Wd, #imm{, LSL #shift}    ; 32-bit general registers
MOVK  Xd, #imm{, LSL #shift}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

shift Is the amount by which to shift the immediate left:

32-bit general registers
Can be 0 or 16.

64-bit general registers
Can be 0, 16, 32 or 48.

Defaults to zero.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

imm Is the 16-bit unsigned immediate, in the range 0 to 65535.

5.89.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.90 MOVL pseudo-instruction

Load a register with either:

- A 32-bit or 64-bit immediate value.
- Any address.

`MOVL` generates either two or four instructions. If a *Wd* register is specified, `MOVL` generates a `MOV`, `MOVK` pair. If an *Xd* register is specified, `MOVL` generates a `MOV` followed by three `MOVK` instructions. If the assembler can load the register using a single `MOV` instruction, it additionally generates either one or three `NOP`s.

5.90.1 Syntax

`MOVL` *Wd*,*expr*

`MOVL` *Xd*,*expr*

where:

Wd Is the register to load with a 32-bit value.

Xd Is the register to load with a 64-bit value.

expr Can be any one of the following:

symbol	A label in this or another program area.
#constant	Any 32-bit or 64-bit immediate value.
symbol + constant	A label plus a 32-bit or 64-bit immediate value.

5.90.2 Usage

Use the `MOVL` pseudo-instruction to:

- Generate literal constants when an immediate value cannot be generated in a single instruction.
- Load a PC-relative or external address into a register. The address remains valid regardless of where the linker places the ELF section containing the `MOVL`.

  Note

  An address loaded in this way is fixed at link time, so the code is not position-independent.

5.90.3 Examples

```
MOVL w3, #0xABCDEF12   ; loads 0xABCDEF12 into w3
MOVL x1, Trigger+12    ; loads the address that is 12 bytes higher than
                       ; the address Trigger into x1
```

5.90.4 See also

Reference

- *MOV (bitmask immediate)* on page 5-97.
- *MOVK* on page 5-99.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.91 MOVN

Move inverse of shifted 16-bit immediate to register.

This instruction is used by the alias `MOV` (inverted wide immediate).

5.91.1 Syntax

```
MOVN  Wd, #imm{, LSL #shift}    ; 32-bit general registers
MOVN  Xd, #imm{, LSL #shift}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

shift Is the amount by which to shift the immediate left:

32-bit general registers
Can be 0 or 16.

64-bit general registers
Can be 0, 16, 32 or 48.

Defaults to zero.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

imm Is the 16-bit unsigned immediate, in the range 0 to 65535.

5.91.2 See also

Reference

- *MOV (inverted wide immediate)* on page 5-95.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.92 MOVZ

Move shifted 16-bit immediate to register.

This instruction is used by the alias `MOV` (wide immediate).

5.92.1 Syntax

```
MOVZ  Wd, #imm{, LSL #shift}    ; 32-bit general registers
MOVZ  Xd, #imm{, LSL #shift}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

shift Is the amount by which to shift the immediate left:

32-bit general registers
Can be 0 or 16.

64-bit general registers
Can be 0, 16, 32 or 48.

Defaults to zero.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

imm Is the 16-bit unsigned immediate, in the range 0 to 65535.

5.92.2 See also

Reference

- *MOV (wide immediate)* on page 5-96.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.93 MRS

Move from system register.

5.93.1 Syntax

```
MRS  Xt, systemreg
```

Where:

Xt Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

systemreg Is a system register name.

5.93.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.94 MSR (immediate)

Move immediate to process state field.

5.94.1 Syntax

```
MSR  pstatefield, #imm
```

Where:

pstatefield Is a PSTATE field name, and can be one of `SPSel`, `DAIFSet` or `DAIFClr`.

imm Is a 4-bit unsigned immediate, in the range 0 to 15.

5.94.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.95 MSR (register)

Move to system register.

5.95.1 Syntax

```
MSR  systemreg, Xt
```

Where:

systemreg Is a system register name.

Xt Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.95.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.96 MSUB

Multiply-subtract.

This instruction is used by the alias `MNEG`.

5.96.1 Syntax

```
MSUB  Wd, Wn, Wm, Wa    ; 32-bit general registers
MSUB  Xd, Xn, Xm, Xa    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

Wa Is the 32-bit name of the third general-purpose source register holding the minuend.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

Xa Is the 64-bit name of the third general-purpose source register holding the minuend.

5.96.2 See also

Reference

- *MNEG* on page 5-93.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.97 MUL

Multiply.

This instruction is an alias of MADD.

5.97.1 Syntax

```
MUL  Wd, Wn, Wm    ; 32-bit general registers
```

Equivalent to `MADD Wd, Wn, Wm, WZR`

```
MUL  Xd, Xn, Xm    ; 64-bit general registers
```

Equivalent to `MADD Xd, Xn, Xm, XZR`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

5.97.2 See also

Reference

- *MADD* on page 5-92.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.98 MVN

Bitwise NOT (shifted register).

This instruction is an alias of ORN (shifted register).

5.98.1 Syntax

```
MVN  Wd, Wm{, shift #amount}    ; 32-bit general registers
```

Equivalent to `ORN Wd, WZR, Wm{, shift #amount}`

```
MVN  Xd, Xm{, shift #amount}    ; 64-bit general registers
```

Equivalent to `ORN Xd, XZR, Xm{, shift #amount}`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wm Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers
Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

shift Is the optional shift to be applied to the final source, defaulting to LSL, and can be one of LSL, LSR, ASR, or ROR.

5.98.2 See also

Reference

- *ORN (shifted register)* on page 5-115.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.99 NEG (shifted register)

Negate.

This instruction is an alias of SUB (shifted register).

5.99.1 Syntax

```
NEG  Wd, Wm{, shift #amount}    ; 32-bit general registers
```

Equivalent to `SUB Wd, WZR, Wm {, shift #amount}`

```
NEG  Xd, Xm{, shift #amount}    ; 64-bit general registers
```

Equivalent to `SUB Xd, XZR, Xm {, shift #amount}`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wm Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers
Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

shift Is the optional shift type to be applied to the second source operand, defaulting to LSL, and can be one of LSL, LSR, or ASR.

5.99.2 See also

Reference

- *SUB (shifted register)* on page 5-144.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.100 NEGS

Negate, setting the condition flags.

This instruction is an alias of SUBS (shifted register).

5.100.1 Syntax

```
NEGS  Wd, Wm{, shift #amount}    ; 32-bit general registers
```

Equivalent to `SUBS Wd, WZR, Wm {, shift #amount}`

```
NEGS  Xd, Xm{, shift #amount}    ; 64-bit general registers
```

Equivalent to `SUBS Xd, XZR, Xm {, shift #amount}`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wm Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers
Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

shift Is the optional shift type to be applied to the second source operand, defaulting to LSL, and can be one of LSL, LSR, or ASR.

5.100.2 See also

Reference

- *SUBS (shifted register)* on page 5-148.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.101 NGC

Negate with carry.

This instruction is an alias of SBC.

5.101.1 Syntax

```
NGC  Wd, Wm    ; 32-bit general registers
```

Equivalent to `SBC Wd, WZR, Wm`

```
NGC  Xd, Xm    ; 64-bit general registers
```

Equivalent to `SBC Xd, XZR, Xm`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wm Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.101.2 See also

Reference

- *SBC* on page 5-126.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.102 NGCS

Negate with carry, setting the condition flags.

This instruction is an alias of SBCS.

5.102.1 Syntax

```
NGCS  Wd, Wm    ; 32-bit general registers
```

Equivalent to `SBCS Wd, WZR, Wm`

```
NGCS  Xd, Xm    ; 64-bit general registers
```

Equivalent to `SBCS Xd, XZR, Xm`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wm Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.102.2 See also

Reference

- *SBCS* on page 5-127.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.103 NOP

No operation.

This instruction is an alias of `HINT`.

5.103.1 See also

Reference

- *HINT* on page 5-81.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.104 ORN (shifted register)

Bitwise inclusive OR NOT (shifted register).

This instruction is used by the alias `MVN`.

5.104.1 Syntax

```
ORN  Wd, Wn, Wm{, shift #amount}    ; 32-bit general registers
ORN  Xd, Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers
Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift to be applied to the final source, defaulting to `LSL`, and can be one of `LSL`, `LSR`, `ASR`, or `ROR`.

5.104.2 See also

Reference

- *MVN* on page 5-109.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.105 ORR (immediate)

Bitwise inclusive OR (immediate).

This instruction is used by the alias MOV (bitmask immediate).

5.105.1 Syntax

```
ORR  Wd|WSP, Wn, #imm    ; 32-bit general registers
ORR  Xd|SP, Xn, #imm     ; 64-bit general registers
```

Where:

Wd|WSP Is the 32-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Xd|SP Is the 64-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

imm Is the bitmask immediate. Such an immediate is a 32-bit or 64-bit pattern viewed as a vector of identical elements of size $e = 2, 4, 8, 16, 32$, or 64 bits. Each element contains the same sub-pattern: a single run of 1 to e-1 non-zero bits, rotated by 0 to e-1 bits. This mechanism can generate 5,334 unique 64-bit patterns (as 2,667 pairs of pattern and their bitwise inverse). Because the all-zeros and all-ones values cannot be described in this way, the assembler generates an error message.

Note

Logical immediate instructions do not set the condition flags, but interesting results can usually directly control a CBZ, CBNZ, TBZ, or TBNZ conditional branch.

5.105.2 See also

Reference

- *MOV (bitmask immediate)* on page 5-97.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.106 ORR (shifted register)

Bitwise inclusive OR (shifted register).

This instruction is used by the alias `MOV` (register).

5.106.1 Syntax

```
ORR  Wd, Wn, Wm{, shift #amount}    ; 32-bit general registers
ORR  Xd, Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers
Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift to be applied to the final source, defaulting to `LSL`, and can be one of `LSL`, `LSR`, `ASR`, or `ROR`.

5.106.2 See also

Reference

- *MOV (register)* on page 5-98.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.107 RBIT

Reverse bit order.

5.107.1 Syntax

```
RBIT  Wd, Wn    ; 32-bit general registers
RBIT  Xd, Xn    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.107.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.108 RET

Return from subroutine, branches unconditionally to an address in a register, with a hint that this is a subroutine return.

5.108.1 Syntax

```
RET  {Xn}
```

Where:

Xn Is the 64-bit name of the general-purpose register holding the address to be branched to, in the range 0 to 31. Defaults to X30 if absent.

5.108.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.109 REV

Reverse bytes.

5.109.1 Syntax

```
REV  Wd, Wn    ; 32-bit general registers
REV  Xd, Xn    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.109.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.110 REV16

Reverse bytes in 16-bit halfwords.

5.110.1 Syntax

```
REV16  Wd, Wn    ; 32-bit general registers
REV16  Xd, Xn    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.110.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.111 REV32

Reverse bytes in 32-bit words.

5.111.1 Syntax

```
REV32  Xd, Xn
```

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.111.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.112 ROR (immediate)

Rotate right (immediate).

This instruction is an alias of EXTR.

5.112.1 Syntax

```
ROR  Wd, Ws, #shift    ; 32-bit general registers
```

Equivalent to `EXTR Wd, Ws, Ws, #shift`

```
ROR  Xd, Xs, #shift    ; 64-bit general registers
```

Equivalent to `EXTR Xd, Xs, Xs, #shift`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Ws Is the 32-bit name of the general-purpose source register in the range 0 to 31.

shift The value depends on the instruction variant:

32-bit general registers
Is the amount by which to rotate, in the range 0 to 31.

64-bit general registers
Is the amount by which to rotate, in the range 0 to 63.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xs Is the 64-bit name of the general-purpose source register in the range 0 to 31.

5.112.2 See also

Reference

- *EXTR* on page 5-80.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.113 ROR (register)

Rotate right (register).

This instruction is an alias of RORV.

5.113.1 Syntax

```
ROR  Wd, Wn, Wm    ; 32-bit general registers
```

Equivalent to `RORV Wd, Wn, Wm`

```
ROR  Xd, Xn, Xm    ; 64-bit general registers
```

Equivalent to `RORV Xd, Xn, Xm`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 31 in its bottom 5 bits.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 63 in its bottom 6 bits.

5.113.2 See also

Reference

- *RORV* on page 5-125.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.114 RORV

Rotate right variable.

This instruction is used by the alias `ROR` (register).

5.114.1 Syntax

```
RORV  Wd, Wn, Wm    ; 32-bit general registers
RORV  Xd, Xn, Xm    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 31 in its bottom 5 bits.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register in the range 0 to 31. It holds a shift amount from 0 to 63 in its bottom 6 bits.

5.114.2 See also

Reference

- *ROR (register)* on page 5-124.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.115 SBC

Subtract with carry.

This instruction is used by the alias NGC.

5.115.1 Syntax

```
SBC  Wd, Wn, Wm    ; 32-bit general registers
SBC  Xd, Xn, Xm    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

5.115.2 See also

Reference

- *NGC* on page 5-112.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.116 SBCS

Subtract with carry, setting the condition flags.

This instruction is used by the alias NGCS.

5.116.1 Syntax

```
SBCS  Wd, Wn, Wm    ; 32-bit general registers
SBCS  Xd, Xn, Xm    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

5.116.2 See also

Reference

- *NGCS* on page 5-113.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.117 SBFIZ

Signed bitfield insert in zero, with sign replication to left and zeros to right.

This instruction is an alias of SBFM.

5.117.1 Syntax

```
SBFIZ  Wd, Wn, #lsb, #width    ; 32-bit general registers
```

Equivalent to `SBFM Wd, Wn, #(-lsb MOD 32), #(width-1)`

```
SBFIZ  Xd, Xn, #lsb, #width    ; 64-bit general registers
```

Equivalent to `SBFM Xd, Xn, #(-lsb MOD 64), #(width-1)`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

lsb The value depends on the instruction variant:

32-bit general registers
Is the bit number of the lsb of the source bitfield, in the range 0 to 31.

64-bit general registers
Is the bit number of the lsb of the source bitfield, in the range 0 to 63.

width The value depends on the instruction variant:

32-bit general registers
Is the width of the bitfield, in the range 1 to 32-*lsb*.

64-bit general registers
Is the width of the bitfield, in the range 1 to 64-*lsb*.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.117.2 See also

Reference

- *SBFM* on page 5-129.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.118 SBFM

Signed bitfield move, with sign replication to left and zeros to right.

This instruction is used by the aliases:

- ASR (immediate).
- SBFIZ.
- SBFX.
- SXTB.
- SXTH.
- SXTW.

5.118.1 Syntax

```
SBFM  Wd, Wn, #immr, #imms    ; 32-bit general registers
SBFM  Xd, Xn, #immr, #imms    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

immr The value depends on the instruction variant:

32-bit general registers
Is the right rotate amount, in the range 0 to 31.

64-bit general registers
Is the right rotate amount, in the range 0 to 63.

imms The value depends on the instruction variant:

32-bit general registers
Is the leftmost bit number to be moved from the source, in the range 0 to 31.

64-bit general registers
Is the leftmost bit number to be moved from the source, in the range 0 to 63.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.118.2 See also

Reference

- *ASR (immediate)* on page 5-27.
- *SBFIZ* on page 5-128.
- *SBFX* on page 5-131.
- *SXTB* on page 5-150.
- *SXTH* on page 5-151.
- *SXTW* on page 5-152.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.119 SBFX

Signed bitfield extract.

This instruction is an alias of SBFM.

5.119.1 Syntax

```
SBFX  Wd, Wn, #lsb, #width    ; 32-bit general registers
```

Equivalent to `SBFM Wd, Wn, #lsb, #(lsb+width-1)`

```
SBFX  Xd, Xn, #lsb, #width    ; 64-bit general registers
```

Equivalent to `SBFM Xd, Xn, #lsb, #(lsb+width-1)`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

lsb The value depends on the instruction variant:

32-bit general registers
Is the bit number of the lsb of the source bitfield, in the range 0 to 31.

64-bit general registers
Is the bit number of the lsb of the source bitfield, in the range 0 to 63.

width The value depends on the instruction variant:

32-bit general registers
Is the width of the bitfield, in the range 1 to 32-*lsb*.

64-bit general registers
Is the width of the bitfield, in the range 1 to 64-*lsb*.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.119.2 See also

Reference

- *SBFM* on page 5-129.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.120 SDIV

Signed divide.

5.120.1 Syntax

```
SDIV  Wd, Wn, Wm    ; 32-bit general registers
SDIV  Xd, Xn, Xm    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

5.120.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.121 SEV

Send event.

This instruction is an alias of `HINT`.

5.121.1 See also

Reference

- *HINT* on page 5-81.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.122 SEVL

Send event locally.

This instruction is an alias of `HINT`.

5.122.1 See also

Reference

- *HINT* on page 5-81.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.123 SMADDL

Signed multiply-add long.

This instruction is used by the alias `SMULL`.

5.123.1 Syntax

`SMADDL` *Xd*, *Wn*, *Wm*, *Xa*

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

Xa Is the 64-bit name of the third general-purpose source register holding the addend.

5.123.2 See also

Reference

- *SMULL* on page 5-140.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.124 SMC

Supervisor call to allow OS or Hypervisor code to call the Secure Monitor. It generates an exception targeting exception level 3 (EL3).

5.124.1 Syntax

```
SMC  #imm
```

Where:

imm Is a 16-bit unsigned immediate, in the range 0 to 65535. This value is made available to the handler in the Exception Syndrome Register.

5.124.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.125 SMNEGL

Signed multiply-negate long.

This instruction is an alias of SMSUBL.

5.125.1 Syntax

SMNEGL *Xd*, *Wn*, *Wm*

Equivalent to SMSUBL *Xd*, *Wn*, *Wm*, XZR

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

5.125.2 See also

Reference

- *SMSUBL* on page 5-138.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.126 SMSUBL

Signed multiply-subtract long.

This instruction is used by the alias `SMNEGL`.

5.126.1 Syntax

`SMSUBL` *Xd*, *Wn*, *Wm*, *Xa*

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

Xa Is the 64-bit name of the third general-purpose source register holding the minuend.

5.126.2 See also

Reference

- *SMNEGL* on page 5-137.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.127 SMULH

Signed multiply high.

5.127.1 Syntax

```
SMULH  Xd, Xn, Xm
```

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

5.127.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.128 SMULL

Signed multiply long.

This instruction is an alias of SMADDL.

5.128.1 Syntax

SMULL *Xd*, *Wn*, *Wm*

Equivalent to SMADDL *Xd*, *Wn*, *Wm*, XZR

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

5.128.2 See also

Reference

- *SMADDL* on page 5-135.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.129 SUB (extended register)

Subtract (extended register).

5.129.1 Syntax

```
SUB  Wd|WSP, Wn|WSP, Wm{, extend {#amount}}    ; 32-bit general registers
SUB  Xd|SP, Xn|SP, Rm{, extend {#amount}}    ; 64-bit general registers
```

Where:

Wd|WSP Is the 32-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Wn|WSP Is the 32-bit name of the first source general-purpose register or stack pointer, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

extend Is the extension to be applied to the second source operand:

32-bit general registers

Can be one of UXTB, UXTH, LSL|UXTW, UXTX, SXTB, SXTH, SXTW or SXTX.

If *Rd* or *Rn* is WSP then LSL is preferred rather than UXTW, and can be omitted when *amount* is 0. In all other cases *extend* is required and must be UXTW rather than LSL.

64-bit general registers

Can be one of UXTB, UXTH, UXTW, LSL|UXTX, SXTB, SXTH, SXTW or SXTX.

If *Rd* or *Rn* is SP then LSL is preferred rather than UXTX, and can be omitted when *amount* is 0. In all other cases *extend* is required and must be UXTX rather than LSL.

Xd|SP Is the 64-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Xn|SP Is the 64-bit name of the first source general-purpose register or stack pointer, in the range 0 to 31.

R Is a width specifier, and can be either W or X.

m Is the number of the second general-purpose source register, in the range 0 to 30, or the name ZR (31).

amount Is the left shift amount to be applied after extension in the range 0 to 4, defaulting to 0. It must be absent when *extend* is absent, is required when *extend* is LSL, and is optional when *extend* is present but not LSL.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.129.2 Usage

The following table shows valid specifier combinations:

Table 5-8 SUB (64-bit general registers) specifier combinations

R	*extend*
W	SXTB
W	SXTH
W	SXTW
W	UXTB
W	UXTH
W	UXTW
X	LSL|UXTX
X	SXTX

5.129.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.130 SUB (immediate)

Subtract (immediate).

5.130.1 Syntax

```
SUB  Wd|WSP, Wn|WSP, #imm{, shift}    ; 32-bit general registers
SUB  Xd|SP, Xn|SP, #imm{, shift}    ; 64-bit general registers
```

Where:

Wd|WSP Is the 32-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Wn|WSP Is the 32-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

Xd|SP Is the 64-bit name of the destination general-purpose register or stack pointer, in the range 0 to 31.

Xn|SP Is the 64-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

imm Is an unsigned immediate, in the range 0 to 4095.

shift Is the optional left shift to apply to the immediate, defaulting to `LSL #0`, and can be either `LSL #0` or `LSL #12`.

5.130.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.131 SUB (shifted register)

Subtract (shifted register).

This instruction is used by the alias NEG (shifted register).

5.131.1 Syntax

```
SUB  Wd, Wn, Wm{, shift #amount}    ; 32-bit general registers
SUB  Xd, Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers
Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift type to be applied to the second source operand, defaulting to LSL, and can be one of LSL, LSR, or ASR.

5.131.2 See also

Reference

- *NEG (shifted register)* on page 5-110.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.132 SUBS (extended register)

Subtract (extended register), setting the condition flags.

This instruction is used by the alias CMP (extended register).

5.132.1 Syntax

```
SUBS  Wd, Wn|WSP, Wm{, extend {#amount}}    ; 32-bit general registers
SUBS  Xd, Xn|SP, Rm{, extend {#amount}}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn|WSP Is the 32-bit name of the first source general-purpose register or stack pointer, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

extend Is the extension to be applied to the second source operand:

32-bit general registers

Can be one of UXTB, UXTH, LSL|UXTW, UXTX, SXTB, SXTH, SXTW or SXTX.

If *Rn* is WSP then LSL is preferred rather than UXTW, and can be omitted when *amount* is 0. In all other cases *extend* is required and must be UXTW rather than LSL.

64-bit general registers

Can be one of UXTB, UXTH, UXTW, LSL|UXTX, SXTB, SXTH, SXTW or SXTX.

If *Rn* is SP then LSL is preferred rather than UXTX, and can be omitted when *amount* is 0. In all other cases *extend* is required and must be UXTX rather than LSL.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn|SP Is the 64-bit name of the first source general-purpose register or stack pointer, in the range 0 to 31.

R Is a width specifier, and can be either W or X.

m Is the number of the second general-purpose source register, in the range 0 to 30, or the name ZR (31).

amount Is the left shift amount to be applied after extension in the range 0 to 4, defaulting to 0. It must be absent when *extend* is absent, is required when *extend* is LSL, and is optional when *extend* is present but not LSL.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.132.2 Usage

The following table shows valid specifier combinations:

Table 5-9 SUBS (64-bit general registers) specifier combinations

R	*extend*
W	SXTB
W	SXTH
W	SXTW
W	UXTB
W	UXTH
W	UXTW
X	LSL|UXTX
X	SXTX

5.132.3 See also

Reference

- *CMP (extended register)* on page 5-56.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.133 SUBS (immediate)

Subtract (immediate), setting the condition flags.

This instruction is used by the alias CMP (immediate).

5.133.1 Syntax

```
SUBS  Wd, Wn|WSP, #imm{, shift}    ; 32-bit general registers
SUBS  Xd, Xn|SP, #imm{, shift}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn|WSP Is the 32-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn|SP Is the 64-bit name of the source general-purpose register or stack pointer, in the range 0 to 31.

imm Is an unsigned immediate, in the range 0 to 4095.

shift Is the optional left shift to apply to the immediate, defaulting to LSL #0, and can be either LSL #0 or LSL #12.

5.133.2 See also

Reference

- *CMP (immediate)* on page 5-58.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.134 SUBS (shifted register)

Subtract (shifted register), setting the condition flags.

This instruction is used by the aliases:

- CMP (shifted register).
- NEGS.

5.134.1 Syntax

```
SUBS  Wd, Wn, Wm{, shift #amount}    ; 32-bit general registers
SUBS  Xd, Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers

Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers

Is the shift amount, in the range 0 to 63, defaulting to 0.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift type to be applied to the second source operand, defaulting to LSL, and can be one of LSL, LSR, or ASR.

5.134.2 See also

Reference

- *CMP (shifted register)* on page 5-59.
- *NEGS* on page 5-111.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.135 SVC

Supervisor call to allow application code to call the OS. It generates an exception targeting exception level 1 (EL1).

5.135.1 Syntax

```
SVC  #imm
```

Where:

imm Is a 16-bit unsigned immediate, in the range 0 to 65535. This value is made available to the handler in the Exception Syndrome Register.

5.135.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.136 SXTB

Signed extend byte.

This instruction is an alias of SBFM.

5.136.1 Syntax

```
SXTB  Wd, Wn    ; 32-bit general registers
```

Equivalent to `SBFM Wd, Wn, #0, #7`

```
SXTB  Xd, Wn    ; 64-bit general registers
```

Equivalent to `SBFM Xd, Xn, #0, #7`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

5.136.2 See also

Reference

- *SBFM* on page 5-129.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.137 SXTH

Signed extend halfword.

This instruction is an alias of SBFM.

5.137.1 Syntax

```
SXTH  Wd, Wn    ; 32-bit general registers
```

Equivalent to `SBFM Wd, Wn, #0, #15`

```
SXTH  Xd, Wn    ; 64-bit general registers
```

Equivalent to `SBFM Xd, Xn, #0, #15`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

5.137.2 See also

Reference

- *SBFM* on page 5-129.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.138 SXTW

Signed extend word.

This instruction is an alias of SBFM.

5.138.1 Syntax

`SXTW  Xd, Wn`

Equivalent to `SBFM Xd, Xn, #0, #31`

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

5.138.2 See also

Reference

- *SBFM* on page 5-129.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.139 SYS

System instruction.

This instruction is used by the aliases:

- AT.
- DC.
- IC.
- TLBI.

5.139.1 Syntax

```
SYS  #op1, Cn, Cm, #op2{, Xt}
```

Where:

op1 Is a 3-bit unsigned immediate, in the range 0 to 7.

Cn Is a name C*n*, with *n* in the range 0 to 15.

Cm Is a name C*m*, with *m* in the range 0 to 15.

op2 Is a 3-bit unsigned immediate, in the range 0 to 7.

Xt Is the 64-bit name of the optional general-purpose source register, defaulting to 31.

5.139.2 See also

Reference

- *AT* on page 5-29.
- *DC* on page 5-69.
- *IC* on page 5-84.
- *TLBI* on page 5-157.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.140 SYSL

System instruction with result.

5.140.1 Syntax

```
SYSL  Xt, #op1, Cn, Cm, #op2
```

Where:

Xt Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

op1 Is a 3-bit unsigned immediate, in the range 0 to 7.

Cn Is a name C*n*, with *n* in the range 0 to 15.

Cm Is a name C*m*, with *m* in the range 0 to 15.

op2 Is a 3-bit unsigned immediate, in the range 0 to 7.

5.140.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.141 TBNZ

Test bit and branch if nonzero to a label at a PC-relative offset, without affecting the condition flags, and with a hint that this is not a subroutine call or return.

5.141.1 Syntax

```
TBNZ  Rt, #imm, label
```

Where:

R Is a width specifier, and can be either `W` or `X`.

In assembler source code an `X` specifier is always permitted, but a `W` specifier is only permitted when the bit number is less than 32.

t Is the number of the general-purpose register to be tested, in the range 0 to 30, or the name ZR (31).

imm Is the bit number to be tested, in the range 0 to 63.

label Is the program label to be conditionally branched to. It is an offset from the address of this instruction, in the range ±32KB.

5.141.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.142 TBZ

Test bit and branch if zero to a label at a PC-relative offset, without affecting the condition flags, and with a hint that this is not a subroutine call or return.

5.142.1 Syntax

```
TBZ  Rt, #imm, label
```

Where:

R Is a width specifier, and can be either `W` or `X`.

In assembler source code an `X` specifier is always permitted, but a `W` specifier is only permitted when the bit number is less than 32.

t Is the number of the general-purpose register to be tested, in the range 0 to 30, or the name ZR (31).

imm Is the bit number to be tested, in the range 0 to 63.

label Is the program label to be conditionally branched to. It is an offset from the address of this instruction, in the range ±32KB.

5.142.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.143 TLBI

TLB invalidate operation.

This instruction is an alias of SYS.

5.143.1 Syntax

`TLBI  tlbi_op{, Xt}`

Equivalent to `SYS #op1, Cn, Cm, #op2{, Xt}`

Where:

op1 Is a 3-bit unsigned immediate, in the range 0 to 7.

Cn Is a name C*n*, with *n* in the range 0 to 15.

Cm Is a name C*m*, with *m* in the range 0 to 15.

op2 Is a 3-bit unsigned immediate, in the range 0 to 7.

tlbi_op Is a TLBI operation name, as listed for the TLBI system operation group specified by the parameters *op1*, *Cn*, *Cm*, and *op2*.

Xt Is the 64-bit name of the optional general-purpose source register, defaulting to 31.

5.143.2 See also

Reference

- *SYS* on page 5-153.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.144 TST (immediate)

Test bits (immediate), setting the condition flags and discarding the result.

This instruction is an alias of ANDS (immediate).

5.144.1 Syntax

```
TST  Wn, #imm    ; 32-bit general registers
```

Equivalent to `ANDS WZR, Wn, #imm`

```
TST  Xn, #imm    ; 64-bit general registers
```

Equivalent to `ANDS XZR, Xn, #imm`

Where:

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

imm Is the bitmask immediate. Such an immediate is a 32-bit or 64-bit pattern viewed as a vector of identical elements of size *e* = 2, 4, 8, 16, 32, or 64 bits. Each element contains the same sub-pattern: a single run of 1 to *e*-1 non-zero bits, rotated by 0 to *e*-1 bits. This mechanism can generate 5,334 unique 64-bit patterns (as 2,667 pairs of pattern and their bitwise inverse). Because the all-zeros and all-ones values cannot be described in this way, the assembler generates an error message.

Note

Logical immediate instructions do not set the condition flags, but interesting results can usually directly control a CBZ, CBNZ, TBZ, or TBNZ conditional branch.

5.144.2 See also

Reference

- *ANDS (immediate)* on page 5-24.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.145 TST (shifted register)

Test bits (shifted register), setting the condition flags and discarding the result.

This instruction is an alias of ANDS (shifted register).

5.145.1 Syntax

```
TST  Wn, Wm{, shift #amount}    ; 32-bit general registers
```

Equivalent to `ANDS WZR, Wn, Wm{, shift #amount}`

```
TST  Xn, Xm{, shift #amount}    ; 64-bit general registers
```

Equivalent to `ANDS XZR, Xn, Xm{, shift #amount}`

Where:

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

amount The value depends on the instruction variant:

32-bit general registers
Is the shift amount, in the range 0 to 31, defaulting to 0.

64-bit general registers
Is the shift amount, in the range 0 to 63, defaulting to 0.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

shift Is the optional shift to be applied to the final source, defaulting to LSL, and can be one of LSL, LSR, ASR, or ROR.

5.145.2 See also

Reference

- *ANDS (shifted register)* on page 5-25.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.146 UBFIZ

Unsigned bitfield insert in zero, with zeros to left and right.

This instruction is an alias of UBFM.

5.146.1 Syntax

```
UBFIZ  Wd, Wn, #lsb, #width    ; 32-bit general registers
```

Equivalent to `UBFM Wd, Wn, #(-lsb MOD 32), #(width-1)`

```
UBFIZ  Xd, Xn, #lsb, #width    ; 64-bit general registers
```

Equivalent to `UBFM Xd, Xn, #(-lsb MOD 64), #(width-1)`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

lsb The value depends on the instruction variant:

32-bit general registers
Is the bit number of the lsb of the source bitfield, in the range 0 to 31.

64-bit general registers
Is the bit number of the lsb of the source bitfield, in the range 0 to 63.

width The value depends on the instruction variant:

32-bit general registers
Is the width of the bitfield, in the range 1 to 32-*lsb*.

64-bit general registers
Is the width of the bitfield, in the range 1 to 64-*lsb*.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.146.2 See also

Reference

- *UBFM* on page 5-161.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.147 UBFM

Unsigned bitfield move, with zeros to left and right.

This instruction is used by the aliases:

- LSL (immediate).
- LSR (immediate).
- UBFIZ.
- UBFX.
- UXTB.
- UXTH.

5.147.1 Syntax

```
UBFM  Wd, Wn, #immr, #imms    ; 32-bit general registers
UBFM  Xd, Xn, #immr, #imms    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

immr The value depends on the instruction variant:

32-bit general registers
Is the right rotate amount, in the range 0 to 31.

64-bit general registers
Is the right rotate amount, in the range 0 to 63.

imms The value depends on the instruction variant:

32-bit general registers
Is the leftmost bit number to be moved from the source, in the range 0 to 31.

64-bit general registers
Is the leftmost bit number to be moved from the source, in the range 0 to 63.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.147.2 See also

Reference

- *LSL (immediate)* on page 5-87.
- *LSR (immediate)* on page 5-90.
- *UBFIZ* on page 5-160.
- *UBFX* on page 5-163.
- *UXTB* on page 5-170.
- *UXTH* on page 5-171.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.148 UBFX

Unsigned bitfield extract.

This instruction is an alias of UBFM.

5.148.1 Syntax

```
UBFX  Wd, Wn, #lsb, #width    ; 32-bit general registers
```

Equivalent to `UBFM Wd, Wn, #lsb, #(lsb+width-1)`

```
UBFX  Xd, Xn, #lsb, #width    ; 64-bit general registers
```

Equivalent to `UBFM Xd, Xn, #lsb, #(lsb+width-1)`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

lsb The value depends on the instruction variant:

32-bit general registers
Is the bit number of the lsb of the source bitfield, in the range 0 to 31.

64-bit general registers
Is the bit number of the lsb of the source bitfield, in the range 0 to 63.

width The value depends on the instruction variant:

32-bit general registers
Is the width of the bitfield, in the range 1 to 32-*lsb*.

64-bit general registers
Is the width of the bitfield, in the range 1 to 64-*lsb*.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

5.148.2 See also

Reference

- *UBFM* on page 5-161.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.149 UDIV

Unsigned divide.

5.149.1 Syntax

```
UDIV  Wd, Wn, Wm    ; 32-bit general registers
UDIV  Xd, Xn, Xm    ; 64-bit general registers
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register, in the range 0 to 31.

5.149.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.150 UMADDL

Unsigned multiply-add long.

This instruction is used by the alias `UMULL`.

5.150.1 Syntax

`UMADDL` *Xd*, *Wn*, *Wm*, *Xa*

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

Xa Is the 64-bit name of the third general-purpose source register holding the addend.

5.150.2 See also

Reference

- *UMULL* on page 5-169.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.151 UMNEGL

Unsigned multiply-negate long.

This instruction is an alias of `UMSUBL`.

5.151.1 Syntax

`UMNEGL` *Xd*, *Wn*, *Wm*

Equivalent to `UMSUBL` *Xd*, *Wn*, *Wm*, `XZR`

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

5.151.2 See also

Reference

- *UMSUBL* on page 5-167.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.152 UMSUBL

Unsigned multiply-subtract long.

This instruction is used by the alias `UMNEGL`.

5.152.1 Syntax

`UMSUBL` *Xd*, *Wn*, *Wm*, *Xa*

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

Xa Is the 64-bit name of the third general-purpose source register holding the minuend.

5.152.2 See also

Reference

- *UMNEGL* on page 5-166.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.153 UMULH

Unsigned multiply high.

5.153.1 Syntax

```
UMULH  Xd, Xn, Xm
```

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Xn Is the 64-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Xm Is the 64-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

5.153.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.154 UMULL

Unsigned multiply long.

This instruction is an alias of UMADDL.

5.154.1 Syntax

```
UMULL  Xd, Wn, Wm
```

Equivalent to `UMADDL Xd, Wn, Wm, XZR`

Where:

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the first general-purpose source register holding the multiplicand, in the range 0 to 31.

Wm Is the 32-bit name of the second general-purpose source register holding the multiplier. The register name can be in the range 0 to 31.

5.154.2 See also

Reference

- *UMADDL* on page 5-165.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.155 UXTB

Unsigned extend byte.

This instruction is an alias of UBFM.

5.155.1 Syntax

```
UXTB  Wd, Wn
```

Equivalent to `UBFM Wd, Wn, #0, #7`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

5.155.2 See also

Reference

- *UBFM* on page 5-161.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.156 UXTH

Unsigned extend halfword.

This instruction is an alias of UBFM.

5.156.1 Syntax

```
UXTH  Wd, Wn
```

Equivalent to `UBFM Wd, Wn, #0, #15`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

5.156.2 See also

Reference

- *UBFM* on page 5-161.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.157 WFE

Wait for event.

This instruction is an alias of `HINT`.

5.157.1 See also

Reference

- *HINT* on page 5-81.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.158 WFI

Wait for interrupt.

This instruction is an alias of `HINT`.

5.158.1 See also

Reference

- *HINT* on page 5-81.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

5.159 YIELD

Yield hint.

This instruction is an alias of `HINT`.

5.159.1 See also

Reference

- *HINT* on page 5-81.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Chapter 6
A64 Data Transfer Instructions

The following topic gives a summary of the A64 data transfer instructions and pseudo-instructions supported by the ARM assembler:

- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *Register restrictions for A64 instructions* on page 6-5.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.1 A64 data transfer instructions in alphabetical order

The following A64 data transfer instructions and pseudo-instructions are supported:

Table 6-1 Location of data transfer instructions

Mnemonic	Brief description	See
LDAR	Load-acquire register	page 6-6
LDARB	Load-acquire register byte	page 6-7
LDARH	Load-acquire register halfword	page 6-8
LDAXP	Load-acquire exclusive pair of registers	page 6-9
LDAXR	Load-acquire exclusive register	page 6-10
LDAXRB	Load-acquire exclusive register byte	page 6-11
LDAXRH	Load-acquire exclusive register halfword	page 6-12
LDNP (SIMD and FP)	Load pair of SIMD and FP registers, with non-temporal hint	page 6-13
LDNP	Load pair of registers, with non-temporal hint	page 6-14
LDP (SIMD and FP)	Load pair of SIMD and FP registers	page 6-15
LDP	Load pair of registers	page 6-17
LDPSW	Load pair of registers signed word	page 6-18
LDR (immediate, SIMD and FP)	Load SIMD and FP register (immediate offset)	page 6-19
LDR (immediate)	Load register (immediate offset)	page 6-21
LDR (literal, SIMD and FP)	Load SIMD and FP register (PC-relative literal)	page 6-22
LDR (literal)	Load register (PC-relative literal)	page 6-23
LDR pseudo-instruction	Load a register	page 6-24
LDR (register, SIMD and FP)	Load SIMD and FP register (register offset)	page 6-26
LDR (register)	Load register (register offset)	page 6-28
LDRB (immediate)	Load register byte (immediate offset)	page 6-29
LDRB (register)	Load register byte (register offset)	page 6-30
LDRH (immediate)	Load register halfword (register offset)	page 6-31
LDRH (register)	Load register halfword (register offset)	page 6-32
LDRSB (immediate)	Load register signed byte (immediate offset)	page 6-33
LDRSB (register)	Load register signed byte (register offset)	page 6-34
LDRSH (immediate)	Load register signed halfword (immediate offset)	page 6-35
LDRSH (register)	Load register signed halfword (register offset)	page 6-36
LDRSW (immediate)	Load register signed word (immediate offset)	page 6-37
LDRSW (literal)	Load register signed word (PC-relative literal)	page 6-38
LDRSW (register)	Load register signed word (register offset)	page 6-39

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 6-1 Location of data transfer instructions (continued)

Mnemonic	Brief description	See
LDTR	Load register (unprivileged)	page 6-40
LDTRB	Load register byte (unprivileged)	page 6-41
LDTRH	Load register halfword (unprivileged)	page 6-42
LDTRSB	Load register signed byte (unprivileged)	page 6-43
LDTRSH	Load register signed halfword (unprivileged)	page 6-44
LDTRSW	Load register signed word (unprivileged)	page 6-45
LDUR (SIMD and FP)	Load SIMD and FP register (unscaled offset)	page 6-46
LDUR	Load register (unscaled offset)	page 6-47
LDURB	Load register byte (unscaled offset)	page 6-48
LDURH	Load register halfword (unscaled offset)	page 6-49
LDURSB	Load register signed byte (unscaled offset)	page 6-50
LDURSH	Load register signed halfword (unscaled offset)	page 6-51
LDURSW	Load register signed word (unscaled offset)	page 6-52
LDXP	Load exclusive pair of registers	page 6-53
LDXR	Load exclusive register	page 6-54
LDXRB	Load exclusive register byte	page 6-55
LDXRH	Load exclusive register halfword	page 6-56
PRFM (immediate)	Prefetch memory (immediate offset)	page 6-57
PRFM (literal)	Prefetch memory (PC-relative offset)	page 6-59
PRFM (register)	Prefetch memory (register offset)	page 6-60
PRFUM	Prefetch memory (unscaled offset)	page 6-62
STLR	Store-release register	page 6-64
STLRB	Store-release register byte	page 6-65
STLRH	Store-release register halfword	page 6-66
STLXP	Store-release exclusive pair of registers, returning status	page 6-67
STLXR	Store-release exclusive register, returning status	page 6-68
STLXRB	Store-release exclusive register byte, returning status	page 6-69
STLXRH	Store-release exclusive register halfword, returning status	page 6-70
STNP (SIMD and FP)	Store pair of SIMD and FP registers, with non-temporal hint	page 6-71
STNP	Store pair of registers, with non-temporal hint	page 6-72
STP (SIMD and FP)	Store pair of SIMD and FP registers	page 6-73
STP	Store pair of registers	page 6-75

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 6-1 Location of data transfer instructions (continued)

Mnemonic	Brief description	See
STR (immediate, SIMD and FP)	Store SIMD and FP register (immediate offset)	page 6-76
STR (immediate)	Store register (immediate offset)	page 6-78
STR (register, SIMD and FP)	Store SIMD and FP register (register offset)	page 6-79
STR (register)	Store register (register offset)	page 6-81
STRB (immediate)	Store register byte (immediate offset)	page 6-82
STRB (register)	Store register byte (register offset)	page 6-83
STRH (immediate)	Store register halfword (immediate offset)	page 6-84
STRH (register)	Store register halfword (register offset)	page 6-85
STTR	Store register (unprivileged)	page 6-86
STTRB	Store register byte (unprivileged)	page 6-87
STTRH	Store register halfword (unprivileged)	page 6-88
STUR (SIMD and FP)	Store SIMD and FP register (unscaled offset)	page 6-89
STUR	Store register (unscaled offset)	page 6-90
STURB	Store register byte (unscaled offset)	page 6-91
STURH	Store register halfword (unscaled offset)	page 6-92
STXP	Store exclusive pair of registers, returning status	page 6-93
STXR	Store exclusive register, returning status	page 6-94
STXRB	Store exclusive register byte, returning status	page 6-95
STXRH	Store exclusive register halfword, returning status	page 6-96

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.2 Register restrictions for A64 instructions

In A64 instructions, the range of general-purpose integer registers is as follows:

- W0-W30 for 32-bit registers.
- X0-X30 for 64-bit registers.

You cannot refer to register 31 by number. In a few instructions, you can refer to it using one of the following names:

`WSP` the current stack pointer in a 32-bit context.

`SP` the current stack pointer in a 64-bit context.

`WZR` the zero register in a 32-bit context.

`XZR` the zero register in a 64-bit context.

You can only use one of these names if it is mentioned in the Syntax section for the instruction.

You cannot refer to the Program Counter (PC) explicitly by name or by number.

6.2.1 See also

Concepts

Using the Assembler:

- *Registers in AArch64 state* on page 5-2
- *Program Counter in AArch64 state* on page 5-8.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.3 LDAR

Load-acquire register.

6.3.1 Syntax

```
LDAR  Wt, [Xn|SP{,#0}]    ; 32-bit general registers
LDAR  Xt, [Xn|SP{,#0}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.3.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.4 LDARB

Load-acquire register byte.

6.4.1 Syntax

```
LDARB  Wt, [Xn|SP{,#0}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.4.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.5 LDARH

Load-acquire register halfword.

6.5.1 Syntax

```
LDARH  Wt, [Xn|SP{,#0}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.5.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.6 LDAXP

Load-acquire exclusive pair of registers.

6.6.1 Syntax

```
LDAXP  Wt1, Wt2, [Xn|SP{,#0}]    ; 32-bit general registers
LDAXP  Xt1, Xt2, [Xn|SP{,#0}]    ; 64-bit general registers
```

Where:

Wt1 Is the 32-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Wt2 Is the 32-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

Xt1 Is the 64-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Xt2 Is the 64-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.6.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.7 LDAXR

Load-acquire exclusive register.

6.7.1 Syntax

```
LDAXR  Wt, [Xn|SP{,#0}]    ; 32-bit general registers
LDAXR  Xt, [Xn|SP{,#0}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.7.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.8 LDAXRB

Load-acquire exclusive register byte.

6.8.1 Syntax

```
LDAXRB  Wt, [Xn|SP{,#0}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.8.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.9 LDAXRH

Load-acquire exclusive register halfword.

6.9.1 Syntax

```
LDAXRH  Wt, [Xn|SP{,#0}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.9.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.10 LDNP (SIMD and FP)

Load pair of SIMD and FP registers, with non-temporal hint.

6.10.1 Syntax

```
LDNP  St1, St2, [Xn|SP{, #imm}]    ; 32-bit FP/SIMD registers, Signed offset
LDNP  Dt1, Dt2, [Xn|SP{, #imm}]    ; 64-bit FP/SIMD registers, Signed offset
LDNP  Qt1, Qt2, [Xn|SP{, #imm}]    ; 128-bit FP/SIMD registers, Signed offset
```

Where:

St1 Is the 32-bit name of the first SIMD and FP register to be transferred, in the range 0 to 31.

St2 Is the 32-bit name of the second SIMD and FP register to be transferred, in the range 0 to 31.

imm The value depends on the instruction variant:

32-bit FP/SIMD registers

Is the optional signed immediate byte offset, a multiple of 4 in the range -256 to 252, defaulting to 0.

64-bit FP/SIMD registers

Is the optional signed immediate byte offset, a multiple of 8 in the range -512 to 504, defaulting to 0.

128-bit FP/SIMD registers

Is the optional signed immediate byte offset, a multiple of 16 in the range -1024 to 1008, defaulting to 0.

Dt1 Is the 64-bit name of the first SIMD and FP register to be transferred, in the range 0 to 31.

Dt2 Is the 64-bit name of the second SIMD and FP register to be transferred, in the range 0 to 31.

Qt1 Is the 128-bit name of the first SIMD and FP register to be transferred, in the range 0 to 31.

Qt2 Is the 128-bit name of the second SIMD and FP register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.10.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.11 LDNP

Load pair of registers, with non-temporal hint.

6.11.1 Syntax

```
LDNP  Wt1, Wt2, [Xn|SP{, #imm}]    ; 32-bit general registers, Signed offset
LDNP  Xt1, Xt2, [Xn|SP{, #imm}]    ; 64-bit general registers, Signed offset
```

Where:

Wt1 Is the 32-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Wt2 Is the 32-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

imm The value depends on the instruction variant:

32-bit general registers

Is the optional signed immediate byte offset, a multiple of 4 in the range -256 to 252, defaulting to 0.

64-bit general registers

Is the optional signed immediate byte offset, a multiple of 8 in the range -512 to 504, defaulting to 0.

Xt1 Is the 64-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Xt2 Is the 64-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.11.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.12 LDP (SIMD and FP)

Load pair of SIMD and FP registers.

6.12.1 Syntax

```
LDP  St1, St2, [Xn|SP], #imm    ; 32-bit FP/SIMD registers, Post-index
LDP  Dt1, Dt2, [Xn|SP], #imm    ; 64-bit FP/SIMD registers, Post-index
LDP  Qt1, Qt2, [Xn|SP], #imm    ; 128-bit FP/SIMD registers, Post-index
LDP  St1, St2, [Xn|SP, #imm]!    ; 32-bit FP/SIMD registers, Pre-index
LDP  Dt1, Dt2, [Xn|SP, #imm]!    ; 64-bit FP/SIMD registers, Pre-index
LDP  Qt1, Qt2, [Xn|SP, #imm]!    ; 128-bit FP/SIMD registers, Pre-index
LDP  St1, St2, [Xn|SP{, #imm}]    ; 32-bit FP/SIMD registers, Signed offset
LDP  Dt1, Dt2, [Xn|SP{, #imm}]    ; 64-bit FP/SIMD registers, Signed offset
LDP  Qt1, Qt2, [Xn|SP{, #imm}]    ; 128-bit FP/SIMD registers, Signed offset
```

Where:

St1 Is the 32-bit name of the first SIMD and FP register to be transferred, in the range 0 to 31.

St2 Is the 32-bit name of the second SIMD and FP register to be transferred, in the range 0 to 31.

imm The value depends on the instruction variant:

32-bit FP/SIMD registers

For the post-index and pre-index variant is the signed immediate byte offset, a multiple of 4 in the range -256 to 252.

For the signed offset variant is the optional signed immediate byte offset, a multiple of 4 in the range -256 to 252, defaulting to 0.

64-bit FP/SIMD registers

For the post-index and pre-index variant is the signed immediate byte offset, a multiple of 8 in the range -512 to 504.

For the signed offset variant is the optional signed immediate byte offset, a multiple of 8 in the range -512 to 504, defaulting to 0.

128-bit FP/SIMD registers

For the post-index and pre-index variant is the signed immediate byte offset, a multiple of 16 in the range -1024 to 1008.

For the signed offset variant is the optional signed immediate byte offset, a multiple of 16 in the range -1024 to 1008, defaulting to 0.

Dt1 Is the 64-bit name of the first SIMD and FP register to be transferred, in the range 0 to 31.

Dt2 Is the 64-bit name of the second SIMD and FP register to be transferred, in the range 0 to 31.

Qt1 Is the 128-bit name of the first SIMD and FP register to be transferred, in the range 0 to 31.

Qt2 Is the 128-bit name of the second SIMD and FP register to be transferred, in the range 0 to 31.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.12.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.13 LDP

Load pair of registers.

6.13.1 Syntax

```
LDP  Wt1, Wt2, [Xn|SP], #imm    ; 32-bit general registers, Post-index
LDP  Xt1, Xt2, [Xn|SP], #imm    ; 64-bit general registers, Post-index
LDP  Wt1, Wt2, [Xn|SP, #imm]!    ; 32-bit general registers, Pre-index
LDP  Xt1, Xt2, [Xn|SP, #imm]!    ; 64-bit general registers, Pre-index
LDP  Wt1, Wt2, [Xn|SP{, #imm}]    ; 32-bit general registers, Signed offset
LDP  Xt1, Xt2, [Xn|SP{, #imm}]    ; 64-bit general registers, Signed offset
```

Where:

Wt1 Is the 32-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Wt2 Is the 32-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

imm The value depends on the instruction variant:

32-bit general registers

For the post-index and pre-index variant is the signed immediate byte offset, a multiple of 4 in the range -256 to 252.

For the signed offset variant is the optional signed immediate byte offset, a multiple of 4 in the range -256 to 252, defaulting to 0.

64-bit general registers

For the post-index and pre-index variant is the signed immediate byte offset, a multiple of 8 in the range -512 to 504.

For the signed offset variant is the optional signed immediate byte offset, a multiple of 8 in the range -512 to 504, defaulting to 0.

Xt1 Is the 64-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Xt2 Is the 64-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.13.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.14 LDPSW

Load pair of registers signed word.

6.14.1 Syntax

```
LDPSW  Xt1, Xt2, [Xn|SP], #imm    ; Post-index general registers
LDPSW  Xt1, Xt2, [Xn|SP, #imm]!    ; Pre-index general registers
LDPSW  Xt1, Xt2, [Xn|SP{, #imm}]    ; Signed offset general registers
```

Where:

imm
: The value depends on the instruction variant:

  Post-index and Pre-index general registers
  : For the post-index and pre-index variant is the signed immediate byte offset, a multiple of 4 in the range -256 to 252.

  Signed offset general registers
  : For the signed offset variant is the optional signed immediate byte offset, a multiple of 4 in the range -256 to 252, defaulting to 0.

Xt1
: Is the 64-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Xt2
: Is the 64-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

Xn|SP
: Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.14.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.15 LDR (immediate, SIMD and FP)

Load SIMD and FP register (immediate offset).

6.15.1 Syntax

```
LDR  Bt, [Xn|SP], #simm    ; 8-bit FP/SIMD registers, Post-index
LDR  Ht, [Xn|SP], #simm    ; 16-bit FP/SIMD registers, Post-index
LDR  St, [Xn|SP], #simm    ; 32-bit FP/SIMD registers, Post-index
LDR  Dt, [Xn|SP], #simm    ; 64-bit FP/SIMD registers, Post-index
LDR  Qt, [Xn|SP], #simm    ; 128-bit FP/SIMD registers, Post-index
LDR  Bt, [Xn|SP, #simm]!    ; 8-bit FP/SIMD registers, Pre-index
LDR  Ht, [Xn|SP, #simm]!    ; 16-bit FP/SIMD registers, Pre-index
LDR  St, [Xn|SP, #simm]!    ; 32-bit FP/SIMD registers, Pre-index
LDR  Dt, [Xn|SP, #simm]!    ; 64-bit FP/SIMD registers, Pre-index
LDR  Qt, [Xn|SP, #simm]!    ; 128-bit FP/SIMD registers, Pre-index
LDR  Bt, [Xn|SP{, #pimm}]    ; 8-bit FP/SIMD registers
LDR  Ht, [Xn|SP{, #pimm}]    ; 16-bit FP/SIMD registers
LDR  St, [Xn|SP{, #pimm}]    ; 32-bit FP/SIMD registers
LDR  Dt, [Xn|SP{, #pimm}]    ; 64-bit FP/SIMD registers
LDR  Qt, [Xn|SP{, #pimm}]    ; 128-bit FP/SIMD registers
```

Where:

Bt Is the 8-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

simm Is the signed immediate byte offset, in the range -256 to 255.

Ht Is the 16-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

St Is the 32-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Dt Is the 64-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Qt Is the 128-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

pimm The value depends on the instruction variant:

8-bit FP/SIMD registers

Is the optional positive immediate byte offset, in the range 0 to 4095, defaulting to 0.

16-bit FP/SIMD registers

Is the optional positive immediate byte offset, a multiple of 2 in the range 0 to 8190, defaulting to 0.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

32-bit FP/SIMD registers

Is the optional positive immediate byte offset, a multiple of 4 in the range 0 to 16380, defaulting to 0.

64-bit FP/SIMD registers

Is the optional positive immediate byte offset, a multiple of 8 in the range 0 to 32760, defaulting to 0.

128-bit FP/SIMD registers

Is the optional positive immediate byte offset, a multiple of 16 in the range 0 to 65520, defaulting to 0.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.15.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.16 LDR (immediate)

Load register (immediate offset).

6.16.1 Syntax

```
LDR  Wt, [Xn|SP], #simm    ; 32-bit general registers, Post-index
LDR  Xt, [Xn|SP], #simm    ; 64-bit general registers, Post-index
LDR  Wt, [Xn|SP, #simm]!    ; 32-bit general registers, Pre-index
LDR  Xt, [Xn|SP, #simm]!    ; 64-bit general registers, Pre-index
LDR  Wt, [Xn|SP{, #pimm}]    ; 32-bit general registers
LDR  Xt, [Xn|SP{, #pimm}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

simm Is the signed immediate byte offset, in the range -256 to 255.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

pimm The value depends on the instruction variant:

32-bit general registers

Is the optional positive immediate byte offset, a multiple of 4 in the range 0 to 16380, defaulting to 0.

64-bit general registers

Is the optional positive immediate byte offset, a multiple of 8 in the range 0 to 32760, defaulting to 0.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.16.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.17 LDR (literal, SIMD and FP)

Load SIMD and FP register (PC-relative literal).

6.17.1 Syntax

```
LDR  St, label    ; 32-bit FP/SIMD registers
LDR  Dt, label    ; 64-bit FP/SIMD registers
LDR  Qt, label    ; 128-bit FP/SIMD registers
```

Where:

St Is the 32-bit name of the SIMD and FP register to be loaded, in the range 0 to 31.

Dt Is the 64-bit name of the SIMD and FP register to be loaded, in the range 0 to 31.

Qt Is the 128-bit name of the SIMD and FP register to be loaded, in the range 0 to 31.

label Is the program label from which the data is to be loaded. It is an offset from the address of this instruction, in the range ±1MB.

6.17.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.18 LDR (literal)

Load register (PC-relative literal).

6.18.1 Syntax

```
LDR  Wt, label    ; 32-bit general registers
LDR  Xt, label    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be loaded, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be loaded, in the range 0 to 31.

label Is the program label from which the data is to be loaded. It is an offset from the address of this instruction, in the range ±1MB.

6.18.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.19 LDR pseudo-instruction

Load a register with either:

- A 32-bit or 64-bit immediate value.
- An address.

Note

This description is for the LDR pseudo-instruction only, and not for the LDR instruction.

6.19.1 Syntax

```
LDR Wd, =expr
LDR Xd, =expr
LDR Wd, =label_expr
LDR Xd, =label_expr
```

where:

Wd Is the register to load with a 32-bit value.

Xd Is the register to load with a 64-bit value.

expr Evaluates to a numeric value.

label_expr Is a PC-relative or external expression of an address in the form of a label plus or minus a numeric value.

6.19.2 Usage

When using the LDR pseudo-instruction, the assembler places the value of *expr* or *label_expr* in a literal pool and generates a PC-relative LDR instruction that reads the constant from the literal pool.

Note

- An address loaded in this way is fixed at link time, so the code is not position-independent.
- The address holding the constant remains valid regardless of where the linker places the ELF section containing the LDR instruction.

If *label_expr* is an external expression, or is not contained in the current section, the assembler places a linker relocation directive in the object file. The linker generates the address at link time.

If *label_expr* is a local label, the assembler places a linker relocation directive in the object file and generates a symbol for that local label. The address is generated at link time.

The offset from the PC to the value in the literal pool must be less than ±1MB . You are responsible for ensuring that there is a literal pool within range.

6.19.3 Examples

```
LDR     w1,=0xfff    ; loads 0xfff into W1
                     ; =>  LDR w1,[pc,offset_to_litpool]
                     ;     ...
                     ;     litpool DCD 4095
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

```
LDR     x2,=place    ; loads the address of
                     ; place into X2
                     ; =>  LDR x2,[pc,offset_to_litpool]
                     ;     ...
                     ;     litpool DCQ place
```

6.19.4 See also

Concepts

Using the Assembler

- *Numeric constants* on page 10-5.
- *Register-relative and PC-relative expressions* on page 10-7.
- *Numeric local labels* on page 10-12.
- *Load immediate 32-bit values to a register using LDR Rd, =const* on page 7-11.

Reference

- *LTORG* on page 10-65.
- *MOVL pseudo-instruction* on page 5-100.
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Other information

- *ARM Architecture Reference Manual*
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.architecture.reference/.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.20 LDR (register, SIMD and FP)

Load SIMD and FP register (register offset).

6.20.1 Syntax

```
LDR  Bt, [Xn|SP, Rm{, extend {amount}}]    ; 8-bit FP/SIMD registers
LDR  Ht, [Xn|SP, Rm{, extend {amount}}]    ; 16-bit FP/SIMD registers
LDR  St, [Xn|SP, Rm{, extend {amount}}]    ; 32-bit FP/SIMD registers
LDR  Dt, [Xn|SP, Rm{, extend {amount}}]    ; 64-bit FP/SIMD registers
LDR  Qt, [Xn|SP, Rm{, extend {amount}}]    ; 128-bit FP/SIMD registers
```

Where:

Bt Is the 8-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

amount Is the index shift amount, optional and defaulting to #0 when *extend* is not LSL:

8-bit FP/SIMD registers
Must be #0.

16-bit FP/SIMD registers
Can be one of #0 or #1.

32-bit FP/SIMD registers
Can be one of #0 or #2.

64-bit FP/SIMD registers
Can be one of #0 or #3.

128-bit FP/SIMD registers
Can be one of #0 or #4.

Ht Is the 16-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

St Is the 32-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Dt Is the 64-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Qt Is the 128-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

R Is the index width specifier, and can be either W or X.

m Is the number of the general-purpose index register, in the range 0 to 30, or the name ZR (31).

extend Is the index extend/shift specifier, defaulting to LSL, and can be one of the values shown in Usage.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.20.2 Usage

The following table shows valid specifier combinations:

Table 6-2 LDR (register, SIMD and FP) specifier combinations

R	*extend*
W	SXTW
W	UXTW
X	LSL
X	SXTX

6.20.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.21 LDR (register)

Load register (register offset).

6.21.1 Syntax

```
LDR  Wt, [Xn|SP, Rm{, extend {amount}}]    ; 32-bit general registers
LDR  Xt, [Xn|SP, Rm{, extend {amount}}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

amount Is the index shift amount, optional and defaulting to #0 when *extend* is not LSL:

32-bit general registers
Can be one of #0 or #2.

64-bit general registers
Can be one of #0 or #3.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

R Is the index width specifier, and can be either W or X.

m Is the number of the general-purpose index register, in the range 0 to 30, or the name ZR (31).

extend Is the index extend/shift specifier, defaulting to LSL, and can be one of the values shown in Usage.

6.21.2 Usage

The following table shows valid specifier combinations:

Table 6-3 LDR (register) specifier combinations

R	*extend*
W	SXTW
W	UXTW
X	LSL
X	SXTX

6.21.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.22 LDRB (immediate)

Load register byte (immediate offset).

6.22.1 Syntax

```
LDRB  Wt, [Xn|SP], #simm    ; Post-index general registers
LDRB  Wt, [Xn|SP, #simm]!    ; Pre-index general registers
LDRB  Wt, [Xn|SP{, #pimm}]    ; Unsigned offset general registers
```

Where:

simm Is the signed immediate byte offset, in the range -256 to 255.

pimm Is the optional positive immediate byte offset, in the range 0 to 4095, defaulting to 0.

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.22.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.23 LDRB (register)

Load register byte (register offset).

6.23.1 Syntax

```
LDRB  Wt, [Xn|SP, Rm{, extend {amount}}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

R Is the index width specifier, and can be either W or X.

m Is the number of the general-purpose index register, in the range 0 to 30, or the name ZR (31).

extend Is the index extend/shift specifier, defaulting to LSL, and can be one of the values shown in Usage.

amount Is the index shift amount, optional and defaulting to #0 when *extend* is not LSL.

6.23.2 Usage

The following table shows valid specifier combinations:

Table 6-4 LDRB (register) specifier combinations

R	*extend*
W	SXTW
W	UXTW
X	LSL
X	SXTX

6.23.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.24 LDRH (immediate)

Load register halfword (register offset).

6.24.1 Syntax

```
LDRH  Wt, [Xn|SP], #simm    ; Post-index general registers
LDRH  Wt, [Xn|SP, #simm]!    ; Pre-index general registers
LDRH  Wt, [Xn|SP{, #pimm}]    ; Unsigned offset general registers
```

Where:

simm Is the signed immediate byte offset, in the range -256 to 255.

pimm Is the optional positive immediate byte offset, a multiple of 2 in the range 0 to 8190, defaulting to 0.

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.24.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.25 LDRH (register)

Load register halfword (register offset).

6.25.1 Syntax

```
LDRH  Wt, [Xn|SP, Rm{, extend {amount}}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

R Is the index width specifier, and can be either W or X.

m Is the number of the general-purpose index register, in the range 0 to 30, or the name ZR (31).

extend Is the index extend/shift specifier, defaulting to LSL, and can be one of the values shown in Usage.

amount Is the index shift amount, optional and defaulting to #0 when *extend* is not LSL, and can be either #0 or #1.

6.25.2 Usage

The following table shows valid specifier combinations:

Table 6-5 LDRH (register) specifier combinations

R	*extend*
W	SXTW
W	UXTW
X	LSL
X	SXTX

6.25.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.26 LDRSB (immediate)

Load register signed byte (immediate offset).

6.26.1 Syntax

```
LDRSB  Wt, [Xn|SP], #simm    ; 32-bit general registers, Post-index
LDRSB  Xt, [Xn|SP], #simm    ; 64-bit general registers, Post-index
LDRSB  Wt, [Xn|SP, #simm]!    ; 32-bit general registers, Pre-index
LDRSB  Xt, [Xn|SP, #simm]!    ; 64-bit general registers, Pre-index
LDRSB  Wt, [Xn|SP{, #pimm}]    ; 32-bit general registers
LDRSB  Xt, [Xn|SP{, #pimm}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

simm Is the signed immediate byte offset, in the range -256 to 255.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

pimm Is the optional positive immediate byte offset, in the range 0 to 4095, defaulting to 0.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.26.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.27 LDRSB (register)

Load register signed byte (register offset).

6.27.1 Syntax

```
LDRSB  Wt, [Xn|SP, Rm{, extend {amount}}]    ; 32-bit general registers
LDRSB  Xt, [Xn|SP, Rm{, extend {amount}}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

R Is the index width specifier, and can be either W or X.

m Is the number of the general-purpose index register, in the range 0 to 30, or the name ZR (31).

extend Is the index extend/shift specifier, defaulting to LSL, and can be one of the values shown in Usage.

amount Is the index shift amount, optional and defaulting to #0 when *extend* is not LSL.

6.27.2 Usage

The following table shows valid specifier combinations:

Table 6-6 LDRSB (register) specifier combinations

R	*extend*
W	SXTW
W	UXTW
X	LSL
X	SXTX

6.27.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.28 LDRSH (immediate)

Load register signed halfword (immediate offset).

6.28.1 Syntax

```
LDRSH  Wt, [Xn|SP], #simm    ; 32-bit general registers, Post-index
LDRSH  Xt, [Xn|SP], #simm    ; 64-bit general registers, Post-index
LDRSH  Wt, [Xn|SP, #simm]!    ; 32-bit general registers, Pre-index
LDRSH  Xt, [Xn|SP, #simm]!    ; 64-bit general registers, Pre-index
LDRSH  Wt, [Xn|SP{, #pimm}]    ; 32-bit general registers
LDRSH  Xt, [Xn|SP{, #pimm}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

simm Is the signed immediate byte offset, in the range -256 to 255.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

pimm Is the optional positive immediate byte offset, a multiple of 2 in the range 0 to 8190, defaulting to 0.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.28.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.29 LDRSH (register)

Load register signed halfword (register offset).

6.29.1 Syntax

```
LDRSH  Wt, [Xn|SP, Rm{, extend {amount}}]    ; 32-bit general registers
LDRSH  Xt, [Xn|SP, Rm{, extend {amount}}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

R Is the index width specifier, and can be either `W` or `X`.

m Is the number of the general-purpose index register, in the range 0 to 30, or the name ZR (31).

extend Is the index extend/shift specifier, defaulting to LSL, and can be one of the values shown in Usage.

amount Is the index shift amount, optional and defaulting to #0 when *extend* is not `LSL`, and can be either `#0` or `#1`.

6.29.2 Usage

The following table shows valid specifier combinations:

Table 6-7 LDRSH (register) specifier combinations

R	*extend*
W	SXTW
W	UXTW
X	LSL
X	SXTX

6.29.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.30 LDRSW (immediate)

Load register signed word (immediate offset).

6.30.1 Syntax

```
LDRSW  Xt, [Xn|SP], #simm    ; Post-index general registers
LDRSW  Xt, [Xn|SP, #simm]!    ; Pre-index general registers
LDRSW  Xt, [Xn|SP{, #pimm}]    ; Unsigned offset general registers
```

Where:

simm Is the signed immediate byte offset, in the range -256 to 255.

pimm Is the optional positive immediate byte offset, a multiple of 4 in the range 0 to 16380, defaulting to 0.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.30.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.31 LDRSW (literal)

Load register signed word (PC-relative literal).

6.31.1 Syntax

```
LDRSW  Xt, label
```

Where:

Xt Is the 64-bit name of the general-purpose register to be loaded, in the range 0 to 31.

label Is the program label from which the data is to be loaded. It is an offset from the address of this instruction, in the range ±1MB.

6.31.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.32 LDRSW (register)

Load register signed word (register offset).

6.32.1 Syntax

```
LDRSW  Xt, [Xn|SP, Rm{, extend {amount}}]
```

Where:

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

R Is the index width specifier, and can be either `W` or `X`.

m Is the number of the general-purpose index register, in the range 0 to 30, or the name ZR (31).

extend Is the index extend/shift specifier, defaulting to LSL, and can be one of the values shown in Usage.

amount Is the index shift amount, optional and defaulting to #0 when *extend* is not `LSL`, and can be either `#0` or `#2`.

6.32.2 Usage

The following table shows valid specifier combinations:

Table 6-8 LDRSW (register) specifier combinations

R	*extend*
W	SXTW
W	UXTW
X	LSL
X	SXTX

6.32.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.33 LDTR

Load register (unprivileged).

6.33.1 Syntax

```
LDTR  Wt, [Xn|SP{, #simm}]    ; 32-bit general registers
LDTR  Xt, [Xn|SP{, #simm}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.33.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.34 LDTRB

Load register byte (unprivileged).

6.34.1 Syntax

```
LDTRB  Wt, [Xn|SP{, #simm}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.34.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.35 LDTRH

Load register halfword (unprivileged).

6.35.1 Syntax

```
LDTRH  Wt, [Xn|SP{, #simm}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.35.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.36 LDTRSB

Load register signed byte (unprivileged).

6.36.1 Syntax

```
LDTRSB  Wt, [Xn|SP{, #simm}]    ; 32-bit general registers
LDTRSB  Xt, [Xn|SP{, #simm}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.36.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.37 LDTRSH

Load register signed halfword (unprivileged).

6.37.1 Syntax

```
LDTRSH  Wt, [Xn|SP{, #simm}]    ; 32-bit general registers
LDTRSH  Xt, [Xn|SP{, #simm}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.37.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.38 LDTRSW

Load register signed word (unprivileged).

6.38.1 Syntax

```
LDTRSW  Xt, [Xn|SP{, #simm}]
```

Where:

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.38.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.39 LDUR (SIMD and FP)

Load SIMD and FP register (unscaled offset).

6.39.1 Syntax

```
LDUR  Bt, [Xn|SP{, #simm}]    ; 8-bit FP/SIMD registers
LDUR  Ht, [Xn|SP{, #simm}]    ; 16-bit FP/SIMD registers
LDUR  St, [Xn|SP{, #simm}]    ; 32-bit FP/SIMD registers
LDUR  Dt, [Xn|SP{, #simm}]    ; 64-bit FP/SIMD registers
LDUR  Qt, [Xn|SP{, #simm}]    ; 128-bit FP/SIMD registers
```

Where:

Bt Is the 8-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Ht Is the 16-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

St Is the 32-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Dt Is the 64-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Qt Is the 128-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.39.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.40 LDUR

Load register (unscaled offset).

6.40.1 Syntax

```
LDUR  Wt, [Xn|SP{, #simm}]    ; 32-bit general registers
LDUR  Xt, [Xn|SP{, #simm}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.40.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.41 LDURB

Load register byte (unscaled offset).

6.41.1 Syntax

```
LDURB  Wt, [Xn|SP{, #simm}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.41.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.42 LDURH

Load register halfword (unscaled offset).

6.42.1 Syntax

```
LDURH  Wt, [Xn|SP{, #simm}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.42.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.43 LDURSB

Load register signed byte (unscaled offset).

6.43.1 Syntax

```
LDURSB  Wt, [Xn|SP{, #simm}]    ; 32-bit general registers
LDURSB  Xt, [Xn|SP{, #simm}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.43.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.44 LDURSH

Load register signed halfword (unscaled offset).

6.44.1 Syntax

```
LDURSH  Wt, [Xn|SP{, #simm}]    ; 32-bit general registers
LDURSH  Xt, [Xn|SP{, #simm}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.44.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.45 LDURSW

Load register signed word (unscaled offset).

6.45.1 Syntax

```
LDURSW  Xt, [Xn|SP{, #simm}]
```

Where:

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.45.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.46 LDXP

Load exclusive pair of registers.

6.46.1 Syntax

```
LDXP  Wt1, Wt2, [Xn|SP{,#0}]    ; 32-bit general registers
LDXP  Xt1, Xt2, [Xn|SP{,#0}]    ; 64-bit general registers
```

Where:

Wt1 Is the 32-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Wt2 Is the 32-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

Xt1 Is the 64-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Xt2 Is the 64-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.46.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.47 LDXR

Load exclusive register.

6.47.1 Syntax

```
LDXR  Wt, [Xn|SP{,#0}]    ; 32-bit general registers
LDXR  Xt, [Xn|SP{,#0}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.47.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.48 LDXRB

Load exclusive register byte.

6.48.1 Syntax

```
LDXRB  Wt, [Xn|SP{,#0}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.48.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.49 LDXRH

Load exclusive register halfword.

6.49.1 Syntax

```
LDXRH  Wt, [Xn|SP{,#0}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.49.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.50 PRFM (immediate)

Prefetch memory (immediate offset).

6.50.1 Syntax

```
PRFM  prfop, [Xn|SP{, #pimm}]
```

Where:

prfop Is the prefetch operation, and contains the following fields without spaces between them:

```
type target policy
```

type Can be one of:

Table 6-9 PRFM (immediate) *type* options

option	meaning
PLD	prefetch for load
PST	prefetch for store

target Can be one of:

Table 6-10 PRFM (immediate) *target* options

option	meaning
L1	Level 1 cache
L2	Level 2 cache
L3	Level 3 cache

policy Can be one of:

Table 6-11 PRFM (immediate) *policy* options

option	meaning
KEEP	keep in cache
STRM	Streaming data

For example, `PLDL1KEEP`.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

pimm Is the optional positive immediate byte offset, a multiple of 8 in the range 0 to 32760, defaulting to 0.

6.50.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.51 PRFM (literal)

Prefetch memory (PC-relative offset).

6.51.1 Syntax

```
PRFM  prfop, label
```

Where:

prfop Is the prefetch operation, and contains the following fields without spaces between them:

```
type target policy
```

type Can be one of:

Table 6-12 PRFM (literal) *type* options

option	meaning
PLD	prefetch for load
PST	prefetch for store

target Can be one of:

Table 6-13 PRFM (literal) *target* options

option	meaning
L1	Level 1 cache
L2	Level 2 cache
L3	Level 3 cache

policy Can be one of:

Table 6-14 PRFM (literal) *policy* options

option	meaning
KEEP	keep in cache
STRM	Streaming data

For example, `PLDL1KEEP`.

label Is the program label from which the data is to be loaded. It is an offset from the address of this instruction, in the range ±1MB.

6.51.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.52 PRFM (register)

Prefetch memory (register offset).

6.52.1 Syntax

```
PRFM  prfop, [Xn|SP, Rm{, extend {amount}}]
```

Where:

prfop Is the prefetch operation, and contains the following fields without spaces between them:

```
type target policy
```

type Can be one of:

Table 6-15 PRFM (register) *type* options

option	meaning
PLD	prefetch for load
PST	prefetch for store

target Can be one of:

Table 6-16 PRFM (register) *target* options

option	meaning
L1	Level 1 cache
L2	Level 2 cache
L3	Level 3 cache

policy Can be one of:

Table 6-17 PRFM (register) *policy* options

option	meaning
KEEP	keep in cache
STRM	Streaming data

For example, `PLDL1KEEP`.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

R Is the index width specifier, and can be either `W` or `X`.

m Is the number of the general-purpose index register, in the range 0 to 30, or the name ZR (31).

extend Is the index extend/shift specifier, defaulting to LSL, and can be one of the values shown in Usage.

amount Is the index shift amount, optional and defaulting to #0 when *extend* is not `LSL`, and can be either `#0` or `#3`.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.52.2 Usage

The following table shows valid specifier combinations:

Table 6-18 PRFM (register) specifier combinations

R	*extend*
W	SXTW
W	UXTW
X	LSL
X	SXTX

6.52.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.53 PRFUM

Prefetch memory (unscaled offset).

6.53.1 Syntax

`PRFUM prfop, [Xn|SP{, #simm}]`

Where:

prfop Is the prefetch operation, and contains the following fields without spaces between them:

type target policy

type Can be one of:

Table 6-19 PRFUM *type* options

option	meaning
PLD	prefetch for load
PST	prefetch for store

target Can be one of:

Table 6-20 PRFUM *target* options

option	meaning
L1	Level 1 cache
L2	Level 2 cache
L3	Level 3 cache

policy Can be one of:

Table 6-21 PRFUM *policy* options

option	meaning
KEEP	keep in cache
STRM	Streaming data

For example, `PLDL1KEEP`.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.53.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.54 STLR

Store-release register.

6.54.1 Syntax

```
STLR  Wt, [Xn|SP{,#0}]    ; 32-bit general registers
STLR  Xt, [Xn|SP{,#0}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.54.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.55 STLRB

Store-release register byte.

6.55.1 Syntax

```
STLRB  Wt, [Xn|SP{,#0}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.55.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.56 STLRH

Store-release register halfword.

6.56.1 Syntax

```
STLRH  Wt, [Xn|SP{,#0}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.56.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.57 STLXP

Store-release exclusive pair of registers, returning status.

6.57.1 Syntax

```
STLXP  Ws, Wt1, Wt2, [Xn|SP{,#0}]    ; 32-bit general registers
STLXP  Ws, Xt1, Xt2, [Xn|SP{,#0}]    ; 64-bit general registers
```

Where:

Wt1 Is the 32-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Wt2 Is the 32-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

Xt1 Is the 64-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Xt2 Is the 64-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

Ws Is the 32-bit name of the general-purpose register into which the status result of the store exclusive is written.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.57.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.58 STLXR

Store-release exclusive register, returning status.

6.58.1 Syntax

```
STLXR  Ws, Wt, [Xn|SP{,#0}]    ; 32-bit general registers
STLXR  Ws, Xt, [Xn|SP{,#0}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Ws Is the 32-bit name of the general-purpose register into which the status result of the store exclusive is written.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.58.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.59 STLXRB

Store-release exclusive register byte, returning status.

6.59.1 Syntax

```
STLXRB  Ws, Wt, [Xn|SP{,#0}]
```

Where:

Ws Is the 32-bit name of the general-purpose register into which the status result of the store exclusive is written.

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.59.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.60 STLXRH

Store-release exclusive register halfword, returning status.

6.60.1 Syntax

```
STLXRH  Ws, Wt, [Xn|SP{,#0}]
```

Where:

Ws Is the 32-bit name of the general-purpose register into which the status result of the store exclusive is written.

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.60.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.61 STNP (SIMD and FP)

Store pair of SIMD and FP registers, with non-temporal hint.

6.61.1 Syntax

```
STNP  St1, St2, [Xn|SP{, #imm}]    ; 32-bit FP/SIMD registers, Signed offset
STNP  Dt1, Dt2, [Xn|SP{, #imm}]    ; 64-bit FP/SIMD registers, Signed offset
STNP  Qt1, Qt2, [Xn|SP{, #imm}]    ; 128-bit FP/SIMD registers, Signed offset
```

Where:

St1 Is the 32-bit name of the first SIMD and FP register to be transferred, in the range 0 to 31.

St2 Is the 32-bit name of the second SIMD and FP register to be transferred, in the range 0 to 31.

imm The value depends on the instruction variant:

32-bit FP/SIMD registers

Is the optional signed immediate byte offset, a multiple of 4 in the range -256 to 252, defaulting to 0.

64-bit FP/SIMD registers

Is the optional signed immediate byte offset, a multiple of 8 in the range -512 to 504, defaulting to 0.

128-bit FP/SIMD registers

Is the optional signed immediate byte offset, a multiple of 16 in the range -1024 to 1008, defaulting to 0.

Dt1 Is the 64-bit name of the first SIMD and FP register to be transferred, in the range 0 to 31.

Dt2 Is the 64-bit name of the second SIMD and FP register to be transferred, in the range 0 to 31.

Qt1 Is the 128-bit name of the first SIMD and FP register to be transferred, in the range 0 to 31.

Qt2 Is the 128-bit name of the second SIMD and FP register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.61.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.62 STNP

Store pair of registers, with non-temporal hint.

6.62.1 Syntax

```
STNP  Wt1, Wt2, [Xn|SP{, #imm}]    ; 32-bit general registers, Signed offset
STNP  Xt1, Xt2, [Xn|SP{, #imm}]    ; 64-bit general registers, Signed offset
```

Where:

Wt1 Is the 32-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Wt2 Is the 32-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

imm The value depends on the instruction variant:

32-bit general registers

Is the optional signed immediate byte offset, a multiple of 4 in the range -256 to 252, defaulting to 0.

64-bit general registers

Is the optional signed immediate byte offset, a multiple of 8 in the range -512 to 504, defaulting to 0.

Xt1 Is the 64-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Xt2 Is the 64-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.62.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.63 STP (SIMD and FP)

Store pair of SIMD and FP registers.

6.63.1 Syntax

```
STP  St1, St2, [Xn|SP], #imm    ; 32-bit FP/SIMD registers, Post-index
STP  Dt1, Dt2, [Xn|SP], #imm    ; 64-bit FP/SIMD registers, Post-index
STP  Qt1, Qt2, [Xn|SP], #imm    ; 128-bit FP/SIMD registers, Post-index
STP  St1, St2, [Xn|SP, #imm]!    ; 32-bit FP/SIMD registers, Pre-index
STP  Dt1, Dt2, [Xn|SP, #imm]!    ; 64-bit FP/SIMD registers, Pre-index
STP  Qt1, Qt2, [Xn|SP, #imm]!    ; 128-bit FP/SIMD registers, Pre-index
STP  St1, St2, [Xn|SP{, #imm}]    ; 32-bit FP/SIMD registers, Signed offset
STP  Dt1, Dt2, [Xn|SP{, #imm}]    ; 64-bit FP/SIMD registers, Signed offset
STP  Qt1, Qt2, [Xn|SP{, #imm}]    ; 128-bit FP/SIMD registers, Signed offset
```

Where:

St1 Is the 32-bit name of the first SIMD and FP register to be transferred, in the range 0 to 31.

St2 Is the 32-bit name of the second SIMD and FP register to be transferred, in the range 0 to 31.

imm The value depends on the instruction variant:

32-bit FP/SIMD registers

For the post-index and pre-index variant is the signed immediate byte offset, a multiple of 4 in the range -256 to 252.

For the signed offset variant is the optional signed immediate byte offset, a multiple of 4 in the range -256 to 252, defaulting to 0.

64-bit FP/SIMD registers

For the post-index and pre-index variant is the signed immediate byte offset, a multiple of 8 in the range -512 to 504.

For the signed offset variant is the optional signed immediate byte offset, a multiple of 8 in the range -512 to 504, defaulting to 0.

128-bit FP/SIMD registers

For the post-index and pre-index variant is the signed immediate byte offset, a multiple of 16 in the range -1024 to 1008.

For the signed offset variant is the optional signed immediate byte offset, a multiple of 16 in the range -1024 to 1008, defaulting to 0.

Dt1 Is the 64-bit name of the first SIMD and FP register to be transferred, in the range 0 to 31.

Dt2 Is the 64-bit name of the second SIMD and FP register to be transferred, in the range 0 to 31.

Qt1 Is the 128-bit name of the first SIMD and FP register to be transferred, in the range 0 to 31.

Qt2 Is the 128-bit name of the second SIMD and FP register to be transferred, in the range 0 to 31.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.63.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.64 STP

Store pair of registers.

6.64.1 Syntax

```
STP  Wt1, Wt2, [Xn|SP], #imm    ; 32-bit general registers, Post-index
STP  Xt1, Xt2, [Xn|SP], #imm    ; 64-bit general registers, Post-index
STP  Wt1, Wt2, [Xn|SP, #imm]!    ; 32-bit general registers, Pre-index
STP  Xt1, Xt2, [Xn|SP, #imm]!    ; 64-bit general registers, Pre-index
STP  Wt1, Wt2, [Xn|SP{, #imm}]    ; 32-bit general registers, Signed offset
STP  Xt1, Xt2, [Xn|SP{, #imm}]    ; 64-bit general registers, Signed offset
```

Where:

Wt1
Is the 32-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Wt2
Is the 32-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

imm
The value depends on the instruction variant:

32-bit general registers

For the post-index and pre-index variant is the signed immediate byte offset, a multiple of 4 in the range -256 to 252.

For the signed offset variant is the optional signed immediate byte offset, a multiple of 4 in the range -256 to 252, defaulting to 0.

64-bit general registers

For the post-index and pre-index variant is the signed immediate byte offset, a multiple of 8 in the range -512 to 504.

For the signed offset variant is the optional signed immediate byte offset, a multiple of 8 in the range -512 to 504, defaulting to 0.

Xt1
Is the 64-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Xt2
Is the 64-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

Xn|SP
Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.64.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.65 STR (immediate, SIMD and FP)

Store SIMD and FP register (immediate offset).

6.65.1 Syntax

```
STR  Bt, [Xn|SP], #simm    ; 8-bit FP/SIMD registers, Post-index
STR  Ht, [Xn|SP], #simm    ; 16-bit FP/SIMD registers, Post-index
STR  St, [Xn|SP], #simm    ; 32-bit FP/SIMD registers, Post-index
STR  Dt, [Xn|SP], #simm    ; 64-bit FP/SIMD registers, Post-index
STR  Qt, [Xn|SP], #simm    ; 128-bit FP/SIMD registers, Post-index
STR  Bt, [Xn|SP, #simm]!    ; 8-bit FP/SIMD registers, Pre-index
STR  Ht, [Xn|SP, #simm]!    ; 16-bit FP/SIMD registers, Pre-index
STR  St, [Xn|SP, #simm]!    ; 32-bit FP/SIMD registers, Pre-index
STR  Dt, [Xn|SP, #simm]!    ; 64-bit FP/SIMD registers, Pre-index
STR  Qt, [Xn|SP, #simm]!    ; 128-bit FP/SIMD registers, Pre-index
STR  Bt, [Xn|SP{, #pimm}]    ; 8-bit FP/SIMD registers
STR  Ht, [Xn|SP{, #pimm}]    ; 16-bit FP/SIMD registers
STR  St, [Xn|SP{, #pimm}]    ; 32-bit FP/SIMD registers
STR  Dt, [Xn|SP{, #pimm}]    ; 64-bit FP/SIMD registers
STR  Qt, [Xn|SP{, #pimm}]    ; 128-bit FP/SIMD registers
```

Where:

Bt Is the 8-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

simm Is the signed immediate byte offset, in the range -256 to 255.

Ht Is the 16-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

St Is the 32-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Dt Is the 64-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Qt Is the 128-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

pimm The value depends on the instruction variant:

8-bit FP/SIMD registers

Is the optional positive immediate byte offset, in the range 0 to 4095, defaulting to 0.

16-bit FP/SIMD registers

Is the optional positive immediate byte offset, a multiple of 2 in the range 0 to 8190, defaulting to 0.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

32-bit FP/SIMD registers

Is the optional positive immediate byte offset, a multiple of 4 in the range 0 to 16380, defaulting to 0.

64-bit FP/SIMD registers

Is the optional positive immediate byte offset, a multiple of 8 in the range 0 to 32760, defaulting to 0.

128-bit FP/SIMD registers

Is the optional positive immediate byte offset, a multiple of 16 in the range 0 to 65520, defaulting to 0.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.65.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.66 STR (immediate)

Store register (immediate offset).

6.66.1 Syntax

```
STR  Wt, [Xn|SP], #simm    ; 32-bit general registers, Post-index
STR  Xt, [Xn|SP], #simm    ; 64-bit general registers, Post-index
STR  Wt, [Xn|SP, #simm]!    ; 32-bit general registers, Pre-index
STR  Xt, [Xn|SP, #simm]!    ; 64-bit general registers, Pre-index
STR  Wt, [Xn|SP{, #pimm}]    ; 32-bit general registers
STR  Xt, [Xn|SP{, #pimm}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

simm Is the signed immediate byte offset, in the range -256 to 255.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

pimm The value depends on the instruction variant:

32-bit general registers

Is the optional positive immediate byte offset, a multiple of 4 in the range 0 to 16380, defaulting to 0.

64-bit general registers

Is the optional positive immediate byte offset, a multiple of 8 in the range 0 to 32760, defaulting to 0.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.66.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.67 STR (register, SIMD and FP)

Store SIMD and FP register (register offset).

6.67.1 Syntax

```
STR  Bt, [Xn|SP, Rm{, extend {amount}}]    ; 8-bit FP/SIMD registers
STR  Ht, [Xn|SP, Rm{, extend {amount}}]    ; 16-bit FP/SIMD registers
STR  St, [Xn|SP, Rm{, extend {amount}}]    ; 32-bit FP/SIMD registers
STR  Dt, [Xn|SP, Rm{, extend {amount}}]    ; 64-bit FP/SIMD registers
STR  Qt, [Xn|SP, Rm{, extend {amount}}]    ; 128-bit FP/SIMD registers
```

Where:

Bt Is the 8-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

amount Is the index shift amount, optional and defaulting to #0 when *extend* is not LSL:

8-bit FP/SIMD registers
Must be #0.

16-bit FP/SIMD registers
Can be one of #0 or #1.

32-bit FP/SIMD registers
Can be one of #0 or #2.

64-bit FP/SIMD registers
Can be one of #0 or #3.

128-bit FP/SIMD registers
Can be one of #0 or #4.

Ht Is the 16-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

St Is the 32-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Dt Is the 64-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Qt Is the 128-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

R Is the index width specifier, and can be either W or X.

m Is the number of the general-purpose index register, in the range 0 to 30, or the name ZR (31).

extend Is the index extend/shift specifier, defaulting to LSL, and can be one of the values shown in Usage.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.67.2 Usage

The following table shows valid specifier combinations:

Table 6-22 STR (register, SIMD and FP) specifier combinations

R	*extend*
W	SXTW
W	UXTW
X	LSL
X	SXTX

6.67.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.68 STR (register)

Store register (register offset).

6.68.1 Syntax

```
STR  Wt, [Xn|SP, Rm{, extend {amount}}]    ; 32-bit general registers
STR  Xt, [Xn|SP, Rm{, extend {amount}}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

amount Is the index shift amount, optional and defaulting to #0 when *extend* is not LSL:

32-bit general registers
Can be one of #0 or #2.

64-bit general registers
Can be one of #0 or #3.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

R Is the index width specifier, and can be either W or X.

m Is the number of the general-purpose index register, in the range 0 to 30, or the name ZR (31).

extend Is the index extend/shift specifier, defaulting to LSL, and can be one of the values shown in Usage.

6.68.2 Usage

The following table shows valid specifier combinations:

Table 6-23 STR (register) specifier combinations

R	*extend*
W	SXTW
W	UXTW
X	LSL
X	SXTX

6.68.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.69 STRB (immediate)

Store register byte (immediate offset).

6.69.1 Syntax

```
STRB  Wt, [Xn|SP], #simm    ; Post-index general registers
STRB  Wt, [Xn|SP, #simm]!    ; Pre-index general registers
STRB  Wt, [Xn|SP{, #pimm}]    ; Unsigned offset general registers
```

Where:

simm Is the signed immediate byte offset, in the range -256 to 255.

pimm Is the optional positive immediate byte offset, in the range 0 to 4095, defaulting to 0.

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.69.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.70 STRB (register)

Store register byte (register offset).

6.70.1 Syntax

```
STRB  Wt, [Xn|SP, Rm{, extend {amount}}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

R Is the index width specifier, and can be either W or X.

m Is the number of the general-purpose index register, in the range 0 to 30, or the name ZR (31).

extend Is the index extend/shift specifier, defaulting to LSL, and can be one of the values shown in Usage.

amount Is the index shift amount, optional and defaulting to #0 when *extend* is not LSL.

6.70.2 Usage

The following table shows valid specifier combinations:

Table 6-24 STRB (register) specifier combinations

R	*extend*
W	SXTW
W	UXTW
X	LSL
X	SXTX

6.70.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.71 STRH (immediate)

Store register halfword (immediate offset).

6.71.1 Syntax

```
STRH  Wt, [Xn|SP], #simm    ; Post-index general registers
STRH  Wt, [Xn|SP, #simm]!    ; Pre-index general registers
STRH  Wt, [Xn|SP{, #pimm}]    ; Unsigned offset general registers
```

Where:

simm Is the signed immediate byte offset, in the range -256 to 255.

pimm Is the optional positive immediate byte offset, a multiple of 2 in the range 0 to 8190, defaulting to 0.

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.71.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.72 STRH (register)

Store register halfword (register offset).

6.72.1 Syntax

```
STRH  Wt, [Xn|SP, Rm{, extend {amount}}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

R Is the index width specifier, and can be either W or X.

m Is the number of the general-purpose index register, in the range 0 to 30, or the name ZR (31).

extend Is the index extend/shift specifier, defaulting to LSL, and can be one of the values shown in Usage.

amount Is the index shift amount, optional and defaulting to #0 when *extend* is not LSL, and can be either #0 or #1.

6.72.2 Usage

The following table shows valid specifier combinations:

Table 6-25 STRH (register) specifier combinations

R	*extend*
W	SXTW
W	UXTW
X	LSL
X	SXTX

6.72.3 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.73 STTR

Store register (unprivileged).

6.73.1 Syntax

```
STTR  Wt, [Xn|SP{, #simm}]    ; 32-bit general registers
STTR  Xt, [Xn|SP{, #simm}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.73.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.74 STTRB

Store register byte (unprivileged).

6.74.1 Syntax

```
STTRB  Wt, [Xn|SP{, #simm}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.74.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.75 STTRH

Store register halfword (unprivileged).

6.75.1 Syntax

```
STTRH  Wt, [Xn|SP{, #simm}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.75.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.76 STUR (SIMD and FP)

Store SIMD and FP register (unscaled offset).

6.76.1 Syntax

```
STUR  Bt, [Xn|SP{, #simm}]    ; 8-bit FP/SIMD registers
STUR  Ht, [Xn|SP{, #simm}]    ; 16-bit FP/SIMD registers
STUR  St, [Xn|SP{, #simm}]    ; 32-bit FP/SIMD registers
STUR  Dt, [Xn|SP{, #simm}]    ; 64-bit FP/SIMD registers
STUR  Qt, [Xn|SP{, #simm}]    ; 128-bit FP/SIMD registers
```

Where:

Bt Is the 8-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Ht Is the 16-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

St Is the 32-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Dt Is the 64-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Qt Is the 128-bit name of the SIMD and FP register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.76.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.77 STUR

Store register (unscaled offset).

6.77.1 Syntax

```
STUR  Wt, [Xn|SP{, #simm}]    ; 32-bit general registers
STUR  Xt, [Xn|SP{, #simm}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.77.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.78 STURB

Store register byte (unscaled offset).

6.78.1 Syntax

```
STURB  Wt, [Xn|SP{, #simm}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.78.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.79 STURH

Store register halfword (unscaled offset).

6.79.1 Syntax

```
STURH  Wt, [Xn|SP{, #simm}]
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

simm Is the optional signed immediate byte offset, in the range -256 to 255, defaulting to 0.

6.79.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.80 STXP

Store exclusive pair of registers, returning status.

6.80.1 Syntax

```
STXP  Ws, Wt1, Wt2, [Xn|SP{,#0}]    ; 32-bit general registers
STXP  Ws, Xt1, Xt2, [Xn|SP{,#0}]    ; 64-bit general registers
```

Where:

Wt1 Is the 32-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Wt2 Is the 32-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

Xt1 Is the 64-bit name of the first general-purpose register to be transferred, in the range 0 to 31.

Xt2 Is the 64-bit name of the second general-purpose register to be transferred, in the range 0 to 31.

Ws Is the 32-bit name of the general-purpose register into which the status result of the store exclusive is written.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.80.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.81 STXR

Store exclusive register, returning status.

6.81.1 Syntax

```
STXR  Ws, Wt, [Xn|SP{,#0}]    ; 32-bit general registers
STXR  Ws, Xt, [Xn|SP{,#0}]    ; 64-bit general registers
```

Where:

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xt Is the 64-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Ws Is the 32-bit name of the general-purpose register into which the status result of the store exclusive is written.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.81.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.82 STXRB

Store exclusive register byte, returning status.

6.82.1 Syntax

```
STXRB  Ws, Wt, [Xn|SP{,#0}]
```

Where:

Ws Is the 32-bit name of the general-purpose register into which the status result of the store exclusive is written.

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.82.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

6.83 STXRH

Store exclusive register halfword, returning status.

6.83.1 Syntax

```
STXRH  Ws, Wt, [Xn|SP{,#0}]
```

Where:

Ws Is the 32-bit name of the general-purpose register into which the status result of the store exclusive is written.

Wt Is the 32-bit name of the general-purpose register to be transferred, in the range 0 to 31.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

6.83.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Chapter 7
A64 Floating-point Instructions

The following topic gives a summary of the A64 floating-point instructions supported by the ARM assembler:

- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.1 A64 floating-point instructions in alphabetical order

The following A64 floating-point instructions are supported:

Table 7-1 Location of floating-point instructions

Mnemonic	Brief description	See
FABS (scalar)	Floating-point absolute value	page 7-4
FADD (scalar)	Floating-point add	page 7-5
FCCMP	Floating-point conditional quiet compare, setting condition flags to result of comparison or an immediate value	page 7-6
FCCMPE	Floating-point conditional signaling compare, setting condition flags to result of comparison or an immediate value	page 7-7
FCMP	Floating-point quiet compare	page 7-8
FCMPE	Floating-point signaling compare	page 7-9
FCSEL	Floating-point conditional select	page 7-10
FCVT	Floating-point convert precision	page 7-11
FCVTAS (scalar)	Floating-point convert to signed integer, rounding to nearest with ties to away	page 7-12
FCVTAU (scalar)	Floating-point convert to unsigned integer, rounding to nearest with ties to away	page 7-13
FCVTMS (scalar)	Floating-point convert to signed integer, rounding toward minus infinity	page 7-14
FCVTMU (scalar)	Floating-point convert to unsigned integer, rounding toward minus infinity	page 7-15
FCVTNS (scalar)	Floating-point convert to signed integer, rounding to nearest with ties to even	page 7-16
FCVTNU (scalar)	Floating-point convert to unsigned integer, rounding to nearest with ties to even	page 7-17
FCVTPS (scalar)	Floating-point convert to signed integer, rounding toward positive infinity	page 7-18
FCVTPU (scalar)	Floating-point convert to unsigned integer, rounding toward positive infinity	page 7-19
FCVTZS (scalar, fixed-point)	Floating-point convert to signed fixed-point, rounding toward zero	page 7-20
FCVTZS (scalar, integer)	Floating-point convert to signed integer, rounding toward zero	page 7-21
FCVTZU (scalar, fixed-point)	Floating-point convert to unsigned fixed-point, rounding toward zero	page 7-22
FCVTZU (scalar, integer)	Floating-point convert to unsigned integer, rounding toward zero	page 7-23
FDIV (scalar)	Floating-point divide	page 7-24
FMADD	Floating-point fused multiply-add	page 7-25

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 7-1 Location of floating-point instructions (continued)

Mnemonic	Brief description	See
FMAX (scalar)	Floating-point maximum	page 7-26
FMAXNM (scalar)	Floating-point maximum number	page 7-27
FMIN (scalar)	Floating-point minimum	page 7-28
FMINNM (scalar)	Floating-point minimum number	page 7-29
FMOV (register)	Floating-point move register without conversion	page 7-30
FMOV (general)	Floating-point move to or from general-purpose register without conversion	page 7-31
FMOV (scalar, immediate)	Floating-point move immediate	page 7-32
FMSUB	Floating-point fused multiply-subtract	page 7-33
FMUL (scalar)	Floating-point multiply	page 7-34
FNEG (scalar)	Floating-point negate	page 7-35
FNMADD	Floating-point negated fused multiply-add	page 7-36
FNMSUB	Floating-point negated fused multiply-subtract	page 7-37
FNMUL	Floating-point multiply-negate	page 7-38
FRINTA (scalar)	Floating-point round to integral, to nearest with ties to away	page 7-39
FRINTI (scalar)	Floating-point round to integral, using current rounding mode	page 7-40
FRINTM (scalar)	Floating-point round to integral, toward minus infinity	page 7-41
FRINTN (scalar)	Floating-point round to integral, to nearest with ties to even	page 7-42
FRINTP (scalar)	Floating-point round to integral, toward positive infinity	page 7-43
FRINTX (scalar)	Floating-point round to integral exact, using current rounding mode	page 7-44
FRINTZ (scalar)	Floating-point round to integral, toward zero	page 7-45
FSQRT (scalar)	Floating-point square root	page 7-46
FSUB (scalar)	Floating-point subtract	page 7-47
SCVTF (scalar, fixed-point)	Signed fixed-point convert to floating-point	page 7-48
SCVTF (scalar, integer)	Signed integer convert to floating-point	page 7-49
UCVTF (scalar, fixed-point)	Unsigned fixed-point convert to floating-point	page 7-50
UCVTF (scalar, integer)	Unsigned integer convert to floating-point	page 7-51

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.2 FABS (scalar)

Floating-point absolute value.

7.2.1 Syntax

```
FABS  Sd, Sn    ; Single-precision
FABS  Dd, Dn    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.2.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.3 FADD (scalar)

Floating-point add.

7.3.1 Syntax

```
FADD  Sd, Sn, Sm    ; Single-precision
FADD  Dd, Dn, Dm    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

7.3.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.4 FCCMP

Floating-point conditional quiet compare, setting condition flags to result of comparison or an immediate value.

7.4.1 Syntax

```
FCCMP  Sn, Sm, #nzcv, cond    ; Single-precision
FCCMP  Dn, Dm, #nzcv, cond    ; Double-precision
```

Where:

Sn Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

nzcv Is the flag bit specifier, an immediate in the range 0 to 15, giving the alternative state for the 4-bit NZCV condition flags. Bit 3 is the N flag, bit 2 is the Z flag, bit 1 is the C flag, and bit 0 is the V flag.

cond Is one of the standard conditions.

7.4.2 See also

Reference

- *Condition codes* on page 3-26
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.5 FCCMPE

Floating-point conditional signaling compare, setting condition flags to result of comparison or an immediate value.

7.5.1 Syntax

```
FCCMPE  Sn, Sm, #nzcv, cond    ; Single-precision
FCCMPE  Dn, Dm, #nzcv, cond    ; Double-precision
```

Where:

Sn Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

nzcv Is the flag bit specifier, an immediate in the range 0 to 15, giving the alternative state for the 4-bit NZCV condition flags. Bit 3 is the N flag, bit 2 is the Z flag, bit 1 is the C flag, and bit 0 is the V flag.

cond Is one of the standard conditions.

7.5.2 See also

Reference

- *Condition codes* on page 3-26
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.6 FCMP

Floating-point quiet compare.

7.6.1 Syntax

```
FCMP  Sn, Sm    ; Single-precision
FCMP  Sn, #0.0    ; Single-precision, zero
FCMP  Dn, Dm    ; Double-precision
FCMP  Dn, #0.0    ; Double-precision, zero
```

Where:

Sn The value depends on the instruction variant:

Single-precision

Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Single-precision, zero

Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dn The value depends on the instruction variant:

Double-precision

Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Double-precision, zero

Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

7.6.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.7 FCMPE

Floating-point signaling compare.

7.7.1 Syntax

```
FCMPE  Sn, Sm    ; Single-precision
FCMPE  Sn, #0.0    ; Single-precision, zero
FCMPE  Dn, Dm    ; Double-precision
FCMPE  Dn, #0.0    ; Double-precision, zero
```

Where:

Sn The value depends on the instruction variant:

Single-precision

Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Single-precision, zero

Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dn The value depends on the instruction variant:

Double-precision

Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Double-precision, zero

Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

7.7.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.8 FCSEL

Floating-point conditional select.

7.8.1 Syntax

```
FCSEL  Sd, Sn, Sm, cond    ; Single-precision
FCSEL  Dd, Dn, Dm, cond    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

cond Is one of the standard conditions.

7.8.2 See also

Reference

- *Condition codes* on page 3-26
- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.9 FCVT

Floating-point convert precision.

7.9.1 Syntax

```
FCVT  Sd, Hn    ; Half-precision to single-precision
FCVT  Dd, Hn    ; Half-precision to double-precision
FCVT  Hd, Sn    ; Single-precision to half-precision
FCVT  Dd, Sn    ; Single-precision to double-precision
FCVT  Hd, Dn    ; Double-precision to half-precision
FCVT  Sd, Dn    ; Double-precision to single-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Hn Is the 16-bit name of the SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Hd Is the 16-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.9.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.10 FCVTAS (scalar)

Floating-point convert to signed integer, rounding to nearest with ties to away.

7.10.1 Syntax

```
FCVTAS  Wd, Sn    ; Single-precision to 32-bit
FCVTAS  Xd, Sn    ; Single-precision to 64-bit
FCVTAS  Wd, Dn    ; Double-precision to 32-bit
FCVTAS  Xd, Dn    ; Double-precision to 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.10.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.11 FCVTAU (scalar)

Floating-point convert to unsigned integer, rounding to nearest with ties to away.

7.11.1 Syntax

```
FCVTAU  Wd, Sn    ; Single-precision to 32-bit
FCVTAU  Xd, Sn    ; Single-precision to 64-bit
FCVTAU  Wd, Dn    ; Double-precision to 32-bit
FCVTAU  Xd, Dn    ; Double-precision to 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.11.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.12 FCVTMS (scalar)

Floating-point convert to signed integer, rounding toward minus infinity.

7.12.1 Syntax

```
FCVTMS  Wd, Sn    ; Single-precision to 32-bit
FCVTMS  Xd, Sn    ; Single-precision to 64-bit
FCVTMS  Wd, Dn    ; Double-precision to 32-bit
FCVTMS  Xd, Dn    ; Double-precision to 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.12.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.13 FCVTMU (scalar)

Floating-point convert to unsigned integer, rounding toward minus infinity.

7.13.1 Syntax

```
FCVTMU  Wd, Sn    ; Single-precision to 32-bit
FCVTMU  Xd, Sn    ; Single-precision to 64-bit
FCVTMU  Wd, Dn    ; Double-precision to 32-bit
FCVTMU  Xd, Dn    ; Double-precision to 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.13.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.14 FCVTNS (scalar)

Floating-point convert to signed integer, rounding to nearest with ties to even.

7.14.1 Syntax

```
FCVTNS  Wd, Sn    ; Single-precision to 32-bit
FCVTNS  Xd, Sn    ; Single-precision to 64-bit
FCVTNS  Wd, Dn    ; Double-precision to 32-bit
FCVTNS  Xd, Dn    ; Double-precision to 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.14.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.15 FCVTNU (scalar)

Floating-point convert to unsigned integer, rounding to nearest with ties to even.

7.15.1 Syntax

```
FCVTNU  Wd, Sn    ; Single-precision to 32-bit
FCVTNU  Xd, Sn    ; Single-precision to 64-bit
FCVTNU  Wd, Dn    ; Double-precision to 32-bit
FCVTNU  Xd, Dn    ; Double-precision to 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.15.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.16 FCVTPS (scalar)

Floating-point convert to signed integer, rounding toward positive infinity.

7.16.1 Syntax

```
FCVTPS  Wd, Sn    ; Single-precision to 32-bit
FCVTPS  Xd, Sn    ; Single-precision to 64-bit
FCVTPS  Wd, Dn    ; Double-precision to 32-bit
FCVTPS  Xd, Dn    ; Double-precision to 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.16.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.17 FCVTPU (scalar)

Floating-point convert to unsigned integer, rounding toward positive infinity.

7.17.1 Syntax

```
FCVTPU  Wd, Sn    ; Single-precision to 32-bit
FCVTPU  Xd, Sn    ; Single-precision to 64-bit
FCVTPU  Wd, Dn    ; Double-precision to 32-bit
FCVTPU  Xd, Dn    ; Double-precision to 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.17.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.18 FCVTZS (scalar, fixed-point)

Floating-point convert to signed fixed-point, rounding toward zero.

7.18.1 Syntax

```
FCVTZS  Wd, Sn, #fbits    ; Single-precision to 32-bit
FCVTZS  Xd, Sn, #fbits    ; Single-precision to 64-bit
FCVTZS  Wd, Dn, #fbits    ; Double-precision to 32-bit
FCVTZS  Xd, Dn, #fbits    ; Double-precision to 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

fbits The value depends on the instruction variant:

32-bit Is the number of bits after the binary point in the fixed-point destination, in the range 1 to 32.

64-bit Is the number of bits after the binary point in the fixed-point destination, in the range 1 to 64.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.18.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.19 FCVTZS (scalar, integer)

Floating-point convert to signed integer, rounding toward zero.

7.19.1 Syntax

```
FCVTZS  Wd, Sn    ; Single-precision to 32-bit
FCVTZS  Xd, Sn    ; Single-precision to 64-bit
FCVTZS  Wd, Dn    ; Double-precision to 32-bit
FCVTZS  Xd, Dn    ; Double-precision to 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.19.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.20 FCVTZU (scalar, fixed-point)

Floating-point convert to unsigned fixed-point, rounding toward zero.

7.20.1 Syntax

```
FCVTZU  Wd, Sn, #fbits    ; Single-precision to 32-bit
FCVTZU  Xd, Sn, #fbits    ; Single-precision to 64-bit
FCVTZU  Wd, Dn, #fbits    ; Double-precision to 32-bit
FCVTZU  Xd, Dn, #fbits    ; Double-precision to 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

fbits The value depends on the instruction variant:

32-bit Is the number of bits after the binary point in the fixed-point destination, in the range 1 to 32.

64-bit Is the number of bits after the binary point in the fixed-point destination, in the range 1 to 64.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.20.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.21 FCVTZU (scalar, integer)

Floating-point convert to unsigned integer, rounding toward zero.

7.21.1 Syntax

```
FCVTZU  Wd, Sn    ; Single-precision to 32-bit
FCVTZU  Xd, Sn    ; Single-precision to 64-bit
FCVTZU  Wd, Dn    ; Double-precision to 32-bit
FCVTZU  Xd, Dn    ; Double-precision to 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.21.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.22 FDIV (scalar)

Floating-point divide.

7.22.1 Syntax

```
FDIV  Sd, Sn, Sm    ; Single-precision
FDIV  Dd, Dn, Dm    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

7.22.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.23 FMADD

Floating-point fused multiply-add.

7.23.1 Syntax

```
FMADD  Sd, Sn, Sm, Sa    ; Single-precision
FMADD  Dd, Dn, Dm, Da    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register holding the multiplicand, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register holding the multiplier. The register name can be in the range 0 to 31.

Sa Is the 32-bit name of the third SIMD and FP source register holding the addend.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register holding the multiplicand, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register holding the multiplier. The register name can be in the range 0 to 31.

Da Is the 64-bit name of the third SIMD and FP source register holding the addend.

7.23.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.24 FMAX (scalar)

Floating-point maximum.

7.24.1 Syntax

```
FMAX  Sd, Sn, Sm    ; Single-precision
FMAX  Dd, Dn, Dm    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

7.24.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.25 FMAXNM (scalar)

Floating-point maximum number.

7.25.1 Syntax

```
FMAXNM  Sd, Sn, Sm    ; Single-precision
FMAXNM  Dd, Dn, Dm    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

7.25.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.26 FMIN (scalar)

Floating-point minimum.

7.26.1 Syntax

```
FMIN  Sd, Sn, Sm    ; Single-precision
FMIN  Dd, Dn, Dm    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

7.26.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.27 FMINNM (scalar)

Floating-point minimum number.

7.27.1 Syntax

```
FMINNM  Sd, Sn, Sm    ; Single-precision
FMINNM  Dd, Dn, Dm    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

7.27.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.28 FMOV (register)

Floating-point move register without conversion.

7.28.1 Syntax

```
FMOV  Sd, Sn    ; Single-precision
FMOV  Dd, Dn    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.28.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.29 FMOV (general)

Floating-point move to or from general-purpose register without conversion.

7.29.1 Syntax

```
FMOV  Sd, Wn    ; 32-bit to single-precision
FMOV  Wd, Sn    ; Single-precision to 32-bit
FMOV  Dd, Xn    ; 64-bit to double-precision
FMOV  Vd.D[1], Xn    ; 64-bit to top half of 128-bit
FMOV  Xd, Dn    ; Double-precision to 64-bit
FMOV  Xd, Vn.D[1]    ; Top half of 128-bit to 64-bit
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

7.29.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.30 FMOV (scalar, immediate)

Floating-point move immediate.

7.30.1 Syntax

```
FMOV  Sd, #imm    ; Single-precision
FMOV  Dd, #imm    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

imm Is a signed floating-point constant with 3-bit exponent and normalized 4 bits of precision.

7.30.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.31 FMSUB

Floating-point fused multiply-subtract.

7.31.1 Syntax

```
FMSUB  Sd, Sn, Sm, Sa    ; Single-precision
FMSUB  Dd, Dn, Dm, Da    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register holding the multiplicand, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register holding the multiplier. The register name can be in the range 0 to 31.

Sa Is the 32-bit name of the third SIMD and FP source register holding the minuend.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register holding the multiplicand, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register holding the multiplier. The register name can be in the range 0 to 31.

Da Is the 64-bit name of the third SIMD and FP source register holding the minuend.

7.31.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.32 FMUL (scalar)

Floating-point multiply.

7.32.1 Syntax

```
FMUL  Sd, Sn, Sm    ; Single-precision
FMUL  Dd, Dn, Dm    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

7.32.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.33 FNEG (scalar)

Floating-point negate.

7.33.1 Syntax

```
FNEG  Sd, Sn    ; Single-precision
FNEG  Dd, Dn    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.33.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.34 FNMADD

Floating-point negated fused multiply-add.

7.34.1 Syntax

```
FNMADD  Sd, Sn, Sm, Sa    ; Single-precision
FNMADD  Dd, Dn, Dm, Da    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register holding the multiplicand, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register holding the multiplier. The register name can be in the range 0 to 31.

Sa Is the 32-bit name of the third SIMD and FP source register holding the addend.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register holding the multiplicand, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register holding the multiplier. The register name can be in the range 0 to 31.

Da Is the 64-bit name of the third SIMD and FP source register holding the addend.

7.34.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.35 FNMSUB

Floating-point negated fused multiply-subtract.

7.35.1 Syntax

```
FNMSUB  Sd, Sn, Sm, Sa    ; Single-precision
FNMSUB  Dd, Dn, Dm, Da    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register holding the multiplicand, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register holding the multiplier. The register name can be in the range 0 to 31.

Sa Is the 32-bit name of the third SIMD and FP source register holding the minuend.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register holding the multiplicand, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register holding the multiplier. The register name can be in the range 0 to 31.

Da Is the 64-bit name of the third SIMD and FP source register holding the minuend.

7.35.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.36 FNMUL

Floating-point multiply-negate.

7.36.1 Syntax

```
FNMUL  Sd, Sn, Sm    ; Single-precision
FNMUL  Dd, Dn, Dm    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

7.36.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.37 FRINTA (scalar)

Floating-point round to integral, to nearest with ties to away.

7.37.1 Syntax

```
FRINTA  Sd, Sn    ; Single-precision
FRINTA  Dd, Dn    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.37.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.38 FRINTI (scalar)

Floating-point round to integral, using current rounding mode.

7.38.1 Syntax

```
FRINTI  Sd, Sn    ; Single-precision
FRINTI  Dd, Dn    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.38.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.39 FRINTM (scalar)

Floating-point round to integral, toward minus infinity.

7.39.1 Syntax

```
FRINTM  Sd, Sn    ; Single-precision
FRINTM  Dd, Dn    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.39.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.40 FRINTN (scalar)

Floating-point round to integral, to nearest with ties to even.

7.40.1 Syntax

```
FRINTN  Sd, Sn    ; Single-precision
FRINTN  Dd, Dn    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.40.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.41 FRINTP (scalar)

Floating-point round to integral, toward positive infinity.

7.41.1 Syntax

```
FRINTP  Sd, Sn    ; Single-precision
FRINTP  Dd, Dn    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.41.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.42 FRINTX (scalar)

Floating-point round to integral exact, using current rounding mode.

7.42.1 Syntax

```
FRINTX  Sd, Sn    ; Single-precision
FRINTX  Dd, Dn    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.42.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.43 FRINTZ (scalar)

Floating-point round to integral, toward zero.

7.43.1 Syntax

```
FRINTZ  Sd, Sn    ; Single-precision
FRINTZ  Dd, Dn    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.43.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.44 FSQRT (scalar)

Floating-point square root.

7.44.1 Syntax

```
FSQRT  Sd, Sn    ; Single-precision
FSQRT  Dd, Dn    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the SIMD and FP source register, in the range 0 to 31.

7.44.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.45 FSUB (scalar)

Floating-point subtract.

7.45.1 Syntax

```
FSUB  Sd, Sn, Sm    ; Single-precision
FSUB  Dd, Dn, Dm    ; Double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Sn Is the 32-bit name of the first SIMD and FP source register, in the range 0 to 31.

Sm Is the 32-bit name of the second SIMD and FP source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Dn Is the 64-bit name of the first SIMD and FP source register, in the range 0 to 31.

Dm Is the 64-bit name of the second SIMD and FP source register, in the range 0 to 31.

7.45.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.46 SCVTF (scalar, fixed-point)

Signed fixed-point convert to floating-point.

7.46.1 Syntax

```
SCVTF  Sd, Wn, #fbits    ; 32-bit to single-precision
SCVTF  Dd, Wn, #fbits    ; 32-bit to double-precision
SCVTF  Sd, Xn, #fbits    ; 64-bit to single-precision
SCVTF  Dd, Xn, #fbits    ; 64-bit to double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

fbits The value depends on the instruction variant:

32-bit Is the number of bits after the binary point in the fixed-point source, in the range 1 to 32.

64-bit Is the number of bits after the binary point in the fixed-point source, in the range 1 to 64.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

7.46.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.47 SCVTF (scalar, integer)

Signed integer convert to floating-point.

7.47.1 Syntax

```
SCVTF  Sd, Wn    ; 32-bit to single-precision
SCVTF  Dd, Wn    ; 32-bit to double-precision
SCVTF  Sd, Xn    ; 64-bit to single-precision
SCVTF  Dd, Xn    ; 64-bit to double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

7.47.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.48 UCVTF (scalar, fixed-point)

Unsigned fixed-point convert to floating-point.

7.48.1 Syntax

```
UCVTF  Sd, Wn, #fbits    ; 32-bit to single-precision
UCVTF  Dd, Wn, #fbits    ; 32-bit to double-precision
UCVTF  Sd, Xn, #fbits    ; 64-bit to single-precision
UCVTF  Dd, Xn, #fbits    ; 64-bit to double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

fbits The value depends on the instruction variant:

32-bit Is the number of bits after the binary point in the fixed-point source, in the range 1 to 32.

64-bit Is the number of bits after the binary point in the fixed-point source, in the range 1 to 64.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

7.48.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

7.49 UCVTF (scalar, integer)

Unsigned integer convert to floating-point.

7.49.1 Syntax

```
UCVTF  Sd, Wn    ; 32-bit to single-precision
UCVTF  Dd, Wn    ; 32-bit to double-precision
UCVTF  Sd, Xn    ; 64-bit to single-precision
UCVTF  Dd, Xn    ; 64-bit to double-precision
```

Where:

Sd Is the 32-bit name of the SIMD and FP destination register, in the range 0 to 31.

Wn Is the 32-bit name of the general-purpose source register, in the range 0 to 31.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

Xn Is the 64-bit name of the general-purpose source register, in the range 0 to 31.

7.49.2 See also

Reference

- *A64 general instructions in alphabetical order* on page 5-2.
- *A64 data transfer instructions in alphabetical order* on page 6-2.
- *A64 floating-point instructions in alphabetical order* on page 7-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Chapter 8
A64 SIMD Scalar Instructions

The following topic gives a summary of the A64 SIMD scalar instructions supported by the ARM assembler:

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.1 A64 SIMD scalar instructions in alphabetical order

The following A64 SIMD scalar instructions are supported:

Table 8-1 Location of SIMD scalar instructions

Mnemonic	Brief description	See
ABS (scalar)	Absolute value	page 8-7
ADD (scalar)	Add	page 8-8
ADDP (scalar)	Add pair of elements	page 8-9
CMEQ (scalar, register)	Compare bitwise equal, setting destination vector element to all ones if the condition holds, else zero	page 8-10
CMEQ (scalar, zero)	Compare bitwise equal to zero, setting destination vector element to all ones if the condition holds, else zero	page 8-11
CMGE (scalar, register)	Compare signed greater than or equal	page 8-12
CMGE (scalar, zero)	Compare signed greater than or equal to zero, setting destination vector element to all ones if the condition holds, else zero	page 8-13
CMGT (scalar, register)	Compare signed greater than, setting destination vector element to all ones if the condition holds, else zero	page 8-14
CMGT (scalar, zero)	Compare signed greater than zero, setting destination vector element to all ones if the condition holds, else zero	page 8-15
CMHI (scalar, register)	Compare unsigned higher, setting destination vector element to all ones if the condition holds, else zero	page 8-16
CMHS (scalar, register)	Compare unsigned higher or same, setting destination vector element to all ones if the condition holds, else zero	page 8-17
CMLE (scalar, zero)	Compare signed less than or equal to zero, setting destination vector element to all ones if the condition holds, else zero	page 8-18
CMLT (scalar, zero)	Compare signed less than zero, setting destination vector element to all ones if the condition holds, else zero	page 8-19
CMTST (scalar)	Compare bitwise test bits nonzero, setting destination vector element to all ones if the condition holds, else zero	page 8-20
DUP (scalar, element)	Duplicate vector element to scalar	page 8-21
FABD (scalar)	Floating-point absolute difference	page 8-22
FACGE (scalar)	Floating-point absolute compare greater than or equal	page 8-23
FACGT (scalar)	Floating-point absolute compare greater than	page 8-24
FADDP (scalar)	Floating-point add pair of elements	page 8-25
FCMEQ (scalar, register)	Floating-point compare equal, setting destination vector element to all ones if the condition holds, else zero	page 8-26
FCMEQ (scalar, zero)	Floating-point compare equal to zero, setting destination vector element to all ones if the condition holds, else zero	page 8-27

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 8-1 Location of SIMD scalar instructions (continued)

Mnemonic	Brief description	See
FCMGE (scalar, register)	Floating-point compare greater than or equal, setting destination vector element to all ones if the condition holds, else zero	page 8-28
FCMGE (scalar, zero)	Floating-point compare greater than or equal to zero, setting destination vector element to all ones if the condition holds, else zero	page 8-29
FCMGT (scalar, register)	Floating-point compare greater than, setting destination vector element to all ones if the condition holds, else zero	page 8-30
FCMGT (scalar, zero)	Floating-point compare greater than zero, setting destination vector element to all ones if the condition holds, else zero	page 8-31
FCMLE (scalar, zero)	Floating-point compare less than or equal to zero, setting destination vector element to all ones if the condition holds, else zero	page 8-32
FCMLT (scalar, zero)	Floating-point compare less than zero, setting destination vector element to all ones if the condition holds, else zero	page 8-33
FCVTAS (scalar)	Floating-point convert to signed integer, rounding to nearest with ties to away	page 8-34
FCVTAU (scalar)	Floating-point convert to unsigned integer, rounding to nearest with ties to away	page 8-35
FCVTMS (scalar)	Floating-point convert to signed integer, rounding toward minus infinity	page 8-36
FCVTMU (scalar)	Floating-point convert to unsigned integer, rounding toward minus infinity	page 8-37
FCVTNS (scalar)	Floating-point convert to signed integer, rounding to nearest with ties to even	page 8-38
FCVTNU (scalar)	Floating-point convert to unsigned integer, rounding to nearest with ties to even	page 8-39
FCVTPS (scalar)	Floating-point convert to signed integer, rounding toward positive infinity	page 8-40
FCVTPU (scalar)	Floating-point convert to unsigned integer, rounding toward positive infinity	page 8-41
FCVTXN (scalar)	Floating-point convert to lower precision narrow, rounding to odd	page 8-42
FCVTZS (scalar, fixed-point)	Floating-point convert to signed fixed-point, rounding toward zero	page 8-43
FCVTZS (scalar, integer)	Floating-point convert to signed integer, rounding toward zero	page 8-44
FCVTZU (scalar, fixed-point)	Floating-point convert to unsigned fixed-point, rounding toward zero	page 8-45
FCVTZU (scalar, integer)	Floating-point convert to unsigned integer, rounding toward zero	page 8-46

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 8-1 Location of SIMD scalar instructions (continued)

Mnemonic	Brief description	See
FMAXNMP (scalar)	Floating-point maximum number of pair of elements	page 8-47
FMAXP (scalar)	Floating-point maximum of pair of elements	page 8-48
FMINNMP (scalar)	Floating-point minimum number of pair of elements	page 8-49
FMINP (scalar)	Floating-point minimum of pair of elements	page 8-50
FMLA (scalar, by element)	Floating-point fused multiply-add to accumulator (by element)	page 8-51
FMLS (scalar, by element)	Floating-point fused multiply-subtract from accumulator (by element)	page 8-52
FMUL (scalar, by element)	Floating-point multiply (by element)	page 8-53
FMULX (scalar, by element)	Floating-point multiply extended (by element)	page 8-54
FMULX (scalar)	Floating-point multiply extended	page 8-55
FRECPE (scalar)	Floating-point reciprocal estimate	page 8-56
FRECPS (scalar)	Floating-point reciprocal step	page 8-57
FRECPX (scalar)	Floating-point reciprocal exponent	page 8-58
FRSQRTE (scalar)	Floating-point reciprocal square root estimate	page 8-59
FRSQRTS (scalar)	Floating-point reciprocal square root step	page 8-60
MOV (scalar)	Move vector element to scalar	page 8-61
NEG (scalar)	Negate	page 8-62
SCVTF (scalar, fixed-point)	Signed fixed-point convert to floating-point	page 8-63
SCVTF (scalar, integer)	Signed integer convert to floating-point	page 8-64
SHL (scalar)	Shift left (immediate)	page 8-65
SLI (scalar)	Shift left and insert (immediate)	page 8-66
SQABS (scalar)	Signed saturating absolute value	page 8-67
SQADD (scalar)	Signed saturating add	page 8-68
SQDMLAL (scalar, by element)	Signed saturating doubling multiply-add long (by element)	page 8-69
SQDMLAL (scalar)	Signed saturating doubling multiply-add long	page 8-70
SQDMLSL (scalar, by element)	Signed saturating doubling multiply-subtract long (by element)	page 8-71
SQDMLSL (scalar)	Signed saturating doubling multiply-subtract long	page 8-72
SQDMULH (scalar, by element)	Signed saturating doubling multiply returning high half (by element)	page 8-73
SQDMULH (scalar)	Signed saturating doubling multiply returning high half	page 8-74
SQDMULL (scalar, by element)	Signed saturating doubling multiply long (by element)	page 8-75
SQDMULL (scalar)	Signed saturating doubling multiply long	page 8-76

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 8-1 Location of SIMD scalar instructions (continued)

Mnemonic	Brief description	See
SQNEG (scalar)	Signed saturating negate	page 8-77
SQRDMULH (scalar, by element)	Signed saturating rounding doubling multiply returning high half (by element)	page 8-78
SQRDMULH (scalar)	Signed saturating rounding doubling multiply returning high half	page 8-79
SQRSHL (scalar)	Signed saturating rounding shift left (register)	page 8-80
SQRSHRN (scalar)	Signed saturating rounded shift right narrow (immediate)	page 8-81
SQRSHRUN (scalar)	Signed saturating rounded shift right unsigned narrow (immediate)	page 8-82
SQSHL (scalar, immediate)	Signed saturating shift left (immediate)	page 8-83
SQSHL (scalar, register)	Signed saturating shift left (register)	page 8-84
SQSHLU (scalar)	Signed saturating shift left unsigned (immediate)	page 8-85
SQSHRN (scalar)	Signed saturating shift right narrow (immediate)	page 8-86
SQSHRUN (scalar)	Signed saturating shift right unsigned narrow (immediate)	page 8-87
SQSUB (scalar)	Signed saturating subtract	page 8-88
SQXTN (scalar)	Signed saturating extract narrow	page 8-89
SQXTUN (scalar)	Signed saturating extract unsigned narrow	page 8-90
SRI (scalar)	Shift right and insert (immediate)	page 8-91
SRSHL (scalar)	Signed rounding shift left (register)	page 8-92
SRSHR (scalar)	Signed rounding shift right (immediate)	page 8-93
SRSRA (scalar)	Signed rounding shift right and accumulate (immediate)	page 8-94
SSHL (scalar)	Signed shift left (register)	page 8-95
SSHR (scalar)	Signed shift right (immediate)	page 8-96
SSRA (scalar)	Signed shift right and accumulate (immediate)	page 8-97
SUB (scalar)	Subtract	page 8-98
SUQADD (scalar)	Signed saturating accumulate of unsigned value	page 8-99
UCVTF (scalar, fixed-point)	Unsigned fixed-point convert to floating-point	page 8-100
UCVTF (scalar, integer)	Unsigned integer convert to floating-point	page 8-101
UQADD (scalar)	Unsigned saturating add	page 8-102
UQRSHL (scalar)	Unsigned saturating rounding shift left (register)	page 8-103
UQRSHRN (scalar)	Unsigned saturating rounded shift right narrow (immediate)	page 8-104
UQSHL (scalar, immediate)	Unsigned saturating shift left (immediate)	page 8-105
UQSHL (scalar, register)	Unsigned saturating shift left (register)	page 8-106

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 8-1 Location of SIMD scalar instructions (continued)

Mnemonic	Brief description	See
UQSHRN (scalar)	Unsigned saturating shift right narrow (immediate)	page 8-107
UQSUB (scalar)	Unsigned saturating subtract	page 8-108
UQXTN (scalar)	Unsigned saturating extract narrow	page 8-109
URSHL (scalar)	Unsigned rounding shift left (register)	page 8-110
URSHR (scalar)	Unsigned rounding shift right (immediate)	page 8-111
URSRA (scalar)	Unsigned rounding shift right and accumulate (immediate)	page 8-112
USHL (scalar)	Unsigned shift left (register)	page 8-113
USHR (scalar)	Unsigned shift right (immediate)	page 8-114
USQADD (scalar)	Unsigned saturating accumulate of signed value	page 8-115
USRA (scalar)	Unsigned shift right and accumulate (immediate)	page 8-116

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.2 ABS (scalar)

Absolute value.

8.2.1 Syntax

```
ABS  Vd, Vn
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.2.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.3 ADD (scalar)

Add.

8.3.1 Syntax

```
ADD  Vd, Vn, Vm
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.3.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.4 ADDP (scalar)

Add pair of elements.

8.4.1 Syntax

```
ADDP  Vd, Vn.T
```

Where:

V Is the destination width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is the source arrangement specifier, `2D`.

8.4.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.5 CMEQ (scalar, register)

Compare bitwise equal, setting destination vector element to all ones if the condition holds, else zero.

8.5.1 Syntax

```
CMEQ  Vd, Vn, Vm
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.5.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.6 CMEQ (scalar, zero)

Compare bitwise equal to zero, setting destination vector element to all ones if the condition holds, else zero.

8.6.1 Syntax

```
CMEQ  Vd, Vn, #0
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.6.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.7 CMGE (scalar, register)

Compare signed greater than or equal.

8.7.1 Syntax

```
CMGE  Vd, Vn, Vm
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.7.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.8 CMGE (scalar, zero)

Compare signed greater than or equal to zero, setting destination vector element to all ones if the condition holds, else zero.

8.8.1 Syntax

```
CMGE  Vd, Vn, #0
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.8.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.9 CMGT (scalar, register)

Compare signed greater than, setting destination vector element to all ones if the condition holds, else zero.

8.9.1 Syntax

```
CMGT  Vd, Vn, Vm
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.9.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.10 CMGT (scalar, zero)

Compare signed greater than zero, setting destination vector element to all ones if the condition holds, else zero.

8.10.1 Syntax

```
CMGT  Vd, Vn, #0
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.10.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.11 CMHI (scalar, register)

Compare unsigned higher, setting destination vector element to all ones if the condition holds, else zero.

8.11.1 Syntax

```
CMHI  Vd, Vn, Vm
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.11.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.12 CMHS (scalar, register)

Compare unsigned higher or same, setting destination vector element to all ones if the condition holds, else zero.

8.12.1 Syntax

```
CMHS  Vd, Vn, Vm
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.12.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.13 CMLE (scalar, zero)

Compare signed less than or equal to zero, setting destination vector element to all ones if the condition holds, else zero.

8.13.1 Syntax

```
CMLE  Vd, Vn, #0
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.13.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.14 CMLT (scalar, zero)

Compare signed less than zero, setting destination vector element to all ones if the condition holds, else zero.

8.14.1 Syntax

```
CMLT  Vd, Vn, #0
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.14.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.15 CMTST (scalar)

Compare bitwise test bits nonzero, setting destination vector element to all ones if the condition holds, else zero.

8.15.1 Syntax

```
CMTST  Vd, Vn, Vm
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.15.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.16 DUP (scalar, element)

Duplicate vector element to scalar.

This instruction is used by the alias `MOV` (scalar).

8.16.1 Syntax

```
DUP  Vd, Vn.T[index]
```

Where:

V Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

T Is the element width specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

index Is the element index, in the range shown in Usage.

8.16.2 Usage

The following table shows valid specifier combinations:

Table 8-2 DUP (Scalar) specifier combinations

V	*T*	*index*
B	B	0 to 15
H	H	0 to 7
S	S	0 to 3
D	D	0 or 1

8.16.3 See also

Reference

- *MOV (scalar)* on page 8-61.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.17 FABD (scalar)

Floating-point absolute difference.

8.17.1 Syntax

```
FABD  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.17.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.18 FACGE (scalar)

Floating-point absolute compare greater than or equal.

8.18.1 Syntax

```
FACGE  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.18.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.19 FACGT (scalar)

Floating-point absolute compare greater than.

8.19.1 Syntax

```
FACGT  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.19.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.20 FADDP (scalar)

Floating-point add pair of elements.

8.20.1 Syntax

```
FADDP  Vd, Vn.T
```

Where:

V Is the destination width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is the source arrangement specifier, and can be either `2S` or `2D`.

8.20.2 Usage

The following table shows valid specifier combinations:

Table 8-3 FADDP specifier combinations

V	*T*
S	2S
D	2D

8.20.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.21 FCMEQ (scalar, register)

Floating-point compare equal, setting destination vector element to all ones if the condition holds, else zero.

8.21.1 Syntax

```
FCMEQ  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.21.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.22 FCMEQ (scalar, zero)

Floating-point compare equal to zero, setting destination vector element to all ones if the condition holds, else zero.

8.22.1 Syntax

```
FCMEQ  Vd, Vn, #0.0
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.22.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.23 FCMGE (scalar, register)

Floating-point compare greater than or equal, setting destination vector element to all ones if the condition holds, else zero.

8.23.1 Syntax

```
FCMGE  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.23.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.24 FCMGE (scalar, zero)

Floating-point compare greater than or equal to zero, setting destination vector element to all ones if the condition holds, else zero.

8.24.1 Syntax

```
FCMGE  Vd, Vn, #0.0
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.24.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.25 FCMGT (scalar, register)

Floating-point compare greater than, setting destination vector element to all ones if the condition holds, else zero.

8.25.1 Syntax

```
FCMGT  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.25.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.26 FCMGT (scalar, zero)

Floating-point compare greater than zero, setting destination vector element to all ones if the condition holds, else zero.

8.26.1 Syntax

```
FCMGT  Vd, Vn, #0.0
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.26.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.27 FCMLE (scalar, zero)

Floating-point compare less than or equal to zero, setting destination vector element to all ones if the condition holds, else zero.

8.27.1 Syntax

```
FCMLE  Vd, Vn, #0.0
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.27.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.28 FCMLT (scalar, zero)

Floating-point compare less than zero, setting destination vector element to all ones if the condition holds, else zero.

8.28.1 Syntax

```
FCMLT  Vd, Vn, #0.0
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.28.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.29 FCVTAS (scalar)

Floating-point convert to signed integer, rounding to nearest with ties to away.

8.29.1 Syntax

```
FCVTAS  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.29.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.30 FCVTAU (scalar)

Floating-point convert to unsigned integer, rounding to nearest with ties to away.

8.30.1 Syntax

```
FCVTAU  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.30.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.31 FCVTMS (scalar)

Floating-point convert to signed integer, rounding toward minus infinity.

8.31.1 Syntax

```
FCVTMS  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.31.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.32 FCVTMU (scalar)

Floating-point convert to unsigned integer, rounding toward minus infinity.

8.32.1 Syntax

```
FCVTMU  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.32.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.33 FCVTNS (scalar)

Floating-point convert to signed integer, rounding to nearest with ties to even.

8.33.1 Syntax

```
FCVTNS  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.33.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.34 FCVTNU (scalar)

Floating-point convert to unsigned integer, rounding to nearest with ties to even.

8.34.1 Syntax

```
FCVTNU  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.34.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.35 FCVTPS (scalar)

Floating-point convert to signed integer, rounding toward positive infinity.

8.35.1 Syntax

```
FCVTPS  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.35.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.36 FCVTPU (scalar)

Floating-point convert to unsigned integer, rounding toward positive infinity.

8.36.1 Syntax

```
FCVTPU  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.36.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.37 FCVTXN (scalar)

Floating-point convert to lower precision narrow, rounding to odd.

8.37.1 Syntax

```
FCVTXN  Vbd, Van
```

Where:

Vb Is the destination width specifier, `S`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Va Is the source width specifier, `D`.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.37.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.38 FCVTZS (scalar, fixed-point)

Floating-point convert to signed fixed-point, rounding toward zero.

8.38.1 Syntax

```
FCVTZS  Vd, Vn, #fbits
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

fbits Is the number of fractional bits, in the range 1 to the operand width.

8.38.2 Usage

The following table shows valid specifier combinations:

Table 8-4 FCVTZS (Scalar) specifier combinations

V	*fbits*
S	1 to 32
D	1 to 64

8.38.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.39 FCVTZS (scalar, integer)

Floating-point convert to signed integer, rounding toward zero.

8.39.1 Syntax

```
FCVTZS  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.39.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.40 FCVTZU (scalar, fixed-point)

Floating-point convert to unsigned fixed-point, rounding toward zero.

8.40.1 Syntax

```
FCVTZU  Vd, Vn, #fbits
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

fbits Is the number of fractional bits, in the range 1 to the operand width.

8.40.2 Usage

The following table shows valid specifier combinations:

Table 8-5 FCVTZU (Scalar) specifier combinations

V	*fbits*
S	1 to 32
D	1 to 64

8.40.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.41 FCVTZU (scalar, integer)

Floating-point convert to unsigned integer, rounding toward zero.

8.41.1 Syntax

```
FCVTZU  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.41.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.42 FMAXNMP (scalar)

Floating-point maximum number of pair of elements.

8.42.1 Syntax

```
FMAXNMP  Vd, Vn.T
```

Where:

V Is the destination width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is the source arrangement specifier, and can be either `2S` or `2D`.

8.42.2 Usage

The following table shows valid specifier combinations:

Table 8-6 FMAXNMP specifier combinations

V	*T*
S	2S
D	2D

8.42.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.43 FMAXP (scalar)

Floating-point maximum of pair of elements.

8.43.1 Syntax

```
FMAXP  Vd, Vn.T
```

Where:

V Is the destination width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is the source arrangement specifier, and can be either `2S` or `2D`.

8.43.2 Usage

The following table shows valid specifier combinations:

Table 8-7 FMAXP specifier combinations

V	*T*
S	2S
D	2D

8.43.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.44 FMINNMP (scalar)

Floating-point minimum number of pair of elements.

8.44.1 Syntax

```
FMINNMP  Vd, Vn.T
```

Where:

V Is the destination width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is the source arrangement specifier, and can be either `2S` or `2D`.

8.44.2 Usage

The following table shows valid specifier combinations:

Table 8-8 FMINNMP specifier combinations

V	*T*
S	2S
D	2D

8.44.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.45 FMINP (scalar)

Floating-point minimum of pair of elements.

8.45.1 Syntax

```
FMINP  Vd, Vn.T
```

Where:

V Is the destination width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is the source arrangement specifier, and can be either `2S` or `2D`.

8.45.2 Usage

The following table shows valid specifier combinations:

Table 8-9 FMINP specifier combinations

V	*T*
S	2S
D	2D

8.45.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.46 FMLA (scalar, by element)

Floating-point fused multiply-add to accumulator (by element).

8.46.1 Syntax

```
FMLA  Vd, Vn, Vm.Ts[index]
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the SIMD and FP source register in the range 0 to 31.

Ts Is the element width specifier, and can be either `S` or `D`.

index Is the element index, in the range shown in Usage.

8.46.2 Usage

The following table shows valid specifier combinations:

Table 8-10 FMLA (Scalar) specifier combinations

V	*Ts*	*index*
S	S	0 to 3
D	D	0 or 1

8.46.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.47 FMLS (scalar, by element)

Floating-point fused multiply-subtract from accumulator (by element).

8.47.1 Syntax

```
FMLS  Vd, Vn, Vm.Ts[index]
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the SIMD and FP source register in the range 0 to 31.

Ts Is the element width specifier, and can be either `S` or `D`.

index Is the element index, in the range shown in Usage.

8.47.2 Usage

The following table shows valid specifier combinations:

Table 8-11 FMLS (Scalar) specifier combinations

V	*Ts*	*index*
S	S	0 to 3
D	D	0 or 1

8.47.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.48 FMUL (scalar, by element)

Floating-point multiply (by element).

8.48.1 Syntax

```
FMUL  Vd, Vn, Vm.Ts[index]
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the SIMD and FP source register in the range 0 to 31.

Ts Is the element width specifier, and can be either `S` or `D`.

index Is the element index, in the range shown in Usage.

8.48.2 Usage

The following table shows valid specifier combinations:

Table 8-12 FMUL (Scalar) specifier combinations

V	*Ts*	*index*
S	S	0 to 3
D	D	0 or 1

8.48.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.49 FMULX (scalar, by element)

Floating-point multiply extended (by element).

8.49.1 Syntax

```
FMULX  Vd, Vn, Vm.Ts[index]
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the SIMD and FP source register in the range 0 to 31.

Ts Is the element width specifier, and can be either `S` or `D`.

index Is the element index, in the range shown in Usage.

8.49.2 Usage

The following table shows valid specifier combinations:

Table 8-13 FMULX (Scalar) specifier combinations

V	*Ts*	*index*
S	S	0 to 3
D	D	0 or 1

8.49.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.50 FMULX (scalar)

Floating-point multiply extended.

8.50.1 Syntax

```
FMULX  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.50.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.51 FRECPE (scalar)

Floating-point reciprocal estimate.

8.51.1 Syntax

```
FRECPE  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.51.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.52 FRECPS (scalar)

Floating-point reciprocal step.

8.52.1 Syntax

```
FRECPS  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.52.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.53 FRECPX (scalar)

Floating-point reciprocal exponent.

8.53.1 Syntax

```
FRECPX  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.53.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.54 FRSQRTE (scalar)

Floating-point reciprocal square root estimate.

8.54.1 Syntax

```
FRSQRTE  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.54.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.55 FRSQRTS (scalar)

Floating-point reciprocal square root step.

8.55.1 Syntax

```
FRSQRTS  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.55.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.56 MOV (scalar)

Move vector element to scalar.

This instruction is an alias of DUP (element).

8.56.1 Syntax

```
MOV  Vd, Vn.T[index]
```

Equivalent to `DUP Vd, Vn.T[index]`

Where:

V Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is the element width specifier, and can be one of the values shown in Usage.

index Is the element index, in the range shown in Usage.

8.56.2 Usage

The following table shows valid specifier combinations:

Table 8-14 MOV specifier combinations

V	*T*	*index*
B	B	0 to 15
H	H	0 to 7
S	S	0 to 3
D	D	0 or 1

8.56.3 See also

Reference

- *DUP (scalar, element)* on page 8-21.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.57 NEG (scalar)

Negate.

8.57.1 Syntax

`NEG` *Vd*, *Vn*

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.57.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.58 SCVTF (scalar, fixed-point)

Signed fixed-point convert to floating-point.

8.58.1 Syntax

```
SCVTF  Vd, Vn, #fbits
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

fbits Is the number of fractional bits, in the range 1 to the operand width.

8.58.2 Usage

The following table shows valid specifier combinations:

Table 8-15 SCVTF (Scalar) specifier combinations

V	*fbits*
S	1 to 32
D	1 to 64

8.58.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.59 SCVTF (scalar, integer)

Signed integer convert to floating-point.

8.59.1 Syntax

```
SCVTF  Vd, Vn
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.59.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.60 SHL (scalar)

Shift left (immediate).

8.60.1 Syntax

```
SHL  Vd, Vn, #shift
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the left shift amount, in the range 0 to 63.

8.60.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.61 SLI (scalar)

Shift left and insert (immediate).

8.61.1 Syntax

```
SLI  Vd, Vn, #shift
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the left shift amount, in the range 0 to 63.

8.61.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.62 SQABS (scalar)

Signed saturating absolute value.

8.62.1 Syntax

```
SQABS  Vd, Vn
```

Where:

V Is a width specifier, and can be one of `B`, `H`, `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.62.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.63 SQADD (scalar)

Signed saturating add.

8.63.1 Syntax

```
SQADD  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be one of `B`, `H`, `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.63.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.64 SQDMLAL (scalar, by element)

Signed saturating doubling multiply-add long (by element).

8.64.1 Syntax

```
SQDMLAL  Vad, Vbn, Vm.Ts[index]
```

Where:

Va Is the destination width specifier, and can be either S or D.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vb Is the source width specifier, and can be either H or S.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

Ts Is the element width specifier, and can be either H or S.

index Is the element index, in the range shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is H, then *Vm* must be in the range V0 to V15.
- If *Ts* is S, then *Vm* must be in the range V0 to V31.

8.64.2 Usage

The following table shows valid specifier combinations:

Table 8-16 SQDMLAL (Scalar) specifier combinations

Va	*Vb*	*Ts*	*index*
S	H	H	0 to 7
D	S	S	0 to 3

8.64.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.65 SQDMLAL (scalar)

Signed saturating doubling multiply-add long.

8.65.1 Syntax

```
SQDMLAL  Vad, Vbn, Vbm
```

Where:

Va Is the destination width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vb Is the source width specifier, and can be either `H` or `S`.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.65.2 Usage

The following table shows valid specifier combinations:

Table 8-17 SQDMLAL (Scalar) specifier combinations

Va	*Vb*
S	H
D	S

8.65.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.66 SQDMLSL (scalar, by element)

Signed saturating doubling multiply-subtract long (by element).

8.66.1 Syntax

```
SQDMLSL  Vad, Vbn, Vm.Ts[index]
```

Where:

Va Is the destination width specifier, and can be either S or D.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vb Is the source width specifier, and can be either H or S.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

Ts Is the element width specifier, and can be either H or S.

index Is the element index, in the range shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is H, then *Vm* must be in the range V0 to V15.
- If *Ts* is S, then *Vm* must be in the range V0 to V31.

8.66.2 Usage

The following table shows valid specifier combinations:

Table 8-18 SQDMLSL (Scalar) specifier combinations

Va	*Vb*	*Ts*	*index*
S	H	H	0 to 7
D	S	S	0 to 3

8.66.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.67 SQDMLSL (scalar)

Signed saturating doubling multiply-subtract long.

8.67.1 Syntax

```
SQDMLSL  Vad, Vbn, Vbm
```

Where:

Va Is the destination width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vb Is the source width specifier, and can be either `H` or `S`.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.67.2 Usage

The following table shows valid specifier combinations:

Table 8-19 SQDMLSL (Scalar) specifier combinations

Va	*Vb*
S	H
D	S

8.67.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.68 SQDMULH (scalar, by element)

Signed saturating doubling multiply returning high half (by element).

8.68.1 Syntax

```
SQDMULH  Vd, Vn, Vm.Ts[index]
```

Where:

V Is a width specifier, and can be either H or S.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

Ts Is the element width specifier, and can be either H or S.

index Is the element index, in the range shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is H, then *Vm* must be in the range V0 to V15.
- If *Ts* is S, then *Vm* must be in the range V0 to V31.

8.68.2 Usage

The following table shows valid specifier combinations:

Table 8-20 SQDMULH (Scalar) specifier combinations

V	*Ts*	*index*
H	H	0 to 7
S	S	0 to 3

8.68.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.69 SQDMULH (scalar)

Signed saturating doubling multiply returning high half.

8.69.1 Syntax

```
SQDMULH  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be either `H` or `S`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.69.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.70 SQDMULL (scalar, by element)

Signed saturating doubling multiply long (by element).

8.70.1 Syntax

```
SQDMULL  Vad, Vbn, Vm.Ts[index]
```

Where:

Va Is the destination width specifier, and can be either S or D.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vb Is the source width specifier, and can be either H or S.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

Ts Is the element width specifier, and can be either H or S.

index Is the element index, in the range shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is H, then *Vm* must be in the range V0 to V15.
- If *Ts* is S, then *Vm* must be in the range V0 to V31.

8.70.2 Usage

The following table shows valid specifier combinations:

Table 8-21 SQDMULL (Scalar) specifier combinations

Va	*Vb*	*Ts*	*index*
S	H	H	0 to 7
D	S	S	0 to 3

8.70.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.71 SQDMULL (scalar)

Signed saturating doubling multiply long.

8.71.1 Syntax

```
SQDMULL  Vad, Vbn, Vbm
```

Where:

Va Is the destination width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vb Is the source width specifier, and can be either `H` or `S`.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.71.2 Usage

The following table shows valid specifier combinations:

Table 8-22 SQDMULL (Scalar) specifier combinations

Va	*Vb*
S	H
D	S

8.71.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.72 SQNEG (scalar)

Signed saturating negate.

8.72.1 Syntax

```
SQNEG  Vd, Vn
```

Where:

V Is a width specifier, and can be one of `B`, `H`, `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.72.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.73 SQRDMULH (scalar, by element)

Signed saturating rounding doubling multiply returning high half (by element).

8.73.1 Syntax

```
SQRDMULH  Vd, Vn, Vm.Ts[index]
```

Where:

V Is a width specifier, and can be either H or S.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

Ts Is the element width specifier, and can be either H or S.

index Is the element index, in the range shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is H, then *Vm* must be in the range V0 to V15.
- If *Ts* is S, then *Vm* must be in the range V0 to V31.

8.73.2 Usage

The following table shows valid specifier combinations:

Table 8-23 SQRDMULH (Scalar) specifier combinations

V	*Ts*	*index*
H	H	0 to 7
S	S	0 to 3

8.73.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.74 SQRDMULH (scalar)

Signed saturating rounding doubling multiply returning high half.

8.74.1 Syntax

```
SQRDMULH  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be either `H` or `S`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.74.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.75 SQRSHL (scalar)

Signed saturating rounding shift left (register).

8.75.1 Syntax

```
SQRSHL  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be one of `B`, `H`, `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.75.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.76 SQRSHRN (scalar)

Signed saturating rounded shift right narrow (immediate).

8.76.1 Syntax

```
SQRSHRN  Vbd, Van, #shift
```

Where:

Vb Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Va Is the source width specifier, and can be one of the values shown in Usage.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the destination operand width in bits, and can be one of the values shown in Usage.

8.76.2 Usage

The following table shows valid specifier combinations:

Table 8-24 SQRSHRN (Scalar) specifier combinations

Vb	*Va*	*shift*
B	H	1 to 8
H	S	1 to 16
S	D	1 to 32

8.76.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.77 SQRSHRUN (scalar)

Signed saturating rounded shift right unsigned narrow (immediate).

8.77.1 Syntax

```
SQRSHRUN  Vbd, Van, #shift
```

Where:

Vb Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Va Is the source width specifier, and can be one of the values shown in Usage.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the destination operand width in bits, and can be one of the values shown in Usage.

8.77.2 Usage

The following table shows valid specifier combinations:

Table 8-25 SQRSHRUN (Scalar) specifier combinations

Vb	*Va*	*shift*
B	H	1 to 8
H	S	1 to 16
S	D	1 to 32

8.77.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.78 SQSHL (scalar, immediate)

Signed saturating shift left (immediate).

8.78.1 Syntax

```
SQSHL  Vd, Vn, #shift
```

Where:

V Is a width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the left shift amount, in the range 0 to the operand width in bits minus 1, and can be one of the values shown in Usage.

8.78.2 Usage

The following table shows valid specifier combinations:

Table 8-26 SQSHL (Scalar) specifier combinations

V	*shift*
B	0 to 7
H	0 to 15
S	0 to 31
D	0 to 63

8.78.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.79 SQSHL (scalar, register)

Signed saturating shift left (register).

8.79.1 Syntax

`SQSHL` *`Vd`*`,` *`Vn`*`,` *`Vm`*

Where:

`V` Is a width specifier, and can be one of `B`, `H`, `S` or `D`.

`d` Is the number of the SIMD and FP destination register, in the range 0 to 31.

`n` Is the number of the first SIMD and FP source register, in the range 0 to 31.

`m` Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.79.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.80 SQSHLU (scalar)

Signed saturating shift left unsigned (immediate).

8.80.1 Syntax

```
SQSHLU  Vd, Vn, #shift
```

Where:

V Is a width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the left shift amount, in the range 0 to the operand width in bits minus 1, and can be one of the values shown in Usage.

8.80.2 Usage

The following table shows valid specifier combinations:

Table 8-27 SQSHLU (Scalar) specifier combinations

V	*shift*
B	0 to 7
H	0 to 15
S	0 to 31
D	0 to 63

8.80.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.81 SQSHRN (scalar)

Signed saturating shift right narrow (immediate).

8.81.1 Syntax

```
SQSHRN  Vbd, Van, #shift
```

Where:

Vb Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Va Is the source width specifier, and can be one of the values shown in Usage.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the destination operand width in bits, and can be one of the values shown in Usage.

8.81.2 Usage

The following table shows valid specifier combinations:

Table 8-28 SQSHRN (Scalar) specifier combinations

Vb	*Va*	*shift*
B	H	1 to 8
H	S	1 to 16
S	D	1 to 32

8.81.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.82 SQSHRUN (scalar)

Signed saturating shift right unsigned narrow (immediate).

8.82.1 Syntax

```
SQSHRUN  Vbd, Van, #shift
```

Where:

Vb Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Va Is the source width specifier, and can be one of the values shown in Usage.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the destination operand width in bits, and can be one of the values shown in Usage.

8.82.2 Usage

The following table shows valid specifier combinations:

Table 8-29 SQSHRUN (Scalar) specifier combinations

Vb	*Va*	*shift*
B	H	1 to 8
H	S	1 to 16
S	D	1 to 32

8.82.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.83 SQSUB (scalar)

Signed saturating subtract.

8.83.1 Syntax

```
SQSUB  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be one of `B`, `H`, `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.83.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.84 SQXTN (scalar)

Signed saturating extract narrow.

8.84.1 Syntax

`SQXTN` *Vbd*, *Van*

Where:

Vb Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Va Is the source width specifier, and can be one of the values shown in Usage.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.84.2 Usage

The following table shows valid specifier combinations:

Table 8-30 SQXTN (Scalar) specifier combinations

Vb	*Va*
B	H
H	S
S	D

8.84.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.85 SQXTUN (scalar)

Signed saturating extract unsigned narrow.

8.85.1 Syntax

```
SQXTUN  Vbd, Van
```

Where:

Vb Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Va Is the source width specifier, and can be one of the values shown in Usage.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.85.2 Usage

The following table shows valid specifier combinations:

Table 8-31 SQXTUN (Scalar) specifier combinations

Vb	*Va*
B	H
H	S
S	D

8.85.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.86 SRI (scalar)

Shift right and insert (immediate).

8.86.1 Syntax

```
SRI  Vd, Vn, #shift
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to 64.

8.86.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.87 SRSHL (scalar)

Signed rounding shift left (register).

8.87.1 Syntax

```
SRSHL  Vd, Vn, Vm
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.87.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.88 SRSHR (scalar)

Signed rounding shift right (immediate).

8.88.1 Syntax

```
SRSHR  Vd, Vn, #shift
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to 64.

8.88.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.89 SRSRA (scalar)

Signed rounding shift right and accumulate (immediate).

8.89.1 Syntax

```
SRSRA  Vd, Vn, #shift
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to 64.

8.89.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.90 SSHL (scalar)

Signed shift left (register).

8.90.1 Syntax

```
SSHL  Vd, Vn, Vm
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.90.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.91 SSHR (scalar)

Signed shift right (immediate).

8.91.1 Syntax

```
SSHR  Vd, Vn, #shift
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to 64.

8.91.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.92 SSRA (scalar)

Signed shift right and accumulate (immediate).

8.92.1 Syntax

```
SSRA  Vd, Vn, #shift
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to 64.

8.92.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.93 SUB (scalar)

Subtract.

8.93.1 Syntax

```
SUB  Vd, Vn, Vm
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.93.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.94 SUQADD (scalar)

Signed saturating accumulate of unsigned value.

8.94.1 Syntax

```
SUQADD  Vd, Vn
```

Where:

V Is a width specifier, and can be one of `B`, `H`, `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.94.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.95 UCVTF (scalar, fixed-point)

Unsigned fixed-point convert to floating-point.

8.95.1 Syntax

```
UCVTF  Vd, Vn, #fbits
```

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

fbits Is the number of fractional bits, in the range 1 to the operand width.

8.95.2 Usage

The following table shows valid specifier combinations:

Table 8-32 UCVTF (Scalar) specifier combinations

V	*fbits*
S	1 to 32
D	1 to 64

8.95.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.96 UCVTF (scalar, integer)

Unsigned integer convert to floating-point.

8.96.1 Syntax

`UCVTF` *Vd*, *Vn*

Where:

V Is a width specifier, and can be either `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.96.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.97 UQADD (scalar)

Unsigned saturating add.

8.97.1 Syntax

```
UQADD  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be one of `B`, `H`, `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.97.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.98 UQRSHL (scalar)

Unsigned saturating rounding shift left (register).

8.98.1 Syntax

`UQRSHL` *Vd*, *Vn*, *Vm*

Where:

V Is a width specifier, and can be one of `B`, `H`, `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.98.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.99 UQRSHRN (scalar)

Unsigned saturating rounded shift right narrow (immediate).

8.99.1 Syntax

```
UQRSHRN  Vbd, Van, #shift
```

Where:

Vb Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Va Is the source width specifier, and can be one of the values shown in Usage.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the destination operand width in bits, and can be one of the values shown in Usage.

8.99.2 Usage

The following table shows valid specifier combinations:

Table 8-33 UQRSHRN (Scalar) specifier combinations

Vb	*Va*	*shift*
B	H	1 to 8
H	S	1 to 16
S	D	1 to 32

8.99.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.100 UQSHL (scalar, immediate)

Unsigned saturating shift left (immediate).

8.100.1 Syntax

```
UQSHL  Vd, Vn, #shift
```

Where:

V Is a width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the left shift amount, in the range 0 to the operand width in bits minus 1, and can be one of the values shown in Usage.

8.100.2 Usage

The following table shows valid specifier combinations:

Table 8-34 UQSHL (Scalar) specifier combinations

V	*shift*
B	0 to 7
H	0 to 15
S	0 to 31
D	0 to 63

8.100.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.101 UQSHL (scalar, register)

Unsigned saturating shift left (register).

8.101.1 Syntax

```
UQSHL  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be one of `B`, `H`, `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.101.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.102 UQSHRN (scalar)

Unsigned saturating shift right narrow (immediate).

8.102.1 Syntax

```
UQSHRN  Vbd, Van, #shift
```

Where:

Vb Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Va Is the source width specifier, and can be one of the values shown in Usage.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the destination operand width in bits, and can be one of the values shown in Usage.

8.102.2 Usage

The following table shows valid specifier combinations:

Table 8-35 UQSHRN (Scalar) specifier combinations

Vb	*Va*	*shift*
B	H	1 to 8
H	S	1 to 16
S	D	1 to 32

8.102.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.103 UQSUB (scalar)

Unsigned saturating subtract.

8.103.1 Syntax

```
UQSUB  Vd, Vn, Vm
```

Where:

V Is a width specifier, and can be one of `B`, `H`, `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.103.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.104 UQXTN (scalar)

Unsigned saturating extract narrow.

8.104.1 Syntax

`UQXTN` *Vbd*, *Van*

Where:

Vb Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Va Is the source width specifier, and can be one of the values shown in Usage.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.104.2 Usage

The following table shows valid specifier combinations:

Table 8-36 UQXTN (Scalar) specifier combinations

Vb	*Va*
B	H
H	S
S	D

8.104.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.105 URSHL (scalar)

Unsigned rounding shift left (register).

8.105.1 Syntax

```
URSHL  Vd, Vn, Vm
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.105.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.106 URSHR (scalar)

Unsigned rounding shift right (immediate).

8.106.1 Syntax

```
URSHR  Vd, Vn, #shift
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to 64.

8.106.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.107 URSRA (scalar)

Unsigned rounding shift right and accumulate (immediate).

8.107.1 Syntax

```
URSRA  Vd, Vn, #shift
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to 64.

8.107.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.108 USHL (scalar)

Unsigned shift left (register).

8.108.1 Syntax

```
USHL  Vd, Vn, Vm
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

m Is the number of the second SIMD and FP source register, in the range 0 to 31.

8.108.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.109 USHR (scalar)

Unsigned shift right (immediate).

8.109.1 Syntax

```
USHR  Vd, Vn, #shift
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to 64.

8.109.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.110 USQADD (scalar)

Unsigned saturating accumulate of signed value.

8.110.1 Syntax

```
USQADD  Vd, Vn
```

Where:

V Is a width specifier, and can be one of `B`, `H`, `S` or `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the SIMD and FP source register, in the range 0 to 31.

8.110.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

8.111 USRA (scalar)

Unsigned shift right and accumulate (immediate).

8.111.1 Syntax

```
USRA  Vd, Vn, #shift
```

Where:

V Is a width specifier, `D`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

n Is the number of the first SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to 64.

8.111.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Chapter 9
A64 SIMD Vector Instructions

The following topic gives a summary of the A64 SIMD vector instructions supported by the ARM assembler:

- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.1 A64 SIMD vector instructions in alphabetical order

The following A64 SIMD vector instructions are supported:

Table 9-1 Location of SIMD Vector instructions

Mnemonic	Brief description	See
ABS (vector)	Absolute value	page 9-12
ADD (vector)	Add	page 9-13
ADDHN, ADDHN2 (vector)	Add returning high narrow	page 9-14
ADDP (vector)	Add pairwise	page 9-15
ADDV (vector)	Add across vector	page 9-16
AND (vector)	Bitwise AND	page 9-17
BIC (vector, immediate)	Bitwise bit clear (immediate)	page 9-18
BIC (vector, register)	Bitwise bit clear (register)	page 9-19
BIF (vector)	Bitwise insert if false	page 9-20
BIT (vector)	Bitwise insert if true	page 9-21
BSL (vector)	Bitwise select	page 9-22
CLS (vector)	Count leading sign bits	page 9-23
CLZ (vector)	Count leading zero bits	page 9-24
CMEQ (vector, register)	Compare bitwise equal, setting destination vector element to all ones if the condition holds, else zero	page 9-25
CMEQ (vector, zero)	Compare bitwise equal to zero, setting destination vector element to all ones if the condition holds, else zero	page 9-26
CMGE (vector, register)	Compare signed greater than or equal	page 9-27
CMGE (vector, zero)	Compare signed greater than or equal to zero, setting destination vector element to all ones if the condition holds, else zero	page 9-28
CMGT (vector, register)	Compare signed greater than, setting destination vector element to all ones if the condition holds, else zero	page 9-29
CMGT (vector, zero)	Compare signed greater than zero, setting destination vector element to all ones if the condition holds, else zero	page 9-30
CMHI (vector, register)	Compare unsigned higher, setting destination vector element to all ones if the condition holds, else zero	page 9-31
CMHS (vector, register)	Compare unsigned higher or same, setting destination vector element to all ones if the condition holds, else zero	page 9-32
CMLE (vector, zero)	Compare signed less than or equal to zero, setting destination vector element to all ones if the condition holds, else zero	page 9-33
CMLT (vector, zero)	Compare signed less than zero, setting destination vector element to all ones if the condition holds, else zero	page 9-34

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 9-1 Location of SIMD Vector instructions (continued)

Mnemonic	Brief description	See
CMTST (vector)	Compare bitwise test bits nonzero, setting destination vector element to all ones if the condition holds, else zero	page 9-35
CNT (vector)	Population count per byte	page 9-36
DUP (vector, element)	Duplicate vector element to vector	page 9-37
DUP (vector, general)	Duplicate general-purpose register to vector	page 9-38
EOR (vector)	Bitwise exclusive OR	page 9-39
EXT (vector)	Extract vector from pair of vectors	page 9-40
FABD (vector)	Floating-point absolute difference	page 9-41
FABS (vector)	Floating-point absolute value	page 9-42
FACGE (vector)	Floating-point absolute compare greater than or equal	page 9-43
FACGT (vector)	Floating-point absolute compare greater than	page 9-44
FADD (vector)	Floating-point add	page 9-45
FADDP (vector)	Floating-point add pairwise	page 9-46
FCMEQ (vector, register)	Floating-point compare equal, setting destination vector element to all ones if the condition holds, else zero	page 9-47
FCMEQ (vector, zero)	Floating-point compare equal to zero, setting destination vector element to all ones if the condition holds, else zero	page 9-48
FCMGE (vector, register)	Floating-point compare greater than or equal, setting destination vector element to all ones if the condition holds, else zero	page 9-49
FCMGE (vector, zero)	Floating-point compare greater than or equal to zero, setting destination vector element to all ones if the condition holds, else zero	page 9-50
FCMGT (vector, register)	Floating-point compare greater than, setting destination vector element to all ones if the condition holds, else zero	page 9-51
FCMGT (vector, zero)	Floating-point compare greater than zero, setting destination vector element to all ones if the condition holds, else zero	page 9-52
FCMLE (vector, zero)	Floating-point compare less than or equal to zero, setting destination vector element to all ones if the condition holds, else zero	page 9-53
FCMLT (vector, zero)	Floating-point compare less than zero, setting destination vector element to all ones if the condition holds, else zero	page 9-54
FCVTAS (vector)	Floating-point convert to signed integer, rounding to nearest with ties to away	page 9-55
FCVTAU (vector)	Floating-point convert to unsigned integer, rounding to nearest with ties to away	page 9-56
FCVTL, FCVTL2 (vector)	Floating-point convert to higher precision long	page 9-57

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 9-1 Location of SIMD Vector instructions (continued)

Mnemonic	Brief description	See
FCVTMS (vector)	Floating-point convert to signed integer, rounding toward minus infinity	page 9-58
FCVTMU (vector)	Floating-point convert to unsigned integer, rounding toward minus infinity	page 9-59
FCVTN, FCVTN2 (vector)	Floating-point convert to lower precision narrow	page 9-60
FCVTNS (vector)	Floating-point convert to signed integer, rounding to nearest with ties to even	page 9-61
FCVTNU (vector)	Floating-point convert to unsigned integer, rounding to nearest with ties to even	page 9-62
FCVTPS (vector)	Floating-point convert to signed integer, rounding toward positive infinity	page 9-63
FCVTPU (vector)	Floating-point convert to unsigned integer, rounding toward positive infinity	page 9-64
FCVTXN, FCVTXN2 (vector)	Floating-point convert to lower precision narrow, rounding to odd	page 9-65
FCVTZS (vector, fixed-point)	Floating-point convert to signed fixed-point, rounding toward zero	page 9-66
FCVTZS (vector, integer)	Floating-point convert to signed integer, rounding toward zero	page 9-67
FCVTZU (vector, fixed-point)	Floating-point convert to unsigned fixed-point, rounding toward zero	page 9-68
FCVTZU (vector, integer)	Floating-point convert to unsigned integer, rounding toward zero	page 9-69
FDIV (vector)	Floating-point divide	page 9-70
FMAX (vector)	Floating-point maximum	page 9-71
FMAXNM (vector)	Floating-point maximum number	page 9-72
FMAXNMP (vector)	Floating-point maximum number pairwise	page 9-73
FMAXNMV (vector)	Floating-point maximum number across vector	page 9-74
FMAXP (vector)	Floating-point maximum pairwise	page 9-75
FMAXV (vector)	Floating-point maximum across vector	page 9-76
FMIN (vector)	Floating-point minimum	page 9-77
FMINNM (vector)	Floating-point minimum number	page 9-78
FMINNMP (vector)	Floating-point minimum number pairwise	page 9-79
FMINNMV (vector)	Floating-point minimum number across vector	page 9-80
FMINP (vector)	Floating-point minimum pairwise	page 9-81
FMINV (vector)	Floating-point minimum across vector	page 9-82
FMLA (vector, by element)	Floating-point fused multiply-add to accumulator (by element)	page 9-83

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 9-1 Location of SIMD Vector instructions (continued)

Mnemonic	Brief description	See
FMLA (vector)	Floating-point fused multiply-add to accumulator	page 9-84
FMLS (vector, by element)	Floating-point fused multiply-subtract from accumulator (by element)	page 9-85
FMLS (vector)	Floating-point fused multiply-subtract from accumulator	page 9-86
FMOV (vector, immediate)	Floating-point move immediate	page 9-87
FMUL (vector, by element)	Floating-point multiply (by element)	page 9-88
FMUL (vector)	Floating-point multiply	page 9-89
FMULX (vector, by element)	Floating-point multiply extended (by element)	page 9-90
FMULX (vector)	Floating-point multiply extended	page 9-91
FNEG (vector)	Floating-point negate	page 9-92
FRECPE (vector)	Floating-point reciprocal estimate	page 9-93
FRECPS (vector)	Floating-point reciprocal step	page 9-94
FRINTA (vector)	Floating-point round to integral, to nearest with ties to away	page 9-95
FRINTI (vector)	Floating-point round to integral, using current rounding mode	page 9-96
FRINTM (vector)	Floating-point round to integral, toward minus infinity	page 9-97
FRINTN (vector)	Floating-point round to integral, to nearest with ties to even	page 9-98
FRINTP (vector)	Floating-point round to integral, toward positive infinity	page 9-99
FRINTX (vector)	Floating-point round to integral exact, using current rounding mode	page 9-100
FRINTZ (vector)	Floating-point round to integral, toward zero	page 9-101
FRSQRTE (vector)	Floating-point reciprocal square root estimate	page 9-102
FRSQRTS (vector)	Floating-point reciprocal square root step	page 9-103
FSQRT (vector)	Floating-point square root	page 9-104
FSUB (vector)	Floating-point subtract	page 9-105
INS (vector, element)	Insert vector element from another vector element	page 9-106
INS (vector, general)	Insert vector element from general-purpose register	page 9-107
LD1 (vector, multiple structures)	Load multiple 1-element structures to one, two, three or four registers	page 9-108
LD1 (vector, single structure)	Load single 1-element structure to one lane of one register	page 9-111
LD1R (vector)	Load single 1-element structure and replicate to all lanes (of one register)	page 9-112
LD2 (vector, multiple structures)	Load multiple 2-element structures to two registers	page 9-113
LD2 (vector, single structure)	Load single 2-element structure to one lane of two registers	page 9-114

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 9-1 Location of SIMD Vector instructions (continued)

Mnemonic	Brief description	See
LD2R (vector)	Load single 2-element structure and replicate to all lanes of two registers	page 9-115
LD3 (vector, multiple structures)	Load multiple 3-element structures to three registers	page 9-117
LD3 (vector, single structure)	Load single 3-element structure to one lane of three registers)	page 9-118
LD3R (vector)	Load single 3-element structure and replicate to all lanes of three registers	page 9-120
LD4 (vector, multiple structures)	Load multiple 4-element structures to four registers	page 9-122
LD4 (vector, single structure)	Load single 4-element structure to one lane of four registers	page 9-123
LD4R (vector)	Load single 4-element structure and replicate to all lanes of four registers	page 9-125
MLA (vector, by element)	Multiply-add to accumulator (by element)	page 9-127
MLA (vector)	Multiply-add to accumulator	page 9-128
MLS (vector, by element)	Multiply-subtract from accumulator (by element)	page 9-129
MLS (vector)	Multiply-subtract from accumulator	page 9-130
MOV (vector, element)	Move vector element to another vector element	page 9-131
MOV (vector, from general)	Move general-purpose register to a vector element	page 9-132
MOV (vector)	Move vector	page 9-133
MOV (vector, to general)	Move vector element to general-purpose register	page 9-134
MOVI (vector)	Move immediate	page 9-135
MUL (vector, by element)	Multiply (by element)	page 9-136
MUL (vector)	Multiply	page 9-137
MVN (vector)	Bitwise NOT	page 9-138
MVNI (vector)	Move inverted immediate	page 9-139
NEG (vector)	Negate	page 9-140
NOT (vector)	Bitwise NOT	page 9-141
ORN (vector)	Bitwise inclusive OR NOT	page 9-142
ORR (vector, immediate)	Bitwise inclusive OR (immediate)	page 9-143
ORR (vector, register)	Bitwise inclusive OR (register)	page 9-144
PMUL (vector)	Polynomial multiply	page 9-145
PMULL, PMULL2 (vector)	Polynomial multiply long	page 9-146
RADDHN, RADDHN2 (vector)	Rounding add returning high narrow	page 9-147
RBIT (vector)	Reverse bit order	page 9-148
REV16 (vector)	Reverse elements in 16-bit halfwords	page 9-149

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 9-1 Location of SIMD Vector instructions (continued)

Mnemonic	Brief description	See
REV32 (vector)	Reverse elements in 32-bit words	page 9-150
REV64 (vector)	Reverse elements in 64-bit doublewords	page 9-151
RSHRN, RSHRN2 (vector)	Rounding shift right narrow (immediate)	page 9-152
RSUBHN, RSUBHN2 (vector)	Rounding subtract returning high narrow	page 9-153
SABA (vector)	Signed absolute difference and accumulate	page 9-154
SABAL, SABAL2 (vector)	Signed absolute difference and accumulate long	page 9-155
SABD (vector)	Signed absolute difference	page 9-156
SABDL, SABDL2 (vector)	Signed absolute difference long	page 9-157
SADALP (vector)	Signed add and accumulate long pairwise	page 9-158
SADDL, SADDL2 (vector)	Signed add long	page 9-159
SADDLP (vector)	Signed add long pairwise	page 9-160
SADDLV (vector)	Signed add long across vector	page 9-161
SADDW, SADDW2 (vector)	Signed add wide	page 9-162
SCVTF (vector, fixed-point)	Signed fixed-point convert to floating-point	page 9-163
SCVTF (vector, integer)	Signed integer convert to floating-point	page 9-164
SHADD (vector)	Signed halving add	page 9-165
SHL (vector)	Shift left (immediate)	page 9-166
SHLL, SHLL2 (vector)	Shift left long (by element size)	page 9-167
SHRN, SHRN2 (vector)	Shift right narrow (immediate)	page 9-168
SHSUB (vector)	Signed halving subtract	page 9-169
SLI (vector)	Shift left and insert (immediate)	page 9-170
SMAX (vector)	Signed maximum	page 9-171
SMAXP (vector)	Signed maximum pairwise	page 9-172
SMAXV (vector)	Signed maximum across vector	page 9-173
SMIN (vector)	Signed minimum	page 9-174
SMINP (vector)	Signed minimum pairwise	page 9-175
SMINV (vector)	Signed minimum across vector	page 9-176
SMLAL, SMLAL2 (vector, by element)	Signed multiply-add long (by element)	page 9-177
SMLAL, SMLAL2 (vector)	Signed multiply-add long	page 9-178
SMLSL, SMLSL2 (vector, by element)	Signed multiply-subtract long (by element)	page 9-179
SMLSL, SMLSL2 (vector)	Signed multiply-subtract long	page 9-180

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 9-1 Location of SIMD Vector instructions (continued)

Mnemonic	Brief description	See
SMOV (vector)	Signed move vector element to general-purpose register	page 9-181
SMULL, SMULL2 (vector, by element)	Signed multiply long (by element)	page 9-182
SMULL, SMULL2 (vector)	Signed multiply long	page 9-183
SQABS (vector)	Signed saturating absolute value	page 9-184
SQADD (vector)	Signed saturating add	page 9-185
SQDMLAL, SQDMLAL2 (vector, by element)	Signed saturating doubling multiply-add long (by element)	page 9-186
SQDMLAL, SQDMLAL2 (vector)	Signed saturating doubling multiply-add long	page 9-187
SQDMLSL, SQDMLSL2 (vector, by element)	Signed saturating doubling multiply-subtract long (by element)	page 9-188
SQDMLSL, SQDMLSL2 (vector)	Signed saturating doubling multiply-subtract long	page 9-189
SQDMULH (vector, by element)	Signed saturating doubling multiply returning high half (by element)	page 9-190
SQDMULH (vector)	Signed saturating doubling multiply returning high half	page 9-191
SQDMULL, SQDMULL2 (vector, by element)	Signed saturating doubling multiply long (by element)	page 9-192
SQDMULL, SQDMULL2 (vector)	Signed saturating doubling multiply long	page 9-193
SQNEG (vector)	Signed saturating negate	page 9-194
SQRDMULH (vector, by element)	Signed saturating rounding doubling multiply returning high half (by element)	page 9-195
SQRDMULH (vector)	Signed saturating rounding doubling multiply returning high half	page 9-196
SQRSHL (vector)	Signed saturating rounding shift left (register)	page 9-197
SQRSHRN, SQRSHRN2 (vector)	Signed saturating rounded shift right narrow (immediate)	page 9-198
SQRSHRUN, SQRSHRUN2 (vector)	Signed saturating rounded shift right unsigned narrow (immediate)	page 9-199
SQSHL (vector, immediate)	Signed saturating shift left (immediate)	page 9-200
SQSHL (vector, register)	Signed saturating shift left (register)	page 9-201
SQSHLU (vector)	Signed saturating shift left unsigned (immediate)	page 9-202
SQSHRN, SQSHRN2 (vector)	Signed saturating shift right narrow (immediate)	page 9-203
SQSHRUN, SQSHRUN2 (vector)	Signed saturating shift right unsigned narrow (immediate)	page 9-204
SQSUB (vector)	Signed saturating subtract	page 9-205
SQXTN, SQXTN2 (vector)	Signed saturating extract narrow	page 9-206
SQXTUN, SQXTUN2 (vector)	Signed saturating extract unsigned narrow	page 9-207
SRHADD (vector)	Signed rounding halving add	page 9-208

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 9-1 Location of SIMD Vector instructions (continued)

Mnemonic	Brief description	See
SRI (vector)	Shift right and insert (immediate)	page 9-209
SRSHL (vector)	Signed rounding shift left (register)	page 9-210
SRSHR (vector)	Signed rounding shift right (immediate)	page 9-211
SRSRA (vector)	Signed rounding shift right and accumulate (immediate)	page 9-212
SSHL (vector)	Signed shift left (register)	page 9-213
SSHLL, SSHLL2 (vector)	Signed shift left long (immediate)	page 9-214
SSHR (vector)	Signed shift right (immediate)	page 9-215
SSRA (vector)	Signed shift right and accumulate (immediate)	page 9-216
SSUBL, SSUBL2 (vector)	Signed subtract long	page 9-217
SSUBW, SSUBW2 (vector)	Signed subtract wide	page 9-218
ST1 (vector, multiple structures)	Store multiple 1-element structures from one, two three or four registers	page 9-219
ST1 (vector, single structure)	Store single 1-element structure from one lane of one register	page 9-222
ST2 (vector, multiple structures)	Store multiple 2-element structures from two registers	page 9-223
ST2 (vector, single structure)	Store single 2-element structure from one lane of two registers	page 9-224
ST3 (vector, multiple structures)	Store multiple 3-element structures from three registers	page 9-225
ST3 (vector, single structure)	Store single 3-element structure from one lane of three registers	page 9-226
ST4 (vector, multiple structures)	Store multiple 4-element structures from four registers	page 9-227
ST4 (vector, single structure)	Store single 4-element structure from one lane of four registers	page 9-228
SUB (vector)	Subtract	page 9-230
SUBHN, SUBHN2 (vector)	Subtract returning high narrow	page 9-231
SUQADD (vector)	Signed saturating accumulate of unsigned value	page 9-232
SXTL, SXTL2 (vector)	Signed extend long	page 9-233
TBL (vector)	Table vector lookup	page 9-234
TBX (vector)	Table vector lookup extension	page 9-235
TRN1 (vector)	Transpose vectors (primary)	page 9-236
TRN2 (vector)	Transpose vectors (secondary)	page 9-237
UABA (vector)	Unsigned absolute difference and accumulate	page 9-238
UABAL, UABAL2 (vector)	Unsigned absolute difference and accumulate long	page 9-239
UABD (vector)	Unsigned absolute difference	page 9-240
UABDL, UABDL2 (vector)	Unsigned absolute difference long	page 9-241

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 9-1 Location of SIMD Vector instructions (continued)

Mnemonic	Brief description	See
UADALP (vector)	Unsigned add and accumulate long pairwise	page 9-242
UADDL, UADDL2 (vector)	Unsigned add long	page 9-243
UADDLP (vector)	Unsigned add long pairwise	page 9-244
UADDLV (vector)	Unsigned sum long across vector	page 9-245
UADDW, UADDW2 (vector)	Unsigned add wide	page 9-246
UCVTF (vector, fixed-point)	Unsigned fixed-point convert to floating-point	page 9-247
UCVTF (vector, integer)	Unsigned integer convert to floating-point	page 9-248
UHADD (vector)	Unsigned halving add	page 9-249
UHSUB (vector)	Unsigned halving subtract	page 9-250
UMAX (vector)	Unsigned maximum	page 9-251
UMAXP (vector)	Unsigned maximum pairwise	page 9-252
UMAXV (vector)	Unsigned maximum across vector	page 9-253
UMIN (vector)	Unsigned minimum	page 9-254
UMINP (vector)	Unsigned minimum pairwise	page 9-255
UMINV (vector)	Unsigned minimum across vector	page 9-256
UMLAL, UMLAL2 (vector, by element)	Unsigned multiply-add long (by element)	page 9-257
UMLAL, UMLAL2 (vector)	Unsigned multiply-add long	page 9-258
UMLSL, UMLSL2 (vector, by element)	Unsigned multiply-subtract long (by element)	page 9-259
UMLSL, UMLSL2 (vector)	Unsigned multiply-subtract long	page 9-260
UMOV (vector)	Unsigned move vector element to general-purpose register	page 9-261
UMULL, UMULL2 (vector, by element)	Unsigned multiply long (by element)	page 9-262
UMULL, UMULL2 (vector)	Unsigned multiply long	page 9-263
UQADD (vector)	Unsigned saturating add	page 9-264
UQRSHL (vector)	Unsigned saturating rounding shift left (register)	page 9-265
UQRSHRN, UQRSHRN2 (vector)	Unsigned saturating rounded shift right narrow (immediate)	page 9-266
UQSHL (vector, immediate)	Unsigned saturating shift left (immediate)	page 9-267
UQSHL (vector, register)	Unsigned saturating shift left (register)	page 9-268
UQSHRN, UQSHRN2 (vector)	Unsigned saturating shift right narrow (immediate)	page 9-269
UQSUB (vector)	Unsigned saturating subtract	page 9-270
UQXTN, UQXTN2 (vector)	Unsigned saturating extract narrow	page 9-271

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 9-1 Location of SIMD Vector instructions (continued)

Mnemonic	Brief description	See
URECPE (vector)	Unsigned reciprocal estimate	page 9-272
URHADD (vector)	Unsigned rounding halving add	page 9-273
URSHL (vector)	Unsigned rounding shift left (register)	page 9-274
URSHR (vector)	Unsigned rounding shift right (immediate)	page 9-275
URSQRTE (vector)	Unsigned reciprocal square root estimate	page 9-276
URSRA (vector)	Unsigned rounding shift right and accumulate (immediate)	page 9-277
USHL (vector)	Unsigned shift left (register)	page 9-278
USHLL, USHLL2 (vector)	Unsigned shift left long (immediate)	page 9-279
USHR (vector)	Unsigned shift right (immediate)	page 9-280
USQADD (vector)	Unsigned saturating accumulate of signed value	page 9-281
USRA (vector)	Unsigned shift right and accumulate (immediate)	page 9-282
USUBL, USUBL2 (vector)	Unsigned subtract long	page 9-283
USUBW, USUBW2 (vector)	Unsigned subtract wide	page 9-284
UXTL, UXTL2 (vector)	Unsigned extend long	page 9-285
UZP1 (vector)	Unzip vectors (primary)	page 9-286
UZP2 (vector)	Unzip vectors (secondary)	page 9-287
XTN, XTN2 (vector)	Extract narrow	page 9-288
ZIP1 (vector)	Zip vectors (primary)	page 9-289
ZIP2 (vector)	Zip vectors (secondary)	page 9-290

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.2 ABS (vector)

Absolute value.

9.2.1 Syntax

```
ABS  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.2.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.3 ADD (vector)

Add.

9.3.1 Syntax

```
ADD  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.3.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.4 ADDHN, ADDHN2 (vector)

Add returning high narrow.

9.4.1 Syntax

```
ADDHN{2}  Vd.Tb, Vn.Ta, Vm.Ta
```

Where:

2
: Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd
: Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb
: Is an arrangement specifier, and can be one of the values shown in Usage.

Vn
: Is the name of the first SIMD and FP source register, in the range 0 to 31.

Ta
: Is an arrangement specifier, and can be one of the values shown in Usage.

Vm
: Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.4.2 Usage

The following table shows valid specifier combinations:

Table 9-2 ADDHN, ADDHN2 specifier combinations

Q	*Tb*	*Ta*
-	8B	8H
2	16B	8H
-	4H	4S
2	8H	4S
-	2S	2D
2	4S	2D

9.4.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.5 ADDP (vector)

Add pairwise.

9.5.1 Syntax

```
ADDP  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.5.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.6 ADDV (vector)

Add across vector.

9.6.1 Syntax

```
ADDV  Vd, Vn.T
```

Where:

V Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

9.6.2 Usage

The following table shows valid specifier combinations:

Table 9-3 ADDV specifier combinations

V	*T*
B	8B
B	16B
H	4H
H	8H
S	4S

9.6.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.7 AND (vector)

Bitwise AND.

9.7.1 Syntax

```
AND  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `8B` or `16B`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.7.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.8 BIC (vector, immediate)

Bitwise bit clear (immediate).

9.8.1 Syntax

```
BIC  Vd.T, #imm8{, LSL #amount}    ; 16-bit
BIC  Vd.T, #imm8{, LSL #amount}    ; 32-bit
```

Where:

T
Is an arrangement specifier:

16-bit Can be one of `4H` or `8H`.

32-bit Can be one of `2S` or `4S`.

amount
Is the shift amount:

16-bit Can be one of `0` or `8`.

32-bit Can be one of `0`, `8`, `16` or `24`.

Defaults to zero if LSL is omitted.

Vd
Is the name of the SIMD and FP register, in the range 0 to 31.

imm8
Is an 8-bit immediate.

9.8.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.9 BIC (vector, register)

Bitwise bit clear (register).

9.9.1 Syntax

```
BIC  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `8B` or `16B`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.9.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.10 BIF (vector)

Bitwise insert if false.

9.10.1 Syntax

```
BIF  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `8B` or `16B`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.10.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.11 BIT (vector)

Bitwise insert if true.

9.11.1 Syntax

```
BIT  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `8B` or `16B`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.11.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.12 BSL (vector)

Bitwise select.

9.12.1 Syntax

```
BSL  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `8B` or `16B`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.12.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.13 CLS (vector)

Count leading sign bits.

9.13.1 Syntax

```
CLS  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S` or `4S`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.13.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.14 CLZ (vector)

Count leading zero bits.

9.14.1 Syntax

```
CLZ  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S` or `4S`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.14.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.15 CMEQ (vector, register)

Compare bitwise equal, setting destination vector element to all ones if the condition holds, else zero.

9.15.1 Syntax

```
CMEQ  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.15.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.16 CMEQ (vector, zero)

Compare bitwise equal to zero, setting destination vector element to all ones if the condition holds, else zero.

9.16.1 Syntax

```
CMEQ  Vd.T, Vn.T, #0
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.16.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.17 CMGE (vector, register)

Compare signed greater than or equal.

9.17.1 Syntax

```
CMGE  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.17.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.18 CMGE (vector, zero)

Compare signed greater than or equal to zero, setting destination vector element to all ones if the condition holds, else zero.

9.18.1 Syntax

```
CMGE  Vd.T, Vn.T, #0
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.18.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.19 CMGT (vector, register)

Compare signed greater than, setting destination vector element to all ones if the condition holds, else zero.

9.19.1 Syntax

```
CMGT  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.19.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.20 CMGT (vector, zero)

Compare signed greater than zero, setting destination vector element to all ones if the condition holds, else zero.

9.20.1 Syntax

```
CMGT  Vd.T, Vn.T, #0
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.20.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.21 CMHI (vector, register)

Compare unsigned higher, setting destination vector element to all ones if the condition holds, else zero.

9.21.1 Syntax

```
CMHI  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.21.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.22 CMHS (vector, register)

Compare unsigned higher or same, setting destination vector element to all ones if the condition holds, else zero.

9.22.1 Syntax

```
CMHS  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.22.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.23 CMLE (vector, zero)

Compare signed less than or equal to zero, setting destination vector element to all ones if the condition holds, else zero.

9.23.1 Syntax

```
CMLE  Vd.T, Vn.T, #0
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.23.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.24 CMLT (vector, zero)

Compare signed less than zero, setting destination vector element to all ones if the condition holds, else zero.

9.24.1 Syntax

```
CMLT  Vd.T, Vn.T, #0
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.24.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.25 CMTST (vector)

Compare bitwise test bits nonzero, setting destination vector element to all ones if the condition holds, else zero.

9.25.1 Syntax

```
CMTST  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.25.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.26 CNT (vector)

Population count per byte.

9.26.1 Syntax

```
CNT  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `8B` or `16B`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.26.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.27 DUP (vector, element)

Duplicate vector element to vector.

This instruction is used by the alias `MOV` (scalar).

9.27.1 Syntax

`DUP  Vd.T, Vn.Ts[index]`

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Ts Is an element size specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

index Is the element index, in the range shown in Usage.

9.27.2 Usage

The following table shows valid specifier combinations:

Table 9-4 DUP (Vector) specifier combinations

T	*Ts*	*index*
8B	B	0 to 15
16B	B	0 to 15
4H	H	0 to 7
8H	H	0 to 7
2S	S	0 to 3
4S	S	0 to 3
2D	D	0 or 1

9.27.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.28 DUP (vector, general)

Duplicate general-purpose register to vector.

9.28.1 Syntax

```
DUP  Vd.T, Rn
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

R Is the width specifier for the general-purpose source register, and can be either `W` or `X`.

n Is the number, in the range 0 to 30, or the name ZR (31).

9.28.2 Usage

The following table shows valid specifier combinations:

Table 9-5 DUP specifier combinations

T	*R*
8B	W
16B	W
4H	W
8H	W
2S	W
4S	W
2D	X

9.28.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.29 EOR (vector)

Bitwise exclusive OR.

9.29.1 Syntax

```
EOR  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `8B` or `16B`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.29.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.30 EXT (vector)

Extract vector from pair of vectors.

9.30.1 Syntax

```
EXT  Vd.T, Vn.T, Vm.T, #index
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either 8B or 16B.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

index Is the lowest numbered byte element to be extracted in the range shown in Usage.

9.30.2 Usage

The following table shows valid specifier combinations:

Table 9-6 EXT specifier combinations

T	*index*
8B	0 to 7
16B	0 to 15

9.30.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.31 FABD (vector)

Floating-point absolute difference.

9.31.1 Syntax

```
FABD  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.31.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.32 FABS (vector)

Floating-point absolute value.

9.32.1 Syntax

```
FABS  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.32.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.33 FACGE (vector)

Floating-point absolute compare greater than or equal.

9.33.1 Syntax

```
FACGE  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.33.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.34 FACGT (vector)

Floating-point absolute compare greater than.

9.34.1 Syntax

```
FACGT  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.34.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.35 FADD (vector)

Floating-point add.

9.35.1 Syntax

```
FADD  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.35.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.36 FADDP (vector)

Floating-point add pairwise.

9.36.1 Syntax

```
FADDP  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.36.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.37 FCMEQ (vector, register)

Floating-point compare equal, setting destination vector element to all ones if the condition holds, else zero.

9.37.1 Syntax

```
FCMEQ  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.37.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.38 FCMEQ (vector, zero)

Floating-point compare equal to zero, setting destination vector element to all ones if the condition holds, else zero.

9.38.1 Syntax

```
FCMEQ  Vd.T, Vn.T, #0.0
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.38.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.39 FCMGE (vector, register)

Floating-point compare greater than or equal, setting destination vector element to all ones if the condition holds, else zero.

9.39.1 Syntax

```
FCMGE  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.39.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.40 FCMGE (vector, zero)

Floating-point compare greater than or equal to zero, setting destination vector element to all ones if the condition holds, else zero.

9.40.1 Syntax

```
FCMGE  Vd.T, Vn.T, #0.0
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.40.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.41 FCMGT (vector, register)

Floating-point compare greater than, setting destination vector element to all ones if the condition holds, else zero.

9.41.1 Syntax

```
FCMGT  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.41.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.42 FCMGT (vector, zero)

Floating-point compare greater than zero, setting destination vector element to all ones if the condition holds, else zero.

9.42.1 Syntax

```
FCMGT  Vd.T, Vn.T, #0.0
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.42.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.43 FCMLE (vector, zero)

Floating-point compare less than or equal to zero, setting destination vector element to all ones if the condition holds, else zero.

9.43.1 Syntax

```
FCMLE  Vd.T, Vn.T, #0.0
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.43.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.44 FCMLT (vector, zero)

Floating-point compare less than zero, setting destination vector element to all ones if the condition holds, else zero.

9.44.1 Syntax

```
FCMLT  Vd.T, Vn.T, #0.0
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.44.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.45 FCVTAS (vector)

Floating-point convert to signed integer, rounding to nearest with ties to away.

9.45.1 Syntax

```
FCVTAS  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.45.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.46 FCVTAU (vector)

Floating-point convert to unsigned integer, rounding to nearest with ties to away.

9.46.1 Syntax

```
FCVTAU  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.46.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.47 FCVTL, FCVTL2 (vector)

Floating-point convert to higher precision long.

9.47.1 Syntax

`FCVTL{2}` *`Vd.Ta, Vn.Tb`*

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

`Vd` Is the name of the SIMD and FP destination register, in the range 0 to 31.

`Ta` Is an arrangement specifier, and can be either `4S` or `2D`.

`Vn` Is the name of the SIMD and FP source register, in the range 0 to 31.

`Tb` Is an arrangement specifier, and can be one of the values shown in Usage.

9.47.2 Usage

The following table shows valid specifier combinations:

Table 9-7 FCVTL, FCVTL2 specifier combinations

Q	*Ta*	*Tb*
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.47.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.48 FCVTMS (vector)

Floating-point convert to signed integer, rounding toward minus infinity.

9.48.1 Syntax

```
FCVTMS  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.48.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.49 FCVTMU (vector)

Floating-point convert to unsigned integer, rounding toward minus infinity.

9.49.1 Syntax

```
FCVTMU  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.49.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.50 FCVTN, FCVTN2 (vector)

Floating-point convert to lower precision narrow.

9.50.1 Syntax

`FCVTN{2}` *Vd.Tb*, *Vn.Ta*

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be either `4S` or `2D`.

9.50.2 Usage

The following table shows valid specifier combinations:

Table 9-8 FCVTN, FCVTN2 specifier combinations

Q	*Tb*	*Ta*
-	4H	4S
2	8H	4S
-	2S	2D
2	4S	2D

9.50.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.51 FCVTNS (vector)

Floating-point convert to signed integer, rounding to nearest with ties to even.

9.51.1 Syntax

```
FCVTNS  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.51.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.52 FCVTNU (vector)

Floating-point convert to unsigned integer, rounding to nearest with ties to even.

9.52.1 Syntax

```
FCVTNU  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.52.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.53 FCVTPS (vector)

Floating-point convert to signed integer, rounding toward positive infinity.

9.53.1 Syntax

```
FCVTPS  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.53.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.54 FCVTPU (vector)

Floating-point convert to unsigned integer, rounding toward positive infinity.

9.54.1 Syntax

```
FCVTPU  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.54.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.55 FCVTXN, FCVTXN2 (vector)

Floating-point convert to lower precision narrow, rounding to odd.

9.55.1 Syntax

```
FCVTXN{2}  Vd.Tb, Vn.Ta
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be either `2S` or `4S`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, `2D`.

9.55.2 Usage

The following table shows valid specifier combinations:

Table 9-9 FCVTXN{2} (Vector) specifier combinations

Q	*Tb*	*Ta*
-	2S	2D
2	4S	2D

9.55.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.56 FCVTZS (vector, fixed-point)

Floating-point convert to signed fixed-point, rounding toward zero.

9.56.1 Syntax

```
FCVTZS  Vd.T, Vn.T, #fbits
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

fbits Is the number of fractional bits, in the range 1 to the element width.

9.56.2 Usage

The following table shows valid specifier combinations:

Table 9-10 FCVTZS (Vector) specifier combinations

T	*fbits*
2S	1 to 32
4S	1 to 32
2D	1 to 64

9.56.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.57 FCVTZS (vector, integer)

Floating-point convert to signed integer, rounding toward zero.

9.57.1 Syntax

```
FCVTZS  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.57.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.58 FCVTZU (vector, fixed-point)

Floating-point convert to unsigned fixed-point, rounding toward zero.

9.58.1 Syntax

```
FCVTZU  Vd.T, Vn.T, #fbits
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

fbits Is the number of fractional bits, in the range 1 to the element width.

9.58.2 Usage

The following table shows valid specifier combinations:

Table 9-11 FCVTZU (Vector) specifier combinations

T	*fbits*
2S	1 to 32
4S	1 to 32
2D	1 to 64

9.58.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.59 FCVTZU (vector, integer)

Floating-point convert to unsigned integer, rounding toward zero.

9.59.1 Syntax

```
FCVTZU  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.59.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.60 FDIV (vector)

Floating-point divide.

9.60.1 Syntax

```
FDIV  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.60.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.61 FMAX (vector)

Floating-point maximum.

9.61.1 Syntax

```
FMAX  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.61.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.62 FMAXNM (vector)

Floating-point maximum number.

9.62.1 Syntax

```
FMAXNM  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.62.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.63 FMAXNMP (vector)

Floating-point maximum number pairwise.

9.63.1 Syntax

```
FMAXNMP  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.63.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.64 FMAXNMV (vector)

Floating-point maximum number across vector.

9.64.1 Syntax

```
FMAXNMV  Vd, Vn.T
```

Where:

V Is the destination width specifier, S.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is an arrangement specifier, 4S.

9.64.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.65 FMAXP (vector)

Floating-point maximum pairwise.

9.65.1 Syntax

```
FMAXP  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.65.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.66 FMAXV (vector)

Floating-point maximum across vector.

9.66.1 Syntax

```
FMAXV  Vd, Vn.T
```

Where:

V Is the destination width specifier, `S`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is an arrangement specifier, `4S`.

9.66.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.67 FMIN (vector)

Floating-point minimum.

9.67.1 Syntax

```
FMIN  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.67.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.68 FMINNM (vector)

Floating-point minimum number.

9.68.1 Syntax

```
FMINNM  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.68.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.69 FMINNMP (vector)

Floating-point minimum number pairwise.

9.69.1 Syntax

```
FMINNMP  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.69.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.70 FMINNMV (vector)

Floating-point minimum number across vector.

9.70.1 Syntax

```
FMINNMV  Vd, Vn.T
```

Where:

V Is the destination width specifier, `S`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is an arrangement specifier, `4S`.

9.70.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.71 FMINP (vector)

Floating-point minimum pairwise.

9.71.1 Syntax

```
FMINP  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.71.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.72 FMINV (vector)

Floating-point minimum across vector.

9.72.1 Syntax

```
FMINV  Vd, Vn.T
```

Where:

V Is the destination width specifier, `S`.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is an arrangement specifier, `4S`.

9.72.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.73 FMLA (vector, by element)

Floating-point fused multiply-add to accumulator (by element).

9.73.1 Syntax

```
FMLA  Vd.T, Vn.T, Vm.Ts[index]
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register in the range 0 to 31.

Ts Is an element size specifier, and can be either `S` or `D`.

index Is the element index, in the range shown in Usage.

9.73.2 Usage

The following table shows valid specifier combinations:

Table 9-12 FMLA (Vector) specifier combinations

T	*Ts*	*index*
2S	S	0 to 3
4S	S	0 to 3
2D	D	0 or 1

9.73.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.74 FMLA (vector)

Floating-point fused multiply-add to accumulator.

9.74.1 Syntax

```
FMLA  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.74.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.75 FMLS (vector, by element)

Floating-point fused multiply-subtract from accumulator (by element).

9.75.1 Syntax

```
FMLS  Vd.T, Vn.T, Vm.Ts[index]
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register in the range 0 to 31.

Ts Is an element size specifier, and can be either `S` or `D`.

index Is the element index, in the range shown in Usage.

9.75.2 Usage

The following table shows valid specifier combinations:

Table 9-13 FMLS (Vector) specifier combinations

T	*Ts*	*index*
2S	S	0 to 3
4S	S	0 to 3
2D	D	0 or 1

9.75.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.76 FMLS (vector)

Floating-point fused multiply-subtract from accumulator.

9.76.1 Syntax

```
FMLS  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.76.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.77 FMOV (vector, immediate)

Floating-point move immediate.

9.77.1 Syntax

```
FMOV  Vd.T, #imm    ; Single-precision
FMOV  Vd.2D, #imm    ; Double-precision
```

Where:

T Is an arrangement specifier, and can be either 2S or 4S.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

imm Is a floating-point constant with sign, 3-bit exponent and normalized 4 bits of precision.

9.77.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.78 FMUL (vector, by element)

Floating-point multiply (by element).

9.78.1 Syntax

```
FMUL  Vd.T, Vn.T, Vm.Ts[index]
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register in the range 0 to 31.

Ts Is an element size specifier, and can be either `S` or `D`.

index Is the element index, in the range shown in Usage.

9.78.2 Usage

The following table shows valid specifier combinations:

Table 9-14 FMUL (Vector) specifier combinations

T	*Ts*	*index*
2S	S	0 to 3
4S	S	0 to 3
2D	D	0 or 1

9.78.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.79 FMUL (vector)

Floating-point multiply.

9.79.1 Syntax

```
FMUL  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.79.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.80 FMULX (vector, by element)

Floating-point multiply extended (by element).

9.80.1 Syntax

```
FMULX  Vd.T, Vn.T, Vm.Ts[index]
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register in the range 0 to 31.

Ts Is an element size specifier, and can be either `S` or `D`.

index Is the element index, in the range shown in Usage.

9.80.2 Usage

The following table shows valid specifier combinations:

Table 9-15 FMULX (Vector) specifier combinations

T	*Ts*	*index*
2S	S	0 to 3
4S	S	0 to 3
2D	D	0 or 1

9.80.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.81 FMULX (vector)

Floating-point multiply extended.

9.81.1 Syntax

```
FMULX  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.81.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.82 FNEG (vector)

Floating-point negate.

9.82.1 Syntax

```
FNEG  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.82.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.83 FRECPE (vector)

Floating-point reciprocal estimate.

9.83.1 Syntax

```
FRECPE  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.83.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.84 FRECPS (vector)

Floating-point reciprocal step.

9.84.1 Syntax

```
FRECPS  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.84.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.85 FRINTA (vector)

Floating-point round to integral, to nearest with ties to away.

9.85.1 Syntax

```
FRINTA  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.85.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.86 FRINTI (vector)

Floating-point round to integral, using current rounding mode.

9.86.1 Syntax

```
FRINTI  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.86.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.87 FRINTM (vector)

Floating-point round to integral, toward minus infinity.

9.87.1 Syntax

```
FRINTM  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.87.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.88 FRINTN (vector)

Floating-point round to integral, to nearest with ties to even.

9.88.1 Syntax

```
FRINTN  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.88.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.89 FRINTP (vector)

Floating-point round to integral, toward positive infinity.

9.89.1 Syntax

```
FRINTP  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.89.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.90 FRINTX (vector)

Floating-point round to integral exact, using current rounding mode.

9.90.1 Syntax

```
FRINTX  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.90.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.91 FRINTZ (vector)

Floating-point round to integral, toward zero.

9.91.1 Syntax

```
FRINTZ  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.91.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.92 FRSQRTE (vector)

Floating-point reciprocal square root estimate.

9.92.1 Syntax

```
FRSQRTE  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 2S, 4S or 2D.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.92.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.93 FRSQRTS (vector)

Floating-point reciprocal square root step.

9.93.1 Syntax

```
FRSQRTS  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.93.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.94 FSQRT (vector)

Floating-point square root.

9.94.1 Syntax

```
FSQRT  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.94.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.95 FSUB (vector)

Floating-point subtract.

9.95.1 Syntax

```
FSUB  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.95.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.96 INS (vector, element)

Insert vector element from another vector element.

This instruction is used by the alias `MOV` (element).

9.96.1 Syntax

`INS` *`Vd.Ts[index1], Vn.Ts[index2]`*

Where:

`Vd` Is the name of the SIMD and FP destination register, in the range 0 to 31.

`Ts` Is an element size specifier, and can be one of the values shown in Usage.

`index1` Is the destination element index, in the range shown in Usage.

`Vn` Is the name of the SIMD and FP source register, in the range 0 to 31.

`index2` Is the source element index in the range shown in Usage.

9.96.2 Usage

The following table shows valid specifier combinations:

Table 9-16 INS specifier combinations

Ts	*index1*	*index2*
B	0 to 15	0 to 15
H	0 to 7	0 to 7
S	0 to 3	0 to 3
D	0 or 1	0 or 1

9.96.3 See also

Reference

- *MOV (vector, element)* on page 9-131.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.97 INS (vector, general)

Insert vector element from general-purpose register.

This instruction is used by the alias `MOV` (from general).

9.97.1 Syntax

`INS  Vd.Ts[index], Rn`

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ts Is an element size specifier, and can be one of the values shown in Usage.

index Is the element index, in the range shown in Usage.

R Is the width specifier for the general-purpose source register, and can be either `W` or `X`.

n Is the number, in the range 0 to 30, or the name ZR (31).

9.97.2 Usage

The following table shows valid specifier combinations:

Table 9-17 INS specifier combinations

Ts	*index*	*R*
B	0 to 15	W
H	0 to 7	W
S	0 to 3	W
D	0 or 1	X

9.97.3 See also

Reference

- *MOV (vector, from general)* on page 9-132.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.98 LD1 (vector, multiple structures)

Load multiple 1-element structures to one, two, three or four registers.

9.98.1 Syntax

```
LD1  { Vt.T }, [Xn|SP]    ; One register

LD1  { Vt.T, Vt2.T }, [Xn|SP]    ; Two registers

LD1  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP]    ; Three registers

LD1  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP]    ; Four registers

LD1  { Vt.T }, [Xn|SP], imm    ; One register, immediate offset, Post-index

LD1  { Vt.T }, [Xn|SP], Xm    ; One register, register offset, Post-index

LD1  { Vt.T, Vt2.T }, [Xn|SP], imm    ; Two registers, immediate offset, Post-index

LD1  { Vt.T, Vt2.T }, [Xn|SP], Xm    ; Two registers, register offset, Post-index

LD1  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP], imm    ; Three registers, immediate offset,
Post-index

LD1  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP], Xm    ; Three registers, register offset,
Post-index

LD1  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP], imm    ; Four registers, immediate offset,
Post-index

LD1  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP], Xm    ; Four registers, register offset,
Post-index
```

Where:

Vt
: Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2
: Is the name of the second SIMD and FP register to be transferred.

Vt3
: Is the name of the third SIMD and FP register to be transferred.

Vt4
: Is the name of the fourth SIMD and FP register to be transferred.

imm
: Is the post-index immediate offset:

  One register, immediate offset
  : Can be one of #8 or #16.

  Two registers, immediate offset
  : Can be one of #16 or #32.

  Three registers, immediate offset
  : Can be one of #24 or #48.

  Four registers, immediate offset
  : Can be one of #32 or #64.

Xm
: Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

T
: Is an arrangement specifier, and can be one of the values shown in Usage.

Xn|SP
: Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Note

Vt, *Vt2*, *Vt3*, and *Vt4* must be consecutive registers. The next consecutive register after V31 is V0.

9.98.2 Usage

The following table shows valid specifier combinations:

Table 9-18 LD1 (One register, immediate offset) specifier combinations

T	*imm*
8B	#8
16B	#16
4H	#8
8H	#16
2S	#8
4S	#16
1D	#8
2D	#16

Table 9-19 LD1 (Two registers, immediate offset) specifier combinations

T	*imm*
8B	#16
16B	#32
4H	#16
8H	#32
2S	#16
4S	#32
1D	#16
2D	#32

Table 9-20 LD1 (Three registers, immediate offset) specifier combinations

T	*imm*
8B	#24
16B	#48
4H	#24
8H	#48
2S	#24

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 9-20 LD1 (Three registers, immediate offset) specifier combinations (continued)

T	*imm*
4S	#48
1D	#24
2D	#48

Table 9-21 LD1 (Four registers, immediate offset) specifier combinations

T	*imm*
8B	#32
16B	#64
4H	#32
8H	#64
2S	#32
4S	#64
1D	#32
2D	#64

9.98.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.99 LD1 (vector, single structure)

Load single 1-element structure to one lane of one register.

9.99.1 Syntax

```
LD1  { Vt.B }[index], [Xn|SP]    ; 8-bit
LD1  { Vt.H }[index], [Xn|SP]    ; 16-bit
LD1  { Vt.S }[index], [Xn|SP]    ; 32-bit
LD1  { Vt.D }[index], [Xn|SP]    ; 64-bit
LD1  { Vt.B }[index], [Xn|SP], #1    ; 8-bit, immediate offset, Post-index
LD1  { Vt.B }[index], [Xn|SP], Xm    ; 8-bit, register offset, Post-index
LD1  { Vt.H }[index], [Xn|SP], #2    ; 16-bit, immediate offset, Post-index
LD1  { Vt.H }[index], [Xn|SP], Xm    ; 16-bit, register offset, Post-index
LD1  { Vt.S }[index], [Xn|SP], #4    ; 32-bit, immediate offset, Post-index
LD1  { Vt.S }[index], [Xn|SP], Xm    ; 32-bit, register offset, Post-index
LD1  { Vt.D }[index], [Xn|SP], #8    ; 64-bit, immediate offset, Post-index
LD1  { Vt.D }[index], [Xn|SP], Xm    ; 64-bit, register offset, Post-index
```

Where:

Vt
Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

index
The value depends on the instruction variant:

8-bit
Is the element index, in the range 0 to 15.

16-bit
Is the element index, in the range 0 to 7.

32-bit
Is the element index, in the range 0 to 3.

64-bit
Is the element index, and can be either 0 or 1.

Xn|SP
Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Xm
Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

9.99.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.100 LD1R (vector)

Load single 1-element structure and replicate to all lanes (of one register).

9.100.1 Syntax

```
LD1R  { Vt.T }, [Xn|SP]    ; No offset
LD1R  { Vt.T }, [Xn|SP], imm    ; Immediate offset, Post-index
LD1R  { Vt.T }, [Xn|SP], Xm    ; Register offset, Post-index
```

Where:

imm Is the post-index immediate offset, and can be one of the values shown in Usage.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

9.100.2 Usage

The following table shows valid specifier combinations:

Table 9-22 LD1R (Immediate offset) specifier combinations

T	*imm*
8B	#1
16B	#1
4H	#2
8H	#2
2S	#4
4S	#4
1D	#8
2D	#8

9.100.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.101 LD2 (vector, multiple structures)

Load multiple 2-element structures to two registers.

9.101.1 Syntax

```
LD2  { Vt.T, Vt2.T }, [Xn|SP]    ; No offset
LD2  { Vt.T, Vt2.T }, [Xn|SP], imm    ; Immediate offset, Post-index
LD2  { Vt.T, Vt2.T }, [Xn|SP], Xm    ; Register offset, Post-index
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

imm Is the post-index immediate offset, and can be either #16 or #32.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Note

Vt and *Vt2* must be consecutive registers. The next consecutive register after V31 is V0.

9.101.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.102 LD2 (vector, single structure)

Load single 2-element structure to one lane of two registers.

9.102.1 Syntax

```
LD2  { Vt.B, Vt2.B }[index], [Xn|SP]    ; 8-bit
LD2  { Vt.H, Vt2.H }[index], [Xn|SP]    ; 16-bit
LD2  { Vt.S, Vt2.S }[index], [Xn|SP]    ; 32-bit
LD2  { Vt.D, Vt2.D }[index], [Xn|SP]    ; 64-bit
LD2  { Vt.B, Vt2.B }[index], [Xn|SP], #2    ; 8-bit, immediate offset, Post-index
LD2  { Vt.B, Vt2.B }[index], [Xn|SP], Xm    ; 8-bit, register offset, Post-index
LD2  { Vt.H, Vt2.H }[index], [Xn|SP], #4    ; 16-bit, immediate offset, Post-index
LD2  { Vt.H, Vt2.H }[index], [Xn|SP], Xm    ; 16-bit, register offset, Post-index
LD2  { Vt.S, Vt2.S }[index], [Xn|SP], #8    ; 32-bit, immediate offset, Post-index
LD2  { Vt.S, Vt2.S }[index], [Xn|SP], Xm    ; 32-bit, register offset, Post-index
LD2  { Vt.D, Vt2.D }[index], [Xn|SP], #16    ; 64-bit, immediate offset, Post-index
LD2  { Vt.D, Vt2.D }[index], [Xn|SP], Xm    ; 64-bit, register offset, Post-index
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

index The value depends on the instruction variant:

- **8-bit** Is the element index, in the range 0 to 15.
- **16-bit** Is the element index, in the range 0 to 7.
- **32-bit** Is the element index, in the range 0 to 3.
- **64-bit** Is the element index, and can be either 0 or 1.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

Note

Vt and *Vt2* must be consecutive registers. The next consecutive register after V31 is V0.

9.102.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.103 LD2R (vector)

Load single 2-element structure and replicate to all lanes of two registers.

9.103.1 Syntax

```
LD2R  { Vt.T, Vt2.T }, [Xn|SP]    ; No offset
LD2R  { Vt.T, Vt2.T }, [Xn|SP], imm    ; Immediate offset, Post-index
LD2R  { Vt.T, Vt2.T }, [Xn|SP], Xm    ; Register offset, Post-index
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

imm Is the post-index immediate offset, and can be one of the values shown in Usage.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Note

Vt and *Vt2* must be consecutive registers. The next consecutive register after V31 is V0.

9.103.2 Usage

The following table shows valid specifier combinations:

Table 9-23 LD2R (Immediate offset) specifier combinations

T	*imm*
8B	#2
16B	#2
4H	#4
8H	#4
2S	#8
4S	#8
1D	#16
2D	#16

9.103.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.104 LD3 (vector, multiple structures)

Load multiple 3-element structures to three registers.

9.104.1 Syntax

```
LD3  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP]    ; No offset
LD3  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP], imm    ; Immediate offset, Post-index
LD3  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP], Xm    ; Register offset, Post-index
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

Vt3 Is the name of the third SIMD and FP register to be transferred.

imm Is the post-index immediate offset, and can be either #24 or #48.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Note

Vt, *Vt2*, and *Vt3* must be consecutive registers. The next consecutive register after V31 is V0.

9.104.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.105 LD3 (vector, single structure)

Load single 3-element structure to one lane of three registers).

9.105.1 Syntax

```
LD3  { Vt.B, Vt2.B, Vt3.B }[index], [Xn|SP]    ; 8-bit

LD3  { Vt.H, Vt2.H, Vt3.H }[index], [Xn|SP]    ; 16-bit

LD3  { Vt.S, Vt2.S, Vt3.S }[index], [Xn|SP]    ; 32-bit

LD3  { Vt.D, Vt2.D, Vt3.D }[index], [Xn|SP]    ; 64-bit

LD3  { Vt.B, Vt2.B, Vt3.B }[index], [Xn|SP], #3    ; 8-bit, immediate offset,
Post-index

LD3  { Vt.B, Vt2.B, Vt3.B }[index], [Xn|SP], Xm    ; 8-bit, register offset, Post-index

LD3  { Vt.H, Vt2.H, Vt3.H }[index], [Xn|SP], #6    ; 16-bit, immediate offset,
Post-index

LD3  { Vt.H, Vt2.H, Vt3.H }[index], [Xn|SP], Xm    ; 16-bit, register offset,
Post-index

LD3  { Vt.S, Vt2.S, Vt3.S }[index], [Xn|SP], #12    ; 32-bit, immediate offset,
Post-index

LD3  { Vt.S, Vt2.S, Vt3.S }[index], [Xn|SP], Xm    ; 32-bit, register offset,
Post-index

LD3  { Vt.D, Vt2.D, Vt3.D }[index], [Xn|SP], #24    ; 64-bit, immediate offset,
Post-index

LD3  { Vt.D, Vt2.D, Vt3.D }[index], [Xn|SP], Xm    ; 64-bit, register offset,
Post-index
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

Vt3 Is the name of the third SIMD and FP register to be transferred.

index The value depends on the instruction variant:

- **8-bit** Is the element index, in the range 0 to 15.
- **16-bit** Is the element index, in the range 0 to 7.
- **32-bit** Is the element index, in the range 0 to 3.
- **64-bit** Is the element index, and can be either 0 or 1.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

Note

Vt, *Vt2*, and *Vt3* must be consecutive registers. The next consecutive register after V31 is V0.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.105.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.106 LD3R (vector)

Load single 3-element structure and replicate to all lanes of three registers.

9.106.1 Syntax

```
LD3R  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP]    ; No offset
LD3R  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP], imm    ; Immediate offset, Post-index
LD3R  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP], Xm    ; Register offset, Post-index
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

Vt3 Is the name of the third SIMD and FP register to be transferred.

imm Is the post-index immediate offset, and can be one of the values shown in Usage.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Note

Vt, *Vt2*, and *Vt3* must be consecutive registers. The next consecutive register after V31 is V0.

9.106.2 Usage

The following table shows valid specifier combinations:

Table 9-24 LD3R (Immediate offset) specifier combinations

T	*imm*
8B	#3
16B	#3
4H	#6
8H	#6
2S	#12
4S	#12
1D	#24
2D	#24

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.106.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.107 LD4 (vector, multiple structures)

Load multiple 4-element structures to four registers.

9.107.1 Syntax

```
LD4  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP]     ; No offset
LD4  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP], imm    ; Immediate offset, Post-index
LD4  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP], Xm    ; Register offset, Post-index
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

Vt3 Is the name of the third SIMD and FP register to be transferred.

Vt4 Is the name of the fourth SIMD and FP register to be transferred.

imm Is the post-index immediate offset, and can be either `#32` or `#64`.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S`, `4S` or `2D`.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Note

Vt, *Vt2*, *Vt3*, and *Vt4* must be consecutive registers. The next consecutive register after V31 is V0.

9.107.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.108 LD4 (vector, single structure)

Load single 4-element structure to one lane of four registers.

9.108.1 Syntax

```
LD4  { Vt.B, Vt2.B, Vt3.B, Vt4.B }[index], [Xn|SP]    ; 8-bit

LD4  { Vt.H, Vt2.H, Vt3.H, Vt4.H }[index], [Xn|SP]    ; 16-bit

LD4  { Vt.S, Vt2.S, Vt3.S, Vt4.S }[index], [Xn|SP]    ; 32-bit

LD4  { Vt.D, Vt2.D, Vt3.D, Vt4.D }[index], [Xn|SP]    ; 64-bit

LD4  { Vt.B, Vt2.B, Vt3.B, Vt4.B }[index], [Xn|SP], #4    ; 8-bit, immediate offset,
Post-index

LD4  { Vt.B, Vt2.B, Vt3.B, Vt4.B }[index], [Xn|SP], Xm    ; 8-bit, register offset,
Post-index

LD4  { Vt.H, Vt2.H, Vt3.H, Vt4.H }[index], [Xn|SP], #8    ; 16-bit, immediate offset,
Post-index

LD4  { Vt.H, Vt2.H, Vt3.H, Vt4.H }[index], [Xn|SP], Xm    ; 16-bit, register offset,
Post-index

LD4  { Vt.S, Vt2.S, Vt3.S, Vt4.S }[index], [Xn|SP], #16    ; 32-bit, immediate offset,
Post-index

LD4  { Vt.S, Vt2.S, Vt3.S, Vt4.S }[index], [Xn|SP], Xm    ; 32-bit, register offset,
Post-index

LD4  { Vt.D, Vt2.D, Vt3.D, Vt4.D }[index], [Xn|SP], #32    ; 64-bit, immediate offset,
Post-index

LD4  { Vt.D, Vt2.D, Vt3.D, Vt4.D }[index], [Xn|SP], Xm    ; 64-bit, register offset,
Post-index
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

Vt3 Is the name of the third SIMD and FP register to be transferred.

Vt4 Is the name of the fourth SIMD and FP register to be transferred.

index The value depends on the instruction variant:

- **8-bit** Is the element index, in the range 0 to 15.
- **16-bit** Is the element index, in the range 0 to 7.
- **32-bit** Is the element index, in the range 0 to 3.
- **64-bit** Is the element index, and can be either 0 or 1.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Note

Vt, *Vt2*, *Vt3*, and *Vt4* must be consecutive registers. The next consecutive register after V31 is V0.

9.108.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.109 LD4R (vector)

Load single 4-element structure and replicate to all lanes of four registers.

9.109.1 Syntax

```
LD4R  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP]    ; No offset
LD4R  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP], imm    ; Immediate offset, Post-index
LD4R  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP], Xm    ; Register offset, Post-index
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

Vt3 Is the name of the third SIMD and FP register to be transferred.

Vt4 Is the name of the fourth SIMD and FP register to be transferred.

imm Is the post-index immediate offset, and can be one of the values shown in Usage.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Note

Vt, *Vt2*, *Vt3*, and *Vt4* must be consecutive registers. The next consecutive register after V31 is V0.

9.109.2 Usage

The following table shows valid specifier combinations:

Table 9-25 LD4R (Immediate offset) specifier combinations

T	*imm*
8B	#4
16B	#4
4H	#8
8H	#8
2S	#16
4S	#16
1D	#32
2D	#32

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.109.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.110 MLA (vector, by element)

Multiply-add to accumulator (by element).

9.110.1 Syntax

```
MLA  Vd.T, Vn.T, Vm.Ts[index]
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is H, then *Vm* must be in the range V0 to V15.
- If *Ts* is S, then *Vm* must be in the range V0 to V31.

Ts Is an element size specifier, and can be either H or S.

index Is the element index, in the range shown in Usage.

9.110.2 Usage

The following table shows valid specifier combinations:

Table 9-26 MLA specifier combinations

T	*Ts*	*index*
4H	H	0 to 7
8H	H	0 to 7
2S	S	0 to 3
4S	S	0 to 3

9.110.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.111 MLA (vector)

Multiply-add to accumulator.

9.111.1 Syntax

```
MLA  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.111.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.112 MLS (vector, by element)

Multiply-subtract from accumulator (by element).

9.112.1 Syntax

```
MLS  Vd.T, Vn.T, Vm.Ts[index]
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is H, then *Vm* must be in the range V0 to V15.
- If *Ts* is S, then *Vm* must be in the range V0 to V31.

Ts Is an element size specifier, and can be either H or S.

index Is the element index, in the range shown in Usage.

9.112.2 Usage

The following table shows valid specifier combinations:

Table 9-27 MLS specifier combinations

T	*Ts*	*index*
4H	H	0 to 7
8H	H	0 to 7
2S	S	0 to 3
4S	S	0 to 3

9.112.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.113 MLS (vector)

Multiply-subtract from accumulator.

9.113.1 Syntax

```
MLS  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.113.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.114 MOV (vector, element)

Move vector element to another vector element.

This instruction is an alias of INS (element).

9.114.1 Syntax

```
MOV  Vd.Ts[index1], Vn.Ts[index2]
```

Equivalent to `INS Vd.Ts[index1], Vn.Ts[index2]`

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ts Is an element size specifier, and can be one of the values shown in Usage.

index1 Is the destination element index, in the range shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

index2 Is the source element index in the range shown in Usage.

9.114.2 Usage

The following table shows valid specifier combinations:

Table 9-28 MOV specifier combinations

Ts	*index1*	*index2*
B	0 to 15	0 to 15
H	0 to 7	0 to 7
S	0 to 3	0 to 3
D	0 or 1	0 or 1

9.114.3 See also

Reference

- *INS (vector, element)* on page 9-106.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.115 MOV (vector, from general)

Move general-purpose register to a vector element.

This instruction is an alias of INS (general).

9.115.1 Syntax

```
MOV  Vd.Ts[index], Rn
```

Equivalent to `INS Vd.Ts[index], Rn`

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ts Is an element size specifier, and can be one of the values shown in Usage.

index Is the element index, in the range shown in Usage.

R Is the width specifier for the general-purpose source register, and can be either W or X.

n Is the number, in the range 0 to 30, or the name ZR (31).

9.115.2 Usage

The following table shows valid specifier combinations:

Table 9-29 MOV specifier combinations

Ts	*index*	*R*
B	0 to 15	W
H	0 to 7	W
S	0 to 3	W
D	0 or 1	X

9.115.3 See also

Reference

- *INS (vector, general)* on page 9-107.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.116 MOV (vector)

Move vector.

This instruction is an alias of ORR (vector, register).

9.116.1 Syntax

```
MOV  Vd.T, Vn.T
```

Equivalent to `ORR Vd.T, Vn.T, Vn.T`

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either 8B or 16B.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

9.116.2 See also

Reference

- *ORR (vector, register)* on page 9-144.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.117 MOV (vector, to general)

Move vector element to general-purpose register.

This instruction is an alias of UMOV.

9.117.1 Syntax

```
MOV  Wd, Vn.S[index]    ; 32-bit
```

Equivalent to `UMOV Wd, Vn.S[index]`

```
MOV  Xd, Vn.D[index]    ; 64-bit
```

Equivalent to `UMOV Xd, Vn.D[index]`

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

index The value depends on the instruction variant:

32-bit Is the element index, in the range shown in Usage.

64-bit Is the element index and can be either 0 or 1.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.117.2 See also

Reference

- *UMOV (vector)* on page 9-261.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.118 MOVI (vector)

Move immediate.

9.118.1 Syntax

```
MOVI  Vd.T, #imm8{, LSL #0}    ; 8-bit
MOVI  Vd.T, #imm8{, LSL #amount}    ; 16-bit shifted immediate
MOVI  Vd.T, #imm8{, LSL #amount}    ; 32-bit shifted immediate
MOVI  Vd.T, #imm8, MSL #amount    ; 32-bit shifting ones
MOVI  Dd, #imm    ; 64-bit scalar
MOVI  Vd.2D, #imm    ; 64-bit vector
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier:

8-bit Can be one of `8B` or `16B`.

16-bit shifted immediate
Can be one of `4H` or `8H`.

32-bit shifted immediate
Can be one of `2S` or `4S`.

32-bit shifting ones
Can be one of `2S` or `4S`.

imm8 Is an 8-bit immediate.

amount Is the shift amount:

16-bit shifted immediate
Can be one of `0` or `8`.

32-bit shifted immediate
Can be one of `0`, `8`, `16` or `24`.

32-bit shifting ones
Can be one of `8` or `16`.

Defaults to zero if LSL is omitted.

Dd Is the 64-bit name of the SIMD and FP destination register, in the range 0 to 31.

imm Is a 64-bit immediate.

9.118.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.119 MUL (vector, by element)

Multiply (by element).

9.119.1 Syntax

```
MUL  Vd.T, Vn.T, Vm.Ts[index]
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is H, then *Vm* must be in the range V0 to V15.
- If *Ts* is S, then *Vm* must be in the range V0 to V31.

Ts Is an element size specifier, and can be either H or S.

index Is the element index, in the range shown in Usage.

9.119.2 Usage

The following table shows valid specifier combinations:

Table 9-30 MUL specifier combinations

T	*Ts*	*index*
4H	H	0 to 7
8H	H	0 to 7
2S	S	0 to 3
4S	S	0 to 3

9.119.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.120 MUL (vector)

Multiply.

9.120.1 Syntax

```
MUL  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.120.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.121 MVN (vector)

Bitwise NOT.

This instruction is an alias of `NOT`.

9.121.1 Syntax

`MVN` *`Vd.T, Vn.T`*

Equivalent to `NOT` *`Vd.T, Vn.T`*

Where:

`Vd` Is the name of the SIMD and FP destination register, in the range 0 to 31.

`T` Is an arrangement specifier, and can be either `8B` or `16B`.

`Vn` Is the name of the SIMD and FP source register, in the range 0 to 31.

9.121.2 See also

Reference

- *NOT (vector)* on page 9-141.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.122 MVNI (vector)

Move inverted immediate.

9.122.1 Syntax

```
MVNI  Vd.T, #imm8{, LSL #amount}    ; 16-bit shifted immediate
MVNI  Vd.T, #imm8{, LSL #amount}    ; 32-bit shifted immediate
MVNI  Vd.T, #imm8, MSL #amount    ; 32-bit shifting ones
```

Where:

T
: Is an arrangement specifier:

  16-bit shifted immediate
  : Can be one of 4H or 8H.

  32-bit shifted immediate
  : Can be one of 2S or 4S.

  32-bit shifting ones
  : Can be one of 2S or 4S.

amount
: Is the shift amount:

  16-bit shifted immediate
  : Can be one of 0 or 8.

  32-bit shifted immediate
  : Can be one of 0, 8, 16 or 24.

  32-bit shifting ones
  : Can be one of 8 or 16.

  Defaults to zero if LSL is omitted.

Vd
: Is the name of the SIMD and FP destination register, in the range 0 to 31.

imm8
: Is an 8-bit immediate.

9.122.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.123 NEG (vector)

Negate.

9.123.1 Syntax

```
NEG  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.123.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.124 NOT (vector)

Bitwise NOT.

This instruction is used by the alias `MVN`.

9.124.1 Syntax

`NOT` *`Vd.T, Vn.T`*

Where:

`Vd` Is the name of the SIMD and FP destination register, in the range 0 to 31.

`T` Is an arrangement specifier, and can be either `8B` or `16B`.

`Vn` Is the name of the SIMD and FP source register, in the range 0 to 31.

9.124.2 See also

Reference

- *MVN (vector)* on page 9-138.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.125 ORN (vector)

Bitwise inclusive OR NOT.

9.125.1 Syntax

```
ORN  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `8B` or `16B`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.125.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.126 ORR (vector, immediate)

Bitwise inclusive OR (immediate).

9.126.1 Syntax

```
ORR  Vd.T, #imm8{, LSL #amount}    ; 16-bit
ORR  Vd.T, #imm8{, LSL #amount}    ; 32-bit
```

Where:

T
Is an arrangement specifier:

16-bit Can be one of 4H or 8H.

32-bit Can be one of 2S or 4S.

amount
Is the shift amount:

16-bit Can be one of 0 or 8.

32-bit Can be one of 0, 8, 16 or 24.

Defaults to zero if LSL is omitted.

Vd
Is the name of the SIMD and FP register, in the range 0 to 31.

imm8
Is an 8-bit immediate.

9.126.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.127 ORR (vector, register)

Bitwise inclusive OR (register).

This instruction is used by the alias `MOV` (vector).

9.127.1 Syntax

```
ORR  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `8B` or `16B`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.127.2 See also

Reference

- *MOV (vector)* on page 9-133.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.128 PMUL (vector)

Polynomial multiply.

9.128.1 Syntax

```
PMUL  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `8B` or `16B`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.128.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.129 PMULL, PMULL2 (vector)

Polynomial multiply long.

9.129.1 Syntax

```
PMULL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, `8H`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.129.2 Usage

The following table shows valid specifier combinations:

Table 9-31 PMULL, PMULL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B

9.129.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.130 RADDHN, RADDHN2 (vector)

Rounding add returning high narrow.

9.130.1 Syntax

```
RADDHN{2}  Vd.Tb, Vn.Ta, Vm.Ta
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.130.2 Usage

The following table shows valid specifier combinations:

Table 9-32 RADDHN, RADDHN2 specifier combinations

Q	*Tb*	*Ta*
-	8B	8H
2	16B	8H
-	4H	4S
2	8H	4S
-	2S	2D
2	4S	2D

9.130.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.131 RBIT (vector)

Reverse bit order.

9.131.1 Syntax

```
RBIT  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `8B` or `16B`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.131.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.132 REV16 (vector)

Reverse elements in 16-bit halfwords.

9.132.1 Syntax

```
REV16  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `8B` or `16B`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.132.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.133 REV32 (vector)

Reverse elements in 32-bit words.

9.133.1 Syntax

```
REV32  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H` or `8H`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.133.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.134 REV64 (vector)

Reverse elements in 64-bit doublewords.

9.134.1 Syntax

```
REV64  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S` or `4S`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.134.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.135 RSHRN, RSHRN2 (vector)

Rounding shift right narrow (immediate).

9.135.1 Syntax

```
RSHRN{2}  Vd.Tb, Vn.Ta, #shift
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

shift Is the right shift amount, in the range 1 to the destination element width in bits, and can be one of the values shown in Usage.

9.135.2 Usage

The following table shows valid specifier combinations:

Table 9-33 RSHRN, RSHRN2 specifier combinations

Q	*Tb*	*Ta*	*shift*
-	8B	8H	1 to 8
2	16B	8H	1 to 8
-	4H	4S	1 to 16
2	8H	4S	1 to 16
-	2S	2D	1 to 32
2	4S	2D	1 to 32

9.135.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.136 RSUBHN, RSUBHN2 (vector)

Rounding subtract returning high narrow.

9.136.1 Syntax

```
RSUBHN{2}  Vd.Tb, Vn.Ta, Vm.Ta
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.136.2 Usage

The following table shows valid specifier combinations:

Table 9-34 RSUBHN, RSUBHN2 specifier combinations

Q	*Tb*	*Ta*
-	8B	8H
2	16B	8H
-	4H	4S
2	8H	4S
-	2S	2D
2	4S	2D

9.136.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.137 SABA (vector)

Signed absolute difference and accumulate.

9.137.1 Syntax

```
SABA  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.137.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.138 SABAL, SABAL2 (vector)

Signed absolute difference and accumulate long.

9.138.1 Syntax

```
SABAL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

2 — Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd — Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta — Is an arrangement specifier, and can be one of the values shown in Usage.

Vn — Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb — Is an arrangement specifier, and can be one of the values shown in Usage.

Vm — Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.138.2 Usage

The following table shows valid specifier combinations:

Table 9-35 SABAL, SABAL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.138.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.139 SABD (vector)

Signed absolute difference.

9.139.1 Syntax

```
SABD  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.139.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.140 SABDL, SABDL2 (vector)

Signed absolute difference long.

9.140.1 Syntax

```
SABDL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.140.2 Usage

The following table shows valid specifier combinations:

Table 9-36 SABDL, SABDL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.140.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.141 SADALP (vector)

Signed add and accumulate long pairwise.

9.141.1 Syntax

```
SADALP  Vd.Ta, Vn.Tb
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

9.141.2 Usage

The following table shows valid specifier combinations:

Table 9-37 SADALP specifier combinations

Ta	*Tb*
4H	8B
8H	16B
2S	4H
4S	8H
1D	2S
2D	4S

9.141.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.142 SADDL, SADDL2 (vector)

Signed add long.

9.142.1 Syntax

```
SADDL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.142.2 Usage

The following table shows valid specifier combinations:

Table 9-38 SADDL, SADDL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.142.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.143 SADDLP (vector)

Signed add long pairwise.

9.143.1 Syntax

```
SADDLP  Vd.Ta, Vn.Tb
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

9.143.2 Usage

The following table shows valid specifier combinations:

Table 9-39 SADDLP specifier combinations

Ta	*Tb*
4H	8B
8H	16B
2S	4H
4S	8H
1D	2S
2D	4S

9.143.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.144 SADDLV (vector)

Signed add long across vector.

9.144.1 Syntax

```
SADDLV  Vd, Vn.T
```

Where:

V Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

9.144.2 Usage

The following table shows valid specifier combinations:

Table 9-40 SADDLV specifier combinations

V	*T*
H	8B
H	16B
S	4H
S	8H
D	4S

9.144.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.145 SADDW, SADDW2 (vector)

Signed add wide.

9.145.1 Syntax

```
SADDW{2}  Vd.Ta, Vn.Ta, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

9.145.2 Usage

The following table shows valid specifier combinations:

Table 9-41 SADDW, SADDW2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.145.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.146 SCVTF (vector, fixed-point)

Signed fixed-point convert to floating-point.

9.146.1 Syntax

```
SCVTF  Vd.T, Vn.T, #fbits
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

fbits Is the number of fractional bits, in the range 1 to the element width.

9.146.2 Usage

The following table shows valid specifier combinations:

Table 9-42 SCVTF (Vector) specifier combinations

T	*fbits*
2S	1 to 32
4S	1 to 32
2D	1 to 64

9.146.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.147 SCVTF (vector, integer)

Signed integer convert to floating-point.

9.147.1 Syntax

```
SCVTF  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `2S`, `4S` or `2D`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.147.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.148 SHADD (vector)

Signed halving add.

9.148.1 Syntax

```
SHADD  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.148.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.149 SHL (vector)

Shift left (immediate).

9.149.1 Syntax

```
SHL  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the left shift amount, in the range 0 to the element width in bits minus 1, and can be one of the values shown in Usage.

9.149.2 Usage

The following table shows valid specifier combinations:

Table 9-43 SHL (Vector) specifier combinations

T	*shift*
8B	0 to 7
16B	0 to 7
4H	0 to 15
8H	0 to 15
2S	0 to 31
4S	0 to 31
2D	0 to 63

9.149.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.150 SHLL, SHLL2 (vector)

Shift left long (by element size).

9.150.1 Syntax

```
SHLL{2}  Vd.Ta, Vn.Tb, #shift
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

shift Is the left shift amount, which must be equal to the source element width in bits, and can be one of the values shown in Usage.

9.150.2 Usage

The following table shows valid specifier combinations:

Table 9-44 SHLL, SHLL2 specifier combinations

Q	*Ta*	*Tb*	*shift*
-	8H	8B	8
2	8H	16B	8
-	4S	4H	16
2	4S	8H	16
-	2D	2S	32
2	2D	4S	32

9.150.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.151 SHRN, SHRN2 (vector)

Shift right narrow (immediate).

9.151.1 Syntax

```
SHRN{2}  Vd.Tb, Vn.Ta, #shift
```

Where:

2 — Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd — Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb — Is an arrangement specifier, and can be one of the values shown in Usage.

Vn — Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta — Is an arrangement specifier, and can be one of the values shown in Usage.

shift — Is the right shift amount, in the range 1 to the destination element width in bits, and can be one of the values shown in Usage.

9.151.2 Usage

The following table shows valid specifier combinations:

Table 9-45 SHRN, SHRN2 specifier combinations

Q	*Tb*	*Ta*	*shift*
-	8B	8H	1 to 8
2	16B	8H	1 to 8
-	4H	4S	1 to 16
2	8H	4S	1 to 16
-	2S	2D	1 to 32
2	4S	2D	1 to 32

9.151.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.152 SHSUB (vector)

Signed halving subtract.

9.152.1 Syntax

```
SHSUB  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.152.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.153 SLI (vector)

Shift left and insert (immediate).

9.153.1 Syntax

```
SLI  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the left shift amount, in the range 0 to the element width in bits minus 1, and can be one of the values shown in Usage.

9.153.2 Usage

The following table shows valid specifier combinations:

Table 9-46 SLI (Vector) specifier combinations

T	*shift*
8B	0 to 7
16B	0 to 7
4H	0 to 15
8H	0 to 15
2S	0 to 31
4S	0 to 31
2D	0 to 63

9.153.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.154 SMAX (vector)

Signed maximum.

9.154.1 Syntax

```
SMAX  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.154.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.155 SMAXP (vector)

Signed maximum pairwise.

9.155.1 Syntax

```
SMAXP  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.155.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.156 SMAXV (vector)

Signed maximum across vector.

9.156.1 Syntax

```
SMAXV  Vd, Vn.T
```

Where:

V Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

9.156.2 Usage

The following table shows valid specifier combinations:

Table 9-47 SMAXV specifier combinations

V	*T*
B	8B
B	16B
H	4H
H	8H
S	4S

9.156.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.157 SMIN (vector)

Signed minimum.

9.157.1 Syntax

```
SMIN  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.157.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.158 SMINP (vector)

Signed minimum pairwise.

9.158.1 Syntax

```
SMINP  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.158.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.159 SMINV (vector)

Signed minimum across vector.

9.159.1 Syntax

```
SMINV  Vd, Vn.T
```

Where:

V Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

9.159.2 Usage

The following table shows valid specifier combinations:

Table 9-48 SMINV specifier combinations

V	*T*
B	8B
B	16B
H	4H
H	8H
S	4S

9.159.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.160 SMLAL, SMLAL2 (vector, by element)

Signed multiply-add long (by element).

9.160.1 Syntax

`SMLAL{2}  Vd.Ta, Vn.Tb, Vm.Ts[index]`

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be either `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is `H`, then *Vm* must be in the range V0 to V15.
- If *Ts* is `S`, then *Vm* must be in the range V0 to V31.

Ts Is an element size specifier, and can be either `H` or `S`.

index Is the element index, in the range shown in Usage.

9.160.2 Usage

The following table shows valid specifier combinations:

Table 9-49 SMLAL, SMLAL2 specifier combinations

Q	*Ta*	*Tb*	*Ts*	*index*
-	4S	4H	H	0 to 7
2	4S	8H	H	0 to 7
-	2D	2S	S	0 to 3
2	2D	4S	S	0 to 3

9.160.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.161 SMLAL, SMLAL2 (vector)

Signed multiply-add long.

9.161.1 Syntax

```
SMLAL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.161.2 Usage

The following table shows valid specifier combinations:

Table 9-50 SMLAL, SMLAL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.161.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.162 SMLSL, SMLSL2 (vector, by element)

Signed multiply-subtract long (by element).

9.162.1 Syntax

`SMLSL{2}  Vd.Ta, Vn.Tb, Vm.Ts[index]`

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be either `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is `H`, then *Vm* must be in the range V0 to V15.
- If *Ts* is `S`, then *Vm* must be in the range V0 to V31.

Ts Is an element size specifier, and can be either `H` or `S`.

index Is the element index, in the range shown in Usage.

9.162.2 Usage

The following table shows valid specifier combinations:

Table 9-51 SMLSL, SMLSL2 specifier combinations

Q	*Ta*	*Tb*	*Ts*	*index*
-	4S	4H	H	0 to 7
2	4S	8H	H	0 to 7
-	2D	2S	S	0 to 3
2	2D	4S	S	0 to 3

9.162.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.163 SMLSL, SMLSL2 (vector)

Signed multiply-subtract long.

9.163.1 Syntax

```
SMLSL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.163.2 Usage

The following table shows valid specifier combinations:

Table 9-52 SMLSL, SMLSL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.163.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.164 SMOV (vector)

Signed move vector element to general-purpose register.

9.164.1 Syntax

```
SMOV  Wd, Vn.Ts[index]    ; 32-bit
SMOV  Xd, Vn.Ts[index]    ; 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Ts Is an element size specifier:

32-bit Can be one of `B` or `H`.

64-bit Can be one of `B`, `H` or `S`.

index Is the element index, in the range shown in Usage.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.164.2 Usage

The following table shows valid specifier combinations:

Table 9-53 SMOV (32-bit) specifier combinations

Ts	*index*
B	0 to 15
H	0 to 7

Table 9-54 SMOV (64-bit) specifier combinations

Ts	*index*
B	0 to 15
H	0 to 7
S	0 to 3

9.164.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.165 SMULL, SMULL2 (vector, by element)

Signed multiply long (by element).

9.165.1 Syntax

```
SMULL{2}  Vd.Ta, Vn.Tb, Vm.Ts[index]
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be either `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is `H`, then *Vm* must be in the range V0 to V15.
- If *Ts* is `S`, then *Vm* must be in the range V0 to V31.

Ts Is an element size specifier, and can be either `H` or `S`.

index Is the element index, in the range shown in Usage.

9.165.2 Usage

The following table shows valid specifier combinations:

Table 9-55 SMULL, SMULL2 specifier combinations

Q	*Ta*	*Tb*	*Ts*	*index*
-	4S	4H	H	0 to 7
2	4S	8H	H	0 to 7
-	2D	2S	S	0 to 3
2	2D	4S	S	0 to 3

9.165.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.166 SMULL, SMULL2 (vector)

Signed multiply long.

9.166.1 Syntax

```
SMULL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

2 — Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd — Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta — Is an arrangement specifier, and can be one of the values shown in Usage.

Vn — Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb — Is an arrangement specifier, and can be one of the values shown in Usage.

Vm — Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.166.2 Usage

The following table shows valid specifier combinations:

Table 9-56 SMULL, SMULL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.166.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.167 SQABS (vector)

Signed saturating absolute value.

9.167.1 Syntax

```
SQABS  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.167.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.168 SQADD (vector)

Signed saturating add.

9.168.1 Syntax

```
SQADD  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.168.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.169 SQDMLAL, SQDMLAL2 (vector, by element)

Signed saturating doubling multiply-add long (by element).

9.169.1 Syntax

```
SQDMLAL{2}  Vd.Ta, Vn.Tb, Vm.Ts[index]
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be either `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Ts Is an element size specifier, and can be either `H` or `S`.

index Is the element index, in the range shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is `H`, then *Vm* must be in the range V0 to V15.
- If *Ts* is `S`, then *Vm* must be in the range V0 to V31.

9.169.2 Usage

The following table shows valid specifier combinations:

Table 9-57 SQDMLAL{2} (Vector) specifier combinations

Q	*Ta*	*Tb*	*Ts*	*index*
-	4S	4H	H	0 to 7
2	4S	8H	H	0 to 7
-	2D	2S	S	0 to 3
2	2D	4S	S	0 to 3

9.169.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.170 SQDMLAL, SQDMLAL2 (vector)

Signed saturating doubling multiply-add long.

9.170.1 Syntax

`SQDMLAL{2}  Vd.Ta, Vn.Tb, Vm.Tb`

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be either `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.170.2 Usage

The following table shows valid specifier combinations:

Table 9-58 SQDMLAL{2} (Vector) specifier combinations

Q	*Ta*	*Tb*
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.170.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.171 SQDMLSL, SQDMLSL2 (vector, by element)

Signed saturating doubling multiply-subtract long (by element).

9.171.1 Syntax

```
SQDMLSL{2}  Vd.Ta, Vn.Tb, Vm.Ts[index]
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be either `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Ts Is an element size specifier, and can be either `H` or `S`.

index Is the element index, in the range shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is `H`, then *Vm* must be in the range V0 to V15.
- If *Ts* is `S`, then *Vm* must be in the range V0 to V31.

9.171.2 Usage

The following table shows valid specifier combinations:

Table 9-59 SQDMLSL{2} (Vector) specifier combinations

Q	*Ta*	*Tb*	*Ts*	*index*
-	4S	4H	H	0 to 7
2	4S	8H	H	0 to 7
-	2D	2S	S	0 to 3
2	2D	4S	S	0 to 3

9.171.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.172 SQDMLSL, SQDMLSL2 (vector)

Signed saturating doubling multiply-subtract long.

9.172.1 Syntax

`SQDMLSL{2}  Vd.Ta, Vn.Tb, Vm.Tb`

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be either `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.172.2 Usage

The following table shows valid specifier combinations:

Table 9-60 SQDMLSL{2} (Vector) specifier combinations

Q	*Ta*	*Tb*
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.172.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.173 SQDMULH (vector, by element)

Signed saturating doubling multiply returning high half (by element).

9.173.1 Syntax

```
SQDMULH  Vd.T, Vn.T, Vm.Ts[index]
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Ts Is an element size specifier, and can be either H or S.

index Is the element index, in the range shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is H, then *Vm* must be in the range V0 to V15.
- If *Ts* is S, then *Vm* must be in the range V0 to V31.

9.173.2 Usage

The following table shows valid specifier combinations:

Table 9-61 SQDMULH (Vector) specifier combinations

T	*Ts*	*index*
4H	H	0 to 7
8H	H	0 to 7
2S	S	0 to 3
4S	S	0 to 3

9.173.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.174 SQDMULH (vector)

Signed saturating doubling multiply returning high half.

9.174.1 Syntax

```
SQDMULH  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.174.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.175 SQDMULL, SQDMULL2 (vector, by element)

Signed saturating doubling multiply long (by element).

9.175.1 Syntax

```
SQDMULL{2}  Vd.Ta, Vn.Tb, Vm.Ts[index]
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be either `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Ts Is an element size specifier, and can be either `H` or `S`.

index Is the element index, in the range shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is `H`, then *Vm* must be in the range V0 to V15.
- If *Ts* is `S`, then *Vm* must be in the range V0 to V31.

9.175.2 Usage

The following table shows valid specifier combinations:

Table 9-62 SQDMULL{2} (Vector) specifier combinations

Q	*Ta*	*Tb*	*Ts*	*index*
-	4S	4H	H	0 to 7
2	4S	8H	H	0 to 7
-	2D	2S	S	0 to 3
2	2D	4S	S	0 to 3

9.175.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.176 SQDMULL, SQDMULL2 (vector)

Signed saturating doubling multiply long.

9.176.1 Syntax

`SQDMULL{2}  Vd.Ta, Vn.Tb, Vm.Tb`

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be either `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.176.2 Usage

The following table shows valid specifier combinations:

Table 9-63 SQDMULL{2} (Vector) specifier combinations

Q	*Ta*	*Tb*
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.176.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.177 SQNEG (vector)

Signed saturating negate.

9.177.1 Syntax

```
SQNEG  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.177.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.178 SQRDMULH (vector, by element)

Signed saturating rounding doubling multiply returning high half (by element).

9.178.1 Syntax

```
SQRDMULH  Vd.T, Vn.T, Vm.Ts[index]
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Ts Is an element size specifier, and can be either H or S.

index Is the element index, in the range shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is H, then *Vm* must be in the range V0 to V15.
- If *Ts* is S, then *Vm* must be in the range V0 to V31.

9.178.2 Usage

The following table shows valid specifier combinations:

Table 9-64 SQRDMULH (Vector) specifier combinations

T	*Ts*	*index*
4H	H	0 to 7
8H	H	0 to 7
2S	S	0 to 3
4S	S	0 to 3

9.178.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.179 SQRDMULH (vector)

Signed saturating rounding doubling multiply returning high half.

9.179.1 Syntax

```
SQRDMULH  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.179.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.180 SQRSHL (vector)

Signed saturating rounding shift left (register).

9.180.1 Syntax

```
SQRSHL  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.180.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.181 SQRSHRN, SQRSHRN2 (vector)

Signed saturating rounded shift right narrow (immediate).

9.181.1 Syntax

```
SQRSHRN{2}  Vd.Tb, Vn.Ta, #shift
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

shift Is the right shift amount, in the range 1 to the destination element width in bits, and can be one of the values shown in Usage.

9.181.2 Usage

The following table shows valid specifier combinations:

Table 9-65 SQRSHRN{2} (Vector) specifier combinations

Q	*Tb*	*Ta*	*shift*
-	8B	8H	1 to 8
2	16B	8H	1 to 8
-	4H	4S	1 to 16
2	8H	4S	1 to 16
-	2S	2D	1 to 32
2	4S	2D	1 to 32

9.181.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.182 SQRSHRUN, SQRSHRUN2 (vector)

Signed saturating rounded shift right unsigned narrow (immediate).

9.182.1 Syntax

```
SQRSHRUN{2}  Vd.Tb, Vn.Ta, #shift
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

shift Is the right shift amount, in the range 1 to the destination element width in bits, and can be one of the values shown in Usage.

9.182.2 Usage

The following table shows valid specifier combinations:

Table 9-66 SQRSHRUN{2} (Vector) specifier combinations

Q	*Tb*	*Ta*	*shift*
-	8B	8H	1 to 8
2	16B	8H	1 to 8
-	4H	4S	1 to 16
2	8H	4S	1 to 16
-	2S	2D	1 to 32
2	4S	2D	1 to 32

9.182.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.183 SQSHL (vector, immediate)

Signed saturating shift left (immediate).

9.183.1 Syntax

```
SQSHL  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the left shift amount, in the range 0 to the element width in bits minus 1, and can be one of the values shown in Usage.

9.183.2 Usage

The following table shows valid specifier combinations:

Table 9-67 SQSHL (Vector) specifier combinations

T	*shift*
8B	0 to 7
16B	0 to 7
4H	0 to 15
8H	0 to 15
2S	0 to 31
4S	0 to 31
2D	0 to 63

9.183.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.184 SQSHL (vector, register)

Signed saturating shift left (register).

9.184.1 Syntax

```
SQSHL  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.184.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.185 SQSHLU (vector)

Signed saturating shift left unsigned (immediate).

9.185.1 Syntax

```
SQSHLU  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the left shift amount, in the range 0 to the element width in bits minus 1, and can be one of the values shown in Usage.

9.185.2 Usage

The following table shows valid specifier combinations:

Table 9-68 SQSHLU (Vector) specifier combinations

T	*shift*
8B	0 to 7
16B	0 to 7
4H	0 to 15
8H	0 to 15
2S	0 to 31
4S	0 to 31
2D	0 to 63

9.185.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.186 SQSHRN, SQSHRN2 (vector)

Signed saturating shift right narrow (immediate).

9.186.1 Syntax

```
SQSHRN{2}  Vd.Tb, Vn.Ta, #shift
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

shift Is the right shift amount, in the range 1 to the destination element width in bits, and can be one of the values shown in Usage.

9.186.2 Usage

The following table shows valid specifier combinations:

Table 9-69 SQSHRN{2} (Vector) specifier combinations

Q	*Tb*	*Ta*	*shift*
-	8B	8H	1 to 8
2	16B	8H	1 to 8
-	4H	4S	1 to 16
2	8H	4S	1 to 16
-	2S	2D	1 to 32
2	4S	2D	1 to 32

9.186.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.187 SQSHRUN, SQSHRUN2 (vector)

Signed saturating shift right unsigned narrow (immediate).

9.187.1 Syntax

```
SQSHRUN{2}  Vd.Tb, Vn.Ta, #shift
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

shift Is the right shift amount, in the range 1 to the destination element width in bits, and can be one of the values shown in Usage.

9.187.2 Usage

The following table shows valid specifier combinations:

Table 9-70 SQSHRUN{2} (Vector) specifier combinations

Q	*Tb*	*Ta*	*shift*
-	8B	8H	1 to 8
2	16B	8H	1 to 8
-	4H	4S	1 to 16
2	8H	4S	1 to 16
-	2S	2D	1 to 32
2	4S	2D	1 to 32

9.187.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.188 SQSUB (vector)

Signed saturating subtract.

9.188.1 Syntax

```
SQSUB  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.188.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.189 SQXTN, SQXTN2 (vector)

Signed saturating extract narrow.

9.189.1 Syntax

```
SQXTN{2}  Vd.Tb, Vn.Ta
```

Where:

2 — Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd — Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb — Is an arrangement specifier, and can be one of the values shown in Usage.

Vn — Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta — Is an arrangement specifier, and can be one of the values shown in Usage.

9.189.2 Usage

The following table shows valid specifier combinations:

Table 9-71 SQXTN{2} (Vector) specifier combinations

Q	*Tb*	*Ta*
-	8B	8H
2	16B	8H
-	4H	4S
2	8H	4S
-	2S	2D
2	4S	2D

9.189.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.190 SQXTUN, SQXTUN2 (vector)

Signed saturating extract unsigned narrow.

9.190.1 Syntax

```
SQXTUN{2}  Vd.Tb, Vn.Ta
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

9.190.2 Usage

The following table shows valid specifier combinations:

Table 9-72 SQXTUN{2} (Vector) specifier combinations

Q	*Tb*	*Ta*
-	8B	8H
2	16B	8H
-	4H	4S
2	8H	4S
-	2S	2D
2	4S	2D

9.190.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.191 SRHADD (vector)

Signed rounding halving add.

9.191.1 Syntax

```
SRHADD  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of `8B`, `16B`, `4H`, `8H`, `2S` or `4S`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.191.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.192 SRI (vector)

Shift right and insert (immediate).

9.192.1 Syntax

```
SRI  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the element width in bits, and can be one of the values shown in Usage.

9.192.2 Usage

The following table shows valid specifier combinations:

Table 9-73 SRI (Vector) specifier combinations

T	*shift*
8B	1 to 8
16B	1 to 8
4H	1 to 16
8H	1 to 16
2S	1 to 32
4S	1 to 32
2D	1 to 64

9.192.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.193 SRSHL (vector)

Signed rounding shift left (register).

9.193.1 Syntax

```
SRSHL  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.193.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.194 SRSHR (vector)

Signed rounding shift right (immediate).

9.194.1 Syntax

```
SRSHR  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the element width in bits, and can be one of the values shown in Usage.

9.194.2 Usage

The following table shows valid specifier combinations:

Table 9-74 SRSHR (Vector) specifier combinations

T	*shift*
8B	1 to 8
16B	1 to 8
4H	1 to 16
8H	1 to 16
2S	1 to 32
4S	1 to 32
2D	1 to 64

9.194.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.195 SRSRA (vector)

Signed rounding shift right and accumulate (immediate).

9.195.1 Syntax

```
SRSRA  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the element width in bits, and can be one of the values shown in Usage.

9.195.2 Usage

The following table shows valid specifier combinations:

Table 9-75 SRSRA (Vector) specifier combinations

T	*shift*
8B	1 to 8
16B	1 to 8
4H	1 to 16
8H	1 to 16
2S	1 to 32
4S	1 to 32
2D	1 to 64

9.195.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.196 SSHL (vector)

Signed shift left (register).

9.196.1 Syntax

```
SSHL  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.196.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.197 SSHLL, SSHLL2 (vector)

Signed shift left long (immediate).

This instruction is used by the alias SXTL, SXTL2.

9.197.1 Syntax

```
SSHLL{2}  Vd.Ta, Vn.Tb, #shift
```

Where:

2 — Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd — Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta — Is an arrangement specifier, and can be one of the values shown in Usage.

Vn — Is the name of the SIMD and FP source register, in the range 0 to 31.

Tb — Is an arrangement specifier, and can be one of the values shown in Usage.

shift — Is the left shift amount, in the range 0 to the source element width in bits minus 1, and can be one of the values shown in Usage.

9.197.2 Usage

The following table shows valid specifier combinations:

Table 9-76 SSHLL, SSHLL2 specifier combinations

Q	*Ta*	*Tb*	*shift*
-	8H	8B	0 to 7
2	8H	16B	0 to 7
-	4S	4H	0 to 15
2	4S	8H	0 to 15
-	2D	2S	0 to 31
2	2D	4S	0 to 31

9.197.3 See also

Reference

- *SXTL, SXTL2 (vector)* on page 9-233.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.198 SSHR (vector)

Signed shift right (immediate).

9.198.1 Syntax

```
SSHR  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the element width in bits, and can be one of the values shown in Usage.

9.198.2 Usage

The following table shows valid specifier combinations:

Table 9-77 SSHR (Vector) specifier combinations

T	*shift*
8B	1 to 8
16B	1 to 8
4H	1 to 16
8H	1 to 16
2S	1 to 32
4S	1 to 32
2D	1 to 64

9.198.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.199 SSRA (vector)

Signed shift right and accumulate (immediate).

9.199.1 Syntax

```
SSRA  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the element width in bits, and can be one of the values shown in Usage.

9.199.2 Usage

The following table shows valid specifier combinations:

Table 9-78 SSRA (Vector) specifier combinations

T	*shift*
8B	1 to 8
16B	1 to 8
4H	1 to 16
8H	1 to 16
2S	1 to 32
4S	1 to 32
2D	1 to 64

9.199.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.200 SSUBL, SSUBL2 (vector)

Signed subtract long.

9.200.1 Syntax

```
SSUBL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.200.2 Usage

The following table shows valid specifier combinations:

Table 9-79 SSUBL, SSUBL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.200.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.201 SSUBW, SSUBW2 (vector)

Signed subtract wide.

9.201.1 Syntax

```
SSUBW{2}  Vd.Ta, Vn.Ta, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

9.201.2 Usage

The following table shows valid specifier combinations:

Table 9-80 SSUBW, SSUBW2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.201.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.202 ST1 (vector, multiple structures)

Store multiple 1-element structures from one, two three or four registers.

9.202.1 Syntax

```
ST1  { Vt.T }, [Xn|SP]    ; One register

ST1  { Vt.T, Vt2.T }, [Xn|SP]    ; Two registers

ST1  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP]    ; Three registers

ST1  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP]    ; Four registers

ST1  { Vt.T }, [Xn|SP], imm    ; One register, immediate offset, Post-index

ST1  { Vt.T }, [Xn|SP], Xm    ; One register, register offset, Post-index

ST1  { Vt.T, Vt2.T }, [Xn|SP], imm    ; Two registers, immediate offset, Post-index

ST1  { Vt.T, Vt2.T }, [Xn|SP], Xm    ; Two registers, register offset, Post-index

ST1  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP], imm    ; Three registers, immediate offset,
Post-index

ST1  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP], Xm    ; Three registers, register offset,
Post-index

ST1  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP], imm    ; Four registers, immediate offset,
Post-index

ST1  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP], Xm    ; Four registers, register offset,
Post-index
```

Where:

Vt
: Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2
: Is the name of the second SIMD and FP register to be transferred.

Vt3
: Is the name of the third SIMD and FP register to be transferred.

Vt4
: Is the name of the fourth SIMD and FP register to be transferred.

imm
: Is the post-index immediate offset:

  One register, immediate offset
  : Can be one of #8 or #16.

  Two registers, immediate offset
  : Can be one of #16 or #32.

  Three registers, immediate offset
  : Can be one of #24 or #48.

  Four registers, immediate offset
  : Can be one of #32 or #64.

Xm
: Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

T
: Is an arrangement specifier, and can be one of the values shown in Usage.

Xn|SP
: Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Note

Vt, *Vt2*, *Vt3*, and *Vt4* must be consecutive registers. The next consecutive register after V31 is V0.

9.202.2 Usage

The following table shows valid specifier combinations:

Table 9-81 ST1 (One register, immediate offset) specifier combinations

T	*imm*
8B	#8
16B	#16
4H	#8
8H	#16
2S	#8
4S	#16
1D	#8
2D	#16

Table 9-82 ST1 (Two registers, immediate offset) specifier combinations

T	*imm*
8B	#16
16B	#32
4H	#16
8H	#32
2S	#16
4S	#32
1D	#16
2D	#32

Table 9-83 ST1 (Three registers, immediate offset) specifier combinations

T	*imm*
8B	#24
16B	#48
4H	#24
8H	#48
2S	#24

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Table 9-83 ST1 (Three registers, immediate offset) specifier combinations (continued)

T	*imm*
4S	#48
1D	#24
2D	#48

Table 9-84 ST1 (Four registers, immediate offset) specifier combinations

T	*imm*
8B	#32
16B	#64
4H	#32
8H	#64
2S	#32
4S	#64
1D	#32
2D	#64

9.202.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.203 ST1 (vector, single structure)

Store single 1-element structure from one lane of one register.

9.203.1 Syntax

```
ST1  { Vt.B }[index], [Xn|SP]    ; 8-bit
ST1  { Vt.H }[index], [Xn|SP]    ; 16-bit
ST1  { Vt.S }[index], [Xn|SP]    ; 32-bit
ST1  { Vt.D }[index], [Xn|SP]    ; 64-bit
ST1  { Vt.B }[index], [Xn|SP], #1    ; 8-bit, immediate offset, Post-index
ST1  { Vt.B }[index], [Xn|SP], Xm    ; 8-bit, register offset, Post-index
ST1  { Vt.H }[index], [Xn|SP], #2    ; 16-bit, immediate offset, Post-index
ST1  { Vt.H }[index], [Xn|SP], Xm    ; 16-bit, register offset, Post-index
ST1  { Vt.S }[index], [Xn|SP], #4    ; 32-bit, immediate offset, Post-index
ST1  { Vt.S }[index], [Xn|SP], Xm    ; 32-bit, register offset, Post-index
ST1  { Vt.D }[index], [Xn|SP], #8    ; 64-bit, immediate offset, Post-index
ST1  { Vt.D }[index], [Xn|SP], Xm    ; 64-bit, register offset, Post-index
```

Where:

Vt
: Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

index
: The value depends on the instruction variant:

  8-bit Is the element index, in the range 0 to 15.

  16-bit Is the element index, in the range 0 to 7.

  32-bit Is the element index, in the range 0 to 3.

  64-bit Is the element index, and can be either 0 or 1.

Xn|SP
: Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Xm
: Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

9.203.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.204 ST2 (vector, multiple structures)

Store multiple 2-element structures from two registers.

9.204.1 Syntax

```
ST2  { Vt.T, Vt2.T }, [Xn|SP]
ST2  { Vt.T, Vt2.T }, [Xn|SP], imm
ST2  { Vt.T, Vt2.T }, [Xn|SP], Xm
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

imm Is the post-index immediate offset, and can be either #16 or #32.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Note

Vt and *Vt2* must be consecutive registers. The next consecutive register after V31 is V0.

9.204.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.205 ST2 (vector, single structure)

Store single 2-element structure from one lane of two registers.

9.205.1 Syntax

```
ST2  { Vt.B, Vt2.B }[index], [Xn|SP]
ST2  { Vt.H, Vt2.H }[index], [Xn|SP]
ST2  { Vt.S, Vt2.S }[index], [Xn|SP]
ST2  { Vt.D, Vt2.D }[index], [Xn|SP]
ST2  { Vt.B, Vt2.B }[index], [Xn|SP], #2
ST2  { Vt.B, Vt2.B }[index], [Xn|SP], Xm
ST2  { Vt.H, Vt2.H }[index], [Xn|SP], #4
ST2  { Vt.H, Vt2.H }[index], [Xn|SP], Xm
ST2  { Vt.S, Vt2.S }[index], [Xn|SP], #8
ST2  { Vt.S, Vt2.S }[index], [Xn|SP], Xm
ST2  { Vt.D, Vt2.D }[index], [Xn|SP], #16
ST2  { Vt.D, Vt2.D }[index], [Xn|SP], Xm
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

index The value depends on the instruction variant:

- **8-bit** Is the element index, in the range 0 to 15.
- **16-bit** Is the element index, in the range 0 to 7.
- **32-bit** Is the element index, in the range 0 to 3.
- **64-bit** Is the element index, and can be either 0 or 1.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

Note

Vt and *Vt2* must be consecutive registers. The next consecutive register after V31 is V0.

9.205.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.206 ST3 (vector, multiple structures)

Store multiple 3-element structures from three registers.

9.206.1 Syntax

```
ST3  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP]
ST3  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP], imm
ST3  { Vt.T, Vt2.T, Vt3.T }, [Xn|SP], Xm
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

Vt3 Is the name of the third SIMD and FP register to be transferred.

imm Is the post-index immediate offset, and can be either #24 or #48.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Note

Vt, *Vt2*, and *Vt3* must be consecutive registers. The next consecutive register after V31 is V0.

9.206.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.207 ST3 (vector, single structure)

Store single 3-element structure from one lane of three registers.

9.207.1 Syntax

```
ST3  { Vt.B, Vt2.B, Vt3.B }[index], [Xn|SP]
ST3  { Vt.H, Vt2.H, Vt3.H }[index], [Xn|SP]
ST3  { Vt.S, Vt2.S, Vt3.S }[index], [Xn|SP]
ST3  { Vt.D, Vt2.D, Vt3.D }[index], [Xn|SP]
ST3  { Vt.B, Vt2.B, Vt3.B }[index], [Xn|SP], #3
ST3  { Vt.B, Vt2.B, Vt3.B }[index], [Xn|SP], Xm
ST3  { Vt.H, Vt2.H, Vt3.H }[index], [Xn|SP], #6
ST3  { Vt.H, Vt2.H, Vt3.H }[index], [Xn|SP], Xm
ST3  { Vt.S, Vt2.S, Vt3.S }[index], [Xn|SP], #12
ST3  { Vt.S, Vt2.S, Vt3.S }[index], [Xn|SP], Xm
ST3  { Vt.D, Vt2.D, Vt3.D }[index], [Xn|SP], #24
ST3  { Vt.D, Vt2.D, Vt3.D }[index], [Xn|SP], Xm
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

Vt3 Is the name of the third SIMD and FP register to be transferred.

index The value depends on the instruction variant:

- **8-bit** Is the element index, in the range 0 to 15.
- **16-bit** Is the element index, in the range 0 to 7.
- **32-bit** Is the element index, in the range 0 to 3.
- **64-bit** Is the element index, and can be either 0 or 1.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

Note

Vt, *Vt2*, and *Vt3* must be consecutive registers. The next consecutive register after V31 is V0.

9.207.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.208 ST4 (vector, multiple structures)

Store multiple 4-element structures from four registers.

9.208.1 Syntax

```
ST4  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP]
ST4  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP], imm
ST4  { Vt.T, Vt2.T, Vt3.T, Vt4.T }, [Xn|SP], Xm
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

Vt3 Is the name of the third SIMD and FP register to be transferred.

Vt4 Is the name of the fourth SIMD and FP register to be transferred.

imm Is the post-index immediate offset, and can be either #32 or #64.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Note

Vt, *Vt2*, *Vt3*, and *Vt4* must be consecutive registers. The next consecutive register after V31 is V0.

9.208.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.209 ST4 (vector, single structure)

Store single 4-element structure from one lane of four registers.

9.209.1 Syntax

```
ST4  { Vt.B, Vt2.B, Vt3.B, Vt4.B }[index], [Xn|SP]
ST4  { Vt.H, Vt2.H, Vt3.H, Vt4.H }[index], [Xn|SP]
ST4  { Vt.S, Vt2.S, Vt3.S, Vt4.S }[index], [Xn|SP]
ST4  { Vt.D, Vt2.D, Vt3.D, Vt4.D }[index], [Xn|SP]
ST4  { Vt.B, Vt2.B, Vt3.B, Vt4.B }[index], [Xn|SP], #4
ST4  { Vt.B, Vt2.B, Vt3.B, Vt4.B }[index], [Xn|SP], Xm
ST4  { Vt.H, Vt2.H, Vt3.H, Vt4.H }[index], [Xn|SP], #8
ST4  { Vt.H, Vt2.H, Vt3.H, Vt4.H }[index], [Xn|SP], Xm
ST4  { Vt.S, Vt2.S, Vt3.S, Vt4.S }[index], [Xn|SP], #16
ST4  { Vt.S, Vt2.S, Vt3.S, Vt4.S }[index], [Xn|SP], Xm
ST4  { Vt.D, Vt2.D, Vt3.D, Vt4.D }[index], [Xn|SP], #32
ST4  { Vt.D, Vt2.D, Vt3.D, Vt4.D }[index], [Xn|SP], Xm
```

Where:

Vt Is the name of the first or only SIMD and FP register to be transferred, in the range 0 to 31.

Vt2 Is the name of the second SIMD and FP register to be transferred.

Vt3 Is the name of the third SIMD and FP register to be transferred.

Vt4 Is the name of the fourth SIMD and FP register to be transferred.

index The value depends on the instruction variant:

- **8-bit** Is the element index, in the range 0 to 15.
- **16-bit** Is the element index, in the range 0 to 7.
- **32-bit** Is the element index, in the range 0 to 3.
- **64-bit** Is the element index, and can be either 0 or 1.

Xn|SP Is the 64-bit name of the general-purpose base register or stack pointer, in the range 0 to 31.

Xm Is the 64-bit name of the general-purpose post-index register, excluding XZR, in the range 0 to 31.

Note

Vt, *Vt2*, *Vt3*, and *Vt4* must be consecutive registers. The next consecutive register after V31 is V0.

9.209.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.210 SUB (vector)

Subtract.

9.210.1 Syntax

```
SUB  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.210.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.211 SUBHN, SUBHN2 (vector)

Subtract returning high narrow.

9.211.1 Syntax

```
SUBHN{2}  Vd.Tb, Vn.Ta, Vm.Ta
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.211.2 Usage

The following table shows valid specifier combinations:

Table 9-85 SUBHN, SUBHN2 specifier combinations

Q	*Tb*	*Ta*
-	8B	8H
2	16B	8H
-	4H	4S
2	8H	4S
-	2S	2D
2	4S	2D

9.211.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.212 SUQADD (vector)

Signed saturating accumulate of unsigned value.

9.212.1 Syntax

```
SUQADD  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.212.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.213 SXTL, SXTL2 (vector)

Signed extend long.

This instruction is an alias of SSHLL, SSHLL2.

9.213.1 Syntax

SXTL{2} *Vd.Ta*, *Vn.Tb*

Equivalent to SSHLL{2} *Vd.Ta*, *Vn.Tb*, #0

Where:

2 Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

9.213.2 Usage

The following table shows valid specifier combinations:

Table 9-86 SXTL, SXTL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.213.3 See also

Reference

- *SSHLL, SSHLL2 (vector)* on page 9-214.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.214 TBL (vector)

Table vector lookup.

9.214.1 Syntax

```
TBL  Vd.Ta, { Vn.16B }, Vm.Ta    ; Single register table
TBL  Vd.Ta, { Vn.16B, Vn+1.16B }, Vm.Ta    ; Two register table
TBL  Vd.Ta, { Vn.16B, Vn+1.16B, Vn+2.16B }, Vm.Ta    ; Three register table
TBL  Vd.Ta, { Vn.16B, Vn+1.16B, Vn+2.16B, Vn+3.16B }, Vm.Ta    ; Four register table
```

Where:

Vn
The value depends on the instruction variant:

Single register table
Is the name of the SIMD and FP table register, in the range 0 to 31.

Two, Three, or Four register table
Is the name of the first SIMD and FP table register, in the range 0 to 31.

Vn+1
Is the name of the second SIMD and FP table register.

Vn+2
Is the name of the third SIMD and FP table register.

Vn+3
Is the name of the fourth SIMD and FP table register.

Vd
Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta
Is an arrangement specifier, and can be either `8B` or `16B`.

Vm
Is the name of the SIMD and FP index register, in the range 0 to 31.

Note

Vn, *Vn+1*, *Vn+2*, and *Vn+3* must be consecutive registers. The next consecutive register after V31 is V0.

9.214.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.215 TBX (vector)

Table vector lookup extension.

9.215.1 Syntax

```
TBX  Vd.Ta, { Vn.16B }, Vm.Ta    ; Single register table
TBX  Vd.Ta, { Vn.16B, Vn+1.16B }, Vm.Ta    ; Two register table
TBX  Vd.Ta, { Vn.16B, Vn+1.16B, Vn+2.16B }, Vm.Ta    ; Three register table
TBX  Vd.Ta, { Vn.16B, Vn+1.16B, Vn+2.16B, Vn+3.16B }, Vm.Ta    ; Four register table
```

Where:

Vn
: The value depends on the instruction variant:

  Single register table
  : Is the name of the SIMD and FP table register, in the range 0 to 31.

  Two, Three, or Four register table
  : Is the name of the first SIMD and FP table register, in the range 0 to 31.

Vn+1
: Is the name of the second SIMD and FP table register.

Vn+2
: Is the name of the third SIMD and FP table register.

Vn+3
: Is the name of the fourth SIMD and FP table register.

Vd
: Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta
: Is an arrangement specifier, and can be either `8B` or `16B`.

Vm
: Is the name of the SIMD and FP index register, in the range 0 to 31.

Note

Vn, *Vn+1*, *Vn+2*, and *Vn+3* must be consecutive registers. The next consecutive register after V31 is V0.

9.215.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.216 TRN1 (vector)

Transpose vectors (primary).

9.216.1 Syntax

```
TRN1  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.216.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.217 TRN2 (vector)

Transpose vectors (secondary).

9.217.1 Syntax

```
TRN2  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.217.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.218 UABA (vector)

Unsigned absolute difference and accumulate.

9.218.1 Syntax

```
UABA  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.218.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.219 UABAL, UABAL2 (vector)

Unsigned absolute difference and accumulate long.

9.219.1 Syntax

```
UABAL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.219.2 Usage

The following table shows valid specifier combinations:

Table 9-87 UABAL, UABAL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.219.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.220 UABD (vector)

Unsigned absolute difference.

9.220.1 Syntax

```
UABD  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.220.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.221 UABDL, UABDL2 (vector)

Unsigned absolute difference long.

9.221.1 Syntax

```
UABDL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.221.2 Usage

The following table shows valid specifier combinations:

Table 9-88 UABDL, UABDL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.221.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.222 UADALP (vector)

Unsigned add and accumulate long pairwise.

9.222.1 Syntax

```
UADALP  Vd.Ta, Vn.Tb
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

9.222.2 Usage

The following table shows valid specifier combinations:

Table 9-89 UADALP specifier combinations

Ta	*Tb*
4H	8B
8H	16B
2S	4H
4S	8H
1D	2S
2D	4S

9.222.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.223 UADDL, UADDL2 (vector)

Unsigned add long.

9.223.1 Syntax

```
UADDL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.223.2 Usage

The following table shows valid specifier combinations:

Table 9-90 UADDL, UADDL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.223.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.224 UADDLP (vector)

Unsigned add long pairwise.

9.224.1 Syntax

```
UADDLP  Vd.Ta, Vn.Tb
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

9.224.2 Usage

The following table shows valid specifier combinations:

Table 9-91 UADDLP specifier combinations

Ta	*Tb*
4H	8B
8H	16B
2S	4H
4S	8H
1D	2S
2D	4S

9.224.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.225 UADDLV (vector)

Unsigned sum long across vector.

9.225.1 Syntax

```
UADDLV  Vd, Vn.T
```

Where:

V Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

9.225.2 Usage

The following table shows valid specifier combinations:

Table 9-92 UADDLV specifier combinations

V	*T*
H	8B
H	16B
S	4H
S	8H
D	4S

9.225.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.226 UADDW, UADDW2 (vector)

Unsigned add wide.

9.226.1 Syntax

```
UADDW{2}  Vd.Ta, Vn.Ta, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

9.226.2 Usage

The following table shows valid specifier combinations:

Table 9-93 UADDW, UADDW2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.226.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.227 UCVTF (vector, fixed-point)

Unsigned fixed-point convert to floating-point.

9.227.1 Syntax

```
UCVTF  Vd.T, Vn.T, #fbits
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

fbits Is the number of fractional bits, in the range 1 to the element width.

9.227.2 Usage

The following table shows valid specifier combinations:

Table 9-94 UCVTF (Vector) specifier combinations

T	*fbits*
2S	1 to 32
4S	1 to 32
2D	1 to 64

9.227.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.228 UCVTF (vector, integer)

Unsigned integer convert to floating-point.

9.228.1 Syntax

```
UCVTF  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 2S, 4S or 2D.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.228.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.229 UHADD (vector)

Unsigned halving add.

9.229.1 Syntax

```
UHADD  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.229.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.230 UHSUB (vector)

Unsigned halving subtract.

9.230.1 Syntax

```
UHSUB  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.230.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.231 UMAX (vector)

Unsigned maximum.

9.231.1 Syntax

```
UMAX  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.231.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.232 UMAXP (vector)

Unsigned maximum pairwise.

9.232.1 Syntax

```
UMAXP  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.232.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.233 UMAXV (vector)

Unsigned maximum across vector.

9.233.1 Syntax

```
UMAXV  Vd, Vn.T
```

Where:

V Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

9.233.2 Usage

The following table shows valid specifier combinations:

Table 9-95 UMAXV specifier combinations

V	*T*
B	8B
B	16B
H	4H
H	8H
S	4S

9.233.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.234 UMIN (vector)

Unsigned minimum.

9.234.1 Syntax

```
UMIN  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.234.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.235 UMINP (vector)

Unsigned minimum pairwise.

9.235.1 Syntax

```
UMINP  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.235.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.236 UMINV (vector)

Unsigned minimum across vector.

9.236.1 Syntax

```
UMINV  Vd, Vn.T
```

Where:

V Is the destination width specifier, and can be one of the values shown in Usage.

d Is the number of the SIMD and FP destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

9.236.2 Usage

The following table shows valid specifier combinations:

Table 9-96 UMINV specifier combinations

V	*T*
B	8B
B	16B
H	4H
H	8H
S	4S

9.236.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.237 UMLAL, UMLAL2 (vector, by element)

Unsigned multiply-add long (by element).

9.237.1 Syntax

```
UMLAL{2}  Vd.Ta, Vn.Tb, Vm.Ts[index]
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be either `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is `H`, then *Vm* must be in the range V0 to V15.
- If *Ts* is `S`, then *Vm* must be in the range V0 to V31.

Ts Is an element size specifier, and can be either `H` or `S`.

index Is the element index, in the range shown in Usage.

9.237.2 Usage

The following table shows valid specifier combinations:

Table 9-97 UMLAL, UMLAL2 specifier combinations

Q	*Ta*	*Tb*	*Ts*	*index*
-	4S	4H	H	0 to 7
2	4S	8H	H	0 to 7
-	2D	2S	S	0 to 3
2	2D	4S	S	0 to 3

9.237.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.238 UMLAL, UMLAL2 (vector)

Unsigned multiply-add long.

9.238.1 Syntax

```
UMLAL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.238.2 Usage

The following table shows valid specifier combinations:

Table 9-98 UMLAL, UMLAL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.238.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.239 UMLSL, UMLSL2 (vector, by element)

Unsigned multiply-subtract long (by element).

9.239.1 Syntax

`UMLSL{2}  Vd.Ta, Vn.Tb, Vm.Ts[index]`

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be either `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is `H`, then *Vm* must be in the range V0 to V15.
- If *Ts* is `S`, then *Vm* must be in the range V0 to V31.

Ts Is an element size specifier, and can be either `H` or `S`.

index Is the element index, in the range shown in Usage.

9.239.2 Usage

The following table shows valid specifier combinations:

Table 9-99 UMLSL, UMLSL2 specifier combinations

Q	*Ta*	*Tb*	*Ts*	*index*
-	4S	4H	H	0 to 7
2	4S	8H	H	0 to 7
-	2D	2S	S	0 to 3
2	2D	4S	S	0 to 3

9.239.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.240 UMLSL, UMLSL2 (vector)

Unsigned multiply-subtract long.

9.240.1 Syntax

```
UMLSL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.240.2 Usage

The following table shows valid specifier combinations:

Table 9-100 UMLSL, UMLSL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.240.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.241 UMOV (vector)

Unsigned move vector element to general-purpose register.

This instruction is used by the alias `MOV` (to general).

9.241.1 Syntax

```
UMOV  Wd, Vn.Ts[index]    ; 32-bit
UMOV  Xd, Vn.Ts[index]    ; 64-bit
```

Where:

Wd Is the 32-bit name of the general-purpose destination register, in the range 0 to 31.

Ts Is an element size specifier:

32-bit Can be one of `B`, `H` or `S`.

64-bit Must be `D`.

index The value depends on the instruction variant:

32-bit Is the element index, in the range shown in Usage.

64-bit Is the element index and can be either 0 or 1.

Xd Is the 64-bit name of the general-purpose destination register, in the range 0 to 31.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.241.2 Usage

The following table shows valid specifier combinations:

Table 9-101 UMOV (32-bit) specifier combinations

Ts	*index*
B	0 to 15
H	0 to 7
S	0 to 3

9.241.3 See also

Reference

- *MOV (vector, to general)* on page 9-134.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.242 UMULL, UMULL2 (vector, by element)

Unsigned multiply long (by element).

9.242.1 Syntax

```
UMULL{2}  Vd.Ta, Vn.Tb, Vm.Ts[index]
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be either `4S` or `2D`.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vm Is the name of the second SIMD and FP source register:

- If *Ts* is `H`, then *Vm* must be in the range V0 to V15.
- If *Ts* is `S`, then *Vm* must be in the range V0 to V31.

Ts Is an element size specifier, and can be either `H` or `S`.

index Is the element index, in the range shown in Usage.

9.242.2 Usage

The following table shows valid specifier combinations:

Table 9-102 UMULL, UMULL2 specifier combinations

Q	*Ta*	*Tb*	*Ts*	*index*
-	4S	4H	H	0 to 7
2	4S	8H	H	0 to 7
-	2D	2S	S	0 to 3
2	2D	4S	S	0 to 3

9.242.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.243 UMULL, UMULL2 (vector)

Unsigned multiply long.

9.243.1 Syntax

```
UMULL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

2 — Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd — Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta — Is an arrangement specifier, and can be one of the values shown in Usage.

Vn — Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb — Is an arrangement specifier, and can be one of the values shown in Usage.

Vm — Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.243.2 Usage

The following table shows valid specifier combinations:

Table 9-103 UMULL, UMULL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.243.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.244 UQADD (vector)

Unsigned saturating add.

9.244.1 Syntax

```
UQADD  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.244.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.245 UQRSHL (vector)

Unsigned saturating rounding shift left (register).

9.245.1 Syntax

```
UQRSHL  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.245.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.246 UQRSHRN, UQRSHRN2 (vector)

Unsigned saturating rounded shift right narrow (immediate).

9.246.1 Syntax

```
UQRSHRN{2}  Vd.Tb, Vn.Ta, #shift
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

shift Is the right shift amount, in the range 1 to the destination element width in bits, and can be one of the values shown in Usage.

9.246.2 Usage

The following table shows valid specifier combinations:

Table 9-104 UQRSHRN{2} (Vector) specifier combinations

Q	*Tb*	*Ta*	*shift*
-	8B	8H	1 to 8
2	16B	8H	1 to 8
-	4H	4S	1 to 16
2	8H	4S	1 to 16
-	2S	2D	1 to 32
2	4S	2D	1 to 32

9.246.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.247 UQSHL (vector, immediate)

Unsigned saturating shift left (immediate).

9.247.1 Syntax

```
UQSHL  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the left shift amount, in the range 0 to the element width in bits minus 1, and can be one of the values shown in Usage.

9.247.2 Usage

The following table shows valid specifier combinations:

Table 9-105 UQSHL (Vector) specifier combinations

T	*shift*
8B	0 to 7
16B	0 to 7
4H	0 to 15
8H	0 to 15
2S	0 to 31
4S	0 to 31
2D	0 to 63

9.247.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.248 UQSHL (vector, register)

Unsigned saturating shift left (register).

9.248.1 Syntax

```
UQSHL  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.248.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.249 UQSHRN, UQSHRN2 (vector)

Unsigned saturating shift right narrow (immediate).

9.249.1 Syntax

```
UQSHRN{2}  Vd.Tb, Vn.Ta, #shift
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

shift Is the right shift amount, in the range 1 to the destination element width in bits, and can be one of the values shown in Usage.

9.249.2 Usage

The following table shows valid specifier combinations:

Table 9-106 UQSHRN{2} (Vector) specifier combinations

Q	*Tb*	*Ta*	*shift*
-	8B	8H	1 to 8
2	16B	8H	1 to 8
-	4H	4S	1 to 16
2	8H	4S	1 to 16
-	2S	2D	1 to 32
2	4S	2D	1 to 32

9.249.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.250 UQSUB (vector)

Unsigned saturating subtract.

9.250.1 Syntax

```
UQSUB  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.250.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.251 UQXTN, UQXTN2 (vector)

Unsigned saturating extract narrow.

9.251.1 Syntax

```
UQXTN{2}  Vd.Tb, Vn.Ta
```

Where:

2 — Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd — Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb — Is an arrangement specifier, and can be one of the values shown in Usage.

Vn — Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta — Is an arrangement specifier, and can be one of the values shown in Usage.

9.251.2 Usage

The following table shows valid specifier combinations:

Table 9-107 UQXTN{2} (Vector) specifier combinations

Q	*Tb*	*Ta*
-	8B	8H
2	16B	8H
-	4H	4S
2	8H	4S
-	2S	2D
2	4S	2D

9.251.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.252 URECPE (vector)

Unsigned reciprocal estimate.

9.252.1 Syntax

```
URECPE  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `2S` or `4S`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.252.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.253 URHADD (vector)

Unsigned rounding halving add.

9.253.1 Syntax

```
URHADD  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S or 4S.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.253.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.254 URSHL (vector)

Unsigned rounding shift left (register).

9.254.1 Syntax

```
URSHL  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.254.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.255 URSHR (vector)

Unsigned rounding shift right (immediate).

9.255.1 Syntax

```
URSHR  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the element width in bits, and can be one of the values shown in Usage.

9.255.2 Usage

The following table shows valid specifier combinations:

Table 9-108 URSHR (Vector) specifier combinations

T	*shift*
8B	1 to 8
16B	1 to 8
4H	1 to 16
8H	1 to 16
2S	1 to 32
4S	1 to 32
2D	1 to 64

9.255.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.256 URSQRTE (vector)

Unsigned reciprocal square root estimate.

9.256.1 Syntax

```
URSQRTE  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be either `2S` or `4S`.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.256.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.257 URSRA (vector)

Unsigned rounding shift right and accumulate (immediate).

9.257.1 Syntax

```
URSRA  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the element width in bits, and can be one of the values shown in Usage.

9.257.2 Usage

The following table shows valid specifier combinations:

Table 9-109 URSRA (Vector) specifier combinations

T	*shift*
8B	1 to 8
16B	1 to 8
4H	1 to 16
8H	1 to 16
2S	1 to 32
4S	1 to 32
2D	1 to 64

9.257.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.258 USHL (vector)

Unsigned shift left (register).

9.258.1 Syntax

```
USHL  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.258.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.259 USHLL, USHLL2 (vector)

Unsigned shift left long (immediate).

This instruction is used by the alias UXTL, UXTL2.

9.259.1 Syntax

```
USHLL{2}  Vd.Ta, Vn.Tb, #shift
```

Where:

2 — Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd — Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta — Is an arrangement specifier, and can be one of the values shown in Usage.

Vn — Is the name of the SIMD and FP source register, in the range 0 to 31.

Tb — Is an arrangement specifier, and can be one of the values shown in Usage.

shift — Is the left shift amount, in the range 0 to the source element width in bits minus 1, and can be one of the values shown in Usage.

9.259.2 Usage

The following table shows valid specifier combinations:

Table 9-110 USHLL, USHLL2 specifier combinations

Q	*Ta*	*Tb*	*shift*
-	8H	8B	0 to 7
2	8H	16B	0 to 7
-	4S	4H	0 to 15
2	4S	8H	0 to 15
-	2D	2S	0 to 31
2	2D	4S	0 to 31

9.259.3 See also

Reference

- *UXTL, UXTL2 (vector)* on page 9-285.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.260 USHR (vector)

Unsigned shift right (immediate).

9.260.1 Syntax

```
USHR  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the element width in bits, and can be one of the values shown in Usage.

9.260.2 Usage

The following table shows valid specifier combinations:

Table 9-111 USHR (Vector) specifier combinations

T	*shift*
8B	1 to 8
16B	1 to 8
4H	1 to 16
8H	1 to 16
2S	1 to 32
4S	1 to 32
2D	1 to 64

9.260.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.261 USQADD (vector)

Unsigned saturating accumulate of signed value.

9.261.1 Syntax

```
USQADD  Vd.T, Vn.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

9.261.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.262 USRA (vector)

Unsigned shift right and accumulate (immediate).

9.262.1 Syntax

```
USRA  Vd.T, Vn.T, #shift
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

shift Is the right shift amount, in the range 1 to the element width in bits, and can be one of the values shown in Usage.

9.262.2 Usage

The following table shows valid specifier combinations:

Table 9-112 USRA (Vector) specifier combinations

T	*shift*
8B	1 to 8
16B	1 to 8
4H	1 to 16
8H	1 to 16
2S	1 to 32
4S	1 to 32
2D	1 to 64

9.262.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.263 USUBL, USUBL2 (vector)

Unsigned subtract long.

9.263.1 Syntax

```
USUBL{2}  Vd.Ta, Vn.Tb, Vm.Tb
```

Where:

2
: Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd
: Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta
: Is an arrangement specifier, and can be one of the values shown in Usage.

Vn
: Is the name of the first SIMD and FP source register, in the range 0 to 31.

Tb
: Is an arrangement specifier, and can be one of the values shown in Usage.

Vm
: Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.263.2 Usage

The following table shows valid specifier combinations:

Table 9-113 USUBL, USUBL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.263.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.264 USUBW, USUBW2 (vector)

Unsigned subtract wide.

9.264.1 Syntax

```
USUBW{2}  Vd.Ta, Vn.Ta, Vm.Tb
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

9.264.2 Usage

The following table shows valid specifier combinations:

Table 9-114 USUBW, USUBW2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.264.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.265 UXTL, UXTL2 (vector)

Unsigned extend long.

This instruction is an alias of USHLL, USHLL2.

9.265.1 Syntax

`UXTL{2}  Vd.Ta, Vn.Tb`

Equivalent to `USHLL{2} Vd.Ta, Vn.Tb, #0`

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

9.265.2 Usage

The following table shows valid specifier combinations:

Table 9-115 UXTL, UXTL2 specifier combinations

Q	*Ta*	*Tb*
-	8H	8B
2	8H	16B
-	4S	4H
2	4S	8H
-	2D	2S
2	2D	4S

9.265.3 See also

Reference

- *USHLL, USHLL2 (vector)* on page 9-279.
- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.266 UZP1 (vector)

Unzip vectors (primary).

9.266.1 Syntax

```
UZP1  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.266.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.267 UZP2 (vector)

Unzip vectors (secondary).

9.267.1 Syntax

```
UZP2  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.267.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.268 XTN, XTN2 (vector)

Extract narrow.

9.268.1 Syntax

```
XTN{2}  Vd.Tb, Vn.Ta
```

Where:

`2` Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements. See *Q* in the Usage table.

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

Tb Is an arrangement specifier, and can be one of the values shown in Usage.

Vn Is the name of the SIMD and FP source register, in the range 0 to 31.

Ta Is an arrangement specifier, and can be one of the values shown in Usage.

9.268.2 Usage

The following table shows valid specifier combinations:

Table 9-116 XTN, XTN2 specifier combinations

Q	*Tb*	*Ta*
-	8B	8H
2	16B	8H
-	4H	4S
2	8H	4S
-	2S	2D
2	4S	2D

9.268.3 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.269 ZIP1 (vector)

Zip vectors (primary).

9.269.1 Syntax

```
ZIP1  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.269.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

9.270 ZIP2 (vector)

Zip vectors (secondary).

9.270.1 Syntax

```
ZIP2  Vd.T, Vn.T, Vm.T
```

Where:

Vd Is the name of the SIMD and FP destination register, in the range 0 to 31.

T Is an arrangement specifier, and can be one of 8B, 16B, 4H, 8H, 2S, 4S or 2D.

Vn Is the name of the first SIMD and FP source register, in the range 0 to 31.

Vm Is the name of the second SIMD and FP source register, in the range 0 to 31.

9.270.2 See also

Reference

- *A64 SIMD scalar instructions in alphabetical order* on page 8-2.
- *A64 SIMD vector instructions in alphabetical order* on page 9-2.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Chapter 10
Directives Reference

The following topics describe the directives that are provided by the ARM assembler, `armasm`:

- *Alphabetical list of directives* on page 10-2.
- *Symbol definition directives* on page 10-3.
- *Data definition directives* on page 10-4.
- *About assembly control directives* on page 10-5.
- *About frame directives* on page 10-6.
- *Reporting directives* on page 10-7.
- *Instruction set and syntax selection directives* on page 10-8.
- *Miscellaneous directives* on page 10-9.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.1 Alphabetical list of directives

Table 10-1 shows a complete list of the directives. Use it to locate individual directives.

Table 10-1 Location of directives

Directive	See	Directive	See	Directive	See
ALIAS	page 10-10	EQU	page 10-35	MACRO *and* MEND	page 10-66
ALIGN	page 10-11	EXPORT *or* GLOBAL	page 10-36	MAP	page 10-69
ARM *and* CODE32	page 10-16	EXPORTAS	page 10-38	MEND *see* MACRO	page 10-66
AREA	page 10-13	EXTERN	page 10-57	MEXIT	page 10-70
ASSERT	page 10-17	FIELD	page 10-39	NOFP	page 10-71
ATTR	page 10-18	FRAME ADDRESS	page 10-41	OPT	page 10-72
CN	page 10-19	FRAME POP	page 10-42	PRESERVE8 *see* REQUIRE8	page 10-78
CODE16	page 10-16	FRAME PUSH	page 10-43	PROC *see* FUNCTION	page 10-52
COMMON	page 10-20	FRAME REGISTER	page 10-44	QN	page 10-74
CP	page 10-21	FRAME RESTORE	page 10-45	RELOC	page 10-76
DATA	page 10-22	FRAME SAVE	page 10-47	REQUIRE	page 10-77
DCB	page 10-23	FRAME STATE REMEMBER	page 10-48	REQUIRE8 *and* PRESERVE8	page 10-78
DCD *and* DCDU	page 10-24	FRAME STATE RESTORE	page 10-49	RLIST	page 10-80
DCDO	page 10-25	FRAME UNWIND ON *or* OFF	page 10-50	RN	page 10-81
DCFD *and* DCFDU	page 10-26	FUNCTION *or* PROC	page 10-52	ROUT	page 10-82
DCFS *and* DCFSU	page 10-27	GBLA, GBLL, *and* GBLS	page 10-54	SETA, SETL, *and* SETS	page 10-83
DCI	page 10-28	GET *or* INCLUDE	page 10-56	SN	page 10-74
DCO *and* DCOU	page 10-29	GLOBAL *see* EXPORT	page 10-36	SPACE or FILL	page 10-85
DCQ *and* DCQU	page 10-30	IF, ELSE, ENDIF, *and* ELIF	page 10-60	SUBT	page 10-86
DCW *and* DCWU	page 10-31	IMPORT	page 10-57	THUMB	page 10-16
DN	page 10-74	INCBIN	page 10-59	THUMBX	page 10-16
ELIF, ELSE *see* IF	page 10-60	INCLUDE *see* GET	page 10-56	TTL	page 10-86
END	page 10-32	INFO	page 10-62	WHILE *and* WEND	page 10-87
ENDFUNC *or* ENDP	page 10-33	KEEP	page 10-63	WN *and* XN	page 10-88
ENDIF *see* IF	page 10-60	LCLA, LCLL, *and* LCLS	page 10-64		
ENTRY	page 10-34	LTORG	page 10-65		

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.2 Symbol definition directives

The following are symbol definition directives:

- *GBLA, GBLL, and GBLS* on page 10-54
  Declare a global arithmetic, logical, or string variable.
- *LCLA, LCLL, and LCLS* on page 10-64
  Declare a local arithmetic, logical, or string variable.
- *SETA, SETL, and SETS* on page 10-83
  Set the value of an arithmetic, logical, or string variable.
- *RELOC* on page 10-76
  Encode an ELF relocation in an object file.
- *RN* on page 10-81
  Define a name for a specified AArch32 state register.
- *WN and XN* on page 10-88
  Define a name for a specified AArch64 state register.
- *RLIST* on page 10-80
  Define a name for a set of general-purpose AArch32 state registers.
- *CN* on page 10-19
  Define a coprocessor register name.
- *CP* on page 10-21
  Define a coprocessor name.
- *QN, DN, and SN* on page 10-74
  Define a double-precision or single-precision floating-point register name.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.3 Data definition directives

The following directives allocate memory, define data structures, and set initial contents of memory:

- *LTORG* on page 10-65

  Set an origin for a literal pool.

- *MAP* on page 10-69

  Set the origin of a storage map.

- *FIELD* on page 10-39

  Define a field within a storage map.

- *SPACE or FILL* on page 10-85

  Allocate a zeroed block of memory.

- *DCB* on page 10-23

  Allocate bytes of memory, and specify the initial contents.

- *DCD and DCDU* on page 10-24

  Allocate words of memory, and specify the initial contents.

- *DCDO* on page 10-25

  Allocate words of memory, and specify the initial contents as offsets from the static base register.

- *DCFD and DCFDU* on page 10-26

  Allocate doublewords of memory, and specify the initial contents as double-precision floating-point numbers.

- *DCFS and DCFSU* on page 10-27

  Allocate words of memory, and specify the initial contents as single-precision floating-point numbers.

- *DCI* on page 10-28

  Allocate words of memory, and specify the initial contents. Mark the location as code not data.

- *DCQ and DCQU* on page 10-30

  Allocate doublewords of memory, and specify the initial contents.

- *DCO and DCOU* on page 10-29

  Allocate quadwords of memory, and specify the initial contents.

- *DCW and DCWU* on page 10-31

  Allocate halfwords of memory, and specify the initial contents.

- *COMMON* on page 10-20

  Allocate a block of memory at a symbol, and specify the alignment.

- *DATA* on page 10-22

  Mark data within a code section. Obsolete, for backwards compatibility only.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.4 About assembly control directives

The following directives control conditional assembly, looping, inclusions, and macros:

- `MACRO` and `MEND`.
- `MEXIT`.
- `IF`, `ELSE`, `ENDIF`, and `ELIF`.
- `WHILE` and `WEND`.

10.4.1 Nesting directives

The following structures can be nested to a total depth of 256:

- `MACRO` definitions.
- `WHILE...WEND` loops.
- `IF...ELSE...ENDIF` conditional structures.
- `INCLUDE` file inclusions.

The limit applies to all structures taken together, regardless of how they are nested. The limit is not 256 of each type of structure.

10.4.2 See also

Reference

- *MACRO and MEND* on page 10-66.
- *MEXIT* on page 10-70.
- *IF, ELSE, ENDIF, and ELIF* on page 10-60.
- *WHILE and WEND* on page 10-87.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.5 About frame directives

Correct use of the frame directives:

- Enables the `armlink --callgraph` option to calculate stack usage of assembler functions.

  The following are the rules that determine stack usage:

  — If a function is not marked with `PROC` or `ENDP`, stack usage is unknown.

  — If a function is marked with `PROC` or `ENDP` but with no `FRAME PUSH` or `FRAME POP`, stack usage is assumed to be zero. This means that there is no requirement to manually add `FRAME PUSH 0` or `FRAME POP 0`.

  — If a function is marked with `PROC` or `ENDP` and with `FRAME PUSH n` or `FRAME POP n`, stack usage is assumed to be `n` bytes.

- Helps you to avoid errors in function construction, particularly when you are modifying existing code.
- Enables the assembler to alert you to errors in function construction.
- Enables backtracing of function calls during debugging.
- Enables the debugger to profile assembler functions.

If you require profiling of assembler functions, but do not want frame description directives for other purposes:

- You must use the `FUNCTION` and `ENDFUNC`, or `PROC` and `ENDP`, directives.
- You can omit the other `FRAME` directives.
- You only have to use the `FUNCTION` and `ENDFUNC` directives for the functions you want to profile.

In DWARF, the canonical frame address is an address on the stack specifying where the call frame of an interrupted function is located.

10.5.1 See also

Reference

- *FRAME ADDRESS* on page 10-41.
- *FRAME POP* on page 10-42.
- *FRAME PUSH* on page 10-43.
- *FRAME REGISTER* on page 10-44.
- *FRAME RESTORE* on page 10-45.
- *FRAME RETURN ADDRESS* on page 10-46.
- *FRAME SAVE* on page 10-47.
- *FRAME STATE REMEMBER* on page 10-48.
- *FRAME STATE RESTORE* on page 10-49.
- *FRAME UNWIND ON* on page 10-50.
- *FRAME UNWIND OFF* on page 10-51.
- *FUNCTION or PROC* on page 10-52.
- *ENDFUNC or ENDP* on page 10-33.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.6 Reporting directives

The following are reporting directives:

- *ASSERT* on page 10-17

  Generates an error message if an assertion is false during assembly.

- *INFO* on page 10-62

  Generates diagnostic information during assembly.

- *OPT* on page 10-72

  Sets listing options.

- *TTL and SUBT* on page 10-86

  Inserts titles and subtitles in listings.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.7 Instruction set and syntax selection directives

The following are the instruction set and syntax selection directives:

- *ARM, THUMB, CODE16 and CODE32* on page 10-16.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.8 Miscellaneous directives

The following topics describe miscellaneous directives:

- *ALIAS* on page 10-10.
- *ALIGN* on page 10-11.
- *AREA* on page 10-13.
- *ATTR* on page 10-18.
- *END* on page 10-32.
- *ENTRY* on page 10-34.
- *EQU* on page 10-35.
- *EXPORT or GLOBAL* on page 10-36.
- *EXPORTAS* on page 10-38.
- *GET or INCLUDE* on page 10-56.
- *IMPORT and EXTERN* on page 10-57.
- *INCBIN* on page 10-59.
- *KEEP* on page 10-63.
- *NOFP* on page 10-71.
- *REQUIRE* on page 10-77.
- *REQUIRE8 and PRESERVE8* on page 10-78.
- *ROUT* on page 10-82.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.9 ALIAS

The ALIAS directive creates an alias for a symbol.

10.9.1 Syntax

```
ALIAS name, aliasname
```

where:

name is the name of the symbol to create an alias for

aliasname is the name of the alias to be created.

10.9.2 Usage

The symbol *name* must already be defined in the source file before creating an alias for it. Properties of *name* set by the EXPORT directive are not inherited by *aliasname*, so you must use EXPORT on *aliasname* if you want to make the alias available outside the current source file. Apart from the properties set by the EXPORT directive, *name* and *aliasname* are identical.

10.9.3 Example

```
baz
bar PROC
    BX lr
    ENDP
    ALIAS  bar,foo    ; foo is an alias for bar
    EXPORT bar
    EXPORT foo        ; foo and bar have identical properties
                      ; because foo was created using ALIAS
    EXPORT baz        ; baz and bar are not identical
                      ; because the size field of baz is not set
```

10.9.4 Incorrect example

```
    EXPORT bar
    IMPORT car
    ALIAS  bar,foo ; ERROR - bar is not defined yet
    ALIAS  car,boo ; ERROR - car is external
bar PROC
    BX lr
    ENDP
```

10.9.5 See also

Reference

- *Data definition directives* on page 10-4.
- *EXPORT or GLOBAL* on page 10-36.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.10 ALIGN

The ALIGN directive aligns the current location to a specified boundary by padding with zeros or NOP instructions.

10.10.1 Syntax

```
ALIGN {expr{,offset{,pad{,padsize}}}}
```

where:

expr	is a numeric expression evaluating to any power of 2 from 2^0 to 2^{31}
offset	can be any numeric expression
pad	can be any numeric expression
padsize	can be 1, 2 or 4.

10.10.2 Operation

The current location is aligned to the next lowest address of the form:

offset + *n* * *expr*

n is any integer which the assembler selects to minimise padding.

If *expr* is not specified, ALIGN sets the current location to the next word (four byte) boundary. The unused space between the previous and the new current location are filled with:

- Copies of *pad*, if *pad* is specified.
- NOP instructions, if all the following conditions are satisfied:
  - *pad* is not specified.
  - The ALIGN directive follows A32 or T32 instructions.
  - The current section has the CODEALIGN attribute set on the AREA directive.
- Zeros otherwise.

pad is treated as a byte, halfword, or word, according to the value of *padsize*. If *padsize* is not specified, *pad* defaults to bytes in data sections, halfwords in T32 code, or words in A32 code.

10.10.3 Usage

Use ALIGN to ensure that your data and code is aligned to appropriate boundaries. This is typically required in the following circumstances:

- The ADR T32 pseudo-instruction can only load addresses that are word aligned, but a label within T32 code might not be word aligned. Use ALIGN 4 to ensure four-byte alignment of an address within T32 code.
- Use ALIGN to take advantage of caches on some ARM processors. For example, the ARM940T has a cache with 16-byte lines. Use ALIGN 16 to align function entries on 16-byte boundaries and maximize the efficiency of the cache.
- A label on a line by itself can be arbitrarily aligned. Following A32 code is word-aligned (T32 code is halfword aligned). The label therefore does not address the code correctly. Use ALIGN 4 (or ALIGN 2 for T32) before the label.

Alignment is relative to the start of the ELF section where the routine is located. The section must be aligned to the same, or coarser, boundaries. The ALIGN attribute on the AREA directive is specified differently.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.10.4 Examples

```
        AREA    cacheable, CODE, ALIGN=3
rout1   ; code          ; aligned on 8-byte boundary
        ; code
        MOV     pc,lr   ; aligned only on 4-byte boundary
        ALIGN   8       ; now aligned on 8-byte boundary
rout2   ; code
```

In the following example, the ALIGN directive tells the assembler that the next instruction is word aligned and offset by 3 bytes. The 3 byte offset is counted from the previous word aligned address, resulting in the second DCB placed in the last byte of the same word and 2 bytes of padding are to be added.

```
        AREA    OffsetExample, CODE
        DCB     1       ; This example places the two bytes in the first
        ALIGN   4,3     ; and fourth bytes of the same word.
        DCB     1       ; The second DCB is offset by 3 bytes from the first DCB
```

In the following example, the ALIGN directive tells the assembler that the next instruction is word aligned and offset by 2 bytes. Here, the 2 byte offset is counted from the next word aligned address, so the value *n* is set to 1 (*n*=0 clashes with the third DCB). This time three bytes of padding are to be added.

```
        AREA    OffsetExample1, CODE
        DCB     1       ; In this example, n cannot be 0 because it clashes with
        DCB     1       ; the 3rd DCB. The assembler sets n to 1.
        DCB     1
        ALIGN   4,2     ; The next instruction is word aligned and offset by 2.
        DCB     2
```

In the following example, the DCB directive makes the PC misaligned. The ALIGN directive ensures that the label subroutine1 and the following instruction are word aligned.

```
        AREA    Example, CODE, READONLY
start   LDR     r6,=label1
        ; code
        MOV     pc,lr
label1  DCB     1       ; PC now misaligned
        ALIGN           ; ensures that subroutine1 addresses
subroutine1             ; the following instruction.
        MOV r5,#0x5
```

10.10.5 See also

Reference

- *Data definition directives* on page 10-4.
- *AREA* on page 10-13.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.11 AREA

The AREA directive instructs the assembler to assemble a new code or data section. Sections are independent, named, indivisible chunks of code or data that are manipulated by the linker.

10.11.1 Syntax

AREA *sectionname*{,*attr*}{,*attr*}...

where:

sectionname is the name to give to the section.

You can choose any name for your sections. However, names starting with a non-alphabetic character must be enclosed in bars or a missing section name error is generated. For example, |1_DataArea|.

Certain names are conventional. For example, |.text| is used for code sections produced by the C compiler, or for code sections otherwise associated with the C library.

attr are one or more comma-delimited section attributes. Valid attributes are:

ALIGN=*expression*

By default, ELF sections are aligned on a four-byte boundary. *expression* can have any integer value from 0 to 31. The section is aligned on a $2^{expression}$-byte boundary. For example, if *expression* is 10, the section is aligned on a 1KB boundary.

This is not the same as the way that the ALIGN *directive is specified.*

Note

- Do not use ALIGN=0 or ALIGN=1 for A32 code sections.
- Do not use ALIGN=0 for T32 code sections.

ASSOC=*section*

section specifies an associated ELF section. *sectionname* must be included in any link that includes *section*

CODE Contains machine instructions. READONLY is the default.

CODEALIGN

Causes the assembler to insert NOP instructions when the ALIGN directive is used after A32 or T32 instructions within the section, unless the ALIGN directive specifies a different padding.

COMDEF Is a common section definition. This ELF section can contain code or data. It must be identical to any other section of the same name in other source files.

Identical ELF sections with the same name are overlaid in the same section of memory by the linker. If any are different, the linker generates a warning and does not overlay the sections.

COMGROUP=*symbol_name*

Is the signature that makes the AREA part of the named ELF section group. See the GROUP=*symbol_name* for more information. The COMGROUP attribute marks the ELF section group with the GRP_COMDAT flag.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

COMMON Is a common data section. You must not define any code or data in it. It is initialized to zeros by the linker. All common sections with the same name are overlaid in the same section of memory by the linker. They do not all have to be the same size. The linker allocates as much space as is required by the largest common section of each name.

DATA Contains data, not instructions. READWRITE is the default.

FINI_ARRAY

Sets the ELF type of the current area to SHT_FINI_ARRAY.

GROUP=*symbol_name*

Is the signature that makes the AREA part of the named ELF section group. It must be defined by the source file, or a file included by the source file. All AREAS with the same *symbol_name* signature are part of the same group. Sections within a group are kept or discarded together.

INIT_ARRAY

Sets the ELF type of the current area to SHT_INIT_ARRAY.

LINKORDER=*section*

Specifies a relative location for the current section in the image. It ensures that the order of all the sections with the LINKORDER attribute, with respect to each other, is the same as the order of the corresponding named *sections* in the image.

MERGE=*n* Indicates that the linker can merge the current section with other sections with the MERGE=*n* attribute. *n* is the size of the elements in the section, for example *n* is 1 for characters. You must not assume that the section is merged, because the attribute does not force the linker to merge the sections.

NOALLOC Indicates that no memory on the target system is allocated to this area.

NOINIT Indicates that the data section is uninitialized, or initialized to zero. It contains only space reservation directives SPACE or DCB, DCD, DCDU, DCQ, DCQU, DCO, DCOU, DCW, or DCWU with initialized values of zero. You can decide at link time whether an area is uninitialized or zero initialized.

PREINIT_ARRAY

Sets the ELF type of the current area to SHT_PREINIT_ARRAY.

READONLY Indicates that this section must not be written to. This is the default for Code areas.

READWRITE Indicates that this section can be read from and written to. This is the default for Data areas.

SECFLAGS=*n*

Adds one or more ELF flags, denoted by *n*, to the current section.

SECTYPE=*n*

Sets the ELF type of the current section to *n*.

STRINGS Adds the SHF_STRINGS flag to the current section. To use the STRINGS attribute, you must also use the MERGE=1 attribute. The contents of the section must be strings that are nul-terminated using the DCB directive.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.11.2 Usage

Use the AREA directive to subdivide your source file into ELF sections. You can use the same name in more than one AREA directive. All areas with the same name are placed in the same ELF section. Only the attributes of the first AREA directive of a particular name are applied.

In general, ARM recommends that you use separate ELF sections for code and data. However, you can put data in code sections. Large programs can usually be conveniently divided into several code sections. Large independent data sets are also usually best placed in separate sections.

The scope of numeric local labels is defined by AREA directives, optionally subdivided by ROUT directives.

There must be at least one AREA directive for an assembly.

Note

The assembler emits R_ARM_TARGET1 relocations for the DCD and DCDU directives if the directive uses PC-relative expressions and is in any of the PREINIT_ARRAY, FINI_ARRAY, or INIT_ARRAY ELF sections. You can override the relocation using the RELOC directive after each DCD or DCDU directive. If this relocation is used, read-write sections might become read-only sections at link time if the platform ABI permits this.

10.11.3 Example

The following example defines a read-only code section named Example.

```
AREA    Example,CODE,READONLY   ; An example code section.
        ; code
```

10.11.4 See also

Concepts

armasm User Guide:

- *ELF sections and the AREA directive* on page 6-5.

Concepts

armlink User Guide:

- Chapter 4 *Image structure and generation*.

Reference

- *ALIGN* on page 10-11.
- *RELOC* on page 10-76.
- *DCD and DCDU* on page 10-24.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.12 ARM, THUMB, CODE16 and CODE32

The ARM directive and the CODE32 directive are synonyms. They instruct the assembler to interpret subsequent instructions as A32 instructions, using either the UAL or the pre-UAL ARM assembler language syntax.

The THUMB directive instructs the assembler to interpret subsequent instructions as T32 instructions, using the UAL syntax.

The CODE16 directive instructs the assembler to interpret subsequent instructions as T32 instructions, using the pre-UAL assembly language syntax.

If necessary, these directives also insert up to three bytes of padding to align to the next word boundary for A32, or up to one byte of padding to align to the next halfword boundary for T32.

Note

These directives are not supported in AArch64 state.

10.12.1 Syntax

```
ARM
THUMB
CODE16
CODE32
```

10.12.2 Usage

In files that contain code using different instruction sets:

- ARM must precede any A32 code. CODE32 is a synonym for ARM.
- THUMB must precede T32 code written in UAL syntax.
- CODE16 must precede T32 code written in pre-UAL syntax.

These directives do not assemble to any instructions. They also do not change the state. They only instruct the assembler to assemble A32 or T32 instructions as appropriate, and insert padding if necessary.

10.12.3 Example

This example shows how you can use ARM and THUMB directives to switch state and assemble both A32 and T32 instructions in a single area.

```
        AREA ToT32, CODE, READONLY      ; Name this block of code
        ENTRY                           ; Mark first instruction to execute
        ARM                             ; Subsequent instructions are A32
start
        ADR     r0, into_t32 + 1        ; Processor starts in A32 state
        BX      r0                      ; Inline switch to T32 state
        THUMB                           ; Subsequent instructions are T32
into_t32
        MOVS    r0, #10                 ; New-style T32 instructions
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.13 ASSERT

The ASSERT directive generates an error message during assembly if a given assertion is false.

10.13.1 Syntax

ASSERT *logical-expression*

where:

logical-expression

is an assertion that can evaluate to either {TRUE} or {FALSE}.

10.13.2 Usage

Use ASSERT to ensure that any necessary condition is met during assembly.

If the assertion is false an error message is generated and assembly fails.

10.13.3 Example

```
ASSERT  label1 <= label2    ; Tests if the address
                            ; represented by label1
                            ; is <= the address
                            ; represented by label2.
```

10.13.4 See also

Reference

- *INFO* on page 10-62.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.14 ATTR

The ATTR set directives set values for the ABI build attributes.

The ATTR scope directives specify the scope for which the set value applies to.

10.14.1 Syntax

```
ATTR FILESCOPE
ATTR SCOPE name
ATTR settype tagid, value
```

where:

name is a section name or symbol name.

settype can be any of:

- SETVALUE.
- SETSTRING.
- SETCOMPATIBLEWITHVALUE.
- SETCOMPATIBLEWITHSTRING.

tagid is an attribute tag name (or its numerical value) defined in the ABI for the ARM Architecture.

value depends on *settype*:

- Is a 32-bit integer value when *settype* is SETVALUE or SETCOMPATIBLEWITHVALUE.
- Is a nul-terminated string when *settype* is SETSTRING or SETCOMPATIBLEWITHSTRING.

10.14.2 Usage

The ATTR set directives following the ATTR FILESCOPE directive apply to the entire object file. The ATTR set directives following the ATTR SCOPE *name* directive apply only to the named section or symbol.

For tags that expect an integer, you must use SETVALUE or SETCOMPATIBLEWITHVALUE. For tags that expect a string, you must use SETSTRING or SETCOMPATIBLEWITHSTRING.

Use SETCOMPATIBLEWITHVALUE and SETCOMPATIBLEWITHSTRING to set tag values which the object file is also compatible with.

10.14.3 Examples

```
ATTR  SETSTRING Tag_CPU_raw_name, "Cortex-A8"
ATTR  SETVALUE  Tag_VFP_arch, 3   ; VFPv3 instructions were permitted.
ATTR  SETVALUE  10, 3             ; 10 is the numerical value of
                                  ; Tag_VFP_arch.
```

10.14.4 See also

Reference

- *Addenda to, and Errata in, the ABI for the ARM Architecture* http://infocenter.arm.com/help/topic/com.arm.doc.ihi0045-/index.html.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.15 CN

The CN directive defines a name for a coprocessor register.

10.15.1 Syntax

name CN *expr*

where:

name is the name to be defined for the coprocessor register. *name* cannot be the same as any of the predefined names.

expr evaluates to a coprocessor register number from 0 to 15.

10.15.2 Usage

Use CN to allocate convenient names to registers, to help you remember what you use each register for.

Note

Avoid conflicting uses of the same register under different names.

The names c0 to c15 are predefined.

10.15.3 Example

```
power    CN  6        ; defines power as a symbol for
                      ; coprocessor register 6
```

10.15.4 See also

Reference

armasm User Guide:

- *Predeclared core register names in AArch32 state* on page 4-8.
- *Predeclared extension register names in AArch32 state* on page 4-9.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.16 COMMON

The COMMON directive allocates a block of memory, of the defined size, at the specified symbol. You specify how the memory is aligned. If alignment is omitted, the default alignment is 4. If size is omitted, the default size is 0.

You can access this memory as you would any other memory, but no space is allocated in object files.

10.16.1 Syntax

COMMON *symbol*{,*size*{,*alignment*}} {[*attr*]}

where:

symbol is the symbol name. The symbol name is case-sensitive.

size is the number of bytes to reserve.

alignment is the alignment.

attr can be any one of:

- DYNAMIC sets the ELF symbol visibility to STV_DEFAULT.
- PROTECTED sets the ELF symbol visibility to STV_PROTECTED.
- HIDDEN sets the ELF symbol visibility to STV_HIDDEN.
- INTERNAL sets the ELF symbol visibility to STV_INTERNAL.

10.16.2 Usage

The linker allocates the required space as zero initialized memory during the link stage. You cannot define, IMPORT or EXTERN a symbol that has already been created by the COMMON directive. In the same way, if a symbol has already been defined or used with the IMPORT or EXTERN directive, you cannot use the same symbol for the COMMON directive.

10.16.3 Example

```
    LDR       r0, =xyz
    COMMON    xyz,255,4   ; defines 255 bytes of ZI store, word-aligned
```

10.16.4 Incorrect examples

```
    COMMON    foo,4,4
    COMMON    bar,4,4
foo DCD       0           ; cannot define label with same name as COMMON
    IMPORT    bar         ; cannot import label with same name as COMMON
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.17 CP

The CP directive defines a name for a specified coprocessor. The coprocessor number must be within the range 0 to 15.

10.17.1 Syntax

name CP *expr*

where:

name is the name to be assigned to the coprocessor. *name* cannot be the same as any of the predefined names.

expr evaluates to a coprocessor number from 0 to 15.

10.17.2 Usage

Use CP to allocate convenient names to coprocessors, to help you to remember what you use each one for.

Note

Avoid conflicting uses of the same coprocessor under different names.

The names p0 to p15 are predefined for coprocessors 0 to 15.

10.17.3 Example

```
dmu    CP  6       ; defines dmu as a symbol for
                   ; coprocessor 6
```

10.17.4 See also

Reference

armasm User Guide:

- *Predeclared core register names in AArch32 state* on page 4-8.
- *Predeclared extension register names in AArch32 state* on page 4-9.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.18 DATA

The DATA directive is no longer required. It is ignored by the assembler.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.19 DCB

The DCB directive allocates one or more bytes of memory, and defines the initial runtime contents of the memory. = is a synonym for DCB.

10.19.1 Syntax

{*label*} DCB *expr*{,*expr*}...

where:

expr is either:

- A numeric expression that evaluates to an integer in the range –128 to 255.
- A quoted string. The characters of the string are loaded into consecutive bytes of store.

10.19.2 Usage

If DCB is followed by an instruction, use an ALIGN directive to ensure that the instruction is aligned.

10.19.3 Example

Unlike C strings, armasm strings are not nul-terminated. You can construct a nul-terminated C string using DCB as follows:

```
C_string   DCB  "C_string",0
```

10.19.4 See also

Concepts

armasm User Guide:

- *Numeric expressions* on page 10-16.

Reference

- *DCD and DCDU* on page 10-24.
- *DCQ and DCQU* on page 10-30.
- *DCW and DCWU* on page 10-31.
- *SPACE or FILL* on page 10-85.
- *ALIGN* on page 10-11.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.20 DCD and DCDU

The DCD directive allocates one or more words of memory, aligned on four-byte boundaries, and defines the initial runtime contents of the memory.

& is a synonym for DCD.

DCDU is the same, except that the memory alignment is arbitrary.

10.20.1 Syntax

{*label*} DCD{U} *expr*{,*expr*}

where:

expr is either:

- A numeric expression.
- A PC-relative expression.

10.20.2 Usage

DCD inserts up to three bytes of padding before the first defined word, if necessary, to achieve four-byte alignment.

Use DCDU if you do not require alignment.

10.20.3 Examples

```
data1   DCD     1,5,20      ; Defines 3 words containing
                            ; decimal values 1, 5, and 20
data2   DCD     mem06 + 4   ; Defines 1 word containing 4 +
                            ; the address of the label mem06
        AREA    MyData, DATA, READWRITE
        DCB     255         ; Now misaligned ...
data3   DCDU    1,5,20      ; Defines 3 words containing
                            ; 1, 5 and 20, not word aligned
```

10.20.4 See also

Concepts

armasm User Guide:

- *Numeric expressions* on page 10-16.

Reference

- *DCB* on page 10-23.
- *DCI* on page 10-28.
- *DCW and DCWU* on page 10-31.
- *DCQ and DCQU* on page 10-30.
- *SPACE or FILL* on page 10-85.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.21 DCDO

The DCDO directive allocates one or more words of memory, aligned on four-byte boundaries, and defines the initial runtime contents of the memory as an offset from the *static base register*, sb (R9).

10.21.1 Syntax

{*label*} DCDO *expr*{,*expr*}...

where:

expr is a register-relative expression or label. The base register must be sb.

10.21.2 Usage

Use DCDO to allocate space in memory for static base register relative relocatable addresses.

10.21.3 Example

```
IMPORT  externsym
DCDO    externsym   ; 32-bit word relocated by offset of
                    ; externsym from base of SB section.
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.22 DCFD and DCFDU

The DCFD directive allocates memory for word-aligned double-precision floating-point numbers, and defines the initial runtime contents of the memory. Double-precision numbers occupy two words and must be word aligned to be used in arithmetic operations.

DCFDU is the same, except that the memory alignment is arbitrary.

10.22.1 Syntax

{*label*} DCFD{U} *fpliteral*{,*fpliteral*}...

where:

fpliteral is a double-precision floating-point literal.

10.22.2 Usage

The assembler inserts up to three bytes of padding before the first defined number, if necessary, to achieve four-byte alignment.

Use DCFDU if you do not require alignment.

The word order used when converting *fpliteral* to internal form is controlled by the floating-point architecture selected. You cannot use DCFD or DCFDU if you select the `--fpu none` option.

The range for double-precision numbers is:

- Maximum 1.79769313486231571e+308.
- Minimum 2.22507385850720138e–308.

10.22.3 Examples

```
DCFD    1E308,-4E-100
DCFDU   10000,-.1,3.1E26
```

10.22.4 See also

Concepts

armasm User Guide:

- *Floating-point literals* on page 10-18.

Reference

- *DCFS and DCFSU* on page 10-27.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.23 DCFS and DCFSU

The DCFS directive allocates memory for word-aligned single-precision floating-point numbers, and defines the initial runtime contents of the memory. Single-precision numbers occupy one word and must be word aligned to be used in arithmetic operations.

DCFSU is the same, except that the memory alignment is arbitrary.

10.23.1 Syntax

{*label*} DCFS{U} *fpliteral*{,*fpliteral*}...

where:

fpliteral is a single-precision floating-point literal.

10.23.2 Usage

DCFS inserts up to three bytes of padding before the first defined number, if necessary to achieve four-byte alignment.

Use DCFSU if you do not require alignment.

The range for single-precision values is:

- Maximum 3.40282347e+38.
- Minimum 1.17549435e–38.

10.23.3 Examples

```
DCFS    1E3,-4E-9
DCFSU   1.0,-.1,3.1E6
```

10.23.4 See also

Concepts

armasm User Guide:

- *Floating-point literals* on page 10-18.

Reference

- *DCFD and DCFDU* on page 10-26.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.24 DCI

In A32 code, the DCI directive allocates one or more words of memory, aligned on four-byte boundaries, and defines the initial runtime contents of the memory.

In T32 code, the DCI directive allocates one or more halfwords of memory, aligned on two-byte boundaries, and defines the initial runtime contents of the memory.

10.24.1 Syntax

{*label*} DCI{.W} *expr*{,*expr*}

where:

expr is a numeric expression.

.W if present, indicates that four bytes must be inserted in T32 code.

10.24.2 Usage

The DCI directive is very like the DCD or DCW directives, but the location is marked as code instead of data. Use DCI when writing macros for new instructions not supported by the version of the assembler you are using.

In A32 code, DCI inserts up to three bytes of padding before the first defined word, if necessary, to achieve four-byte alignment. In T32 code, DCI inserts an initial byte of padding, if necessary, to achieve two-byte alignment.

You can use DCI to insert a bit pattern into the instruction stream. For example, use:

```
DCI 0x46c0
```

to insert the T32 operation MOV r8,r8.

10.24.3 Example macro

```
MACRO           ; this macro translates newinstr Rd,Rm
                ; to the appropriate machine code
newinst     $Rd,$Rm
DCI         0xe16f0f10 :OR: ($Rd:SHL:12) :OR: $Rm
MEND
```

10.24.4 32-bit T32 example

```
DCI.W   0xf3af8000   ; inserts 32-bit NOP, 2-byte aligned.
```

10.24.5 See also

Concepts

armasm User Guide:

- *Numeric expressions* on page 10-16.

Reference

- *DCD and DCDU* on page 10-24.
- *DCW and DCWU* on page 10-31.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.25 DCO and DCOU

The DCO directive allocates one or more sixteen-byte blocks of memory, aligned on four-byte boundaries, and defines the initial runtime contents of the memory.

DCOU is the same, except that the memory alignment is arbitrary.

10.25.1 Syntax

```
{label} DCO{U} {-}literal{,{-}literal}...
```

where:

literal is a 128-bit numeric literal.

The range of numbers permitted is 0 to $2^{128}-1$.

In addition to the characters normally permitted in a numeric literal, you can prefix *literal* with a minus sign. In this case, the range of numbers permitted is -2^{127} to -1.

The result of specifying *-n* is the same as the result of specifying $2^{128}-n$.

10.25.2 Usage

DCO inserts up to three bytes of padding before the first defined eight-byte block, if necessary, to achieve four-byte alignment.

Use DCOU if you do not require alignment.

10.25.3 See also

Concepts

armasm User Guide:

- *Numeric literals* on page 10-17.

Reference

- *DCB* on page 10-23.
- *DCD and DCDU* on page 10-24.
- *DCW and DCWU* on page 10-31.
- *DCO and DCOU*.
- *SPACE or FILL* on page 10-85.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.26 DCQ and DCQU

The DCQ directive allocates one or more eight-byte blocks of memory, aligned on four-byte boundaries, and defines the initial runtime contents of the memory.

DCQU is the same, except that the memory alignment is arbitrary.

10.26.1 Syntax

```
{label} DCQ{U} {-}literal{,{-}literal}...
```

```
{label} DCQ{U} expression
```

where:

literal is a 64-bit numeric literal.

The range of numbers permitted is 0 to $2^{64}-1$.

In addition to the characters normally permitted in a numeric literal, you can prefix *literal* with a minus sign. In this case, the range of numbers permitted is -2^{63} to -1.

The result of specifying *-n* is the same as the result of specifying $2^{64}-n$.

expression is a numeric expression that evaluates to a 64-bit integer.

10.26.2 Usage

DCQ inserts up to three bytes of padding before the first defined eight-byte block, if necessary, to achieve four-byte alignment.

Use DCQU if you do not require alignment.

10.26.3 Example

```
        AREA    MiscData, DATA, READWRITE
data    DCQ     -225,2_101      ; 2_101 means binary 101.
```

10.26.4 Incorrect example

```
number  EQU     2
        DCQU    number          ; DCQ and DCQU only accept literals not expressions.
```

10.26.5 See also

Concepts

armasm User Guide:

- *Numeric expressions* on page 10-16.
- *Numeric literals* on page 10-17.

Reference

- *DCB* on page 10-23.
- *DCD and DCDU* on page 10-24.
- *DCW and DCWU* on page 10-31.
- *DCO and DCOU* on page 10-29.
- *SPACE or FILL* on page 10-85.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.27 DCW and DCWU

The DCW directive allocates one or more halfwords of memory, aligned on two-byte boundaries, and defines the initial runtime contents of the memory.

DCWU is the same, except that the memory alignment is arbitrary.

10.27.1 Syntax

```
{label} DCW{U} expr{,expr}...
```

where:

expr is a numeric expression that evaluates to an integer in the range –32768 to 65535.

10.27.2 Usage

DCW inserts a byte of padding before the first defined halfword if necessary to achieve two-byte alignment.

Use DCWU if you do not require alignment.

10.27.3 Examples

```
data    DCW     -225,2*number   ; number must already be defined
        DCWU    number+4
```

10.27.4 See also

Concepts

armasm User Guide:

- *Numeric expressions* on page 10-16.

Reference

- *DCB* on page 10-23.
- *DCD and DCDU* on page 10-24.
- *DCQ and DCQU* on page 10-30.
- *SPACE or FILL* on page 10-85.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.28 END

The END directive informs the assembler that it has reached the end of a source file.

10.28.1 Syntax

```
END
```

10.28.2 Usage

Every ARM assembly language source file must end with END on a line by itself.

If the source file has been included in a parent file by a GET directive, the assembler returns to the parent file and continues assembly at the first line following the GET directive.

If END is reached in the top-level source file during the first pass without any errors, the second pass begins.

If END is reached in the top-level source file during the second pass, the assembler finishes the assembly and writes the appropriate output.

10.28.3 See also

Reference

- *GET or INCLUDE* on page 10-56.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.29 ENDFUNC or ENDP

The ENDFUNC directive marks the end of an AAPCS-conforming function. ENDP is a synonym for ENDFUNC.

10.29.1 See also

Reference

- *FUNCTION or PROC* on page 10-52.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.30 ENTRY

The ENTRY directive declares an entry point to a program.

10.30.1 Syntax

```
ENTRY
```

10.30.2 Usage

A program must have an entry point. You can specify an entry point in the following ways:

- Using the ENTRY directive in assembly language source code.
- Providing a main() function in C or C++ source code.
- Using the armlink --entry command line option.

You can declare more than one entry point in a program, although a source file cannot contain more than one ENTRY directive. For example, a program could contain multiple assembly language source files, each with an ENTRY directive. Or it could contain a C or C++ file with a main() function and one or more assembly source files with an ENTRY directive.

If the program contains multiple entry points, then you must select one of them. You do this by exporting the symbol for the ENTRY directive that you want to use as the entry point, then using the armlink --entry option to select the exported symbol.

10.30.3 Example

```
        AREA   ARMex, CODE, READONLY
        ENTRY      ; Entry point for the application
        EXPORT ep1 ; Export the symbol so the linker can find it in the object file
ep1
        ; code

        END
```

When you invoke armlink, if other entry points are declared in the program, then you must specify --entry=ep1, to select ep1.

10.30.4 See also

Concepts

- *Image entry points* on page 4-16 *in armlink User Guide.*

Reference

- *--entry* on page 2-49 *in armlink Reference Guide.*

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.31 EQU

The EQU directive gives a symbolic name to a numeric constant, a register-relative value or a PC-relative value. * is a synonym for EQU.

10.31.1 Syntax

name EQU *expr*{, *type*}

where:

name is the symbolic name to assign to the value.

expr is a register-relative address, a PC-relative address, an absolute address, or a 32-bit or 64-bit integer constant in A32/T32 or A64 code respectively.

type is optional. *type* can be any one of:

- ARM.
- THUMB.
- CODE32.
- CODE16.
- DATA.

You can use *type* only if *expr* is an absolute address. If *name* is exported, the *name* entry in the symbol table in the object file is marked as ARM, THUMB, CODE32, CODE16, or DATA, according to *type*. This can be used by the linker.

10.31.2 Usage

Use EQU to define constants. This is similar to the use of **#define** to define a constant in C.

10.31.3 Examples

```
abc EQU 2               ; assigns the value 2 to the symbol abc.
xyz EQU label+8         ; assigns the address (label+8) to the
                        ; symbol xyz.
fiq EQU 0x1C, CODE32    ; assigns the absolute address 0x1C to
                        ; the symbol fiq, and marks it as code
```

10.31.4 See also

Reference

- *KEEP* on page 10-63.
- *EXPORT or GLOBAL* on page 10-36.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.32 EXPORT or GLOBAL

The EXPORT directive declares a symbol that can be used by the linker to resolve symbol references in separate object and library files. GLOBAL is a synonym for EXPORT.

10.32.1 Syntax

```
EXPORT {[WEAK]}
EXPORT symbol {[SIZE=n]}
EXPORT symbol {[type{,set}]}
EXPORT symbol [attr{,type{,set}}{,SIZE=n}]
EXPORT symbol [WEAK{,attr}{,type{,set}}{,SIZE=n}]
```

where:

symbol is the symbol name to export. The symbol name is case-sensitive. If *symbol* is omitted, all symbols are exported.

WEAK *symbol* is only imported into other sources if no other source exports an alternative *symbol*. If [WEAK] is used without *symbol*, all exported symbols are weak.

attr can be any one of:

- DYNAMIC sets the ELF symbol visibility to STV_DEFAULT.
- PROTECTED sets the ELF symbol visibility to STV_PROTECTED.
- HIDDEN sets the ELF symbol visibility to STV_HIDDEN.
- INTERNAL sets the ELF symbol visibility to STV_INTERNAL.

type specifies the symbol type:

- DATA *symbol* is treated as data when the source is assembled and linked.
- CODE *symbol* is treated as code when the source is assembled and linked.
- ELFTYPE=*n* *symbol* is treated as a particular ELF symbol, as specified by the value of *n*, where *n* can be any number from 0 to 15.

If unspecified, the assembler determines the most appropriate *type*. Usually the assembler determines the correct type so there is no need to specify the *type*.

set specifies the instruction set:

- ARM *symbol* is treated as an A32 symbol.
- THUMB *symbol* is treated as a T32 symbol.

If unspecified, the assembler determines the most appropriate set.

n specifies the size and can be any 32-bit value. If the SIZE attribute is not specified, the assembler calculates the size:

- For PROC and FUNCTION symbols, the size is set to the size of the code until its ENDP or ENDFUNC.
- For other symbols, the size is the size of instruction or data on the same source line. If there is no instruction or data, the size is zero.

10.32.2 Usage

Use EXPORT to give code in other files access to symbols in the current file.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Use the [WEAK] attribute to inform the linker that a different instance of *symbol* takes precedence over this one, if a different one is available from another source. You can use the [WEAK] attribute with any of the symbol visibility attributes.

10.32.3 Example

```
        AREA    Example,CODE,READONLY
        EXPORT  DoAdd           ; Export the function name
                                ; to be used by external
                                ; modules.
DoAdd   ADD     r0,r0,r1
```

Symbol visibility can be overridden for duplicate exports. In the following example, the last EXPORT takes precedence for both binding and visibility:

```
        EXPORT  SymA[WEAK]      ; Export as weak-hidden
        EXPORT  SymA[DYNAMIC]   ; SymA becomes non-weak dynamic.
```

The following examples show the use of the SIZE attribute:

```
EXPORT symA [SIZE=4]
EXPORT symA [DATA, SIZE=4]
```

10.32.4 See also

Reference

- *IMPORT and EXTERN* on page 10-57.
- *ELF for the ARM Architecture ABI*
  http://infocenter/help/topic/com.arm.doc.ihi0044-/index.html.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.33 EXPORTAS

The EXPORTAS directive enables you to export a symbol to the object file, corresponding to a different symbol in the source file.

10.33.1 Syntax

EXPORTAS *symbol1*, *symbol2*

where:

symbol1 is the symbol name in the source file. *symbol1* must have been defined already. It can be any symbol, including an area name, a label, or a constant.

symbol2 is the symbol name you want to appear in the object file.

The symbol names are case-sensitive.

10.33.2 Usage

Use EXPORTAS to change a symbol in the object file without having to change every instance in the source file.

10.33.3 Examples

```
    AREA data1, DATA       ; starts a new area data1
    AREA data2, DATA       ; starts a new area data2
    EXPORTAS data2, data1  ; the section symbol referred to as data2
                           ; appears in the object file string table as data1.
one EQU  2
    EXPORTAS one, two
    EXPORT one             ; the symbol 'two' appears in the object
                           ; file's symbol table with the value 2.
```

10.33.4 See also

Reference

- *EXPORT or GLOBAL* on page 10-36.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.34 FIELD

The FIELD directive describes space within a storage map that has been defined using the MAP directive. # is a synonym for FIELD.

Note

This directive is not supported in A64 code in this release.

10.34.1 Syntax

{*label*} FIELD *expr*

where:

label is an optional label. If specified, *label* is assigned the value of the storage location counter, {VAR}. The storage location counter is then incremented by the value of *expr*.

expr is an expression that evaluates to the number of bytes to increment the storage counter.

10.34.2 Usage

If a storage map is set by a MAP directive that specifies a *base-register*, the base register is implicit in all labels defined by following FIELD directives, until the next MAP directive. These register-relative labels can be quoted in load and store instructions.

10.34.3 Examples

The following example shows how register-relative labels are defined using the MAP and FIELD directives.

```
    MAP     0,r9        ; set {VAR} to the address stored in R9
    FIELD   4           ; increment {VAR} by 4 bytes
Lab FIELD   4           ; set Lab to the address [R9 + 4]
                        ; and then increment {VAR} by 4 bytes
    LDR     r0,Lab      ; equivalent to LDR r0,[r9,#4]
```

When using the MAP and FIELD directives, you must ensure that the values are consistent in both passes. The following example shows a use of MAP and FIELD that cause inconsistent values for the symbol x. In the first pass sym is not defined, so x is at 0x04+R9. In the second pass, sym is defined, so x is at 0x00+R0. This example results in an assembly error.

```
    MAP 0, r0
    if :LNOT: :DEF: sym
      MAP 0, r9
      FIELD 4  ; x is at 0x04+R9 in first pass
    ENDIF
x   FIELD 4    ; x is at 0x00+R0 in second pass
sym LDR r0, x  ; inconsistent values for x results in assembly error
```

10.34.4 See also

Concepts

- *How the assembler works* on page 2-4 in *armasm User Guide*.
- *Directives that can be omitted in pass 2 of the assembler* on page 2-6 in *armasm User Guide*.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Reference

- *MAP* on page 10-69.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.35 FRAME ADDRESS

The FRAME ADDRESS directive describes how to calculate the canonical frame address for following instructions. You can only use it in functions with FUNCTION and ENDFUNC or PROC and ENDP directives.

10.35.1 Syntax

```
FRAME ADDRESS reg[,offset]
```

where:

reg is the register on which the canonical frame address is to be based. This is SP unless the function uses a separate frame pointer.

offset is the offset of the canonical frame address from *reg*. If *offset* is zero, you can omit it.

10.35.2 Usage

Use FRAME ADDRESS if your code alters which register the canonical frame address is based on, or if it changes the offset of the canonical frame address from the register. You must use FRAME ADDRESS immediately after the instruction that changes the calculation of the canonical frame address.

Note

- If your code uses a single instruction to save registers and alter the stack pointer, you can use FRAME PUSH instead of using both FRAME ADDRESS and FRAME SAVE.
- If your code uses a single instruction to load registers and alter the stack pointer, you can use FRAME POP instead of using both FRAME ADDRESS and FRAME RESTORE.

10.35.3 Example

```
_fn     FUNCTION        ; CFA (Canonical Frame Address) is value
                        ; of SP on entry to function
        PUSH    {r4,fp,ip,lr,pc}
        FRAME PUSH {r4,fp,ip,lr,pc}
        SUB     sp,sp,#4            ; CFA offset now changed
        FRAME ADDRESS sp,24         ; - so we correct it
        ADD     fp,sp,#20
        FRAME ADDRESS fp,4          ; New base register
        ; code using fp to base call-frame on, instead of SP
```

10.35.4 See also

Reference

- *FRAME POP* on page 10-42.
- *FRAME PUSH* on page 10-43.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.36 FRAME POP

Use the FRAME POP directive to inform the assembler when the callee reloads registers. You can only use it within functions with FUNCTION and ENDFUNC or PROC and ENDP directives.

You do not have to do this after the last instruction in a function.

10.36.1 Syntax

There are the following alternative syntaxes for FRAME POP:

```
FRAME POP {reglist}
FRAME POP {reglist},n
FRAME POP n
```

where:

reglist is a list of registers restored to the values they had on entry to the function. There must be at least one register in the list.

n is the number of bytes that the stack pointer moves.

10.36.2 Usage

FRAME POP is equivalent to a FRAME ADDRESS and a FRAME RESTORE directive. You can use it when a single instruction loads registers and alters the stack pointer.

You must use FRAME POP immediately after the instruction it refers to.

If *n* is not specified or is zero, the assembler calculates the new offset for the canonical frame address from {*reglist*}. It assumes that:

- Each ARM register popped occupies four bytes on the stack.
- Each floating-point single-precision register popped occupies four bytes on the stack, plus an extra four-byte word for each list.
- Each floating-point double-precision register popped occupies eight bytes on the stack, plus an extra four-byte word for each list.

10.36.3 See also

Reference

- *FRAME ADDRESS* on page 10-41.
- *FRAME RESTORE* on page 10-45.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.37 FRAME PUSH

Use the FRAME PUSH directive to inform the assembler when the callee saves registers, normally at function entry. You can only use it within functions with FUNCTION and ENDFUNC or PROC and ENDP directives.

10.37.1 Syntax

There are two alternative syntaxes for FRAME PUSH:

```
FRAME PUSH {reglist}
FRAME PUSH {reglist},n
FRAME PUSH n
```

where:

reglist is a list of registers stored consecutively below the canonical frame address. There must be at least one register in the list.

n is the number of bytes that the stack pointer moves.

10.37.2 Usage

FRAME PUSH is equivalent to a FRAME ADDRESS and a FRAME SAVE directive. You can use it when a single instruction saves registers and alters the stack pointer.

You must use FRAME PUSH immediately after the instruction it refers to.

If *n* is not specified or is zero, the assembler calculates the new offset for the canonical frame address from {*reglist*}. It assumes that:

- Each ARM register pushed occupies four bytes on the stack.
- Each floating-point single-precision register pushed occupies four bytes on the stack, plus an extra four-byte word for each list.
- Each floating-point double-precision register popped occupies eight bytes on the stack, plus an extra four-byte word for each list.

10.37.3 Example

```
p   PROC ; Canonical frame address is SP + 0
    EXPORT  p
    PUSH    {r4-r6,lr}
         ; SP has moved relative to the canonical frame address,
         ; and registers R4, R5, R6 and LR are now on the stack
    FRAME PUSH {r4-r6,lr}
         ; Equivalent to:
         ; FRAME ADDRESS    sp,16       ; 16 bytes in {R4-R6,LR}
         ; FRAME SAVE    {r4-r6,lr},-16
```

10.37.4 See also

Reference

- *FRAME ADDRESS* on page 10-41.
- *FRAME SAVE* on page 10-47.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.38 FRAME REGISTER

Use the FRAME REGISTER directive to maintain a record of the locations of function arguments held in registers. You can only use it within functions with FUNCTION and ENDFUNC or PROC and ENDP directives.

10.38.1 Syntax

```
FRAME REGISTER reg1,
     reg2
```

where:

reg1 is the register that held the argument on entry to the function.

reg2 is the register in which the value is preserved.

10.38.2 Usage

Use the FRAME REGISTER directive when you use a register to preserve an argument that was held in a different register on entry to a function.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.39 FRAME RESTORE

Use the FRAME RESTORE directive to inform the assembler that the contents of specified registers have been restored to the values they had on entry to the function. You can only use it within functions with FUNCTION and ENDFUNC or PROC and ENDP directives.

10.39.1 Syntax

```
FRAME RESTORE {reglist}
```

where:

reglist is a list of registers whose contents have been restored. There must be at least one register in the list.

10.39.2 Usage

Use FRAME RESTORE immediately after the callee reloads registers from the stack. You do not have to do this after the last instruction in a function.

reglist can contain integer registers or floating-point registers, but not both.

Note

If your code uses a single instruction to load registers and alter the stack pointer, you can use FRAME POP instead of using both FRAME RESTORE and FRAME ADDRESS.

10.39.3 See also

Reference

- *FUNCTION or PROC* on page 10-52.
- *ENDFUNC or ENDP* on page 10-33.
- *FRAME POP* on page 10-42.
- *FRAME ADDRESS* on page 10-41.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.40 FRAME RETURN ADDRESS

The FRAME RETURN ADDRESS directive provides for functions that use a register other than LR for their return address. You can only use it within functions with FUNCTION and ENDFUNC or PROC and ENDP directives.

Note

Any function that uses a register other than LR for its return address is not AAPCS compliant. Such a function must not be exported.

10.40.1 Syntax

```
FRAME RETURN ADDRESS reg
```

where:

reg is the register used for the return address.

10.40.2 Usage

Use the FRAME RETURN ADDRESS directive in any function that does not use LR for its return address. Otherwise, a debugger cannot backtrace through the function.

Use FRAME RETURN ADDRESS immediately after the FUNCTION or PROC directive that introduces the function.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.41 FRAME SAVE

The FRAME SAVE directive describes the location of saved register contents relative to the canonical frame address. You can only use it within functions with FUNCTION and ENDFUNC or PROC and ENDP directives.

10.41.1 Syntax

```
FRAME SAVE {reglist}, offset
```

where:

reglist is a list of registers stored consecutively starting at *offset* from the canonical frame address. There must be at least one register in the list.

10.41.2 Usage

Use FRAME SAVE immediately after the callee stores registers onto the stack.

reglist can include registers which are not required for backtracing. The assembler determines which registers it requires to record in the DWARF call frame information.

Note

If your code uses a single instruction to save registers and alter the stack pointer, you can use FRAME PUSH instead of using both FRAME SAVE and FRAME ADDRESS.

10.41.3 See also

Reference

- *FRAME PUSH* on page 10-43.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.42 FRAME STATE REMEMBER

The FRAME STATE REMEMBER directive saves the current information on how to calculate the canonical frame address and locations of saved register values. You can only use it within functions with FUNCTION and ENDFUNC or PROC and ENDP directives.

10.42.1 Syntax

```
FRAME STATE REMEMBER
```

10.42.2 Usage

During an inline exit sequence the information about calculation of canonical frame address and locations of saved register values can change. After the exit sequence another branch can continue using the same information as before. Use FRAME STATE REMEMBER to preserve this information, and FRAME STATE RESTORE to restore it.

These directives can be nested. Each FRAME STATE RESTORE directive must have a corresponding FRAME STATE REMEMBER directive.

10.42.3 Example

```
        ; function code
        FRAME STATE REMEMBER
            ; save frame state before in-line exit sequence
        POP     {r4-r6,pc}
            ; do not have to FRAME POP here, as control has
            ; transferred out of the function
        FRAME STATE RESTORE
            ; end of exit sequence, so restore state
exitB   ; code for exitB
        POP     {r4-r6,pc}
        ENDP
```

10.42.4 See also

Reference

- *FRAME STATE RESTORE* on page 10-49.
- *FUNCTION or PROC* on page 10-52.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.43 FRAME STATE RESTORE

The FRAME STATE RESTORE directive restores information about how to calculate the canonical frame address and locations of saved register values. You can only use it within functions with FUNCTION and ENDFUNC or PROC and ENDP directives.

10.43.1 Syntax

```
FRAME STATE RESTORE
```

10.43.2 See also

Reference

- *FRAME STATE REMEMBER* on page 10-48.
- *FUNCTION or PROC* on page 10-52.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.44 FRAME UNWIND ON

The `FRAME UNWIND ON` directive instructs the assembler to produce *unwind* tables for this and subsequent functions.

10.44.1 Syntax

```
FRAME UNWIND ON
```

10.44.2 Usage

You can use this directive outside functions. In this case, the assembler produces *unwind* tables for all following functions until it reaches a `FRAME UNWIND OFF` directive.

Note

A `FRAME UNWIND` directive is not sufficient to turn on exception table generation. Furthermore a `FRAME UNWIND` directive, without other `FRAME` directives, is not sufficient information for the assembler to generate the *unwind* information.

10.44.3 See also

Reference

- *--exceptions* on page 2-30.
- *--exceptions_unwind* on page 2-31.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.45 FRAME UNWIND OFF

The `FRAME UNWIND OFF` directive instructs the assembler to produce *nounwind* tables for this and subsequent functions.

10.45.1 Syntax

```
FRAME UNWIND OFF
```

10.45.2 Usage

You can use this directive outside functions. In this case, the assembler produces *nounwind* tables for all following functions until it reaches a `FRAME UNWIND ON` directive.

10.45.3 See also

Reference

- *--exceptions* on page 2-30.
- *--exceptions_unwind* on page 2-31.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.46 FUNCTION or PROC

The FUNCTION directive marks the start of a function. PROC is a synonym for FUNCTION.

10.46.1 Syntax

label FUNCTION [{*reglist1*} [, {*reglist2*}]]

where:

reglist1 is an optional list of callee-saved ARM registers. If *reglist1* is not present, and your debugger checks register usage, it assumes that the AAPCS is in use. If you use empty brackets, this informs the debugger that all ARM registers are caller-saved.

reglist2 is an optional list of callee-saved floating-point registers. If you use empty brackets, this informs the debugger that all floating-point registers are caller-saved.

10.46.2 Usage

Use FUNCTION to mark the start of functions. The assembler uses FUNCTION to identify the start of a function when producing DWARF call frame information for ELF.

FUNCTION sets the canonical frame address to be R13 (SP), and the frame state stack to be empty.

Each FUNCTION directive must have a matching ENDFUNC directive. You must not nest FUNCTION and ENDFUNC pairs, and they must not contain PROC or ENDP directives.

You can use the optional *reglist* parameters to inform the debugger about an alternative procedure call standard, if you are using your own. Not all debuggers support this feature. See your debugger documentation for details.

If you specify an empty *reglist*, using {}, this indicates that all registers for the function are caller-saved. Typically you do this when writing a reset vector where the values in all registers are unknown on execution. This avoids problems in a debugger if it tries to construct a backtrace from the values in the registers.

Note

FUNCTION does not automatically cause alignment to a word boundary (or halfword boundary for T32). Use ALIGN if necessary to ensure alignment, otherwise the call frame might not point to the start of the function.

10.46.3 Examples

```
        ALIGN      ; ensures alignment
dadd    FUNCTION   ; without the ALIGN directive, this might not be word-aligned
        EXPORT  dadd
        PUSH       {r4-r6,lr}    ; this line automatically word-aligned
        FRAME PUSH {r4-r6,lr}
        ; subroutine body
        POP        {r4-r6,pc}
        ENDFUNC
func6   PROC {r4-r8,r12},{D1-D3} ; non-AAPCS-conforming function
        ...
        ENDP
func7   FUNCTION {} ; another non-AAPCS-conforming function
        ...
        ENDFUNC
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.46.4 See also

Reference

- *FRAME ADDRESS* on page 10-41.
- *FRAME STATE RESTORE* on page 10-49.
- *ALIGN* on page 10-11.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.47 GBLA, GBLL, and GBLS

The GBLA directive declares a global arithmetic variable, and initializes its value to 0.

The GBLL directive declares a global logical variable, and initializes its value to {FALSE}.

The GBLS directive declares a global string variable and initializes its value to a null string, "".

10.47.1 Syntax

<gblx> variable

where:

<gblx> is one of GBLA, GBLL, or GBLS.

variable is the name of the variable. *variable* must be unique among symbols within a source file.

10.47.2 Usage

Using one of these directives for a variable that is already defined re-initializes the variable.

The scope of the variable is limited to the source file that contains it.

Set the value of the variable with a SETA, SETL, or SETS directive.

Global variables can also be set with the --predefine assembler command-line option.

10.47.3 Examples

Example 10-1 declares a variable objectsize, sets the value of objectsize to 0xFF, and then uses it later in a SPACE directive.

Example 10-1

```
            GBLA    objectsize    ; declare the variable name
objectsize  SETA    0xFF          ; set its value
            .
            .                     ; other code
            .
            SPACE   objectsize    ; quote the variable
```

Example 10-2 shows how to declare and set a variable when you invoke armasm. Use this when you want to set the value of a variable at assembly time. --pd is a synonym for --predefine.

Example 10-2

```
armasm --cpu=8-A.64 --predefine "objectsize SETA 0xFF" sourcefile -o objectfile
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.47.4 See also

Reference

- *SETA, SETL, and SETS* on page 10-83.
- *LCLA, LCLL, and LCLS* on page 10-64.
- *armasm command-line options* on page 2-3.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.48 GET or INCLUDE

The GET directive includes a file within the file being assembled. The included file is assembled at the location of the GET directive. INCLUDE is a synonym for GET.

10.48.1 Syntax

```
GET filename
```

where:

filename is the name of the file to be included in the assembly. The assembler accepts pathnames in either UNIX or MS-DOS format.

10.48.2 Usage

GET is useful for including macro definitions, EQUs, and storage maps in an assembly. When assembly of the included file is complete, assembly continues at the line following the GET directive.

By default the assembler searches the current place for included files. The current place is the directory where the calling file is located. Use the -i assembler command line option to add directories to the search path. File names and directory names containing spaces must not be enclosed in double quotes (" ").

The included file can contain additional GET directives to include other files.

If the included file is in a different directory from the current place, this becomes the current place until the end of the included file. The previous current place is then restored.

You cannot use GET to include object files.

10.48.3 Examples

```
AREA    Example, CODE, READONLY
GET     file1.s              ; includes file1 if it exists
                             ; in the current place.
GET     c:\project\file2.s   ; includes file2
GET     c:\Program files\file3.s  ; space is permitted
```

10.48.4 See also

Reference

- *INCBIN* on page 10-59.
- *Nesting directives* on page 10-5.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.49 IMPORT and EXTERN

These directives provide the assembler with a name that is not defined in the current assembly.

10.49.1 Syntax

directive symbol {[SIZE=*n*]}

directive symbol {[*type*]}

directive symbol [*attr*{,*type*}{,SIZE=*n*}]

directive symbol [WEAK{,*attr*}{,*type*}{,SIZE=*n*}]

where:

directive can be either:

- IMPORT imports the symbol unconditionally.
- EXTERN imports the symbol only if it is referred to in the current assembly.

symbol is a symbol name defined in a separately assembled source file, object file, or library. The symbol name is case-sensitive.

WEAK prevents the linker generating an error message if the symbol is not defined elsewhere. It also prevents the linker searching libraries that are not already included.

attr can be any one of:

- DYNAMIC sets the ELF symbol visibility to STV_DEFAULT.
- PROTECTED sets the ELF symbol visibility to STV_PROTECTED.
- HIDDEN sets the ELF symbol visibility to STV_HIDDEN.
- INTERNAL sets the ELF symbol visibility to STV_INTERNAL.

type specifies the symbol type:

- DATA *symbol* is treated as data when the source is assembled and linked.
- CODE *symbol* is treated as code when the source is assembled and linked.
- ELFTYPE=*n* *symbol* is treated as a particular ELF symbol, as specified by the value of *n*, where *n* can be any number from 0 to 15.

If unspecified, the linker determines the most appropriate type.

n specifies the size and can be any 32-bit value. If the SIZE attribute is not specified, the assembler calculates the size:

- For PROC and FUNCTION symbols, the size is set to the size of the code until its ENDP or ENDFUNC.
- For other symbols, the size is the size of instruction or data on the same source line. If there is no instruction or data, the size is zero.

10.49.2 Usage

The name is resolved at link time to a symbol defined in a separate object file. The symbol is treated as a program address. If [WEAK] is not specified, the linker generates an error if no corresponding symbol is found at link time.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

If [WEAK] is specified and no corresponding symbol is found at link time:

- If the reference is the destination of a B or BL instruction, the value of the symbol is taken as the address of the following instruction. This makes the B or BL instruction effectively a NOP.
- Otherwise, the value of the symbol is taken as zero.

10.49.3 Example

The example tests to see if the C++ library has been linked, and branches conditionally on the result.

```
AREA    Example, CODE, READONLY
EXTERN  __CPP_INITIALIZE[WEAK]  ; If C++ library linked, gets the address of
                                ; __CPP_INITIALIZE function.
LDR     r0,=__CPP_INITIALIZE    ; If not linked, address is zeroed.
CMP     r0,#0                   ; Test if zero.
BEQ     nocplusplus             ; Branch on the result.
```

The following examples show the use of the SIZE attribute:

```
EXTERN symA [SIZE=4]
EXTERN symA [DATA, SIZE=4]
```

10.49.4 See also

Reference

- *ELF for the ARM Architecture* http://infocenter.arm.com/help/topic/com.arm.doc.ihi0044-/index.html.
- *EXPORT or GLOBAL* on page 10-36.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.50 INCBIN

The INCBIN directive includes a file within the file being assembled. The file is included as it is, without being assembled.

10.50.1 Syntax

INCBIN *filename*

where:

filename is the name of the file to be included in the assembly. The assembler accepts pathnames in either UNIX or MS-DOS format.

10.50.2 Usage

You can use INCBIN to include executable files, literals, or any arbitrary data. The contents of the file are added to the current ELF section, byte for byte, without being interpreted in any way. Assembly continues at the line following the INCBIN directive.

By default, the assembler searches the current place for included files. The current place is the directory where the calling file is located. Use the -i assembler command line option to add directories to the search path. File names and directory names containing spaces must not be enclosed in double quotes (" ").

10.50.3 Example

```
AREA    Example, CODE, READONLY
INCBIN  file1.dat               ; includes file1 if it
                                ; exists in the
                                ; current place.
INCBIN  c:\project\file2.txt    ; includes file2
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.51 IF, ELSE, ENDIF, and ELIF

The IF directive introduces a condition that controls whether to assemble a sequence of instructions and directives. [is a synonym for IF.

The ELSE directive marks the beginning of a sequence of instructions or directives that you want to be assembled if the preceding condition fails. | is a synonym for ELSE.

The ENDIF directive marks the end of a sequence of instructions or directives that you want to be conditionally assembled.] is a synonym for ENDIF.

The ELIF directive creates a structure equivalent to ELSE IF, without the requirement for nesting or repeating the condition.

10.51.1 Syntax

```
IF logical-expression
      … ; code
{ELSE
      … ; code}
ENDIF
```

where:

logical-expression

is an expression that evaluates to either {TRUE} or {FALSE}.

10.51.2 Usage

Use IF with ENDIF, and optionally with ELSE, for sequences of instructions or directives that are only to be assembled or acted on under a specified condition.

IF...ENDIF conditions can be nested.

10.51.3 Using ELIF

Without using ELIF, you can construct a nested set of conditional instructions like this:

```
IF logical-expression
    instructions
ELSE
    IF logical-expression2
        instructions
    ELSE
        IF logical-expression3
            instructions
        ENDIF
    ENDIF
ENDIF
```

A nested structure like this can be nested up to 256 levels deep.

You can write the same structure more simply using ELIF:

```
IF logical-expression
    instructions
ELIF logical-expression2
    instructions
ELIF logical-expression3
    instructions
ENDIF
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

This structure only adds one to the current nesting depth, for the IF...ENDIF pair.

10.51.4 Examples

Example 10-3 assembles the first set of instructions if NEWVERSION is defined, or the alternative set otherwise.

Example 10-3 Assembly conditional on a variable being defined

```
IF :DEF:NEWVERSION
    ; first set of instructions or directives
ELSE
    ; alternative set of instructions or directives
ENDIF
```

Invoking armasm as follows defines NEWVERSION, so the first set of instructions and directives are assembled:

```
armasm --cpu=8-A.64 --predefine "NEWVERSION SETL {TRUE}" test.s
```

Invoking armasm as follows leaves NEWVERSION undefined, so the second set of instructions and directives are assembled:

```
armasm --cpu=8-A.64 test.s
```

Example 10-4 assembles the first set of instructions if NEWVERSION has the value {TRUE}, or the alternative set otherwise.

Example 10-4 Assembly conditional on a variable value

```
IF NEWVERSION = {TRUE}
    ; first set of instructions or directives
ELSE
    ; alternative set of instructions or directives
ENDIF
```

Invoking armasm as follows causes the first set of instructions and directives to be assembled:

```
armasm --cpu=8-A.64 --predefine "NEWVERSION SETL {TRUE}" test.s
```

Invoking armasm as follows causes the second set of instructions and directives to be assembled:

```
armasm --cpu=8-A.64 --predefine "NEWVERSION SETL {FALSE}" test.s
```

10.51.5 See also

Concepts

armasm User Guide:

- *Relational operators* on page 10-27.

Reference

- *Using ELIF* on page 10-60.
- *Nesting directives* on page 10-5.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.52 INFO

The INFO directive supports diagnostic generation on either pass of the assembly.

! is very similar to INFO, but has less detailed reporting.

10.52.1 Syntax

INFO *numeric-expression*, *string-expression*{, *severity*}

where:

numeric-expression

is a numeric expression that is evaluated during assembly. If the expression evaluates to zero:

- No action is taken during pass one.
- *string-expression* is printed as a warning during pass two if *severity* is 1.
- *string-expression* is printed as a message during pass two if *severity* is 0 or not specified.

If the expression does not evaluate to zero:

- *string-expression* is printed as an error message and the assembly fails irrespective of whether *severity* is specified or not (non-zero values for *severity* are reserved in this case).

string-expression

is an expression that evaluates to a string.

severity

is an optional number that controls the severity of the message. Its value can be either 0 or 1. All other values are reserved.

10.52.2 Usage

INFO provides a flexible means of creating custom error messages.

10.52.3 Examples

```
INFO    0, "Version 1.0"
IF endofdata <= label1
    INFO    4, "Data overrun at label1"
ENDIF
```

10.52.4 See also

Concepts

armasm User Guide:

- *Numeric expressions* on page 10-16.
- *String expressions* on page 10-14.

Reference

- *ASSERT* on page 10-17.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.53 KEEP

The KEEP directive instructs the assembler to retain named local labels in the symbol table in the ELF object file.

10.53.1 Syntax

KEEP {*label*}

where:

label is the name of the local label to keep. If *label* is not specified, all named local labels are kept except register-relative labels.

10.53.2 Usage

By default, the only labels that the assembler describes in its output object file are:

- Exported labels.
- Labels that are relocated against.

Use KEEP to preserve local labels. This can help when debugging. Kept labels appear in the ARM debuggers and in linker map files.

KEEP cannot preserve register-relative labels or numeric local labels.

10.53.3 Example

```
label   ADC     r2,r3,r4
        KEEP    label       ; makes label available to debuggers
        ADD     r2,r2,r5
```

10.53.4 See also

Reference

- *MAP* on page 10-69.

Concepts

- *Numeric local labels* on page 10-12.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.54 LCLA, LCLL, and LCLS

The LCLA directive declares a local arithmetic variable, and initializes its value to 0.

The LCLL directive declares a local logical variable, and initializes its value to {FALSE}.

The LCLS directive declares a local string variable, and initializes its value to a null string, "".

10.54.1 Syntax

<lclx> variable

where:

<lclx> is one of LCLA, LCLL, or LCLS.

variable is the name of the variable. *variable* must be unique within the macro that contains it.

10.54.2 Usage

Using one of these directives for a variable that is already defined re-initializes the variable.

The scope of the variable is limited to a particular instantiation of the macro that contains it.

Set the value of the variable with a SETA, SETL, or SETS directive.

10.54.3 Example

```
        MACRO                           ; Declare a macro
$label  message $a                      ; Macro prototype line
        LCLS    err                     ; Declare local string
                                        ; variable err.
err     SETS    "error no: "            ; Set value of err
$label  ; code
        INFO    0, "err":CC::STR:$a     ; Use string
        MEND
```

10.54.4 See also

Reference

- *SETA, SETL, and SETS* on page 10-83.
- *MACRO and MEND* on page 10-66.
- *GBLA, GBLL, and GBLS* on page 10-54.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.55 LTORG

The LTORG directive instructs the assembler to assemble the current literal pool immediately.

10.55.1 Syntax

```
LTORG
```

10.55.2 Usage

The assembler assembles the current literal pool at the end of every code section. The end of a code section is determined by the AREA directive at the beginning of the following section, or the end of the assembly.

These default literal pools can sometimes be out of range of some LDR, VLDR, and WLDR pseudo-instructions. Use LTORG to ensure that a literal pool is assembled within range.

Large programs can require several literal pools. Place LTORG directives after unconditional branches or subroutine return instructions so that the processor does not attempt to execute the constants as instructions.

The assembler word-aligns data in literal pools.

10.55.3 Example

```
        AREA    Example, CODE, READONLY
start   BL      func1
func1                           ; function body
        ; code
        LDR     r1,=0x55555555  ; => LDR R1, [pc, #offset to Literal Pool 1]
        ; code
        MOV     pc,lr           ; end function
        LTORG                   ; Literal Pool 1 contains literal &55555555.
data    SPACE   4200            ; Clears 4200 bytes of memory,
                                ; starting at current location.
        END                     ; Default literal pool is empty.
```

10.55.4 See also

Reference

- *LDR pseudo-instruction* on page 3-86.
- *VLDR pseudo-instruction* on page 4-62.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.56 MACRO and MEND

The MACRO directive marks the start of the definition of a macro. Macro expansion terminates at the MEND directive.

10.56.1 Syntax

These two directives define a macro. The syntax is:

```
          MACRO
{$label}  macroname{$cond} {$parameter{,$parameter}...}
          ; code
          MEND
```

where:

$label is a parameter that is substituted with a symbol given when the macro is invoked. The symbol is usually a label.

macroname is the name of the macro. It must not begin with an instruction or directive name.

$cond is a special parameter designed to contain a condition code. Values other than valid condition codes are permitted.

$parameter is a parameter that is substituted when the macro is invoked. A default value for a parameter can be set using this format:

$parameter="default value"

Double quotes must be used if there are any spaces within, or at either end of, the default value.

10.56.2 Usage

If you start any WHILE...WEND loops or IF...ENDIF conditions within a macro, they must be closed before the MEND directive is reached. You can use MEXIT to enable an early exit from a macro, for example, from within a loop.

Within the macro body, parameters such as *$label*, *$parameter* or *$cond* can be used in the same way as other variables. They are given new values each time the macro is invoked. Parameters must begin with $ to distinguish them from ordinary symbols. Any number of parameters can be used.

$label is optional. It is useful if the macro defines internal labels. It is treated as a parameter to the macro. It does not necessarily represent the first instruction in the macro expansion. The macro defines the locations of any labels.

Use | as the argument to use the default value of a parameter. An empty string is used if the argument is omitted.

In a macro that uses several internal labels, it is useful to define each internal label as the base label with a different suffix.

Use a dot between a parameter and following text, or a following parameter, if a space is not required in the expansion. Do not use a dot between preceding text and a parameter.

You can use the *$cond* parameter for condition codes. Use the unary operator :REVERSE_CC: to find the inverse condition code, and :CC_ENCODING: to find the 4-bit encoding of the condition code.

Macros define the scope of local variables.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Macros can be nested.

10.56.3 Examples

```
 ; macro definition
                MACRO                   ; start macro definition
$label          xmac    $p1,$p2
                ; code
$label.loop1    ; code
                ; code
                BGE     $label.loop1
$label.loop2    ; code
                BL      $p1
                BGT     $label.loop2
                ; code
                ADR     $p2
                ; code
                MEND                    ; end macro definition
 ; macro invocation
abc             xmac    subr1,de        ; invoke macro
                ; code                  ; this is what is
abcloop1        ; code                  ; is produced when
                ; code                  ; the xmac macro is
                BGE     abcloop1        ; expanded
abcloop2        ; code
                BL      subr1
                BGT     abcloop2
                ; code
                ADR     de
                ; code
```

Using a macro to produce assembly-time diagnostics:

```
        MACRO                           ; Macro definition
        diagnose  $param1="default"     ; This macro produces
        INFO      0,"$param1"           ; assembly-time diagnostics
        MEND                            ; (on second assembly pass)
 ; macro expansion
        diagnose            ; Prints blank line at assembly-time
        diagnose "hello"    ; Prints "hello" at assembly-time
        diagnose |          ; Prints "default" at assembly-time
```

Note

When variables are also being passed in as arguments, use of | might leave some variables unsubstituted. To workaround this, define the | in a LCLS or GBLS variable and pass this variable as an argument instead of |. For example:

```
        MACRO               ; Macro definition
        m2 $a,$b=r1,$c      ; The default value for $b is r1
        add $a,$b,$c        ; The macro adds $b and $c and puts result in $a
        MEND                ; Macro end

        MACRO               ; Macro definition
        m1 $a,$b            ; This macro adds $b to r1 and puts result in $a
        LCLS def            ; Declare a local string variable for |
def     SETS "|"            ; Define |
        m2 $a,$def,$b       ; Invoke macro m2 with $def instead of |
                            ; to use the default value for the second argument.
        MEND                ; Macro end
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.56.4 Conditional macro example

```
        AREA    codx, CODE, READONLY

; macro definition

        MACRO
        Return$cond
        [ {ARCHITECTURE} <> "4"
          BX$cond lr
          |
          MOV$cond pc,lr
        ]
        MEND

; macro invocation

fun     PROC
        CMP     r0,#0
        MOVEQ   r0,#1
        ReturnEQ
        MOV     r0,#0
        Return
        ENDP

        END
```

10.56.5 See also

Concepts

armasm User Guide:

- *Use of macros* on page 7-31.
- *Assembly time substitution of variables* on page 10-6.

Reference

- *MEXIT* on page 10-70.
- *Nesting directives* on page 10-5.
- *GBLA, GBLL, and GBLS* on page 10-54.
- *LCLA, LCLL, and LCLS* on page 10-64.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.57 MAP

The MAP directive sets the origin of a storage map to a specified address. The storage-map location counter, {VAR}, is set to the same address. ^ is a synonym for MAP.

Note

This directive is not supported in A64 code in this release.

10.57.1 Syntax

```
MAP expr{,base-register}
```

where:

expr is a numeric or PC-relative expression:

- If *base-register* is not specified, *expr* evaluates to the address where the storage map starts. The storage map location counter is set to this address.
- If *expr* is PC-relative, you must have defined the label before you use it in the map. The map requires the definition of the label during the first pass of the assembler.

base-register

specifies a register. If *base-register* is specified, the address where the storage map starts is the sum of *expr*, and the value in *base-register* at runtime.

10.57.2 Usage

Use the MAP directive in combination with the FIELD directive to describe a storage map.

Specify *base-register* to define register-relative labels. The base register becomes implicit in all labels defined by following FIELD directives, until the next MAP directive. The register-relative labels can be used in load and store instructions.

The MAP directive can be used any number of times to define multiple storage maps.

The {VAR} counter is set to zero before the first MAP directive is used.

10.57.3 Examples

```
MAP     0,r9
MAP     0xff,r9
```

10.57.4 See also

Concepts

- *How the assembler works* on page 2-4 in *armasm User Guide*.
- *Directives that can be omitted in pass 2 of the assembler* on page 2-6 in *armasm User Guide*.

Reference

- *FIELD* on page 10-39.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.58 MEXIT

The MEXIT directive exits a macro definition before the end.

10.58.1 Usage

Use MEXIT when you require an exit from within the body of a macro. Any unclosed WHILE...WEND loops or IF...ENDIF conditions within the body of the macro are closed by the assembler before the macro is exited.

10.58.2 Example

```
        MACRO
$abc    example abc     $param1,$param2
        ; code
        WHILE condition1
            ; code
            IF condition2
                ; code
                MEXIT
            ELSE
                ; code
            ENDIF
        WEND
        ; code
        MEND
```

10.58.3 See also

Reference

- *MACRO and MEND* on page 10-66.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.59 NOFP

The NOFP directive ensures that there are no floating-point instructions in an assembly language source file.

10.59.1 Syntax

```
NOFP
```

10.59.2 Usage

Use NOFP to ensure that no floating-point instructions are used in situations where there is no support for floating-point instructions either in software or in target hardware.

If a floating-point instruction occurs after the NOFP directive, an `Unknown opcode` error is generated and the assembly fails.

If a NOFP directive occurs after a floating-point instruction, the assembler generates the error:

```
Too late to ban floating point instructions
```

and the assembly fails.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.60 OPT

The OPT directive sets listing options from within the source code.

10.60.1 Syntax

OPT *n*

where:

n is the OPT directive setting. Table 10-2 lists valid settings.

Table 10-2 OPT directive settings

OPT *n*	Effect
1	Turns on normal listing.
2	Turns off normal listing.
4	Page throw. Issues an immediate form feed and starts a new page.
8	Resets the line number counter to zero.
16	Turns on listing for SET, GBL and LCL directives.
32	Turns off listing for SET, GBL and LCL directives.
64	Turns on listing of macro expansions.
128	Turns off listing of macro expansions.
256	Turns on listing of macro invocations.
512	Turns off listing of macro invocations.
1024	Turns on the first pass listing.
2048	Turns off the first pass listing.
4096	Turns on listing of conditional directives.
8192	Turns off listing of conditional directives.
16384	Turns on listing of MEND directives.
32768	Turns off listing of MEND directives.

10.60.2 Usage

Specify the --list= assembler option to turn on listing.

By default the --list= option produces a normal listing that includes variable declarations, macro expansions, call-conditioned directives, and MEND directives. The listing is produced on the second pass only. Use the OPT directive to modify the default listing options from within your code.

You can use OPT to format code listings. For example, you can specify a new page before functions and sections.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.60.3 Example

```
        AREA    Example, CODE, READONLY
start   ; code
        ; code
        BL      func1
        ; code
        OPT 4               ; places a page break before func1
func1   ; code
```

10.60.4 See also

Reference

- *--list* on page 2-41.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.61 QN, DN, and SN

The QN directive defines a name for a specified 128-bit extension register.

The DN directive defines a name for a specified 64-bit extension register.

The SN directive defines a name for a specified single-precision floating-point register.

10.61.1 Syntax

name directive expr{.*type*}{[*x*]}

where:

directive is QN, DN, or SN.

name is the name to be assigned to the extension register. *name* cannot be the same as any of the predefined names.

expr Can be:

- An expression that evaluates to a number in the range:
  - 0-15 if you are using QN in A32/T32 Advanced SIMD code.
  - 0-31 otherwise.
- A predefined register name, or a register name that has already been defined in a previous directive.

type is any Advanced SIMD or floating-point datatype.

[*x*] is only available for Advanced SIMD code. [*x*] is a scalar index into a register.

type and [*x*] are *Extended notation*.

10.61.2 Usage

Use QN, DN, or SN to allocate convenient names to extension registers, to help you to remember what you use each one for.

Note

Avoid conflicting uses of the same register under different names.

You cannot specify a vector length in a DN or SN directive.

10.61.3 Examples

```
energy  DN  6   ; defines energy as a symbol for
                ; floating-point double-precision register 6
mass    SN  16  ; defines mass as a symbol for
                ; floating-point single-precision register 16
```

10.61.4 Extended notation examples

```
varA    DN      d1.U16
varB    DN      d2.U16
varC    DN      d3.U16
        VADD    varA,varB,varC       ; VADD.U16 d1,d2,d3
index   DN      d4.U16[0]
result  QN      q5.I32
        VMULL   result,varA,index    ; VMULL.U16 q5,d1,d3[2]
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.61.5 See also

Reference

armasm User Guide:

- *Predeclared core register names in AArch32 state* on page 4-8.
- *Predeclared core register names in AArch64 state* on page 5-6.
- *Predeclared extension register names in AArch32 state* on page 4-9.
- *Predeclared extension register names in AArch64 state* on page 5-7.
- *Extended notation in A32/T32 code* on page 11-25.
- *Extended notation examples* on page 10-74.
- *Advanced SIMD and floating-point data types in A32/T32 instructions* on page 11-17.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.62 RELOC

The RELOC directive explicitly encodes an ELF relocation in an object file.

10.62.1 Syntax

RELOC *n*, *symbol*

RELOC *n*

where:

n must be an integer in the range 0 to 255, or 0 to 2^{32}–1 in A32/T32 or A64 code respectively. In A32/T32 code, you can alternatively specify one of the relocation names defined in the Application Binary Interface for the ARM Architecture.

symbol can be any PC-relative label.

10.62.2 Usage

Use RELOC *n*, *symbol* to create a relocation with respect to the address labeled by *symbol*.

If used immediately after an instruction, RELOC results in a relocation at that instruction. If used immediately after a DCB, DCW, or DCD, or any other data generating directive, RELOC results in a relocation at the start of the data. Any addend to be applied must be encoded in the instruction or in the data.

If the assembler has already emitted a relocation at that place, the relocation is updated with the details in the RELOC directive, for example:

```
DCD     sym2 ; R_ARM_ABS32 to sym32
RELOC   55   ; ... makes it R_ARM_ABS32_NOI
```

RELOC is faulted in all other cases, for example, after any non-data generating directive, LTORG, ALIGN, or as the first thing in an AREA.

Use RELOC *n* to create a relocation with respect to the anonymous symbol, that is, symbol 0 of the symbol table. If you use RELOC *n* without a preceding assembler generated relocation, the relocation is with respect to the anonymous symbol.

10.62.3 Examples

```
IMPORT  impsym
LDR     r0,[pc,#-8]
RELOC   4, impsym
DCD     0
RELOC   2, sym
DCD     0,1,2,3,4       ; the final word is relocated
RELOC   38,sym2         ; R_ARM_TARGET1
DCD     impsym
RELOC   R_ARM_TARGET1   ; relocation code 38
```

10.62.4 See also

Reference

- Application Binary Interface for the ARM Architecture
  http://infocenter.arm.com/help/topic/com.arm.doc.subset.swdev.abi/index.html.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.63 REQUIRE

The REQUIRE directive specifies a dependency between sections.

10.63.1 Syntax

```
REQUIRE label
```

where:

label is the name of the required label.

10.63.2 Usage

Use REQUIRE to ensure that a related section is included, even if it is not directly called. If the section containing the REQUIRE directive is included in a link, the linker also includes the section containing the definition of the specified label.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.64 REQUIRE8 and PRESERVE8

The REQUIRE8 directive specifies that the current file requires eight-byte alignment of the stack. It sets the REQ8 build attribute to inform the linker.

The PRESERVE8 directive specifies that the current file preserves eight-byte alignment of the stack. It sets the PRES8 build attribute to inform the linker.

The linker checks that any code that requires eight-byte alignment of the stack is only called, directly or indirectly, by code that preserves eight-byte alignment of the stack.

10.64.1 Syntax

```
REQUIRE8 {bool}
```

```
PRESERVE8 {bool}
```

where:

bool is an optional Boolean constant, either {TRUE} or {FALSE}.

10.64.2 Usage

Where required, if your code preserves eight-byte alignment of the stack, use PRESERVE8 to set the PRES8 build attribute on your file. If your code does not preserve eight-byte alignment of the stack, use PRESERVE8 {FALSE} to ensure that the PRES8 build attribute is not set. If there are multiple REQUIRE8 or PRESERVE8 directives in a file, the assembler uses the value of the last directive.

Note

If you omit both PRESERVE8 and PRESERVE8 {FALSE}, the assembler decides whether to set the PRES8 build attribute or not, by examining instructions that modify the SP. ARM recommends that you specify PRESERVE8 explicitly.

You can enable a warning with:

```
armasm --diag_warning 1546
```

This gives you warnings like:

```
"test.s", line 37: Warning: A1546W: Stack pointer update potentially
                                breaks 8 byte stack alignment
       37 00000044         STMFD    sp!,{r2,r3,lr}
```

10.64.3 Examples

```
REQUIRE8
REQUIRE8     {TRUE}      ; equivalent to REQUIRE8
REQUIRE8     {FALSE}     ; equivalent to absence of REQUIRE8
PRESERVE8    {TRUE}      ; equivalent to PRESERVE8
PRESERVE8    {FALSE}     ; NOT exactly equivalent to absence of PRESERVE8
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.64.4 See also

Concepts

- *8 Byte Stack Alignment*
  http://infocenter.arm.com/help/topic/com.arm.doc.faqs/ka4127.html.

Reference

- *armasm command-line options* on page 2-3.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.65 RLIST

The RLIST (register list) directive gives a name to a set of general-purpose registers in A32/T32 code.

10.65.1 Syntax

name RLIST {*list-of-registers*}

where:

name is the name to be given to the set of registers. *name* cannot be the same as any of the predefined names.

list-of-registers
is a comma-delimited list of register names and register ranges. The register list must be enclosed in braces.

10.65.2 Usage

Use RLIST to give a name to a set of registers to be transferred by the LDM or STM instructions.

LDM and STM always put the lowest physical register numbers at the lowest address in memory, regardless of the order they are supplied to the LDM or STM instruction. If you have defined your own symbolic register names it can be less apparent that a register list is not in increasing register order.

Use the --diag_warning 1206 assembler option to ensure that the registers in a register list are supplied in increasing register order. If registers are not supplied in increasing register order, a warning is issued.

10.65.3 Example

```
Context RLIST   {r0-r6,r8,r10-r12,pc}
```

10.65.4 See also

Reference

armasm User Guide:

- *Predeclared core register names in AArch32 state* on page 4-8.
- *Predeclared extension register names in AArch32 state* on page 4-9.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.66 RN

The RN directive defines a name for a specified register in A32/T32 code.

10.66.1 Syntax

name RN *expr*

where:

name is the name to be assigned to the register. *name* cannot be the same as any of the predefined names.

expr evaluates to a register number from 0 to 15.

10.66.2 Usage

Use RN to allocate convenient names to registers, to help you to remember what you use each register for. Be careful to avoid conflicting uses of the same register under different names. In A64 code, use WN or XN instead.

10.66.3 Examples

```
regname     RN  11  ; defines regname for register 11
sqr4        RN  r6  ; defines sqr4 for register 6
```

10.66.4 See also

Reference

armasm User Guide:

- *Predeclared core register names in AArch32 state* on page 4-8.
- *Predeclared extension register names in AArch32 state* on page 4-9.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.67 ROUT

The ROUT directive marks the boundaries of the scope of numeric local labels.

10.67.1 Syntax

{*name*} ROUT

where:

name is the name to be assigned to the scope.

10.67.2 Usage

Use the ROUT directive to limit the scope of numeric local labels. This makes it easier for you to avoid referring to a wrong label by accident. The scope of numeric local labels is the whole area if there are no ROUT directives in it.

Use the *name* option to ensure that each reference is to the correct numeric local label. If the name of a label or a reference to a label does not match the preceding ROUT directive, the assembler generates an error message and the assembly fails.

10.67.3 Example

```
            ; code
routineA    ROUT            ; ROUT is not necessarily a routine
            ; code
3routineA   ; code          ; this label is checked
            ; code
            BEQ     %4routineA   ; this reference is checked
            ; code
            BGE     %3      ; refers to 3 above, but not checked
            ; code
4routineA   ; code          ; this label is checked
            ; code
otherstuff  ROUT            ; start of next scope
```

10.67.4 See also

Concepts

armasm User Guide:

- *Numeric local labels* on page 10-12.

Reference

- *AREA* on page 10-13.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.68 SETA, SETL, and SETS

The SETA directive sets the value of a local or global arithmetic variable.

The SETL directive sets the value of a local or global logical variable.

The SETS directive sets the value of a local or global string variable.

10.68.1 Syntax

variable <*setx*> *expr*

where:

<*setx*> is one of SETA, SETL, or SETS.

variable is the name of a variable declared by a GBLA, GBLL, GBLS, LCLA, LCLL, or LCLS directive.

expr is an expression that is:

- Numeric, for SETA.
- Logical, for SETL.
- String, for SETS.

10.68.2 Usage

You must declare *variable* using a global or local declaration directive before using one of these directives.

You can also predefine variable names with the `--predefine` armasm command line option.

10.68.3 Restrictions

The value you can specify using a SETA directive is limited to 32 bits. If you exceed this limit, the assembler reports an error. A possible workaround in A64 code is to use an EQU directive instead of SETA, although EQU defines a constant, whereas GBLA and SETA define a variable.

For example, replace the following code:

```
                GBLA    MyAddress
MyAddress       SETA    0x0000008000000000
```

with:

```
MyAddress       EQU     0x0000008000000000
```

10.68.4 Examples

```
                GBLA    VersionNumber
VersionNumber   SETA    21
                GBLL    Debug
Debug           SETL    {TRUE}
                GBLS    VersionString
VersionString   SETS    "Version 1.0"
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.68.5 See also

Concepts

armasm User Guide:

- *Numeric expressions* on page 10-16.
- *Logical expressions* on page 10-19.
- *String expressions* on page 10-14.

Reference

- *armasm command-line options* on page 2-3.
- *LCLA, LCLL, and LCLS* on page 10-64.
- *GBLA, GBLL, and GBLS* on page 10-54.
- *EQU* on page 10-35.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.69 SPACE or FILL

The SPACE directive reserves a zeroed block of memory. % is a synonym for SPACE.

The FILL directive reserves a block of memory to fill with the given value.

10.69.1 Syntax

{*label*} SPACE *expr*

{*label*} FILL *expr*{,*value*{,*valuesize*}}

where:

label is an optional label.

expr evaluates to the number of bytes to fill or zero.

value evaluates to the value to fill the reserved bytes with. *value* is optional and if omitted, it is 0. *value* must be 0 in a NOINIT area.

valuesize is the size, in bytes, of *value*. It can be any of 1, 2, or 4. *valuesize* is optional and if omitted, it is 1.

10.69.2 Usage

Use the ALIGN directive to align any code following a SPACE or FILL directive.

10.69.3 Example

```
        AREA    MyData, DATA, READWRITE
data1   SPACE   255       ; defines 255 bytes of zeroed store
data2   FILL    50,0xAB,1 ; defines 50 bytes containing 0xAB
```

10.69.4 See also

Concepts

armasm User Guide:

- *Numeric expressions* on page 10-16.

Reference

- *DCB* on page 10-23.
- *DCD and DCDU* on page 10-24.
- *DCDO* on page 10-25.
- *DCW and DCWU* on page 10-31.
- *ALIGN* on page 10-11.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.70 TTL and SUBT

The TTL directive inserts a title at the start of each page of a listing file. The title is printed on each page until a new TTL directive is issued.

The SUBT directive places a subtitle on the pages of a listing file. The subtitle is printed on each page until a new SUBT directive is issued.

10.70.1 Syntax

`TTL` *`title`*

`SUBT` *`subtitle`*

where:

`title` is the title.

`subtitle` is the subtitle.

10.70.2 Usage

Use the TTL directive to place a title at the top of the pages of a listing file. If you want the title to appear on the first page, the TTL directive must be on the first line of the source file.

Use additional TTL directives to change the title. Each new TTL directive takes effect from the top of the next page.

Use SUBT to place a subtitle at the top of the pages of a listing file. Subtitles appear in the line below the titles. If you want the subtitle to appear on the first page, the SUBT directive must be on the first line of the source file.

Use additional SUBT directives to change subtitles. Each new SUBT directive takes effect from the top of the next page.

10.70.3 Examples

```
TTL     First Title     ; places a title on the first
                        ; and subsequent pages of a
                        ; listing file.
SUBT    First Subtitle  ; places a subtitle on the
                        ; second and subsequent pages
                        ; of a listing file.
```


Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.71 WHILE and WEND

The WHILE directive starts a sequence of instructions or directives that are to be assembled repeatedly. The sequence is terminated with a WEND directive.

10.71.1 Syntax

```
WHILE logical-expression

code

WEND
```

where:

logical-expression

is an expression that can evaluate to either {TRUE} or {FALSE}.

10.71.2 Usage

Use the WHILE directive, together with the WEND directive, to assemble a sequence of instructions a number of times. The number of repetitions can be zero.

You can use IF...ENDIF conditions within WHILE...WEND loops.

WHILE...WEND loops can be nested.

10.71.3 Example

```
        GBLA count                  ; declare local variable
count   SETA    1                   ; you are not restricted to
        WHILE   count <= 4          ; such simple conditions
count   SETA    count+1             ; In this case,
            ; code                  ; this code is
            ; code                  ; repeated four times
        WEND
```

10.71.4 See also

Concepts

armasm User Guide:

- *Logical expressions* on page 10-19.

Reference

- *Nesting directives* on page 10-5.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

10.72 WN and XN

The WN directive defines a name for a specified 32-bit register in A64 code.

The XN directive defines a name for a specified 64-bit register in A64 code.

10.72.1 Syntax

name directive expr

where:

name is the name to be assigned to the register. *name* cannot be the same as any of the predefined names.

directive is WN or XN.

expr evaluates to a register number from 0 to 30.

10.72.2 Usage

Use WN and XN to allocate convenient names to registers in A64 code, to help you to remember what you use each register for. Be careful to avoid conflicting uses of the same register under different names.

10.72.3 Examples

```
sqr4        WN w16  ; defines sqr4 for register w16
regname     XN 21   ; defines regname for register x21
```

10.72.4 See also

Reference

armasm User Guide:

- *Predeclared core register names in AArch64 state* on page 5-6.
- *Predeclared extension register names in AArch64 state* on page 5-7.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

Appendix A
Via File Syntax

This appendix describes the syntax of via files accepted by all the ARM development tools:

- *Overview of via files* on page A-2
- *Via file syntax* on page A-3

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

A.1 Overview of via files

Via files are plain text files that contain command-line arguments and options to ARM development tools.

Typically, you can use a via file to overcome the command-line length limitations. However, you might want to create multiple via files that:

- Group similar arguments and options together.
- Contain different sets of arguments and options to be used in different scenarios.

Note

In general, you can use a via file to specify any command-line option to a tool, including `--via`. This means that you can call multiple nested via files from within a via file.

A.1.1 Via file evaluation

When the assembler is invoked it:

1. Replaces the first specified `--via` *`via_file`* argument with the sequence of argument words extracted from the via file, including recursively processing any nested `--via` commands in the via file.
2. Processes any subsequent `--via` *`via_file`* arguments in the same way, in the order they are presented.

That is, via files are processed in the order you specify them, and each via file is processed completely including processing nested via files before processing the next via file.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential

A.2 Via file syntax

Via files must conform to the following syntax rules:

- A via file is a text file containing a sequence of words. Each word in the text file is converted into an argument string and passed to the tool.
- Words are separated by whitespace, or the end of a line, except in delimited strings. For example:

  `--bigend --reduce_paths` (two words)

  `--bigend--reduce_paths` (one word)
- The end of a line is treated as whitespace. For example:

  ```
  --bigend
  --reduce_paths
  ```

  is equivalent to:

  ```
  --bigend --reduce_paths
  ```
- Strings enclosed in quotation marks ("), or apostrophes (') are treated as a single word. Within a quoted word, an apostrophe is treated as an ordinary character. Within an apostrophe delimited word, a quotation mark is treated as an ordinary character.

  Use quotation marks to delimit filenames or path names that contain spaces. For example:

  `--errors C:\My Project\errors.txt` (three words)

  `--errors "C:\My Project\errors.txt"` (two words)

  Use apostrophes to delimit words that contain quotes.
- Characters enclosed in parentheses are treated as a single word. For example:

  `--option(x, y, z)` (one word)

  `--option (x, y, z)` (two words)
- Within quoted or apostrophe delimited strings, you can use a backslash (\) character to escape the quote, apostrophe, and backslash characters.
- A word that occurs immediately next to a delimited word is treated as a single word. For example:

  `--errors"C:\Project\errors.txt"`

  is treated as the single word:

  `--errorsC:\Project\errors.txt`
- Lines beginning with a semicolon (;) or a hash (#) character as the first nonwhitespace character are comment lines. If a semicolon or hash character appears anywhere else in a line, it is not treated as the start of a comment. For example:

  ```
  -o objectname.axf      ;this is not a comment
  ```

  A comment ends at the end of a line, or at the end of the file. There are no multi-line comments, and there are no part-line comments.

Copyright © 2014 ARM. All rights reserved.
Non-Confidential