# Aarch64 V8 most common instructions (Work in Progress - V4.0)

## General conventions

	General conventions				
Subtraction   Sumf(s)   Sumf(s)   rid, rin, opp	rd, rn, rm: W or X registers; o	р2: гед <sup>-</sup>	ister or #imn (n-bit immediate)	if {S} is present flags will be affected	
Subtraction   Sumf(s)   Sumf(s)   rid, rin, opp	Addition	VDD163	VDD(2) 24 25 253	rd = rp + op2	[Voc]
Westhorn				·	
with carry					
Incigned multiply   Mul.   Mul.   A, rn, rn	_			•	
Montgood multiply high   Montgood multiply and high   Montgood multiply high high   Montgood multiply   Mon	ween carry	NGC [3]	Mac(3) 10, 11	I C I III C	(1C3)
management multiply brigh	Unsigned multiply	MUL	MUL rd, rn, rm	rd = rn x rm	
Signed multiply loop	Unsigned multiply long	UMULL	UMULL xd, wn, wm	xd = wn x wm	
Signed multiply Nigh	Unsigned multiply high	UMULH	UMULH xd, xn, xm	xd = <127:64> of xn x xm	
None: For IMMUNH and SMULH, Mult can handle bits addited of the 178-bit results   Multiply and add	Signed multiply long	SMULL	SMULL xd, wn, wm	xd = wm x wn (signed operands)	
Multiply and sub	Signed multiply high	SMULH	SMULL xd, wn, wm	xd = <127:64> of xn x xm (signed operands)	
Multiply and sub	Note: for UMULH and SMULH, M	UL can l	nandle bits <63:0> of the 128-bi	t result	
Multiply and add long   Minco   Mico   Mico   Minco	Multiply and add	MADD	MADD rd, rn, rm, ra	rd = ra + (rn x rm)	
Unsigned multiply and add long DMADDL MADDL MAY, wn, wn, xa	Multiply and sub	MSUB	MSUB rd, rn, rm, ra	rd = ra - (rn x rm)	
Unsigned multiply and sub long   MSKUBEL   M	Multiply and neg	MNEG	MNEG rd, rn, rm	rd = - (rn x rm)	
Unsigned multiply and sub long   MSKUBEL   M	Unsigned multiply and add long	UMADDL	UMADDL xd, wn, wm, xa	xd = xa + (wm x wn)	
Designed multiply and add tong   SMADED   SMAD		UMSUBL		xd = xa - (wm x wn)	
Stoped multiply and add long				xd = - (wm x wn)	
Signed multiply and sub long					
Stoped multiply and neg long   SMNEGL SMNEGL SMNEGL Md, wn, wn					
Integer divide unsigned					
Integer divide signed   SDIV   WDIV rd, rn, rn	J			· · · · · · · · · · · · · · · · · · ·	
Note: the remainder may be computed using the MSUB instruction as numerator - (quotient x denominator)	Integer divide unsigned	UDIV	UDIV rd, rn, rm	rd = rn / rm	
Bltwise logical	Integer divide signed	SDIV	UDIV rd, rn, rm	rd = rn / rm	
SIC   SIC(S) rd, rn, op2	Note: the remainder may be c	omputed	using the MSUB instruction as n	umerator – (quotient x denominator)	
SIC   SIC(S) rd, rn, op2			I		
ORR	Bitwise logical			rd = rn & op2	{Yes}
ORN				rd = rn & ~op2	{Yes}
EOR FOR rd, rn, op2		ORR	ORR rd, rn, op2	rd = rn   op2	
EON rd, rn, op2		ORN	ORN rd, rn, op2	rd = rn   ~op2	
LSL LSL rd, rn, op2 Logical shift left (stuffing zeros enter fron right)  LSR LSR rd, rn, rm  Logical shift right (stuffing zeros enter fron left)  ASR ASR rd, rn, op2 Arithmetic shift right (preserves sign)  ROR ROR ROR rd, rn, op2 Rotater right  MOV MOV rd, op2 rd = op2 Yes  MVN MVN RVN rd, op2 rd = -op2  IST TST rn, op2 rn & op2  Bitfield operations  BFI BFI rd, rn, #lsb, #width Moves a bitfield of #width bits starting at source bit #0 To destination starting at bit #lsb  UBFX UBFZ rd, rn, #lsb, #width Moves a bitfield of #width bits starting at source bit robb to destination starting at bit 0; clears all other rd bits to destination starting at bit 0; clears all other rd bits to destination starting at bit 0; stgn extends the result  Bit operations  CLS CLS rd, rn  CLZ CLZ rd, rn  Cunut leading sign bits  CLS CLS rd, rn  Reverse bits  REV REV rd, rn  Reverse bytes  REV16 RCV16 rd, rn  Reverse bytes  REV28 REV37 rd, rn  Reverse bytes  REV32 RCV37 rm  Reverse words  Store single register  STR  rt, [addr]  Men[addr] = rt  Subtype byte  STRB  wt, [addr]  Men[addr] = wt<15:09  Store string at rg, [addr]  Tt = Men[addr] = wt<15:09  Store string at rg, [addr]  Tt = Men[addr] (only 32-byte containers)  Wt = Sub-type byte containers)  Sub-type signed byte  LORS LORS rt, [addr]  Tt = Sbyte[addr] (only 32-byte containers)  Sub-type signed half word  LORS LORS rt, [addr]  Tt = Swergaldadr] (only for 64-byte containers)		EOR	EOR rd, rn, op2	rd = rn ⊕ op2	
LSR LSR rd, rn, rm Logical shift right (stuffing zeros enter from left)  ASR ASR rd, rn, op2 Arithmetic shift right (preserves sign)  AOR ROR ROR rd, rn, op2 Rotate right  MOV MOV rd, op2 rd = op2 Yes  AVN MVN rd, op2 rd = op2  TST TST rn, op2 rn & op2  Bitfield operations  BFI BFI rd, rn, #lsb, #width Moves a bitfield of #width bits starting at source bit #0 To destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Moves a bitfield of #width bits starting at source bit #1 #1 #1 #1 #1 #1 #1 #1 #1 #1 #1 #1 #1		EON	EON rd, rn, op2	rd = rn ⊕ ~op2	
ASR ASR rd, rn, op2 Arithmetic shift right (preserves sign)  ROR ROR rd, rn, op2 Rotate right  MOV MOV rd, op2 rd = -op2  TST MOV rd, op2 rn & op2  RETURN ROPE rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #0 To destination starting at bit #lsb to destination starting at bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  CLZ CLZ rd, rn Count leading sign bits  CLZ CLZ rd, rn Reverse bits  Reverse bits  Reverse bits  Reverse bits  Reverse bits  Reverse bits  Reverse bytes		LSL	LSL rd, rn, op2	Logical shift left (stuffing zeros enter from right)	
ROR ROR rd, rn, op2 Rotate right  MOV MOV rd, op2 rd = op2  MVN MVN rd, op2 rd = -op2  TST TST rn, op2 rn & op2  Bitfield operations  BFI SFI rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #B To destination starting at bit #lsb  UBFX UBFZ rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #B To destination starting at bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFX rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFX rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX SBFX rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clear all other rd bits  SBFX rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clear all other rd bits  SBFX rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clear all other rd  SBFX rd, rn, #lsb, #width Roves a bitfield of #width bits starting at source bit #lsb to destination starting a		LSR	LSR rd, rn, rm	Logical shift right (stuffing zeros enter from left)	
MOV MOV rd, op2  MVN MVN rd, op2  TST TST rn, op2  BET FST rn, op2  BET BFI rd, rn, #lsb, #width  Woves a bitfield of #width bits starting at source bit #0 fosts tasts  BET BFI rd, rn, #lsb, #width  Woves a bitfield of #width bits starting at source bit #0 fosts tasts  BET BFI rd, rn, #lsb, #width  Woves a bitfield of #width bits starting at source bit #0 fosts tasts  BET BFI rd, rn, #lsb, #width  Woves a bitfield of #width bits starting at source bit #0 fosts  BET BFI rd, rn, #lsb, #width  Woves a bitfield of #width bits starting at source bit #10 fosts  BIS b to destination starting at bit 0; clears all other rd bits  BIS to operations  CLZ CLZ rd, rn  Count leading sign bits  CLZ CLZ rd, rn  Reverse bits  REV REV rd, rn  Reverse bits  REVIG REVIG REVIG rd, rn  Reverse bytes  REVIG REVIG REVIG rn  Reverse bytes  SETOR SETOR TH, [addr]  Subtype byte  STRB wt, [addr]  Mem[addr] = rt  Subtype byte  STRB wt, [addr]  SUBRE wt, [addr]  SUBRE wt, [addr]  SUBRE wt, [addr]  SUBRE wt, [addr]  Tt = Mem[addr]  Tt = Mem[addr]  SUB-type signed byte  LDRB LDRB wt, [addr]  Tt = Sybt[addr] (only 32-byte containers)  SUB-type signed byte  LDRSH LDRSH rt, [addr]  Tt = Sybt(addr) (only 32-byte containers)  Tt = Sybtype signed half word  SUB-type signed half word  LDRSH LDRSH rt, [addr]  Tt = Sher[addr] (only 32-byte containers)  Tt = Sybtype signed half word  SUB-type signed half word  LDRSH LDRSH rt, [addr]  Tt = Sher[addr] (only 32-byte containers)  TT = Mem[addr] (only 92-byte containers)		ASR	ASR rd, rn, op2	Arithmetic shift right (preserves sign)	
MVN MVN rd, op2		ROR	ROR rd, rn, op2	Rotate right	
SET		MOV	MOV rd, op2	rd = op2	Yes
Bitfield operations  BFI BFI rd, rn, #lsb, #width  UBFX  UBFX  UBFZ rd, rn, #lsb, #width  Woves a bitfield of #width bits starting at source bit #0 for destination starting at bit #lsb  UBFX  UBFX  UBFZ rd, rn, #lsb, #width  Woves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX  SBFZ rd, rn, #lsb, #width  Woves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  Bit operations  CLS  CLS rd, rm  Count leading sign bits  CLZ  CLZ rd, rm  Reverse bits  RBIT rd, rn  Reverse bits  REV1 REV rd, rm  Reverse bytes  REV16 REV16 rd, rn  Reverse balf words  REV32 REV32 rd, rn  Reverse words  Store single register  STR  xt, [addr]  Subtype byte  STRB  wt, [addr]  Wen[addr] = rt  Subtype half word  STRH  wt, [addr]  Wen[addr] = wt<7:0>  Store register pair  STP  STP rt, rm, [addr]  Store st and rm in consecutive positions starting at addr  Load single register  LDR  LDR kt, [addr]  xt = Men[addr] (only 32-byte containers)  Sub-type byte signed byte  Sub-type signed byte  LDRSB  LDRSB rt, [addr]  xt = HalfWord[addr] (only 32-byte containers)  Sub-type signed half word  LDRSH  LDRSH rt, [addr]  xt = HalfWord[addr] (only 32-byte containers)  Tt = Men[addr] (load one half word, signed)  Sub-type signed word  LDRSH  LDRSH rt, [addr]  xt = Sword[addr] (only for 64-byte containers)		MVN	MVN rd, op2	rd = ~op2	
UBFX UBFX UBFZ rd, rn, #lsb, #width UBFX UBFZ rd, rn, #lsb, #width UBFX UBFX UBFZ rd, rn, #lsb, #width Ubes a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX UBFZ rd, rn, #lsb, #width Ubes a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; sign extends the result  CLS ULS rd, rm UDFX ULZ ULZ rd, rm UDFX ULZ ULZ rd, rm UDFX UDFX UDFX UDFX UDFX UDFX UDFX UDFX		TST	TST rn, op2	rn & op2	Yes
UBFX UBFX UBFZ rd, rn, #lsb, #width UBFX UBFZ rd, rn, #lsb, #width UBFX UBFX UBFZ rd, rn, #lsb, #width Ubes a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; clears all other rd bits  SBFX UBFZ rd, rn, #lsb, #width Ubes a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; sign extends the result  CLS ULS rd, rm UDFX ULZ ULZ rd, rm UDFX ULZ ULZ rd, rm UDFX UDFX UDFX UDFX UDFX UDFX UDFX UDFX	D:+6:-14+:	DET	DET	M Litti -1.1 -5 M.ii.Jah Litt - Ai Lit MO	
#Ibb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Moves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; sign extends the result.  Bit operations CLS CLS rd, rm Count leading sign bits  CLZ CLZ rd, rm Count leading zero bits  RBIT RBIT rd, rm Reverse bits  REV REV rd, rm Reverse bytes  REV16 REV16 rd, rm Reverse bytes  REV32 REV32 rd, rm Reverse words  Store single register STR rt, [addr] Mem[addr] = rt  Subtype byte STRB wt, [addr] Byte[addr] = wt<7:0>  Subtype half word STRH wt, [addr] HalfWord[addr] = wt<15:0>  Store register pair STP STP rt, rm, [addr] Stores rt and rm in consecutive positions starting at addr  Load single register LDR LDR rt, [addr] wt = Byte[addr] (only 32-byte containers)  Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (signed byte)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)	Bitfield operations	B+1	BFI rd, rn, #lsb, #width		
#Ibb to destination starting at bit 0; clears all other rd bits  SBFX SBFZ rd, rn, #lsb, #width Moves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; sign extends the result.  Bit operations CLS CLS rd, rm Count leading sign bits  CLZ CLZ rd, rm Count leading zero bits  RBIT RBIT rd, rm Reverse bits  REV REV rd, rm Reverse bytes  REV16 REV16 rd, rm Reverse bytes  REV32 REV32 rd, rm Reverse words  Store single register STR rt, [addr] Mem[addr] = rt  Subtype byte STRB wt, [addr] Byte[addr] = wt<7:0>  Subtype half word STRH wt, [addr] HalfWord[addr] = wt<15:0>  Store register pair STP STP rt, rm, [addr] Stores rt and rm in consecutive positions starting at addr  Load single register LDR LDR rt, [addr] wt = Byte[addr] (only 32-byte containers)  Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (signed byte)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)					
SBFX SBFZ rd, rn, #lsb, #width Moves a bitfield of #width bits starting at source bit #lsb to destination starting at bit 0; sign extends the result  Bit operations CLS CLS rd, rm Count leading sign bits  CLZ CLZ rd, rm Reverse bits  RBIT RBIT rd, rm Reverse bits  REV REV rd, rm Reverse bytes  REV16 REV16 rd, rm Reverse bytes  REV32 REV32 rd, rm Reverse words  Store single register STR rt, [addr] Mem[addr] = rt  Subtype byte STRB wt, [addr] Byte[addr] = wt<7:0>  Subtype half word STRH wt, [addr] HalfWord[addr] = wt<15:0>  Store register pair STP STP rt, rm, [addr] Stores rt and rm in consecutive positions starting at addr  Load single register LDR LDR rt, [addr] rt = Mem[addr]  Sub-type byte LDRS LDRSB rt, [addr] rt = Sbyte[addr] (only 32-byte containers)  Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (only 32-byte containers)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (only 32-byte containers)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (only 32-byte containers)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (only 32-byte containers)  Sub-type signed word LDRSH LDRSH rt, [addr] rt = Mem[addr] (only 32-byte containers)		UBFX	UBFZ rd, rn, #lsb, #width		
#1sb to destination starting at bit 0; sign extends the result  Bit operations  CLS  CLS rd, rm  Count leading sign bits  CLZ  CLZ rd, rm  Count leading zero bits  RBIT RBIT rd, rm  Reverse bits  REV REV rd, rm  Reverse bytes  REV16 REV16 rd, rm  Reverse half words  REV32 REV32 rd, rm  Reverse words  Store single register  STR  STR  STR  t, [addr]  Mem[addr] = rt  Subtype byte  STRB  wt, [addr]  Byte[addr] = wt<7:0>  Store register pair  STP  STP rt, rm, [addr]  Store single register  LDR  LDR rt, [addr]  Sub-type byte  LDRS  LDRS LDRS wt, [addr]  wt = Byte[addr] (only 32-byte containers)  Sub-type signed half word  Sub-type signed half word  Sub-type signed word  LDRSH  LDRSH rt, [addr]  wt = HalfWord[addr] (only 32-byte containers)  Sub-type signed half word  Sub-type signed half word  LDRSH LDRSH rt, [addr]  rt = Mem[addr] (only 32-byte containers)  Sub-type signed half word  Sub-type signed half word  LDRSH LDRSH rt, [addr]  rt = Mem[addr] (only 32-byte containers)  Sub-type signed half word  LDRSH LDRSH rt, [addr]  rt = Mem[addr] (load one half word, signed)  Sub-type signed word  LDRSW LDRSW xt, [addr]  rt = Mem[addr] (load one half word, signed)  Sub-type signed word  LDRSW LDRSW xt, [addr]  rt = Mem[addr] (lonly for 64-byte containers)					
result  Bit operations  CLS CLS rd, rm  Count leading sign bits  CLZ CLZ rd, rm  Count leading zero bits  RBIT RBIT rd, rm  Reverse bits  REV REV rd, rm  Reverse bytes  REV16 REV16 rd, rm  Reverse bytes  REV32 REV32 rd, rm  Reverse words  Store single register  STR  SUbtype byte  STRB wt, [addr]  Subtype half word  STRH wt, [addr]  Store register pair  STP  STP rt, rm, [addr]  Store register lDR  SUDRS wt, [addr]  Sub-type byte  LDRS LDRS wt, [addr]  Sub-type signed byte  LDRS LDRS rt, [addr]  Sub-type signed half word  LDRSH LDRSH rt, [addr]  Sub-type signed word  LDRSH LDRSH rt, [addr]  Tt = Mem[addr] (load one half word, signed)  Sub-type signed word  LDRSW t, [addr]  Tt = Mem[addr] (load one half word, signed)  Sub-type signed word  LDRSW t, [addr]  Tt = Mem[addr] (load one half word, signed)  Sub-type signed word  LDRSW t, [addr]  Tt = Mem[addr] (load one half word, signed)  Sub-type signed word  LDRSW t, [addr]  Tt = Mem[addr] (load one half word, signed)  Sub-type signed word  LDRSW t, [addr]  Tt = Mem[addr] (load one half word, signed)		SBFX	SBFZ rd, rn, #lsb, #width		
CLZ CLZ rd, rm Count leading zero bits  RBIT RBIT rd, rm Reverse bits  REV REV rd, rm Reverse bytes  REV16 REV16 rd, rm Reverse half words  REV32 REV32 rd, rm Reverse words  Store single register STR rt, [addr] Mem[addr] = rt  Subtype byte STRB wt, [addr] Byte[addr] = wt<7:0>  Subtype half word STRH wt, [addr] HalfWord[addr] = wt<15:0>  Store register pair STP STP rt, rm, [addr] Stores rt and rm in consecutive positions starting at addr  Load single register LDR LDR rt, [addr] rt = Mem[addr]  Sub-type byte LDRB LDRB wt, [addr] wt = Byte[addr] (only 32-byte containers)  Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (only 32-byte containers)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)					
CLZ CLZ rd, rm Count leading zero bits  RBIT RBIT rd, rm Reverse bits  REV REV rd, rm Reverse bytes  REV16 REV16 rd, rm Reverse half words  REV32 REV32 rd, rm Reverse words  Store single register STR rt, [addr] Mem[addr] = rt  Subtype byte STRB wt, [addr] Byte[addr] = wt<7:0>  Subtype half word STRH wt, [addr] HalfWord[addr] = wt<15:0>  Store register pair STP STP rt, rm, [addr] Stores rt and rm in consecutive positions starting at addr  Load single register LDR LDR rt, [addr] rt = Mem[addr]  Sub-type byte LDRB LDRB wt, [addr] wt = Byte[addr] (only 32-byte containers)  Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (only 32-byte containers)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)	Rit operations	CLS	CLS cd cm	Count leading sign bits	
REV REV rd, rm Reverse bytes  REV REV16 rd, rm Reverse bytes  REV32 REV32 rd, rm Reverse half words  REV32 REV32 rd, rm Reverse words  Store single register STR rt, [addr] Mem[addr] = rt  Subtype byte STRB wt, [addr] Byte[addr] = wt<7:0>  Subtype half word STRH wt, [addr] HalfWord[addr] = wt<15:0>  Store register pair STP STP rt, rm, [addr] Stores rt and rm in consecutive positions starting at addr  Load single register LDR LDR rt, [addr] rt = Mem[addr]  Sub-type byte LDRS LDRS wt, [addr] wt = Byte[addr] (only 32-byte containers)  Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (signed byte)  Sub-type signed half word LDRH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)	DEC OPERACIONS				
REV REV rd, rm Reverse bytes  REV16 REV16 rd, rm Reverse half words  REV32 REV32 rd, rm Reverse words  Store single register STR rt, [addr] Mem[addr] = rt  Subtype byte STRB wt, [addr] Byte[addr] = wt<7:0>  Subtype half word STRH wt, [addr] HalfWord[addr] = wt<15:0>  Store register pair STP STP rt, rm, [addr] Stores rt and rm in consecutive positions starting at addr  Load single register LDR LDR rt, [addr] rt = Mem[addr]  Sub-type byte LDRB LDRB wt, [addr] wt = Byte[addr] (only 32-byte containers)  Sub-type signed byte LDRB LDRSB rt, [addr] rt = Sbyte[addr] (signed byte)  Sub-type signed half word LDRH LDRH wt, [addr] wt = HalfWord[addr] (only 32-byte containers)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)					
REV16 REV16 rd, rm Reverse half words  REV32 REV32 rd, rm Reverse words  Store single register STR rt, [addr] Mem[addr] = rt Subtype byte STRB wt, [addr] Byte[addr] = wt<7:0> Subtype half word STRH wt, [addr] HalfWord[addr] = wt<15:0> Store register pair STP STP rt, rm, [addr] Stores rt and rm in consecutive positions starting at addr Load single register LDR LDR rt, [addr] rt = Mem[addr] Sub-type byte LDRB LDRB wt, [addr] wt = Byte[addr] (only 32-byte containers) Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (signed byte) Sub-type signed half word LDRH LDRH wt, [addr] rt = Mem[addr] (only 32-byte containers) Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed) Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)					
REV32 REV32 rd, rm  Reverse words  Store single register  STR  rt, [addr]  Mem[addr] = rt  Subtype byte  STRB  wt, [addr]  Subtype half word  STRH  wt, [addr]  STP  STP rt, rm, [addr]  Stores rt and rm in consecutive positions starting at addr  Load single register  LDR  LDR LDR rt, [addr]  Sub-type byte  LDRB  LDRB wt, [addr]  wt = Byte[addr] (only 32-byte containers)  Sub-type signed byte  LDRSB  LDRSH  LDRSH rt, [addr]  wt = HalfWord[addr] (only 32-byte containers)  wt = HalfWord[addr] (only 32-byte containers)  Sub-type signed half word  LDRSH  LDRSH rt, [addr]  rt = Mem[addr] (load one half word, signed)  Sub-type signed word  LDRSW  LDRSW xt, [addr]  xt = sword[addr] (only for 64-byte containers)				•	
Store single register  STR rt, [addr] Mem[addr] = rt  Subtype byte STRB wt, [addr] Byte[addr] = wt<7:0>  Subtype half word STRH wt, [addr] HalfWord[addr] = wt<15:0>  Store register pair STP STP rt, rm, [addr] Stores rt and rm in consecutive positions starting at addr  Load single register  LDR LDR rt, [addr] rt = Mem[addr]  Sub-type byte LDRB LDRB wt, [addr] wt = Byte[addr] (only 32-byte containers)  Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (signed byte)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (only 32-byte containers)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)					
Subtype byte STRB wt, [addr] Byte[addr] = wt<7:0> Subtype half word STRH wt, [addr] HalfWord[addr] = wt<15:0> Store register pair STP STP rt, rm, [addr] Stores rt and rm in consecutive positions starting at addr Load single register LDR LDR rt, [addr] rt = Mem[addr] Sub-type byte LDRB LDRB wt, [addr] wt = Byte[addr] (only 32-byte containers) Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (signed byte) Sub-type half word LDRH LDRH wt, [addr] wt = HalfWord[addr] (only 32-byte containers) Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed) Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)		KEV32	KEV32 FO, FM	keverse WOFOS	
Subtype byte STRB wt, [addr] Byte[addr] = wt<7:0> Subtype half word STRH wt, [addr] HalfWord[addr] = wt<15:0> Store register pair STP STP rt, rm, [addr] Stores rt and rm in consecutive positions starting at addr Load single register LDR LDR rt, [addr] rt = Mem[addr] Sub-type byte LDRB LDRB wt, [addr] wt = Byte[addr] (only 32-byte containers) Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (signed byte) Sub-type half word LDRH LDRH wt, [addr] wt = HalfWord[addr] (only 32-byte containers) Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed) Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)	Store single register	STR	rt, [addr]	Mem[addr] = rt	
Subtype half word  STRH wt, [addr] HalfWord[addr] = wt<15:0>  Store register pair  STP STP rt, rm, [addr] Stores rt and rm in consecutive positions starting at addr  Load single register  LDR LDR rt, [addr] rt = Mem[addr]  Sub-type byte  LDRB LDRB wt, [addr] wt = Byte[addr] (only 32-byte containers)  Sub-type signed byte  LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (signed byte)  Sub-type half word  LDRH LDRH wt, [addr] wt = HalfWord[addr] (only 32-byte containers)  Sub-type signed half word  LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word  LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)					
Store register pair  STP STP rt, rm, [addr] Stores rt and rm in consecutive positions starting at addr  Load single register  LDR LDR rt, [addr] rt = Mem[addr]  Sub-type byte LDRB LDRB wt, [addr] wt = Byte[addr] (only 32-byte containers)  Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (signed byte)  Sub-type half word LDRH LDRH wt, [addr] wt = HalfWord[addr] (only 32-byte containers)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)					
Load single register  LDR LDR rt, [addr] rt = Mem[addr]  Sub-type byte LDRB LDRB wt, [addr] wt = Byte[addr] (only 32-byte containers)  Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (signed byte)  Sub-type half word LDRH LDRH wt, [addr] wt = HalfWord[addr] (only 32-byte containers)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)					
Sub-type byte LDRB LDRB wt, [addr] wt = Byte[addr] (only 32-byte containers)  Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (signed byte)  Sub-type half word LDRH LDRH wt, [addr] wt = HalfWord[addr] (only 32-byte containers)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)	- '				
Sub-type signed byte LDRSB LDRSB rt, [addr] rt = Sbyte[addr] (signed byte)  Sub-type half word LDRH LDRH wt, [addr] wt = HalfWord[addr] (only 32-byte containers)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)					
Sub-type half word LDRH LDRH wt, [addr] wt = HalfWord[addr] (only 32-byte containers)  Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)					
Sub-type signed half word LDRSH LDRSH rt, [addr] rt = Mem[addr] (load one half word, signed)  Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)					
Sub-type signed word LDRSW LDRSW xt, [addr] xt = sword[addr] (only for 64-byte containers)					
Load register pair LDP rt, rm, [addr] Loads rt and rm from consecutive positions starting at addr					
	Load register pair	LUP	LUY IT, IM, [addr]	Loads rt and rm from consecutive positions starting at addr	

ranch B B target Jump to target				
Conditional branch	B.CC	B.cc target	If (cc) jump to target	
Compare and branch if zero	CBZ	CBZ rd, target	If (rd=0) jump to target	
Compare and branch if not zero CBNZ CBNZ rd, target				
	1	1		
Conditional select	CSEL	CSEL rd, rn, rm, cc	If (cc) rd = rn else rd = rm	
with increment,	CSINC	CSINC rd, rn, rm, cc	If (cc) rd = rn else rd = rm+1	
with negate,	CSNEG	CSNEG rd, rn, rm, cc	If (cc) rd = rn else rd = -rm	
with invert	CSINV	CSINV rd, rn, rm, cc	If (cc) rd = rn else rd = ~rm	
Conditional set	CSET	CSET rd, cc	If (cc) rd = 1 else rd = 0	
with mask,	CSETM	CSETM rd, cc	If (cc) rd = -1 else rd = 0	
with increment,	CINC	CINC rd, rn, cc	If (cc) rd = rn+1 else rd = rn	
with negate,	CNEG	CNEG rd, rn, cc	If (cc) rd = -rn else rd = rn	
with invert CINV CINV rd, rn, cc If (cc) rd = ~rn else rd = rn				
Compare CMP CMP rd, op2 rd - op2 Yes				Yes
with negative	CMN	CMN rd, op2	rd - (-op2)	Yes
Conditional compare	CCMP	CCMP rd, rn, #im4, cc	If (cc) NZCV = CMP(rd,rn) else NZCV = #imm4	Yes
with negative CCMN CCMP rd, rn, #im4, cc If (cc) NZCV = CMP(rd,-rn) else NZCV = #imm4 Ye			Yes	
Note: for these instructions	rn can	also be an #im5 (5-bit unsigned	immediate value 032)	

## Aarch64 V8 accessory information

## Condition codes (magnitude of operands)

LO	Lower, unsigned	C = 0
HI	Higher, unsigned	C = 1 and Z = 0
LS	Lower or same, unsigned	C = 0 or Z = 1
HS	Higher or same, unsigned	C = 1
LT	Less than, signed	N != V
GT	Greater than, signed	Z = 0 and $N = V$
LE	Less than or equal, signed	Z = 1 and N != V
GE	Greater than or equal, signed	N = V

## Condition codes (direct flags)

EQ	Equal	Z = 1
NE	Not equal	Z = 0
MI	Negative	N = 1
PL	Positive or zero	N = 0
VS	Overflow	V = 1
VC	No overflow	V = 0
CS	Carry	C = 0
СС	No carry	C = 1

# Sub types (suffix of some instructions) R byte 8 bits

SB signed byte         8 bits           H Half word         16 bits           SH signed half word         16 bits           W word         32 bits	B	byte	8 DITS
SH signed half word 16 bits W word 32 bits	SB	signed byte	8 bits
W word 32 bits	Н	Half word	16 bits
	SH	signed half word	16 bits
	W	word	32 bits
SW signed word 32 bits	SW	signed word	32 bits

#### Flags set to 1 when:

N	the result of the last operation was negative, cleared to 0 otherwise
Z	the result of the last operation was zero, cleared to 0 otherwise
С	the last operation resulted in a carry, cleared to 0 otherwise
V	the last operation caused overflow, cleared to 0 otherwise

#### Containers

Χn	64	bit	register
Wn	32	LSB	of Xn

#### Addressing modes (base: register; offset: register or immediate)

Addiess tily Modes	(base, register, orrset, register or thineatat	<del>-</del> )
[base]	MEM[base]	
[base, offset]	MEM[base+offset]	
[base, offset]!	<pre>MEM[base+offset] then base = base + offset</pre>	(pre indexed)
[base], offset	MEM[base] then base = base + offset	(post indexed)

#### Sizes, Assembly and C

	5 c z c 5 , 7 c c 5 c c c				
8	byte	char			
16	Half word	short int			
32	word	int			
64	double word	long int			
128	quad word	-			

#### Op2 processing (applied to Op2 before anything else )

ope processing (c	pperce to ope belove anything eros /
LSL LSR ASR #imm	
SXTW	Sign extension
SXTB {#imm2}	Sign extension after LSL #imm2

#### Calling convention (register use)

Params: X0..X7 Result: X0

Do not use: X8, X16..X18

Unprotected: X9..X15 (callee may corrupt these)
Protected: X19..X28 (callee must preserve these)