

Cortex-M4F Instructions used in *ARM Assembly for Embedded Applications* (ISBN 978-1-09254-223-4)

Function Call/Return	Operation	Notes	Clock Cycles
BL <i>label</i>	$LR \leftarrow \text{return address}; PC \leftarrow \text{address of label}$	BL is used to call a function BX LR is used as function return	2-4
BLX R_n	$LR \leftarrow \text{return address}; PC \leftarrow R_n$		
BX R_n	$PC \leftarrow R_n$		
B <i>label</i>	$PC \leftarrow \text{address of label}$		




Load Integer Constant	Operation	Flags	Notes	Clock Cycles
ADR R_d, label	$R_d \leftarrow \text{address of label}$		$PC-4095 \leq \text{address} \leq PC+4095$	1
MOV{S} $R_d, \text{constant}$	$R_d \leftarrow \text{constant}$	NZ	$0 \leq \text{constant} \leq 255$ (FF ₁₆) & a few others	
MVN{S} $R_d, \text{constant}$	$R_d \leftarrow \sim \text{constant}$	NZ	$0 \leq \text{constant} \leq 255$ (FF ₁₆) & a few others	
MOVW $R_d, \text{constant}$	$R_d \leftarrow \text{constant}$		$0 \leq \text{constant} \leq 65535$ (FFFF ₁₆)	
MOVT $R_d, \text{constant}$	$R_d <31..16> \leftarrow \text{constant}$		$0 \leq \text{constant} \leq 65535$ (FFFF ₁₆)	

Load/Store Memory	Operation	Bits	Notes	Clock Cycles
LDRB $R_d, [\text{address mode}]$	$R_d \leftarrow \text{memory} <7..0>$ (zero extended)	8	$R_d <31..8> \leftarrow 24 \text{ 0's}$	2
LDRSB $R_d, [\text{address mode}]$	$R_d \leftarrow \text{memory} <7..0>$ (sign extended)	8	$R_d <31..8> \leftarrow 24 \text{ copies of } R_d <7>$	
LDRH $R_d, [\text{address mode}]$	$R_d \leftarrow \text{memory} <15..0>$ (zero extended)	16	$R_d <31..16> \leftarrow 16 \text{ 0's}$	
LDRSH $R_d, [\text{address mode}]$	$R_d \leftarrow \text{memory} <15..0>$ (sign extended)	16	$R_d <31..16> \leftarrow 16 \text{ copies of } R_d <16>$	
LDR $R_d, [\text{address mode}]$	$R_d \leftarrow \text{memory} <31..0>$	32		3
LDRD $R_t, R_{t2}, [\text{address mode}]$	$R_{t2}, R_t \leftarrow \text{memory} <63..0>$	64	Can't use register offset adrs mode	
STRB $R_d, [\text{address mode}]$	$R_d \rightarrow \text{memory} <7..0>$	8		
STRH $R_d, [\text{address mode}]$	$R_d \rightarrow \text{memory} <15..0>$	16		
STR $R_d, [\text{address mode}]$	$R_d \rightarrow \text{memory} <31..0>$	32		
STRD $R_t, R_{t2}, [\text{address mode}]$	$R_{t2}, R_t \rightarrow \text{memory} <63..0>$	64	Can't use register offset adrs mode	


Load/Store Multiple		Operation	Notes	Clock Cycles
POP	{register list}	registers ← memory[SP]; SP+=4×#registers	regs: Not SP; PC/LR, but not both	1 + #registers
PUSH	{register list}	SP-=4×#registers; registers → memory[SP]	regs: Neither SP or PC.	
LDMIA	R _n !,{register list}	registers ← memory[R _n]	if "!" is appended, then R _n += 4×#registers	
STMIA	R _n !,{register list}	registers → memory[R _n]		
LDMDB	R _n !,{register list}	registers ← memory[R _n - 4×#registers]	if "!" is appended, then R _n -= 4×#registers	
STMDB	R _n !,{ register list}	registers → memory[R _n - 4×#registers]		



Move / Add / Subtract	Operation	Flags	operand2 options:	Clock Cycles
MOV{S} R_d, R_n	$R_d \leftarrow R_n$	NZ	1. constant 2. R_m (a register) 3. R_m, shift (Any kind of shift)	1
ADD{S} $R_d, R_n, \text{operand2}$	$R_d \leftarrow R_n + \text{operand2}$	NZCV		
ADC{S} $R_d, R_n, \text{operand2}$	$R_d \leftarrow R_n + \text{operand2} + C$	NZCV		
SUB{S} $R_d, R_n, \text{operand2}$	$R_d \leftarrow R_n - \text{operand2}$	NZCV		
SBC{S} $R_d, R_n, \text{operand2}$	$R_d \leftarrow R_n - \text{operand2} + C - 1$	NZCV		
RSB{S} $R_d, R_n, \text{operand2}$	$R_d \leftarrow \text{operand2} - R_n$	NZCV		


Cortex-M4F Instructions used in *ARM Assembly for Embedded Applications* (ISBN 978-1-09254-223-4)

Multiply / Divide	Operation	Flags	Notes	Clock Cycles
MUL{S} R _d , R _n , R _m	R _d ← (R _n x R _m)<31..0>	NZC	32 ← 32×32; C←undefined	1
MLA R _d , R _n , R _m , R _a	R _d ← R _a + (R _n x R _m)<31..0>		32 ← 32 + 32×32	
MLS R _d , R _n , R _m , R _a	R _d ← R _a - (R _n x R _m)<31..0>		32 ← 32 - 32×32	
SMMUL{R} R _d , R _n , R _m	R _d ← (R _n x R _m)<63..32>		Upper half of signed 64-bit product; Append R: Round towards +∞ (Adds 0x80000000 to the 64-bit product)	
SMMLA{R} R _d , R _n , R _m , R _a	R _d ← R _a + (R _n x R _m)<63..32>			
SMMLS{R} R _d , R _n , R _m , R _a	R _d ← R _a - (R _n x R _m)<63..32>			
 MULL R _{dlo} , R _{dhi} , R _n , R _m	R _{dhi} R _{dlo} ← R _n x R _m		S igned/ U nsigned: 64 ← 32x32	
 MLAL R _{dlo} , R _{dhi} , R _n , R _m	R _{dhi} R _{dlo} ← R _{dhi} R _{dlo} + R _n xR _m		S igned/ U nsigned: 64 ← 64 + 32×32	
 DIV R _d , R _n , R _m	R _d ← R _n / R _m		S igned/ U nsigned: 32 ← 32÷32	

Saturating Instructions		Operation	Min	Max	operand2 options	Clock Cycles
SSAT	$R_d, n, operand2$	$R_d \leftarrow operand2$	-2^{n-1}	$2^{n-1}-1$	1. R_m (a register) 2. R_m, ASR constant 3. R_m, LSL constant	1
USAT	$R_d, n, operand2$	$R_d \leftarrow operand2$	0	2^n-1		
QADD	R_d, R_n, R_m	$R_d \leftarrow R_n + R_m$	-2^{31}	$2^{31}-1$	(Q \leftarrow 1 if saturates)	
QSUB	R_d, R_n, R_m	$R_d \leftarrow R_n - R_m$				

SIMD Signed Saturating ADD/SUB	Operation	Min to Max	Notes	Clock Cycles
QADD  R_d, R_n, R_m	$R_d[bits] \leftarrow R_n[bits] + R_m[bits]$	8 : -2^7 to $+2^7-1$ 16 : -2^{15} to $+2^{15}-1$	For bytes 0-3: bits 7..0, 15..8, 23..16, & 31..24 (No flags affected)	1
QSUB  R_d, R_n, R_m	$R_d[bits] \leftarrow R_n[bits] - R_m[bits]$			

SIMD Unsigned Saturating ADD/SUB	Operation	Min to Max	Notes	Clock Cycles
UQADD  R_d, R_n, R_m	$R_d[bits] \leftarrow R_n[bits] + R_m[bits]$	8 : 0 to 2^8-1 16 : 0 to $2^{16}-1$	For halfwords 0 and 1: bits 15..0 & 31..16 (No flags affected)	1
UQSUB  R_d, R_n, R_m	$R_d[bits] \leftarrow R_n[bits] - R_m[bits]$			

SIMD Signed Non-Saturating ADD/SUB	Operation	GE Flags	Notes	Clock Cycles
SADD  R_d, R_n, R_m	$R_d[bits] \leftarrow R_n[bits] + R_m[bits]$	sum ≥ 0 ? 1 : 0	Parallel operations: Four 8-bit operations, or two 16-bit operations	1
SSUB  R_d, R_n, R_m	$R_d[bits] \leftarrow R_n[bits] - R_m[bits]$	diff ≥ 0 ? 1 : 0		

SIMD Unsigned Non-Saturating ADD/SUB	Operation	GE Flags	Notes	Clock Cycles
UADD  R_d, R_n, R_m	$R_d[bits] \leftarrow R_n[bits] + R_m[bits]$	overflow ? 1 : 0	Parallel operations: Four 8-bit operations, or two 16-bit operations	1
USUB  R_d, R_n, R_m	$R_d[bits] \leftarrow R_n[bits] - R_m[bits]$	diff ≥ 0 ? 1 : 0		

Cortex-M4F Instructions used in *ARM Assembly for Embedded Applications* (ISBN 978-1-09254-223-4)

Q and GE Flag Instructions		Operation	Notes	Clock Cycles
SEL	R _d ,R _n ,R _m	R _d [bits] ← (GE[byte] = 1) ? R _n [bits] : R _m [bits]	For bytes 0-3: bits 7..0, 15..8, 23..16, & 31..24	1
MRS	R _d ,APSR	R _d <31..27> ← NZCVQ R _d <19..16> ← GE flags	All other bots of R _d are filled with zeroes.	
MSR	APSR_nzcvq,R _n	NZCVQ ← R _n <31..27>	Other flags in the PSR are not affected.	
MSR	APSR_g,R _n	GE flags ← R _n <19..16>		



SIMD Multiply Instructions		Operation	Notes	Clock Cycles
SMUAD	R_d, R_n, R_m	$R_d \leftarrow R_n\langle 15..00 \rangle \times R_m\langle 15..00 \rangle + R_n\langle 31..16 \rangle \times R_m\langle 31..16 \rangle$	Sets Q flag if an addition or subtraction overflows; does <u>not</u> saturate.	1
SMUSD	R_d, R_n, R_m	$R_d \leftarrow R_n\langle 15..00 \rangle \times R_m\langle 15..00 \rangle - R_n\langle 31..16 \rangle \times R_m\langle 31..16 \rangle$		
SMLAD	R_d, R_n, R_m, R_a	$R_d \leftarrow R_a + R_n\langle 15..00 \rangle \times R_m\langle 15..00 \rangle + R_n\langle 31..16 \rangle \times R_m\langle 31..16 \rangle$		
SMLSD	R_d, R_n, R_m, R_a	$R_d \leftarrow R_a + R_n\langle 15..00 \rangle \times R_m\langle 15..00 \rangle - R_n\langle 31..16 \rangle \times R_m\langle 31..16 \rangle$		
SMLALD	$R_{dlo}, R_{dhi}, R_n, R_m$	$R_{dhi}.R_{dlo} += R_n\langle 15..00 \rangle \times R_m\langle 15..00 \rangle + R_n\langle 31..16 \rangle \times R_m\langle 31..16 \rangle$		
SMLSLD	$R_{dlo}, R_{dhi}, R_n, R_m$	$R_{dhi}.R_{dlo} += R_n\langle 15..00 \rangle \times R_m\langle 15..00 \rangle - R_n\langle 31..16 \rangle \times R_m\langle 31..16 \rangle$		

Appending "X" to instruction mnemonic changes operand2s to $R_n\langle 15..00 \rangle \times R_m\langle 31..16 \rangle$ and $R_n\langle 31..16 \rangle \times R_m\langle 15..00 \rangle$.

Signed Multiply Halfwords		Operation	Notes	Clock Cycles
SMULBB	R_d, R_n, R_m	$R_d \leftarrow R_n\langle 15..00 \rangle \times R_m\langle 15..00 \rangle$	$32 \leftarrow 16 \times 16$	1
SMULBT	R_d, R_n, R_m	$R_d \leftarrow R_n\langle 15..00 \rangle \times R_m\langle 31..16 \rangle$		
SMULTB	R_d, R_n, R_m	$R_d \leftarrow R_n\langle 31..16 \rangle \times R_m\langle 15..00 \rangle$		
SMULTT	R_d, R_n, R_m	$R_d \leftarrow R_n\langle 31..16 \rangle \times R_m\langle 31..16 \rangle$		

Pack Halfwords		Operation	operand2 options:	Notes	Clock Cycles
PKHBT	$R_d, R_n, \text{operand2}$	Btm: $R_d\langle 15..00 \rangle \leftarrow R_n\langle 15..00 \rangle$ Top: $R_d\langle 31..16 \rangle \leftarrow \text{operand2}\langle 31..16 \rangle$	1. R_m (a register) 2. R_m , LSL constant 3. R_m , ASR constant	Shift constants: LSL: 1-31 ASR: 1-32	1
PKHTB	$R_d, R_n, \text{operand2}$	Top: $R_d\langle 31..16 \rangle \leftarrow R_n\langle 31..16 \rangle$ Btm: $R_d\langle 15..00 \rangle \leftarrow \text{operand2}\langle 15..00 \rangle$			

Compare Instructions		Operation	operand2 options:	Notes	Clock Cycles
CMP	$R_n, \text{operand2}$	$R_n - \text{operand2}$	1. constant 2. R_m (a register) 3. R_m , shift (any kind of shift)	Updates: NZCV	1
CMN	$R_n, \text{operand2}$	$R_n + \text{operand2}$		Updates: NZCV	
TST	$R_n, \text{operand2}$	$R_n \& \text{operand2}$		Updates: NZC	
TEQ	$R_n, \text{operand2}$	$R_n \wedge \text{operand2}$		Updates: NZC	

Zero/Sign-Extend Instructions		Operation	operand2 options:	Clock Cycles
 XTB	$R_d, \text{operand2}$	$R_d \leftarrow \text{Sign (S) extend or Unsigned (U) extend operand2}\langle 7..0 \rangle$	1. R_m (a register) 2. R_m , ROR constant (constant=8, 16 or 24)	1
 XTH	$R_d, \text{operand2}$	$R_d \leftarrow \text{Sign (S) extend or Unsigned (U) extend operand2}\langle 15..0 \rangle$		

Cortex-M4F Instructions used in *ARM Assembly for Embedded Applications* (ISBN 978-1-09254-223-4)

Conditional Branch Instructions		Operation	Notes		Clock Cycles
Bcc	label	Branch to label if "cc" is true	"cc" is a condition code		1 (Fail) or 2-4
CBZ	R _n , label	Branch to label if R _n =0	Can't use in an IT block		
CBNZ	R _n , label	Branch to label if R _n ≠0	Can't use in an IT block		
ITC ₁ C ₂ C ₃	condition code	Each c _i is one of T, E, or empty	Controls 1-4 instructions		1

Shift Instructions		Operation	Flags	operand2 options	Notes	Clock Cycles
ASR{S}	R _d ,R _n ,operand2 // 1-32 bits	R _d ← R _n >> operand2 (arithmetic shift right)	NZC	1. constant 2. R _m (a register) ASR & LSR shift 1-32 bits; LSL & ROR shift 1-31 bits. RRX shifts only by 1 bit.	Sign extends	1
LSL{S}	R _d ,R _n ,operand2 // 1-31 bits	R _d ← R _n << operand2 (logical shift left)	NZC		Zero fills	
LSR{S}	R _d ,R _n ,operand2 // 1-32 bits	R _d ← R _n >> operand2 (logical shift right)	NZC			
ROR{S}	R _d ,R _n ,operand2 // 1-31 bits	R _d ← R _n >> operand2 (rotate right)	NZC		right rotate	
RRX{S}	R _d ,R _n // 1 bit	R _d ← R _n >> 1; R _d <31> ← C; C ← R _n <0>	NZC	33-bit rotate w/C		

Bitwise Instructions		Operation	Flags	operand2 options	Notes	Clock Cycles
AND{S}	R _d ,R _n ,operand2	R _d ← R _n & operand2	NZC	1. constant 2. R _m (a register) 3. R _m ,shift (Any kind of shift)		1
ORR{S}	R _d ,R _n ,operand2	R _d ← R _n operand2	NZC			
EOR{S}	R _d ,R _n ,operand2	R _d ← R _n ^ operand2	NZC			
BIC{S}	R _d ,R _n ,operand2	R _d ← R _n & ~operand2	NZC			
ORN{S}	R _d ,R _n ,operand2	R _d ← R _n ~operand2	NZC			
MVN{S}	R _d ,operand2	R _d ← ~operand2	NZC			

Bitfield Instructions		Operation	Notes	Clock Cycles
BFC	R _d ,lsb,width	SelectedBitfieldOf(R _d) ← 0		1
BFI	R _d ,R _n ,lsb,width	SelectedBitfieldOf(R _d) ← LSBitsOf(R _n)		
SBFX	R _d ,R _n ,lsb,width	R _d ← SelectedBitfieldOf(R _n)	Sign extends	
UBFX	R _d ,R _n ,lsb,width	R _d ← SelectedBitfieldOf(R _n)	Zero extends	

Bits / Bytes / Words		Operation	Notes	Clock Cycles
CLZ	R _d ,R _n	R _d ← CountLeadingZeroesOf(R _n)	#leading 0's = 0-32	1
RBIT	R _d ,R _n	R _d ← ReverseBitOrderOf(R _n)		
REV	R _d ,R _n	R _d ← ReverseByteOrderOf(R _n)		

Pseudo-Instructions		Operation	Flags	Replaced by	Clock Cycles
LDR	R _d ,=constant	R _d ← constant		MOV, MVN, MOVW, or LDR	1
NEG	R _d ,R _n	R _d ← -R _n	NZCV	RSBS R _d ,R _n ,0	
CPY	R _d ,R _n	R _d ← R _n		MOV R _d ,R _n	

Cortex-M4F Instructions used in *ARM Assembly for Embedded Applications* (ISBN 978-1-09254-223-4)

Floating-Point PUSH/POP		Operation	Clock Cycles
VPUISH	{FP register list}	SP -= 4 × #registers, copy registers to memory[SP]	1 + #registers
VPOP	{FP register list}	Copy memory[SP] to registers, SP += 4 × # registers	

Floating-Point Load Constant		Operation	Clock Cycles
VMOV	S _d , fpconstant	fpconstant must be $\pm m \times 2^{-n}$, (16 ≤ m ≤ 31; 0 ≤ n ≤ 7)	1

Floating-Point Copy Registers		Operation	Clock Cycles
VMOV	S _d , S _m	S _d ← S _m	1
VMOV	R _d , S _m	R _d ← S _m	
VMOV	S _d , R _m	S _d ← R _m	
VMOV	R _t , R _{t2} , S _m , S _{m+1}	R _t ← S _m ; R _{t2} ← S _{m+1} (S _m , S _{m+1} adjacent regs)	2
VMOV	S _m , S _{m+1} , R _t , R _{t2}	S _m ← R _t ; S _{m+1} ← R _{t2} (S _m , S _{m+1} adjacent regs)	

Floating-Point Load Registers		Operation	Clock Cycles
VLDR	S _d , [R _n]	S _d ← memory32[R _n]	2
VLDR	S _d , [R _n , constant]	S _d ← memory32[R _n + constant]	
VLDR	S _d , label	S _d ← memory32[Address of label]	
VLDR	D _d , [R _n]	D _d ← memory64[R _n]	3
VLDR	D _d , [R _n , constant]	D _d ← memory64[R _n + constant]	
VLDR	D _d , label	D _d ← memory64[Address of label]	
VLDmia	R _n !, {FP register list}	FP registers ← memory, R _n = lowest address; Updates R _n if write-back flag (!) is included.	1 + #registers
VLDmdb	R _n !, {FP register list}	FP registers ← memory, R _n -4 = highest address; Must append (!) and always updates R _n	

Floating-Point Store Registers		Operation	Clock Cycles
VSTR	S _d , [R _n]	S _d → memory32[R _n]	2
VSTR	S _d , [R _n , constant]	S _d → memory32[R _n + constant]	
VSTR	D _d , [R _n]	D _d → memory64[R _n]	3
VSTR	D _d , [R _n , constant]	D _d → memory64[R _n + constant]	
VSTmia	R _n !, {FP register list}	FP registers → memory, R _n = lowest address; Updates R _n if write-back flag (!) is included.	1 + #registers
VSTmdb	R _n !, {FP register list}	FP registers → memory, R _n -4 = highest address; Must append (!) and always updates R _n	

Cortex-M4F Instructions used in *ARM Assembly for Embedded Applications* (ISBN 978-1-09254-223-4)

Floating-Point Convert Representation	Operation	Clock Cycles
VCVT.F32.U32 S_d, S_m	$S_d \leftarrow (\text{float}) S_m$, where S_m is an unsigned integer	1
VCVT.F32.S32 S_d, S_m	$S_d \leftarrow (\text{float}) S_m$, where S_m is a 2's comp integer	
VCVT{R}.U32.F32 S_d, S_m	$S_d \leftarrow (\text{uint32_t}) S_m$	
VCVT{R}.S32.F32 S_d, S_m	$S_d \leftarrow (\text{int32_t}) S_m$	
Rounded if suffix "R" is appended using current rounding mode (FPSCR bits 23 and 22, default is nearest even)		

Floating-Point Arithmetic	Operation	Clock Cycles
VADD.F32 S_d, S_n, S_m	$S_d \leftarrow S_n + S_m$	1
VSUB.F32 S_d, S_n, S_m	$S_d \leftarrow S_n - S_m$	
VNEG.F32 S_d, S_m	$S_d \leftarrow -S_m$	
VABS.F32 S_d, S_m	$S_d \leftarrow S_m $; (clears FPU sign bit, N)	
VMUL.F32 S_d, S_n, S_m	$S_d \leftarrow S_n \times S_m$	
VDIV.F32 S_d, S_n, S_m	$S_d \leftarrow S_n \div S_m$	14
VSQRT.F32 S_d, S_m	$S_d \leftarrow \sqrt{S_m}$	
VMLA.F32 S_d, S_n, S_m	$S_d \leftarrow S_d + S_n \times S_m$	3
VMLS.F32 S_d, S_n, S_m	$S_d \leftarrow S_d - S_n \times S_m$	

Floating-Point Compare	Operation	Clock Cycles
VCMP.F32 S_d, S_m	Computes $S_d - S_m$ and updates <i>FPU Flags</i> in FPSCR	1
VCMP.F32 $S_d, \#0.0$	Computes $S_d - 0$ and updates <i>FPU Flags</i> in FPSCR	
VMRS	APSR_nzcv, FPSCR $\text{Core CPU Flags} \leftarrow \text{FPU Flags}$ (Needed between VCMF.F32 and conditional branch)	

Addressing Modes for floating-point load and store instructions (VLDR & VSTR):

Addressing Mode	Syntax	Meaning	Example
Immediate Offset	$[R_n]$	$\text{address} = R_n$	$[R5]$
	$[R_n, \text{constant}]$	$\text{address} = R_n + \text{constant}$	$[R5, 100]$

Shift Codes:

Any of these may be applied as the "shift" option of "operand2" in Move / Add / Subtract, Compare, and Bitwise Groups.

Shift Code	Meaning	Notes
LSL <i>constant</i>	Logical Shift Left by <i>constant</i> bits	Zero fills; $0 \leq \text{constant} \leq 31$
LSR <i>constant</i>	Logical Shift Right by <i>constant</i> bits	Zero fills; $1 \leq \text{constant} \leq 32$
ASR <i>constant</i>	Arithmetic Shift Right by <i>constant</i> bits	Sign extends; $1 \leq \text{constant} \leq 32$
ROR <i>constant</i>	ROtate Right by <i>constant</i> bits	$1 \leq \text{constant} \leq 32$
RRX	Rotate Right eXtended (with carry) by 1 bit	

Cortex-M4F Instructions used in *ARM Assembly for Embedded Applications* (ISBN 978-1-09254-223-4)

Addressing Modes for *integer* load and store instructions (LDR, STR, etc.):

Any of these may be used with all variations of LDR/STR except LDRD/STRD, which may not use Register Offset Mode.

Addressing Mode	Syntax	Meaning	Example
Immediate Offset	[R _n]	$address = R_n$	[R5]
	[R _n ,constant]	$address = R_n + constant$	[R5,100]
Register Offset	[R _n ,R _m]	$address = R_n + R_m$	[R4,R5]
	[R _n ,R _m ,LSL constant]	$address = R_n + (R_m \ll constant)$	[R4,R5,LSL 3]
Pre-Indexed	[R _n ,constant]!	$R_n \leftarrow R_n + constant; address = R_n$	[R5,100]!
Post-Indexed	[R _n],constant	$address = R_n; R_n \leftarrow R_n + constant$	[R5],100

Condition Codes:

If appended to an FPU instruction within an IT block, the condition code precedes any extension. (E.g., VADDGT.F32)

Condition Code	CMP Meaning	VCMP Meaning	Requirements
EQ (Equal)	==	==	Z = 1
NE (Not Equal)	!=	!= or unordered	Z = 0
HS (Higher or Same)	unsigned ≥	≥ or unordered	C = 1 <i>Note: Synonym for “CS” (Carry Set)</i>
LO (Lower)	unsigned <	<	C = 0 <i>Note: Synonym for “CC” (Carry Clear)</i>
HI (Higher)	unsigned >	> or unordered	C = 1 && Z = 0
LS (Lower or Same)	unsigned ≤	≤	C = 0 Z = 1
GE (Greater Than or Equal)	signed ≥	≥	N = V
LT (Less Than)	signed <	< or unordered	N ≠ V
GT (Greater Than)	signed >	>	Z = 0 && N = V
LE (Less Than or Equal)	signed ≤	≤ or unordered	Z = 1 N ≠ V
CS (Carry Set)	unsigned ≥	≥ or unordered	C = 1 <i>Note: Synonym for “HS” (Higher or Same)</i>
CC (Carry Clear)	unsigned <	<	C = 0 <i>Note: Synonym for “LO” (Lower)</i>
MI (Minus)	negative	<	N = 1
PL (Plus)	non-negative	≥ or unordered	N = 0
VS (Overflow Set)	overflow	unordered	V = 1
VC (Overflow Clear)	no overflow	not unordered	V = 0
AL (Always)	unconditional	unconditional	Always true

- Notes:
1. This is only a partial list of the most commonly-used ARM Cortex-M4 instructions.
 2. Clock Cycle counts do not include delays due to stalls when an instruction must wait for the previous instruction to complete.
 3. There are magnitude restrictions on immediate constants; see ARM documentation for more information.