Cortex-M4F Instructions used in *ARM Assembly for Embedded Applications* (ISBN 978-1-54390-804-6)

Function Call/Return		Operation	Notes	Clock Cycles
BL	*label*	LR ← *return address;* PC ← *address of label*	Function call	2-4
BLX	R_n	LR ← *return address;* PC ← R_n		
BX	R_n	PC ← R_n	BX LR used as function return	
B	*label*	PC ← *address of label*		

Load Integer Constant		Operation	Flags	Notes	Clock Cycles
ADR	R_d,*label*	R_d ← *address of label*		PC-4095 ≤ *address* ≤ PC+4095	1
MOVS	R_d, *constant*	R_d ← *constant*	NZ	0≤*constant*≤255 (FF_{16}) & a few others	
MVN{S}	R_d, *constant*	R_d ← ~*constant*	NZ	0≤*constant*≤255 (FF_{16}) & a few others	
MOVW	R_d, *constant*	R_d ← *constant*		0≤*constant*≤65535 ($FFFF_{16}$)	
MOVT	R_d, *constant*	R_d<31..16> ← *constant*		0≤*constant*≤65535 ($FFFF_{16}$)	

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

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*	Clock Cycles
MOV{S}	R_d,R_n	R_d ← R_n	NZ	*Options:* 1. *Constant (0-255)* 2. *register* 3. *register,shift*	1
ADD{S}	R_d,R_n,*operand2*	R_d ← R_n + *operand2*	NZCV		
ADC{S}	R_d,R_n,*operand2*	R_d ← R_n + *operand2* + C	NZCV		
SUB{S}	R_d,R_n,*operand2*	R_d ← R_n − *operand2*	NZCV		
SBC{S}	R_d,R_n,*operand2*	R_d ← R_n − *operand2* + C − 1	NZCV		
RSB{S}	R_d,R_n,*operand2*	R_d ← *operand2* − R_n	NZCV		

Cortex-M4F Instructions used in *ARM Assembly for Embedded Applications* (ISBN 978-1-54390-804-6)

Multiply / Divide		Operation	Flags	Notes	Clock Cycles
MUL{S}	R_d,R_n,R_m	$R_d \leftarrow (R_n \times R_m)$<31..0>	NZC	32 ← 32×32; C←undefined	1
MLA	R_d,R_n,R_m,R_a	$R_d \leftarrow R_a + (R_n \times R_m)$<31..0>		32 ← 32 + 32×32	
MLS	R_d,R_n,R_m,R_a	$R_d \leftarrow R_a - (R_n \times R_m)$<31..0>		32 ← 32 - 32×32	
SMMUL	R_d,R_n,R_m	$R_d \leftarrow (R_n \times R_m)$<63..32>		Uses upper half of signed 64-bit product	
SMMLA	R_d,R_n,R_m,R_a	$R_d \leftarrow (R_n \times R_m)$<63..32> + R_a			
SMMLS	R_d,R_n,R_m,R_a	$R_d \leftarrow (R_n \times R_m)$<63..32> − R_a			
[S/U]MULL	R_{dlo},R_{dhi},R_n,R_m	$R_{dhi}R_{dlo} \leftarrow R_n \times R_m$		**S**igned/**U**nsigned: 64 ← 32x32	
[S/U]MLAL	R_{dlo},R_{dhi},R_n,R_m	$R_{dhi}R_{dlo} \leftarrow R_{dhi}R_{dlo} + R_n \times R_m$		**S**igned/**U**nsigned: 64 ← 64 + 32×32	
[S/U]DIV	R_d,R_n,R_m	$R_d \leftarrow R_n / R_m$		**S**igned/**U**nsigned: 32 ← 32÷32	2-12

Saturating Instructions		Operation	Min	Max	Notes	Clock Cycles
SSAT	R_d,n,*operand2*	$R_d \leftarrow$ *operand2*	-2^{n-1}	$2^{n-1}-1$	*operand2*: R_m or R_m,ASR *constant* or 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}	$2^{31}-1$		
QSUB	R_d,R_n,R_m	$R_d \leftarrow R_n - R_m$				

SIMD Saturating ADD/SUB		Operation	Min to Max	Notes	Clock Cycles
[Q/UQ]ADD8	R_d,R_n,R_m	R_d[bits] ← R_n[bits] + R_m[bits]	Q: -2^7 to 2^7-1 UQ: 0 to 2^7-1	For bytes 0 through 3: bits 7..0, 15..8, 23..16, and 31..24	1
[Q/UQ]SUB8	R_d,R_n,R_m	R_d[bits] ← R_n[bits] - R_m[bits]			
[Q/UQ]ADD16	R_d,R_n,R_m	R_d[bits] ← R_n[bits] + R_m[bits]	Q: -2^{15} to $2^{15}-1$ UQ: 0 to $2^{16}-1$	For halfwords 0 and 1: bits 15..0 and 31..16	
[Q/UQ]SUB16	R_d,R_n,R_m	R_d[bits] ← R_n[bits] - R_m[bits]			

SIMD Non-Saturating ADD/SUB		Operation	GE Flags	Notes	Clock Cycles
[S/U]ADD8	R_d,R_n,R_m	R_d[bits] ← R_n[bits] + R_m[bits]	S: sum ≥ 0 ? 1 : 0 U: sum ≥ 2^8 ? 1 : 0	For bytes 0 through 3: bits 7..0, 15..8, 23..16, and 31..24	1
[S/U]SUB8	R_d,R_n,R_m	R_d[bits] ← R_n[bits] - R_m[bits]	S: diff ≥ 0 ? 1 : 0 U: diff ≥ 0 ? 1 : 0		
[S/U]ADD16	R_d,R_n,R_m	R_d[bits] ← R_n[bits] + R_m[bits]	S: sum ≥ 0 ? 11 : 00 U: sum ≥ 2^{16} ? 11 : 00	For halfwords 0 and 1: bits 15..0 and 31..16	
[S/U]SUB16	R_d,R_n,R_m	R_d[bits] ← R_n[bits] - R_m[bits]	S: diff ≥ 0 ? 11 : 00 U: diff ≥ 0 ? 11 : 00		

Cortex-M4F Instructions used in *ARM Assembly for Embedded Applications* (ISBN 978-1-54390-804-6)

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 ← R_n<15..00>×R_m<15..00> + R_n<31..16>×R_m<31..16>	Sets Q flag if an addition or subtraction overflows; does <u>not</u> saturate.	1
SMUSD	R_d,R_n,R_m	R_d ← R_n<15..00>×R_m<15..00> − R_n<31..16>×R_m<31..16>		
SMLAD	R_d,R_n,R_m,R_a	R_d ← R_a + R_n<15..00>×R_m<15..00> + R_n<31..16>×R_m<31..16>		
SMLSD	R_d,R_n,R_m,R_a	R_d ← R_a + R_n<15..00>×R_m<15..00> − R_n<31..16>×R_m<31..16>		
SMLALD	R_{dlo},R_{dhi},R_n,R_m	R_{dhi}.R_{dlo} += Rn<15..00>×R_m<15..00> + Rn<31..16>×R_m<31..16>		
SMLSLD	R_{dlo},R_{dhi},R_n,R_m	R_{dhi}.R_{dlo} += Rn<15..00>×R_m<15..00> − Rn<31..16>×R_m<31..16>		

Appending "X" to instruction mnemonic changes operand2s to Rn<15..00>×Rm<31..16> and Rn<31..16> × Rm<15..00>.

Signed Multiply Halfwords		*Operation*	*Notes*	*Clock Cycles*
SMULBB	R_d,R_n,R_m	R_d ← R_n<15..00> × R_m<15..00>	32 ← 16×16	1
SMULBT	R_d,R_n,R_m	R_d ← R_n<15..00> × R_m<31..16>		
SMULTB	R_d,R_n,R_m	R_d ← R_n<31..16> × R_m<15..00>		
SMULTT	R_d,R_n,R_m	R_d ← R_n<31..16> × R_m<31..16>		

Pack Halfwords		*Operation*	*operand2*	*Notes*	*Clock Cycles*
PKHBT	R_d,R_n,*operand2*	**B**tm: R_d<15..00> ← R_n<15..00> **T**op: R_d<31..16> ← *operand2*<31..16>	*Options:* 1. *register* 2. *register,LSL constant* 3. *register,ASR constant*		1
PKHTB	R_d,R_n,*operand2*	**T**op: R_d<31..16> ← R_n<31..16> **B**tm: R_d<15..00> ← *operand2*<15..00>			

Compare Instructions		*Operation*	*operand2*	*Notes*	*Clock Cycles*
CMP	R_n,*operand2*	R_n − *operand2*	*Options:* 1. *constant (0-255)* 2. *register* 3. *register,shift*	Updates: NZCV	1
CMN	R_n,*operand2*	R_n + *operand2*		Updates: NZCV	
TST	R_n,*operand2*	R_n & *operand2*		Updates: NZC	
TEQ	R_n,*operand2*	R_n ^ *operand2*		Updates: NZC	

Cortex-M4F Instructions used in *ARM Assembly for Embedded Applications* (ISBN 978-1-54390-804-6)

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 $R_n \neq 0$	Can't use in an IT block	
IT$c_1c_2c_3$	*condition code*	Each c_i is one of T, E, or *empty*	Controls 1-4 instructions	1

Shift Instructions		Operation	Flags	*operand2*	Notes	Clock Cycles
ASR{S}	R_d,R_n,*operand2*	$R_d \leftarrow R_n >> operand2$ (arithmetic shift right)	NZC	*Options:* 1. *constant* 2. *register*	Sign extends	1
LSL{S}	R_d,R_n,*operand2*	$R_d \leftarrow R_n << operand2$ (logical shift left)	NZC		Zero fills	
LSR{S}	R_d,R_n,*operand2*	$R_d \leftarrow R_n >> operand2$ (logical shift right)	NZC			
ROR{S}	R_d,R_n,*operand2*	$R_d \leftarrow R_n >> operand2$ (rotate right)	NZC		right rotate	
RRX{S}	R_d,R_n	$R_d \leftarrow R_n >> 1$; R_d<31> ← C; C ← R_n<0>	NZC		33-bit rotate w/C	

Bitwise Instructions		Operation	Flags	*operand2*	Notes	Clock Cycles
AND{S}	R_d,R_n,*operand2*	$R_d \leftarrow R_n$ & *operand2*	NZC	*Options:* 1. *constant (0-255)* 2. *register* 3. *register,shift*		1
ORR{S}	R_d,R_n,*operand2*	$R_d \leftarrow R_n$ | *operand2*	NZC			
EOR{S}	R_d,R_n,*operand2*	$R_d \leftarrow R_n$ ^ *operand2*	NZC			
BIC{S}	R_d,R_n,*operand2*	$R_d \leftarrow R_n$ & ~*operand2*	NZC			
ORN{S}	R_d,R_n,*operand2*	$R_d \leftarrow R_n$ | ~*operand2*	NZC			
MVN{S}	R_d,*operand2*	$R_d \leftarrow$ ~*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*	$R_d \leftarrow$ *SelectedBitfieldOf*(R_n)	Sign extends	
UBFX	R_d,R_n,*lsb,width*	$R_d \leftarrow$ *SelectedBitfieldOf*(R_n)	Zero extends	

Bits / Bytes / Words		Operation	Notes	Clock Cycles
CLZ	R_d,R_n	$R_d \leftarrow$ *CountLeadingZeroesOf*(R_n)	#leading 0's = 0-32	1
RBIT	R_d,R_n	$R_d \leftarrow$ *ReverseBitOrderOf*(R_n)		
REV	R_d,R_n	$R_d \leftarrow$ *ReverseByteOrderOf*(R_n)		

Pseudo-Instructions		Operation	Flags	Replaced by	Clock Cycles
LDR	R_d,=*constant*	$R_d \leftarrow$ *constant*	MOVS:NZ	MOV, MOVS, MVN, MOVW, or LDR	1
NEG	R_d,R_n	$R_d \leftarrow -R_n$	NZCV	RSBS R_d,R_n,0	
CPY	R_d,R_n	$R_d \leftarrow R_n$		MOV R_d,R_n	

Cortex-M4F Instructions used in *ARM Assembly for Embedded Applications* (ISBN 978-1-54390-804-6)

Floating-Point PUSH/POP		***Operation***	***Clock Cycles***
VPUSH	{*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			***Clock Cycles***
VMOV	S_d,*fpconstant*	*fpconstant* must be $\pm m \times 2^{-n}$, $(16 \le m \le 31 \ \& \ 0 \le n \le 7)$	1

Floating-Point Copy Registers		***Operation***	***Clock Cycles***
VMOV	S_d,S_m	$S_d \leftarrow S_m$	1
VMOV	R_d,S_m	$R_d \leftarrow S_m$	
VMOV	S_d,R_m	$S_d \leftarrow R_m$	
VMOV	R_t,R_{t2},S_m,S_{m+1}	$R_t \leftarrow S_m$; $R_{t2} \leftarrow S_{m+1}$ (S_m, S_{m+1} adjacent regs)	2
VMOV	S_m,S_{m+1},R_t,R_{t2}	$S_m \leftarrow R_t$; $S_{m+1} \leftarrow R_{t2}$ (S_m, S_{m+1} adjacent regs)	

Floating-Point Load Registers		***Operation***	***Clock Cycles***
VLDR	S_d,[R_n]	$S_d \leftarrow$ *memory32*[R_n]	2
VLDR	S_d,[R_n,*constant*]	$S_d \leftarrow$ *memory32*[R_n + *constant*]	
VLDR	S_d,*label*	$S_d \leftarrow$ *memory32*[Address of *label*]	
VLDR	D_d,[R_n]	$D_d \leftarrow$ *memory64*[R_n]	3
VLDR	D_d,[R_n,*constant*]	$D_d \leftarrow$ *memory64*[R_n + *constant*]	
VLDR	D_d,*label*	$D_d \leftarrow$ *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]	$S_d \rightarrow$ *memory32*[R_n]	2
VSTR	S_d,[R_n,*constant*]	$S_d \rightarrow$ *memory32*[R_n + *constant*]	
VSTR	D_d,[R_n]	$D_d \rightarrow$ *memory64*[R_n]	3
VSTR	D_d,[R_n,*constant*]	$D_d \rightarrow$ *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 978-1-54390-804-6)

Floating-Point Convert Representation		Operation	Clock Cycles
VCVT.F32.U32	S_d,S_m	S_d ← (float) S_m, *where S_m is an unsigned integer*	1
VCVT.F32.S32	S_d,S_m	S_d ← (float) S_m, *where S_m is a 2's comp integer*	
VCVT{R}.U32.F32	S_d,S_m	S_d ← (uint32_t) S_m, *rounded if suffix "R" is appended*	
VCVT{R}.S32.F32	S_d,S_m	S_d ← (int32_t) S_m, *rounded if suffix "R" is appended*	

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	$S_d \leftarrow \lvert S_m \rvert$; *(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 *FPU Flags* in FPSCR	1
VCMP.F32	S_d,#0.0	Computes S_d - 0 and updates *FPU Flags* in FPSCR	
VMRS	APSR_nzcv,FPSCR	*Core CPU Flags* ← *FPU Flags* (Needed between VCMP.F32 and conditional branch)	

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

Addressing Mode	Syntax	Meaning	Example
Immediate Offset	[R_n]	*address* = R_n	[R5]
	[R_n,*constant*]	*address* = R_n + *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 *constant*	**L**ogical **S**hift **L**eft by *constant* bits	Zero fills; 0 ≤ *constant* ≤ 31
LSR *constant*	**L**ogical **S**hift **R**ight by *constant* bits	Zero fills; 1 ≤ *constant* ≤ 32
ASR *constant*	**A**rithmetic **S**hift **R**ight by *constant* bits	Sign extends; 1 ≤ *constant* ≤ 32
ROR *constant*	**RO**tate **R**ight by *constant* bits	1 ≤ *constant* ≤ 32
RRX	**R**otate **R**ight e**X**tended (with carry) by 1 bit	

Cortex-M4F Instructions used in *ARM Assembly for Embedded Applications* (ISBN 978-1-54390-804-6)

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 << *constant*)	[R4,R5,LSL 3]
Pre-Indexed	[R_n,*constant*]!	R_n ← R_n + *constant*; *address* = R_n	[R5,100]!
Post-Indexed	[R_n],*constant*	*address* = R_n; R_n ← 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., VADD**GT**.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 *Note: Synonym for "CS" (Carry Set)*
LO (Lower)	unsigned <	<	C = 0 *Note: Synonym for "CC" (Carry Clear)*
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 *Note: Synonym for "HS" (Higher or Same)*
CC (Carry Clear)	unsigned <	<	C = 0 *Note: Synonym for "LO" (Lower)*
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.