Instruction Set Summary

A.1 Instructions available for both Cortex®-M3 and Cortex-M4

A.1.1 Note

This section is extracted from Cortex®-M3/M4 Devices Generic User Guide with permission from ARM® Ltd.

Note:

- Angle brackets, <>, enclose alternative forms of the operand.
- Braces, {}, enclose optional operands.
- The Operands column is not exhaustive.
- Op2 is a flexible second operand that can be either a register or a constant.
- Most instructions can use an optional condition code suffix.

A.1.2 Flexible second operand (Op2)

Some of the instructions support the Flexible second operand (Op2).

1) `Operand2` *can be a constant:* *`#constant`*

Where *`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 the constants shown above, X and Y are hexadecimal digits.

In addition, in a small number of instructions, `constant` can take a wider range of values. Please refer to the individual instruction descriptions in the Cortex®-M3/M4 Devices Generic User Guide.

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.

2) `Operand2` ***can be a register with optional shift:*** *`Rm {, shift}`*

Where *`Rm`* specifies the register holding the data for the second operand, and *`shift`* is an optional shift to be applied to Rm. It can be one of:

`Shift` **Expression**	**Descriptions**
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.
-	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.

A.1.3 Instruction list

Mnemonic	Operands	Brief Description	Flags
ADC, ADCS	{Rd,} Rn, Op2	Add with Carry	N,Z,C,V
ADD, ADDS	{Rd,} Rn, Op2	Add	N,Z,C,V
ADD, ADDW	{Rd,} Rn, #imm12	Add	-
ADR	Rd, label	Load PC-relative Address (add immediate value with (Align(PC, 4)))	-
AND, ANDS	{Rd,} Rn, Op2	Logical AND	N,Z,C
ASR, ASRS	Rd, Rm, <Rs|#n>	Arithmetic Shift Right	N,Z,C
B	label	Branch	-
BFC	Rd, #lsb, #width	Bit Field Clear	-
BFI	Rd, Rn, #lsb, #width	Bit Field Insert	-
BIC, BICS	{Rd,} Rn, Op2	Bit Clear	N,Z,C
BKPT	#imm	Breakpoint	-
BL	label	Branch with Link	-
BLX	Rm	Branch indirect with Link	-
BX	Rm	Branch indirect	-
CBNZ	Rn, label	Compare and Branch if Non-Zero	-
CBZ	Rn, label	Compare and Branch if Zero	-
CLREX	-	Clear Exclusive	-

Mnemonic	Operands	Brief Description	Flags
CLZ	Rd, Rm	Count Leading Zeros	-
CMN	Rn, Op2	Compare Negative	N,Z,C,V
CMP	Rn, Op2	Compare	N,Z,C,V
CPSID	i	Change Processor State, Disable Interrupts (set PRIMASK)	-
CPSIE	i	Change Processor State, Enable Interrupts (clear PRIMASK)	-
CPSID	f	Change Processor State, Disable Interrupts (set FAULTMASK)	-
CPSIE	f	Change Processor State, Enable Interrupts (clear FAULTMASK)	-
DMB	-	Data Memory Barrier	-
DSB	-	Data Synchronization Barrier	-
EOR, EORS	{Rd,} Rn, Op2	Exclusive OR	N,Z,C
ISB	-	Instruction Synchronization Barrier	-
IT	<cond>	If-Then condition block	-
LDM	Rn{!}, reglist	Load Multiple registers, increment after	-
LDMDB, LDMEA	Rn{!}, reglist	Load Multiple registers, decrement before	-
LDMFD, LDMIA	Rn{!}, reglist	Load Multiple registers, increment after	-
LDR	Rt, [Rn, #offset]	Load Register with word	-
LDRB, LDRBT	Rt, [Rn, #offset]	Load Register with byte	-
LDRD	Rt, Rt2, [Rn, #offset]	Load Register with two words	-
LDREX	Rt, [Rn, #offset]	Load Register Exclusive	-
LDREXB	Rt, [Rn]	Load Register Exclusive with Byte	-
LDREXH	Rt, [Rn]	Load Register Exclusive with Half-word	-
LDRH, LDRHT	Rt, [Rn, #offset]	Load Register with Half-word	-
LDRSB, LDRSBT	Rt, [Rn, #offset]	Load Register with Signed Byte	-
LDRSH, LDRSHT	Rt, [Rn, #offset]	Load Register with Signed Half-word	-
LDRT	Rt, [Rn, #offset]	Load Register with word	-
LSL, LSLS	Rd, Rm, <Rs|#n>	Logical Shift Left	N,Z,C

(*Continued*)

Mnemonic	Operands	Brief Description	Flags
LSR, LSRS	Rd, Rm, <Rs|#n>	Logical Shift Right	N,Z,C
MLA	Rd, Rn, Rm, Ra	Multiply with Accumulate, 32-bit result	-
MLS	Rd, Rn, Rm, Ra	Multiply and Subtract, 32-bit result	-
MOV	Rd, Op2	Move	-
MOVS	Rd, Op2	Move with APSR update	N,Z,C
MOVT	Rd, #imm16	Move Top	-
MOVW, MOV	Rd, #imm16	Move 16-bit constant	N,Z,C
MRS	Rd, spec_reg	Move from Special Register to general register	-
MSR	spec_reg, Rm	Move from general register to Special Register	- /N,Z,C,V
MUL, MULS	{Rd,} Rn, Rm	Multiply, 32-bit result	N,Z
MVN, MVNS	Rd, Op2	Move NOT	N,Z,C
NOP	-	No Operation	-
ORN, ORNS	{Rd,} Rn, Op2	Logical OR NOT	N,Z,C
ORR, ORRS	{Rd,} Rn, Op2	Logical OR	N,Z,C
POP	reglist	Pop registers from stack	-
PUSH	reglist	Push registers onto stack	-
RBIT	Rd, Rn	Reverse Bits	-
REV	Rd, Rn	Reverse byte order in a word	-
REV16	Rd, Rn	Reverse byte order in each half-word	-
REVSH	Rd, Rn	Reverse byte order in bottom half-word and sign extend	-
ROR, RORS	Rd, Rm, <Rs|#n>	Rotate Right	N,Z,C
RRX, RRXS	Rd, Rm	Rotate Right with Extend	N,Z,C
RSB, RSBS	{Rd,} Rn, Op2	Reverse Subtract	N,Z,C,V
SBC, SBCS	{Rd,} Rn, Op2	Subtract with Carry	N,Z,C,V
SBFX	Rd, Rn, #lsb, #width	Signed Bit Field Extract	-
SDIV	{Rd,} Rn, Rm	Signed Divide	-
SEV	-	Send Event	-
SMLAL	RdLo, RdHi, Rn, Rm	Signed Multiply with Accumulate (32 x 32 + 64), 64-bit result	-
SMULL	RdLo, RdHi, Rn, Rm	Signed Multiply (32 x 32), 64-bit result	-
SSAT	Rd, #n, Rm {,shift #s}	Signed Saturate	Q
STM	Rn{!}, reglist	Store Multiple registers, increment after	-
STMDB, STMEA	Rn{!}, reglist	Store Multiple registers, decrement before	-

Mnemonic	Operands	Brief Description	Flags
STMFD, STMIA	Rn{!}, reglist	Store Multiple registers, increment after	-
STR	Rt, [Rn, #offset]	Store Register word	-
STRB, STRBT	Rt, [Rn, #offset]	Store Register byte	-
STRD	Rt, Rt2, [Rn, #offset]	Store Register two words	-
STREX	Rd, Rt, [Rn, #offset]	Store Register Exclusive	-
STREXB	Rd, Rt, [Rn]	Store Register Exclusive Byte	-
STREXH	Rd, Rt, [Rn]	Store Register Exclusive Half-word	-
STRH, STRHT	Rt, [Rn, #offset]	Store Register Half-word	-
STRT	Rt, [Rn, #offset]	Store Register word	-
SUB, SUBS	{Rd,} Rn, Op2	Subtract	N,Z,C,V
SUB, SUBW	{Rd,} Rn, #imm12	Subtract	N,Z,C,V
SVC	#imm	Supervisor Call	-
SXTB	{Rd,} Rm {,ROR #n}	Sign extend a byte	-
SXTH	{Rd,} Rm {,ROR #n}	Sign extend a half-word	-
TBB	[Rn, Rm]	Table Branch Byte	-
TBH	[Rn, Rm, LSL #1]	Table Branch Half-word	-
TEQ	Rn, Op2	Test Equivalence	N,Z,C
TST	Rn, Op2	Test	N,Z,C
UBFX	Rd, Rn, #lsb, #width	Unsigned Bit Field Extract	-
UDIV	{Rd,} Rn, Rm	Unsigned Divide	-
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply with Accumulate (32 x 32 + 64), 64-bit result	-
UMULL	RdLo, RdHi, Rn, Rm	Unsigned Multiply (32 x 32), 64-bit result	-
USAT	Rd, #n, Rm {,shift #s}	Unsigned Saturate	Q
UXTB	{Rd,} Rm {,ROR #n}	Zero extend a Byte	-
UXTH	{Rd,} Rm {,ROR #n}	Zero extend a Half-word	-
WFE	-	Wait For Event	-
WFI	-	Wait For Interrupt	-

A.2 Instructions available for Cortex®-M4

SIMD and saturating instructions:

Mnemonic	Operands	Brief Description	Flags	Figure
SADD8	{Rd,} Rn, Rm	Signed Add 8	GE [3:0]	B.13
SADD16	{Rd,} Rn, Rm	Signed Add 16	GE [3:0]	B.14
SSUB8	{Rd,} Rn, Rm	Signed Subtract 8	GE [3:0]	B.17
SSUB16	{Rd,} Rn, Rm	Signed Subtract 16	GE [3:0]	B.18
SASX	{Rd,} Rn, Rm	Signed Add and Subtract with Exchange	GE [3:0]	B.21
SSAX	{Rd,} Rn, Rm	Signed Subtract and Add with Exchange	GE [3:0]	B.22
QADD8	{Rd,} Rn, Rm	Saturating Add 8	Q	B.5
QADD16	{Rd,} Rn, Rm	Saturating Add 16	Q	B.4
QSUB8	{Rd,} Rn, Rm	Saturating Subtract 8	Q	B.9
QSUB16	{Rd,} Rn, Rm	Saturating Subtract 16	Q	B.8
QASX	{Rd,} Rn, Rm	Saturating Add and Subtract with Exchange	Q	B.10
QSAX	{Rd,} Rn, Rm	Saturating Subtract and Add with Exchange	Q	B.11
SHADD8	{Rd,} Rn, Rm	Signed Halving Add 8		B.15
SHADD16	{Rd,} Rn, Rm	Signed Halving Add 16		B.16
SHSUB8	{Rd,} Rn, Rm	Signed Halving Subtract 8		B.19
SHSUB16	{Rd,} Rn, Rm	Signed Halving Subtract 16		B.20
SHASX	{Rd,} Rn, Rm	Signed Halving Add and Subtract with Exchange		B.23
SHSAX	{Rd,} Rn, Rm	Signed Halving Subtract and Add with Exchange		B.24
UADD8	{Rd,} Rn, Rm	Unsigned Add 8	GE [3:0]	B.69
UADD16	{Rd,} Rn, Rm	Unsigned Add 16	GE [3:0]	B.70
USUB8	{Rd,} Rn, Rm	Unsigned Subtract 8	GE [3:0]	B.73
USUB16	{Rd,} Rn, Rm	Unsigned Subtract 16	GE [3:0]	B.74
UASX	{Rd,} Rn, Rm	Unsigned Add and Subtract with Exchange	GE [3:0]	B.77

Mnemonic	Operands	Brief Description	Flags	Figure
USAX	{Rd,} Rn, Rm	Unsigned Subtract and Add with Exchange	GE [3:0]	B.78
UQADD8	{Rd,} Rn, Rm	Unsigned Saturating Add 8	Q	B.85
UQADD16	{Rd,} Rn, Rm	Unsigned Saturating Add 16	Q	B.84
UQSUB8	{Rd,} Rn, Rm	Unsigned Saturating Subtract 8	Q	B.87
UQSUB16	{Rd,} Rn, Rm	Unsigned Saturating Subtract 16	Q	B.86
UQASX	{Rd,} Rn, Rm	Unsigned Saturating Add and Subtract with Exchange	Q	B.88
UQSAX	{Rd,} Rn, Rm	Unsigned Saturating Subtract and Add with Exchange	Q	B.89
UHADD8	{Rd,} Rn, Rm	Unsigned Halving Add 8		B.71
UHADD16	{Rd,} Rn, Rm	Unsigned Halving Add 16		B.72
UHSUB8	{Rd,} Rn, Rm	Unsigned Halving Subtract 8		B.75
UHSUB16	{Rd,} Rn, Rm	Unsigned Halving Subtract 16		B.76
UHASX	{Rd,} Rn, Rm	Unsigned Halving Add and Subtract with Exchange		B.79
UHSAX	{Rd,} Rn, Rm	Unsigned Halving Subtract and Add with Exchange		B.80
USAD8	{Rd,} Rn, Rm	Unsigned Sum of Absolute Differences		B.81
USADA8	{Rd,} Rn, Rm, Ra	Unsigned Sum of Absolute Differences and Accumulate		B.82
USAT16	Rd, #imm, Rn	Unsigned saturate 2 signed 16-bit values	Q	B.83
SSAT16	Rd, #imm, Rn	Signed saturate 2 signed 16-bit values	Q	B.62
SEL	{Rd,} Rn, Rm	Select bytes base on GE bits		B.25
USAT	{Rd,} #imm, Rn {, LSL #n} {Rd,} #imm, Rn {, ASR #n}	Unsigned saturate (optionally shifted) value	Q	5.12
SSAT	{Rd,} #imm, Rn {, LSL #n} {Rd,} #imm, Rn {, ASR #n}	Signed saturate (optionally shifted) value	Q	5.11
QADD	{Rd,} Rn, Rm	Saturating Add	Q	B.3
QDADD	{Rd,} Rn, Rm	Saturating double and Add	Q	B.6
QSUB	{Rd,} Rn, Rm	Saturating Subtract	Q	B.7
QDSUB	{Rd,} Rn, Rm	Saturating double and Subtract	Q	B.12

Multiply and MAC (Multiply and Accumulate) instructions:

Mnemonic	Operands	Brief Description	Flags	
MUL{S}	{Rd,} Rn, Rm	Unsigned Multiply, 32-bit result	N, Z	
SMULL	RdLo, RdHi, Rn, Rm	Signed Multiply, 64-bit result		B.26
SMLAL	RdLo, RdHi, Rn, Rm	Signed Multiply and Accumulate, 64-bit result		B.27
UMULL	RdLo, RdHi, Rn, Rm	Unsigned Multiply, 64-bit result		B.90
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply and Accumulate, 64-bit result		B.91
UMAAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply Accumulate Accumulate		B.92
SMULBB	{Rd,} Rn, Rm	Signed Multiply (half-words)		B.28
SMULBT	{Rd,} Rn, Rm	Signed Multiply (half-words)		B.29
SMULTB	{Rd,} Rn, Rm	Signed Multiply (half-words)		B.30
SMULTT	{Rd,} Rn, Rm	Signed Multiply (half-words)		B.31
SMLABB	Rd, Rn, Rm, Ra	Signed Multiply Accumulate (half-words)	Q	B.36
SMLABT	Rd, Rn, Rm, Ra	Signed Multiply Accumulate (half-words)	Q	B.37
SMLATB	Rd, Rn, Rm, Ra	Signed Multiply Accumulate (half-words)	Q	B.38
SMLATT	Rd, Rn, Rm, Ra	Signed Multiply Accumulate (half-words)	Q	B.39
SMULWB	Rd, Rn, Rm, Ra	Signed Multiply (word by half-word)		B.40
SMULWT	Rd, Rn, Rm, Ra	Signed Multiply (word by half-word)		B.41
SMLAWB	Rd, Rn, Rm, Ra	Signed Multiply Accumulate (word by half-word)	Q	B.42
SMLAWT	Rd, Rn, Rm, Ra	Signed Multiply Accumulate (word by half-word)	Q	B.43
SMMUL	{Rd,} Rn, Rm	Signed Most significant word Multiply		B.32
SMMULR	{Rd,} Rn, Rm	Signed Most significant word Multiply with rounded result		B.33
SMMLA	Rd, Rn, Rm, Ra	Signed Most significant word Multiply Accumulate		B.34

Mnemonic	Operands	Brief Description	Flags	
SMMLAR	Rd, Rn, Rm, Ra	Signed Most significant word Multiply Accumulate with rounded result		B.35
SMMLS	Rd, Rn, Rm, Ra	Signed Most significant word Multiply Subtract		B.44
SMMLSR	Rd, Rn, Rm, Ra	Signed Most significant word Multiply Subtract with rounded result		B.45
SMLALBB	RdLo, RdHi, Rn, Rm	Signed Multiply Accumulate Long (half-words)		B.46
SMLALBT	RdLo, RdHi, Rn, Rm	Signed Multiply Accumulate Long (half-words)		B.47
SMLALTB	RdLo, RdHi, Rn, Rm	Signed Multiply Accumulate Long (half-words)		B.48
SMLALTT	RdLo, RdHi, Rn, Rm	Signed Multiply Accumulate Long (half-words)		B.49
SMUAD	`{Rd,} Rn, Rm`	Signed Dual Multiply Add	Q	B.50
SMUADX	`{Rd,} Rn, Rm`	Signed Dual Multiply Add with eXchange	Q	B.51
SMUSD	`{Rd,} Rn, Rm`	Signed Dual Multiply Subtract		B.56
SMUSDX	`{Rd,} Rn, Rm`	Signed Dual Multiply Subtract with eXchange		B.57
SMLAD	Rd, Rn, Rm, Ra	Signed Multiply Accumulate Dual	Q	B.52
SMLADX	Rd, Rn, Rm, Ra	Signed Multiply Accumulate Dual with eXchange	Q	B.53
SMLSD	Rd, Rn, Rm, Ra	Signed Multiply Subtract Dual	Q	B.58
SMLSDX	Rd, Rn, Rm, Ra	Signed Multiply Subtract Dual with eXchange	Q	B.59
SMLALD	RdLo, RdHi, Rn, Rm	Signed Multiply Accumulate Long Dual		B.54
SMLALDX	RdLo, RdHi, Rn, Rm	Signed Multiply Accumulate Long Dual with eXchange		B.55
SMLSLD	RdLo, RdHi, Rn, Rm	Signed Multiply Subtract Long Dual		B.60
SMLSLDX	RdLo, RdHi, Rn, Rm	Signed Multiply Subtract Long Dual with eXchange		B.61

Packing and unpacking instructions:

Instructions	Operands	Descriptions	
PKHBT	{Rd,} Rn, Rm {,LSL #imm}	Pack Half-word with lower half from 1st operand and upper half from shifted 2nd operand	B.1
PKHTB	{Rd,} Rn, Rm {,ASR #imm}	Pack Half-word with upper half from 1st operand and lower half from shifted 2nd operand	B.2
SXTB	Rd,Rm {,ROR #n}	Signed eXtend Byte	B.63
SXTH	Rd,Rm {,ROR #n}	Signed eXtend Half-word	B.67
UXTB	Rd,Rm {,ROR #n}	Unsigned eXtend Byte	B.93
UXTH	Rd,Rm {,ROR #n}	Unsigned eXtend Half-word	B.97
SXTB16	Rd,Rm {,ROR #n}	Signed eXtend two bytes to two half-words	B.64
UXTB16	Rd,Rm {,ROR #n}	Unsigned eXtend two bytes to two half-words	B.94
SXTAB	{Rd,} Rn, Rm{,ROR #n}	Signed eXtend and Add byte	B.65
SXTAH	{Rd,} Rn, Rm{,ROR #n}	Signed eXtend and Add half-word	B.68
SXTAB16	{Rd,} Rn, Rm{,ROR #n}	Signed extend two bytes to half-words and dual add	B.66
UXTAB	{Rd,} Rn, Rm{,ROR #n}	Unsigned eXtend and Add byte	B.95
UXTAH	{Rd,} Rn, Rm{,ROR #n}	Unsigned eXtend and Add half-word	B.98
UXTAB16	{Rd,} Rn, Rm{,ROR #n}	Unsigned extend two bytes to half-words and dual add	B.96

A.3 Floating point instructions for Cortex®-M4

The following instructions are available to Cortex®-M4 processor with floating point unit, and after the floating point unit is enabled.

Instructions	Operands	Operations
VABS.F32	Sd, Sm	Floating point Absolute value
VADD.F32	{Sd,} Sn, Sm	Floating point Add
VCMP{E}.F32	Sd, Sm	Compare two floating point registers VCMP : Raise Invalid Operation exception if either operand is a signaling NaN VCMPE : Raise Invalid Operation exception if either operand is any types of NaN
VCMP{E}.F32	Sd, #0.0	Compare a floating point register to zero (#0.0)
VCVT.S32.F32	Sd, Sm	Convert from signed 32-bit integer to floating point (round toward zero rounding mode)

Instructions	Operands	Operations
VCVTR.S32.F32	Sd, Sm	Convert from signed 32-bit integer to floating point (use rounding mode specified by FPCSR)
VCVT.U32.F32	Sd, Sm	Convert from unsigned 32-bit integer to floating point (round toward zero rounding mode)
VCVTR.U32.F32	Sd, Sm	Convert from unsigned 32-bit integer to floating point (use rounding mode specified by FPCSR)
VCVT.F32.S32	Sd, Sm	Convert from floating point to 32-bit signed integer
VCVT.F32.U32	Sd, Sm	Convert from floating point to 32-bit unsigned integer
VCVT.S16.F32	Sd, Sd, #fbit	Convert from signed 16-bit fixed point value to floating point. #fbit range from 1 to 16 (fraction bits)
VCVT.U16.F32	Sd, Sd, #fbit	Convert from unsigned 16-bit fixed point value to floating point. #fbit range from 1 to 16 (fraction bits)
VCVT.S32.F32	Sd, Sd, #fbit	Convert from signed 32-bit fixed point value to floating point. #fbit range from 1 to 32 (fraction bits)
VCVT.U32.F32	Sd, Sd, #fbit	Convert from unsigned 32-bit fixed point value to floating point. #fbit range from 1 to 32 (fraction bits)
VCVT.F32.S16	Sd, Sd, #fbit	Convert from floating point to signed 16-bit fixed point value. #fbit range from 1 to 16 (fraction bits)
VCVT.F32.U16	Sd, Sd, #fbit	Convert from floating point to unsigned 16-bit fixed point value. #fbit range from 1 to 16 (fraction bits)
VCVT.F32.S32	Sd, Sd, #fbit	Convert from floating point to signed 32-bit fixed point value. #fbit range from 1 to 32 (fraction bits)
VCVT.F32.U32	Sd, Sd, #fbit	Convert from floating point to unsigned 32-bit fixed point value. #fbit range from 1 to 32 (fraction bits)
VCVTB.F32.F16	Sd, Sm	Convert from Single precision to Half Precision (use bottom 16-bit, upper 16-bit unaffected)
VCVTF.F32.F16	Sd, Sm	Convert from Single precision to Half Precision (use upper 16-bit, bottom 16-bit unaffected)
VCVTB.F16.F32	Sd, Sm	Convert from Half Precision (use bottom 16-bit) to Single precision
VCVTF.F16.F32	Sd, Sm	Convert from Half Precision (use upper 16-bit) to Single precision
VDIV.F32	{Sd,} Sn, Sm	Floating point Divide
VFMA.F32	Sd, Sn, Sm	Floating point Fused Multiply Accumulate Sd=Sd+(Sn*Sm)

(*Continued*)

Instructions	Operands	Operations
VFMS.F32	Sd, Sn, Sm	Floating point Fused Multiply Subtract Sd=Sd-(Sn*Sm)
VFNMA.F32	Sd, Sn, Sm	Floating point Fused Negate Multiply Accumulate Sd=(-Sd)+(Sn*Sm)
VFNMS.F32	Sd, Sn, Sm	Floating point Fused Negate Multiply Subtract Sd=(-Sd)-(Sn*Sm)
VLDMIA.32	Rn{!}, {S_regs}	Floating point Multiple Load Increment After
VLDMDB.32	Rn{!}, {S_regs}	Floating point Multiple Load Decrement Before
VLDMIA.64	Rn{!}, {D_regs}	Floating point Multiple Load Increment After
VLDMDB.64	Rn{!}, {D_regs}	Floating point Multiple Load Decrement Before
VLDR.32	Sd,[Rn{, #imm}]	Load a single precision data from memory (register + offset)
VLDR.32	Sd, label	Load a single precision data from memory (literal data)
VLDR.32	Sd, [PC, #imm]	Load a single precision data from memory (literal data)
VLDR.64	Dd,[Rn{, #imm}]	Load a double precision data from memory (register + offset)
VLDR.64	Dd, label	Load a double precision data from memory (literal data)
VLDR.64	Dd, [PC, #imm]	Load a double precision data from memory (literal data)
VMLA.F32	Sd, Sn, Sm	Floating point Multiply Accumulate Sd=Sd+(Sn*Sm)
VMLS.F32	Sd, Sn, Sm	Floating point Multiply Subtract Sd=Sd-(Sn*Sm)
VMOV{.F32}	Rt, Sm	Copy floating point (scalar) to ARM core register
VMOV{.F32}	Sn, Rt	Copy ARM core register to floating point (scalar)
VMOV{.F32}	Sd, Sm	Copy floating point register Sm to Sd (single precision)
VMOV	Sm, Sm1, Rt, Rt2	Copy 2 ARM core registers to 2 single precision register
VMOV	Rt, Rt2, Sm, Sm1	Copy 2 single precision register to 2 ARM core registers (Alternate syntax : VMOV Rt, Rt2, Dm)
VMRS.F32	Rt, FPCSR	Copy value in FPSCR, a floating point unit system register to Rt
VMRS	APSR_nzcv, FPCSR	Copy flags from FPSCR to the flags in APSR
VMSR	FPSCR, Rt	Copy Rt to FPSCR, a floating point unit system register
VMOV.F32	Sd, #imm	Move single precision value into floating point register

Instructions	Operands	Operations
VMUL.F32	{Sd,} Sn, Sm	Floating point Multiply
VNEG.F32	Sd, Sm	Floating point Negate
VNMUL	{Sd,} Sn, Sm	Floating point Multiply with negation Sd = - (Sn * Sm)
VNMLA	Sd, Sn, Sm	Floating point Multiply Accumulate with negation Sd = -(Sd + (Sn * Sm))
VNMLS	Sd, Sn, Sm	Floating point Multiply Accumulate with negation Sd = -(Sd - (Sn * Sm))
VPUSH.32	{S_regs}	Floating point single precision register(s) push
VPUSH.64	{D_regs}	Floating point double precision register(s) push
VPOP.32	{D_regs}	Floating point double precision register(s) pop
VPOP.64	{D_regs}	Floating point double precision register(s) pop
VSQRT.F32	Sd, Sm	Floating point Square Root
VSTMIA.32	Rn{!}, <S_regs>	Floating point Multiple Store Increment After
VSTMDB.32	Rn{!}, <S_regs>	Floating point Multiple Store Decrement Before
VSTMIA.64	Rn{!}, <D_regs>	Floating point Multiple Store Increment After
VSTMDB.64	Rn{!}, <D_regs>	Floating point Multiple Store Decrement Before
VSTR.32	Sd,[Rn{, #imm}]	Store a single precision data to memory (register + offset)
VSTR.64	Dd,[Rn{, #imm}]	Store a double precision data to memory (register + offset)
VSUB.F32	{Sd,} Sn, Sm	Floating point Subtract