Vector Floating Point Instruction Set
Quick Reference Card

Key to Tables	
{cond}	See Table **Condition Field** (on ARM side).
<S/D>	S (single precision) or D (double precision).
<S/D/X>	As above, or X (unspecified precision).
Fd, Fn, Fm	Sd, Sn, Sm (single precision), or Dd, Dn, Dm (double precision).

{E}	E : raise exception on any NaN. Without E : raise exception only on signaling NaNs.
{Z}	Round towards zero. Overrides FPSCR rounding mode.
<VFPregs>	A comma separated list of *consecutive* VFP registers, enclosed in braces ({ and }).
<VFPsysreg>	FPSCR, or FPSID.

Operation		Assembler	Exceptions	Action	Notes
Vector arithmetic	Multiply	FMUL<S/D>{cond} Fd, Fn, Fm	IO, OF, UF, IX	Fd := Fn * Fm	
	negative	FNMUL<S/D>{cond} Fd, Fn, Fm	IO, OF, UF, IX	Fd := - (Fn * Fm)	
	accumulate	FMAC<S/D>{cond} Fd, Fn, Fm	IO, OF, UF, IX	Fd := Fd + (Fn * Fm)	
	deduct	FNMAC<S/D>{cond} Fd, Fn, Fm	IO, OF, UF, IX	Fd := Fd - (Fn * Fm)	
	negate and accumulate	FMSC<S/D>{cond} Fd, Fn, Fm	IO, OF, UF, IX	Fd := -Fd + (Fn * Fm)	
	negate and deduct	FNMSC<S/D>{cond} Fd, Fn, Fm	IO, OF, UF, IX	Fd := -Fd - (Fn * Fm)	
	Add	FADD<S/D>{cond} Fd, Fn, Fm	IO, OF, IX	Fd := Fn + Fm	
	Subtract	FSUB<S/D>{cond} Fd, Fn, Fm	IO, OF, IX	Fd := Fn - Fm	
	Divide	FDIV<S/D>{cond} Fd, Fn, Fm	IO, DZ, OF, UF, IX	Fd := Fn / Fm	
	Copy	FCPY<S/D>{cond} Fd, Fm		Fd := Fm	
	Absolute	FABS<S/D>{cond} Fd, Fm		Fd := abs(Fm)	
	Negative	FNEG<S/D>{cond} Fd, Fm		Fd := -Fm	
	Square root	FSQRT<S/D>{cond} Fd, Fm	IO, IX	Fd := sqrt(Fm)	
Scalar compare		FCMP{E}<S/D>{cond} Fd, Fm	IO	Set FPSCR flags on Fd - Fm	Use FMSTAT to transfer flags.
	Compare with zero	FCMP{E}Z<S/D>{cond} Fd	IO	Set FPSCR flags on Fd - 0	Use FMSTAT to transfer flags.
Scalar convert	Single to double	FCVTDS{cond} Dd, Sm	IO	Dd := convertStoD(Sm)	
	Double to single	FCVTSD{cond} Sd, Dm	IO, OF, UF, IX	Sd := convertDtoS(Dm)	
	Unsigned integer to float	FUITO<S/D>{cond} Fd, Sm		Fd := convertUItoF(Sm)	
	Signed integer to float	FSITO<S/D>{cond} Fd, Sm	IX	Fd := convertSItoF(Sm)	
	Float to unsigned integer	FTOUI{Z}<S/D>{cond} Sd, Fm	IO, IX	Sd := convertFtoUI(Fm)	
	Float to signed integer	FTOSI{Z}<S/D>{cond} Sd, Fm	IO, IX	Sd := convertFtoSI(Fm)	
Save VFP registers		FST<S/D>{cond} Fd, [Rn{, #<immed_8*4>}]		[address] := Fd	
	Multiple, unindexed	FSTMIA<S/D/X>{cond} Rn, <VFPregs>		Saves list of VFP registers, starting at address in Rn.	
	increment after	FSTMIA<S/D/X>{cond} Rn!, <VFPregs>		synonym: FSTMEA (empty ascending)	
	decrement before	FSTMDB<S/D/X>{cond} Rn!, <VFPregs>		synonym: FSTMFD (full descending)	
Load VFP registers		FLD<S/D>{cond} Fd, [Rn{, #<immed_8*4>}]		Fd := [address]	
	Multiple, unindexed	FLDMIA<S/D/X>{cond} Rn, <VFPregs>		Loads list of VFP registers, starting at address in Rn.	
	increment after	FLDMIA<S/D/X>{cond} Rn!, <VFPregs>		synonym: FLDMFD (full descending)	
	decrement before	FLDMDB<S/D/X>{cond} Rn!, <VFPregs>		synonym: FLDMEA (empty ascending)	
Transfer registers	ARM to single	FMSR{cond} Sn, Rd		Sn := Rd	
	Single to ARM	FMRS{cond} Rd, Sn		Rd := Sn	
	ARM to lower half of double	FMDLR{cond} Dn, Rd		Dn[31:0] := Rd	Use with FMDHR.
	Lower half of double to ARM	FMRDL{cond} Rd, Dn		Rd := Dn[31:0]	Use with FMRDH.
	ARM to upper half of double	FMDHR{cond} Dn, Rd		Dn[63:32] := Rd	Use with FMDLR.
	Upper half of double to ARM	FMRDH{cond} Rd, Dn		Rd := Dn[63:32]	Use with FMRDL.
	ARM to VFP system register	FMXR{cond} <VFPsysreg>, Rd		VFPsysreg := Rd	Stalls ARM until all VFP ops complete.
	VFP system register to ARM	FMRX{cond} Rd, <VFPsysreg>		Rd := VFPsysreg	Stalls ARM until all VFP ops complete.
	FPSCR flags to CPSR	FMSTAT{cond}		CPSR flags := FPSCR flags	Equivalent to FMRX R15, FPSCR

Exceptions	
IO	Invalid operation
OF	Overflow
UF	Underflow
IX	Inexact result
DZ	Division by zero

FPSCR format								Rounding		(Stride - 1)*3			Vector length - 1				Exception trap enable bits								Cumulative exception bits				
31	30	29	28				24	23	22	21	20		18	17	16		12	11	10	9	8				4	3	2	1	0
N	Z	C	V				FZ	RMODE		STRIDE			LEN				IXE	UFE	OFE	DZE	IOE				IXC	UFC	OFC	DZC	IOC

FZ: 1 = flush to zero mode. Rounding: 0 = round to nearest, 1 = towards +∞, 2 = towards -∞, 3 = towards zero. (Vector length * Stride) must not exceed 4 for double precision operands.

If Fd is S0-S7 or D0-D3, operation is Scalar (regardless of vector length).	If Fd is S8-S31 or D4-D15, and Fm is S0-S7 or D0-D3, operation is Mixed (Fm scalar, others vector).
If Fd is S8-S31 or D4-D15, and Fm is S8-S31 or D4-D15, operation is Vector.	S0-S7 (or D0-D3), S8-S15 (D4-D7), S16-S23 (D8-D11), S24-S31 (D12-D15) each form a circulating bank of registers.

Thumb Instruction Set
Quick Reference Card

All Thumb registers are Lo (R0-R7) except where specified. Hi registers are R8-R15.

Operation		§	Assembler	Update flags	Action	Notes
Move	Immediate		`MOV Rd, #<immed_8>`	✓	Rd := immed_8	8-bit immediate value.
	Lo to Lo		`MOV Rd, Rm`	✓	Rd := Rm	
	Hi to Lo, Lo to Hi, Hi to Hi		`MOV Rd, Rm`	×	Rd := Rm	Not Lo to Lo
Arithmetic	Add		`ADD Rd, Rn, #<immed_3>`	✓	Rd := Rn + immed_3	3-bit immediate value.
	Lo and Lo		`ADD Rd, Rn, Rm`	✓	Rd := Rn + Rm	
	Hi to Lo, Lo to Hi, Hi to Hi		`ADD Rd, Rm`	×	Rd := Rd + Rm	Not Lo to Lo
	immediate		`ADD Rd, #<immed_8>`	✓	Rd := Rd + immed_8	8-bit immediate value.
	with carry		`ADC Rd, Rm`	✓	Rd := Rd + Rm + C-bit	
	value to SP		`ADD SP, #<immed_7*4>`	×	SP := SP + immed_7 * 4	9-bit immediate value (word-aligned).
	form address from SP		`ADD Rd, SP, #<immed_8*4>`	×	Rd := SP + immed_8 * 4	10-bit immediate value (word-aligned).
	form address from PC		`ADD Rd, PC, #<immed_8*4>`	×	Rd := (PC AND 0xFFFFFFFC) + immed_8 * 4	10-bit immediate value (word-aligned).
	Subtract		`SUB Rd, Rn, Rm`	✓	Rd := Rn - Rm	
	immediate 3		`SUB Rd, Rn, #<immed_3>`	✓	Rd := Rn - immed_3	3-bit immediate value.
	immediate 8		`SUB Rd, #<immed_8>`	✓	Rd := Rd - immed_8	8-bit immediate value.
	with carry		`SBC Rd, Rm`	✓	Rd := Rd - Rm - NOT C-bit	
	value from SP		`SUB SP, #<immed_7*4>`	×	SP := SP - immed_7 * 4	9-bit immediate value (word-aligned).
	Negate		`NEG Rd, Rm`	✓	Rd := - Rm	
	Multiply		`MUL Rd, Rm`	✓	Rd := Rm * Rd	
	Compare		`CMP Rn, Rm`	✓	update CPSR flags on Rn - Rm	Can be Lo to Lo, Lo to Hi, Hi to Lo, or Hi to Hi.
	negative		`CMN Rn, Rm`	✓	update CPSR flags on Rn + Rm	
	immediate		`CMP Rn, #<immed_8>`	✓	update CPSR flags on Rn - immed_8	8-bit immediate value.
	No operation		`NOP`	×	R8 := R8	Flags not affected.
Logical	AND		`AND Rd, Rm`	✓	Rd := Rd AND Rm	
	Exclusive OR		`EOR Rd, Rm`	✓	Rd := Rd EOR Rm	
	OR		`ORR Rd, Rm`	✓	Rd := Rd OR Rm	
	Bit clear		`BIC Rd, Rm`	✓	Rd := Rd AND NOT Rm	
	Move NOT		`MVN Rd, Rm`	✓	Rd := NOT Rm	
	Test bits		`TST Rn, Rm`	✓	update CPSR flags on Rn AND Rm	
Shift/rotate	Logical shift left		`LSL Rd, Rm, #<immed_5>`	✓	Rd := Rm << immed_5	5-bit immediate shift. Allowed shifts 0-31.
			`LSL Rd, Rs`	✓	Rd := Rd << Rs	
	Logical shift right		`LSR Rd, Rm, #<immed_5>`	✓	Rd := Rm >> immed_5	5-bit immediate shift. Allowed shifts 1-32.
			`LSR Rd, Rs`	✓	Rd := Rd >> Rs	
	Arithmetic shift right		`ASR Rd, Rm, #<immed_5>`	✓	Rd := Rm ASR immed_5	5-bit immediate shift. Allowed shifts 1-32.
			`ASR Rd, Rs`	✓	Rd := Rd ASR Rs	
	Rotate right		`ROR Rd, Rs`	✓	Rd := Rd ROR Rs	
Branch	Conditional branch		`B{cond} label`		R15 := label	label must be within -252 to +258 bytes of current instruction. See Table Condition Field (ARM side). AL not allowed.
	Unconditional branch		`B label`		R15 := label	label must be within ±2Kb of current instruction.
	Long branch with link		`BL label`		R14 := R15 - 2, R15 := label	Encoded as two Thumb instructions. label must be within ±4Mb of current instruction.
	Branch and exchange		`BX Rm`		R15 := Rm AND 0xFFFFFFFE	Change to ARM state if Rm[0] = 0.
	Branch with link and exchange	5T	`BLX label`		R14 := R15 - 2, R15 := label Change to ARM	Encoded as two Thumb instructions. label must be within ±4Mb of current instruction.
	Branch with link and exchange	5T	`BLX Rm`		R14 := R15 - 2, R15 := Rm AND 0xFFFFFFFE Change to ARM if Rm[0] = 0	
Software Interrupt			`SWI <immed_8>`		Software interrupt processor exception	8-bit immediate value encoded in instruction.
Breakpoint		5T	`BKPT <immed_8>`		Prefetch abort *or* enter debug state	

Thumb Instruction Set
Quick Reference Card

Operation		§	Assembler	Action	Notes
Load	with immediate offset, word		`LDR Rd, [Rn, #<immed_5*4>]`	Rd := [Rn + immed_5 * 4]	
	halfword		`LDRH Rd, [Rn, #<immed_5*2>]`	Rd := ZeroExtend([Rn + immed_5 * 2][15:0])	Clears bits 31:16
	byte		`LDRB Rd, [Rn, #<immed_5>]`	Rd := ZeroExtend([Rn + immed_5][7:0])	Clears bits 31:8
	with register offset, word		`LDR Rd, [Rn, Rm]`	Rd := [Rn + Rm]	
	halfword		`LDRH Rd, [Rn, Rm]`	Rd := ZeroExtend([Rn + Rm][15:0])	Clears bits 31:16
	signed halfword		`LDRSH Rd, [Rn, Rm]`	Rd := SignExtend([Rn + Rm][15:0])	Sets bits 31:16 to bit 15
	byte		`LDRB Rd, [Rn, Rm]`	Rd := ZeroExtend([Rn + Rm][7:0])	Clears bits 31:8
	signed byte		`LDRSB Rd, [Rn, Rm]`	Rd := SignExtend([Rn + Rm][7:0])	Sets bits 31:8 to bit 7
	PC-relative		`LDR Rd, [PC, #<immed_8*4>]`	Rd := [(PC AND 0xFFFFFFFC) + immed_8 * 4]	
	SP-relative		`LDR Rd, [SP, #<immed_8*4>]`	Rd := [SP + immed_8 * 4]	
	Multiple		`LDMIA Rn!, <reglist>`	Loads list of registers	Always updates base register.
Store	with immediate offset, word		`STR Rd, [Rn, #<immed_5*4>]`	[Rn + immed_5 * 4] := Rd	
	halfword		`STRH Rd, [Rn, #<immed_5*2>]`	[Rn + immed_5 * 2][15:0] := Rd[15:0]	Ignores Rd[31:16]
	byte		`STRB Rd, [Rn, #<immed_5>]`	[Rn + immed_5][7:0] := Rd[7:0]	Ignores Rd[31:8]
	with register offset, word		`STR Rd, [Rn, Rm]`	[Rn + Rm] := Rd	
	halfword		`STRH Rd, [Rn, Rm]`	[Rn + Rm][15:0] := Rd[15:0]	Ignores Rd[31:16]
	byte		`STRB Rd, [Rn, Rm]`	[Rn + Rm][7:0] := Rd[7:0]	Ignores Rd[31:8]
	SP-relative, word		`STR Rd, [SP, #<immed_8*4>]`	[SP + immed_8 * 4] := Rd	
	Multiple		`STMIA Rn!, <reglist>`	Stores list of registers	Always updates base register.
Push/ Pop	Push		`PUSH <reglist>`	Push registers onto stack	Full descending stack.
	Push with link		`PUSH <reglist, LR>`	Push LR and registers onto stack	
	Pop		`POP <reglist>`	Pop registers from stack	
	Pop and return		`POP <reglist, PC>`	Pop registers, branch to address loaded to PC	
	Pop and return with exchange	5T	`POP <reglist, PC>`	Pop, branch, and change to ARM state if address[0] = 0	

Document Number

ARM QRC 0001D

Change Log

Issue	Date	By	Change
A	June 1995	BJH	First Release
B	Sept 1996	BJH	Second Release
C	Nov 1998	BJH	Third Release
D	Oct 1999	CKS	Fourth Release

ENGLAND
ARM Ltd
Fulbourn Road
Cherry Hinton
Cambridge CB1 9JN
UK

Telephone: +44 1223 400400
Facsimile: +44 1223 400410
Email: info@arm.com

GERMANY
ARM Ltd
Otto-Hahn Str. 13b
85521 Ottobrun-Riemerling
Munich
Germany

Telephone: +49 89 608 75545
Facsimile: +49 98 608 75599
Email: info@arm.com

USA
ARM Inc
750 University Avenue
Suite 150,
Los Gatos CA 95032
USA

Telephone: +1 408 579 2207
Facsimile: +1 408 579 1205
Email: info@arm.com

JAPAN
ARM KK
Plustaria Building 4F,
3-1-4 Shinyokohama, Kohoku-ku,
Yokohama-shi, 222-0033
Japan

Telephone: +81 45 477 5260
Facsimile: +81 45 477 5261
Email: info@arm.com

KOREA
ARM
Room #1115, Hyundai Building
9-4, Soonae-Dong, Boondang-Ku
Sungnam, Kyunggi-Do
Korea 463-020

Telephone: +82 342 712 8234
Facsimile: +82 342 713 8225
Email: info@arm.com