ARM Instruction Set
Quick Reference Card

Key to Tables	
{cond}	Refer to Table **Condition Field {cond}**
<Oprnd2>	Refer to Table **Operand 2**
<fields>	Refer to Table **PSR fields**
{S}	Updates condition flags if S present
C*, V*	Flag is unpredictable after these instructions in Architecture v4 and earlier
Q	Sticky flag. Always updates on overflow (no S option). Read and reset using MRS and MSR
x,y	B meaning half-register [15:0], or T meaning [31:16]
<immed_8r>	A 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits
<immed_8*4>	A 10-bit constant, formed by left-shifting an 8-bit value by two bits

<a_mode2>	Refer to Table **Addressing Mode 2**
<a_mode2P>	Refer to Table **Addressing Mode 2 (Post-indexed only)**
<a_mode3>	Refer to Table **Addressing Mode 3**
<a_mode4L>	Refer to Table **Addressing Mode 4 (Block load or Stack pop)**
<a_mode4S>	Refer to Table **Addressing Mode 4 (Block store or Stack push)**
<a_mode5>	Refer to Table **Addressing Mode 5**
<reglist>	A comma-separated list of registers, enclosed in braces ({ and })
{!}	Updates base register after data transfer if ! present
§	Refer to Table **ARM architecture versions**

Operation		§	Assembler	S updates	Q	Action	Notes
Move	Move		MOV{cond}{S} Rd, <Oprnd2>	N Z C		Rd := Oprnd2	
	NOT		MVN{cond}{S} Rd, <Oprnd2>	N Z C		Rd := 0xFFFFFFFF EOR Oprnd2	
	SPSR to register	3	MRS{cond} Rd, SPSR			Rd := SPSR	
	CPSR to register	3	MRS{cond} Rd, CPSR			Rd := CPSR	
	register to SPSR	3	MSR{cond} SPSR_<fields>, Rm			SPSR := Rm (selected bytes only)	
	register to CPSR	3	MSR{cond} CPSR_<fields>, Rm			CPSR := Rm (selected bytes only)	
	immediate to SPSR	3	MSR{cond} SPSR_<fields>, #<immed_8r>			SPSR := immed_8r (selected bytes only)	
	immediate to CPSR	3	MSR{cond} CPSR_<fields>, #<immed_8r>			CPSR := immed_8r (selected bytes only)	
Arithmetic	Add		ADD{cond}{S} Rd, Rn, <Oprnd2>	N Z C V		Rd := Rn + Oprnd2	
	with carry		ADC{cond}{S} Rd, Rn, <Oprnd2>	N Z C V		Rd := Rn + Oprnd2 + Carry	
	saturating	5E	QADD{cond} Rd, Rm, Rn		Q	Rd := SAT(Rm + Rn)	No shift/rotate.
	double saturating	5E	QDADD{cond} Rd, Rm, Rn		Q	Rd := SAT(Rm + SAT(Rn * 2))	No shift/rotate.
	Subtract		SUB{cond}{S} Rd, Rn, <Oprnd2>	N Z C V		Rd := Rn - Oprnd2	
	with carry		SBC{cond}{S} Rd, Rn, <Oprnd2>	N Z C V		Rd := Rn - Oprnd2 - NOT(Carry)	
	reverse subtract		RSB{cond}{S} Rd, Rn, <Oprnd2>	N Z C V		Rd := Oprnd2 - Rn	
	reverse subtract with carry		RSC{cond}{S} Rd, Rn, <Oprnd2>	N Z C V		Rd := Oprnd2 - Rn - NOT(Carry)	
	saturating	5E	QSUB{cond} Rd, Rm, Rn		Q	Rd := SAT(Rm - Rn)	No shift/rotate.
	double saturating	5E	QDSUB{cond} Rd, Rm, Rn		Q	Rd := SAT(Rm - SAT(Rn * 2))	No shift/rotate.
	Multiply	2	MUL{cond}{S} Rd, Rm, Rs	N Z C*		Rd := (Rm * Rs)[31:0]	
	accumulate	2	MLA{cond}{S} Rd, Rm, Rs, Rn	N Z C*		Rd := ((Rm * Rs) + Rn)[31:0]	
	unsigned long	M	UMULL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C* V*		RdHi,RdLo := unsigned(Rm * Rs)	
	unsigned accumulate long	M	UMLAL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C* V*		RdHi,RdLo := unsigned(RdHi,RdLo + Rm * Rs)	
	signed long	M	SMULL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C* V*		RdHi,RdLo := signed(Rm * Rs)	
	signed accumulate long	M	SMLAL{cond}{S} RdLo, RdHi, Rm, Rs	N Z C* V*		RdHi,RdLo := signed(RdHi,RdLo + Rm * Rs)	
	signed 16 * 16 bit	5E	SMULxy{cond} Rd, Rm, Rs			Rd := Rm[x] * Rs[y]	No shift/rotate.
	signed 32 * 16 bit	5E	SMULWy{cond} Rd, Rm, Rs			Rd := (Rm * Rs[y])[47:16]	No shift/rotate.
	signed accumulate 16 * 16	5E	SMLAxy{cond} Rd, Rm, Rs, Rn		Q	Rd := Rn + Rm[x] * Rs[y]	No shift/rotate.
	signed accumulate 32 * 16	5E	SMLAWy{cond} Rd, Rm, Rs, Rn		Q	Rd := Rn + (Rm * Rs[y])[47:16]	No shift/rotate.
	signed accumulate long 16 * 16	5E	SMLALxy{cond} RdLo, RdHi, Rm, Rs			RdHi,RdLo := RdHi,RdLo + Rm[x] * Rs[y]	No shift/rotate.
	Count leading zeroes	5	CLZ{cond} Rd, Rm			Rd := number of leading zeroes in Rm	
Logical	Test		TST{cond} Rn, <Oprnd2>	N Z C		Update CPSR flags on Rn AND Oprnd2	
	Test equivalence		TEQ{cond} Rn, <Oprnd2>	N Z C		Update CPSR flags on Rn EOR Oprnd2	
	AND		AND{cond}{S} Rd, Rn, <Oprnd2>	N Z C		Rd := Rn AND Oprnd2	
	EOR		EOR{cond}{S} Rd, Rn, <Oprnd2>	N Z C		Rd := Rn EOR Oprnd2	
	ORR		ORR{cond}{S} Rd, Rn, <Oprnd2>	N Z C		Rd := Rn OR Oprnd2	
	Bit Clear		BIC{cond}{S} Rd, Rn, <Oprnd2>	N Z C		Rd := Rn AND NOT Oprnd2	
	No operation		NOP			R0 := R0	Flags not affected.
	Shift/Rotate						See Table **Operand 2.**
Compare	Compare		CMP{cond} Rn, <Oprnd2>	N Z C V		Update CPSR flags on Rn - Oprnd2	
	negative		CMN{cond} Rn, <Oprnd2>	N Z C V		Update CPSR flags on Rn + Oprnd2	

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	

Proprietary Notice

ARM is the trademark of ARM Ltd.

Neither the whole nor any part of the information contained in, or the product described in, this reference card may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this reference card is subject to continuous developments and improvements. All particulars of the product and its use contained in this reference card are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This reference card is intended only to assist the reader in the use of the product. ARM Ltd shall not be liable for any loss or damage arising from the use of any information in this reference card, or any error or omission in such information, or any incorrect use of the product.

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

ARM Instruction Set
Quick Reference Card

Operation		§	Assembler	Action	Notes
Branch	Branch		`B{cond} label`	R15 := label	label must be within ±32Mb of current instruction.
	with link		`BL{cond} label`	R14 := R15-4, R15 := label	label must be within ±32Mb of current instruction.
	and exchange	4T	`BX{cond} Rm`	R15 := Rm, Change to Thumb if Rm[0] is 1	
	with link and exchange (1)	5T	`BLX label`	R14 := R15 - 4, R15 := label, Change to Thumb	Cannot be conditional.
	with link and exchange (2)	5T	`BLX{cond} Rm`	R14 := R15 - 4, R15 := Rm[31:1] Change to Thumb if Rm[0] is 1	label must be within ±32Mb of current instruction.
Load	Word		`LDR{cond} Rd, <a_mode2>`	Rd := [address]	
	User mode privilege		`LDR{cond}T Rd, <a_mode2P>`		
	branch (and exchange)		`LDR{cond} R15, <a_mode2>`	R15 := [address][31:1] (§ 5T: Change to Thumb if [address][0] is 1)	
	Byte		`LDR{cond}B Rd, <a_mode2>`	Rd := ZeroExtend[byte from address]	
	User mode privilege		`LDR{cond}BT Rd, <a_mode2P>`		
	signed	4	`LDR{cond}SB Rd, <a_mode3>`	Rd := SignExtend[byte from address]	
	Halfword	4	`LDR{cond}H Rd, <a_mode3>`	Rd := ZeroExtent[halfword from address]	
	signed	4	`LDR{cond}SH Rd, <a_mode3>`	Rd := SignExtend[halfword from address]	
Load multiple	Pop, or Block data load		`LDM{cond}<a_mode4L> Rd{!}, <reglist-pc>`	Load list of registers from [Rd]	
	return (and exchange)		`LDM{cond}<a_mode4L> Rd{!}, <reglist+pc>`	Load registers, R15 := [address][31:1] (§ 5T: Change to Thumb if [address][0] is 1)	
	and restore CPSR		`LDM{cond}<a_mode4L> Rd{!}, <reglist+pc>^`	Load registers, branch (§ 5T: and exchange), CPSR := SPSR	Use from exception modes only.
	User mode registers		`LDM{cond}<a_mode4L> Rd, <reglist-pc>^`	Load list of User mode registers from [Rd]	Use from privileged modes only.
Store	Word		`STR{cond} Rd, <a_mode2>`	[address] := Rd	
	User mode privilege		`STR{cond}T Rd, <a_mode2P>`	[address] := Rd	
	Byte		`STR{cond}B Rd, <a_mode2>`	[address][7:0] := Rd[7:0]	
	User mode privilege		`STR{cond}BT Rd, <a_mode2P>`	[address][7:0] := Rd[7:0]	
	Halfword	4	`STR{cond}H Rd, <a_mode3>`	[address][15:0] := Rd[15:0]	
Store multiple	Push, or Block data store		`STM{cond}<a_mode4S> Rd{!}, <reglist>`	Store list of registers to [Rd]	
	User mode registers		`STM{cond}<a_mode4S> Rd{!}, <reglist>^`	Store list of User mode registers to [Rd]	Use from privileged modes only.
Swap	Word	3	`SWP{cond} Rd, Rm, [Rn]`	temp := [Rn], [Rn] := Rm, Rd := temp	
	Byte	3	`SWP{cond}B Rd, Rm, [Rn]`	temp := ZeroExtend([Rn][7:0]), [Rn][7:0] := Rm[7:0], Rd := temp	
Coprocessors	Data operations	2	`CDP{cond} p<cpnum>, <op1>, CRd, CRn, CRm, <op2>`	Coprocessor defined	
		5	`CDP2 p<cpnum>, <op1>, CRd, CRn, CRm, <op2>`		Cannot be conditional.
	Move to ARM reg from coproc	2	`MRC{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>`		
		5	`MRC2 p<cpnum>, <op1>, Rd, CRn, CRm, <op2>`		Cannot be conditional.
	Move to coproc from ARM reg	2	`MCR{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>`		
		5	`MCR2 p<cpnum>, <op1>, Rd, CRn, CRm, <op2>`		Cannot be conditional.
	Load	2	`LDC{cond} p<cpnum>, CRd, <a_mode5>`		
		5	`LDC2 p<cpnum>, CRd, <a_mode5>`		Cannot be conditional.
	Store	2	`STC{cond} p<cpnum>, CRd, <a_mode5>`		
		5	`STC2 p<cpnum>, CRd, <a_mode5>`		Cannot be conditional.
Software interrupt			`SWI{cond} <immed_24>`	Software interrupt processor exception	24-bit value encoded in instruction.
Breakpoint		5	`BKPT <immed_16>`	Prefetch abort *or* enter debug state	Cannot be conditional.

ARM Addressing Modes
Quick Reference Card

Addressing Mode 2 - Word and Unsigned Byte Data Transfer			
Pre-indexed	Immediate offset	`[Rn, #+/-<immed_12>]{!}`	
	Zero offset	`[Rn]`	Equivalent to [Rn,#0]
	Register offset	`[Rn, +/-Rm]{!}`	
	Scaled register offset	`[Rn, +/-Rm, LSL #<immed_5>]{!}`	Allowed shifts 0-31
		`[Rn, +/-Rm, LSR #<immed_5>]{!}`	Allowed shifts 1-32
		`[Rn, +/-Rm, ASR #<immed_5>]{!}`	Allowed shifts 1-32
		`[Rn, +/-Rm, ROR #<immed_5>]{!}`	Allowed shifts 1-31
		`[Rn, +/-Rm, RRX]{!}`	
Post-indexed	Immediate offset	`[Rn], #+/-<immed_12>`	
	Register offset	`[Rn], +/-Rm`	
	Scaled register offset	`[Rn], +/-Rm, LSL #<immed_5>`	Allowed shifts 0-31
		`[Rn], +/-Rm, LSR #<immed_5>`	Allowed shifts 1-32
		`[Rn], +/-Rm, ASR #<immed_5>`	Allowed shifts 1-32
		`[Rn], +/-Rm, ROR #<immed_5>`	Allowed shifts 1-31
		`[Rn], +/-Rm, RRX`	

Addressing Mode 2 (Post-indexed only)			
Post-indexed	Immediate offset	`[Rn], #+/-<immed_12>`	
	Zero offset	`[Rn]`	Equivalent to [Rn],#0
	Register offset	`[Rn], +/-Rm`	
	Scaled register offset	`[Rn], +/-Rm, LSL #<immed_5>`	Allowed shifts 0-31
		`[Rn], +/-Rm, LSR #<immed_5>`	Allowed shifts 1-32
		`[Rn], +/-Rm, ASR #<immed_5>`	Allowed shifts 1-32
		`[Rn], +/-Rm, ROR #<immed_5>`	Allowed shifts 1-31
		`[Rn], +/-Rm, RRX`	

Addressing Mode 3 - Halfword and Signed Byte Data Transfer			
Pre-indexed	Immediate offset	`[Rn, #+/-<immed_8>]{!}`	
	Zero offset	`[Rn]`	Equivalent to [Rn,#0]
	Register	`[Rn, +/-Rm]{!}`	
Post-indexed	Immediate offset	`[Rn], #+/-<immed_8>`	
	Register	`[Rn], +/-Rm`	

Addressing Mode 4 - Multiple Data Transfer			
Block load		**Stack pop**	
`IA`	Increment After	`FD`	Full Descending
`IB`	Increment Before	`ED`	Empty Descending
`DA`	Decrement After	`FA`	Full Ascending
`DB`	Decrement Before	`EA`	Empty Ascending
Block store		**Stack push**	
`IA`	Increment After	`EA`	Empty Ascending
`IB`	Increment Before	`FA`	Full Ascending
`DA`	Decrement After	`ED`	Empty Descending
`DB`	Decrement Before	`FD`	Full Descending

Addressing Mode 5 - Coprocessor Data Transfer			
Pre-indexed	Immediate offset	`[Rn, #+/-<immed_8*4>]{!}`	
	Zero offset	`[Rn]`	Equivalent to [Rn,#0]
Post-indexed	Immediate offset	`[Rn], #+/-<immed_8*4>`	
Unindexed	No offset	`[Rn], {8-bit copro. option}`	

ARM architecture versions	
n	ARM architecture version *n* and above.
*n*T	T variants of ARM architecture version *n* and above.
M	ARM architecture version 3M, and 4 and above excluding xM variants
*n*E	E variants of ARM architecture version *n* and above.

Operand 2		
Immediate value	`#<immed_8r>`	
Logical shift left immediate	`Rm, LSL #<immed_5>`	Allowed shifts 0-31
Logical shift right immediate	`Rm, LSR #<immed_5>`	Allowed shifts 1-32
Arithmetic shift right immediate	`Rm, ASR #<immed_5>`	Allowed shifts 1-32
Rotate right immediate	`Rm, ROR #<immed_5>`	Allowed shifts 1-31
Register	`Rm`	
Rotate right extended	`Rm, RRX`	
Logical shift left register	`Rm, LSL Rs`	
Logical shift right register	`Rm, LSR Rs`	
Arithmetic shift right register	`Rm, ASR Rs`	
Rotate right register	`Rm, ROR Rs`	

PSR fields (use at least one suffix)		
Suffix	**Meaning**	
`c`	Control field mask byte	PSR[7:0]
`f`	Flags field mask byte	PSR[31:24]
`s`	Status field mask byte	PSR[23:16]
`x`	Extension field mask byte	PSR[15:8]

Condition Field {cond}		
Mnemonic	**Description**	**Description (VFP)**
`EQ`	Equal	Equal
`NE`	Not equal	Not equal, or unordered
`CS / HS`	Carry Set / Unsigned higher or same	Greater than or equal, or unordered
`CC / LO`	Carry Clear / Unsigned lower	Less than
`MI`	Negative	Less than
`PL`	Positive or zero	Greater than or equal, or unordered
`VS`	Overflow	Unordered (at least one NaN operand)
`VC`	No overflow	Not unordered
`HI`	Unsigned higher	Greater than, or unordered
`LS`	Unsigned lower or same	Less than or equal
`GE`	Signed greater than or equal	Greater than or equal
`LT`	Signed less than	Less than, or unordered
`GT`	Signed greater than	Greater than
`LE`	Signed less than or equal	Less than or equal, or unordered
`AL`	Always (normally omitted)	Always (normally omitted)

Key to tables	
`{!}`	Updates base register after data transfer if ! present. (Post-indexed always updates.)
`<immed_8r>`	A 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits.
`+/-`	+ or -. (+ may be omitted.)