ARM® and Thumb®-2 Instruction Set
Quick Reference Card

Key to Tables

Rm {, <opsh>}	See Table **Register, optionally shifted by constant**
<Operand2>	See Table **Flexible Operand 2**. Shift and rotate are only available as part of Operand2.
<fields>	See Table **PSR fields**.
<PSR>	Either CPSR (Current Processor Status Register) or SPSR (Saved Processor Status Register)
C*, V*	Flag is unpredictable in Architecture v4 and earlier, unchanged in Architecture v5 and later.
<Rs|sh>	Can be Rs or an immediate shift value. The values allowed for each shift type are the same as those shown in Table **Register, optionally shifted by constant**.
x,y	B meaning half-register [15:0], or T meaning [31:16].
<imm8m>	ARM: a 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits.
	Thumb: a 32-bit constant, formed by left-shifting an 8-bit value by any number of bits, or a bit pattern of one of the forms 0xXYXYXYXY, 0x00XY00XY or 0xXY00XY00.
<prefix>	See Table **Prefixes for Parallel instructions**
{IA|IB|DA|DB}	Increment After, Increment Before, Decrement After, or Decrement Before. IB and DA are not available in Thumb state. If omitted, defaults to IA.
<size>	B, SB, H, or SH, meaning Byte, Signed Byte, Halfword, and Signed Halfword respectively. SB and SH are not available in STR instructions.

<reglist>	A comma-separated list of registers, enclosed in braces { and }.
<reglist-PC>	As <reglist>, must not include the PC.
<reglist+PC>	As <reglist>, including the PC.
+/-	+ or –. (+ may be omitted.)
§	See Table **ARM architecture versions**.
<iflags>	Interrupt flags. One or more of a, i, f (abort, interrupt, fast interrupt).
<p_mode>	See Table **Processor Modes**
SPm	SP for the processor mode specified by <p_mode>
<lsb>	Least significant bit of bitfield.
<width>	Width of bitfield. <width> + <lsb> must be <= 32.
{X}	RsX is Rs rotated 16 bits if X present. Otherwise, RsX is Rs.
{!}	Updates base register after data transfer if ! present (pre-indexed).
{S}	Updates condition flags if S present.
{T}	User mode privilege if T present.
{R}	Rounds result to nearest if R present, otherwise truncates result.

Operation		§	Assembler	S updates	Action	Notes
Add	Add		ADD{S} Rd, Rn, <Operand2>	N Z C V	Rd := Rn + Operand2	N
	with carry		ADC{S} Rd, Rn, <Operand2>	N Z C V	Rd := Rn + Operand2 + Carry	N
	wide	T2	ADD Rd, Rn, #<imm12>		Rd := Rn + imm12, imm12 range 0-4095	T, P
	saturating {doubled}	5E	Q{D}ADD Rd, Rm, Rn		Rd := SAT(Rm + Rn) doubled: Rd := SAT(Rm + SAT(Rn * 2))	Q
Address	Form PC-relative address		ADR Rd, <label>		Rd := <label>, for <label> range from current instruction see Note L	N, L
Subtract	Subtract		SUB{S} Rd, Rn, <Operand2>	N Z C V	Rd := Rn – Operand2	N
	with carry		SBC{S} Rd, Rn, <Operand2>	N Z C V	Rd := Rn – Operand2 – NOT(Carry)	N
	wide	T2	SUB Rd, Rn, #<imm12>	N Z C V	Rd := Rn – imm12, imm12 range 0-4095	T, P
	reverse subtract		RSB{S} Rd, Rn, <Operand2>	N Z C V	Rd := Operand2 – Rn	N
	reverse subtract with carry		RSC{S} Rd, Rn, <Operand2>	N Z C V	Rd := Operand2 – Rn – NOT(Carry)	A
	saturating {doubled}	5E	Q{D}SUB Rd, Rm, Rn		Rd := SAT(Rm – Rn) doubled: Rd := SAT(Rm – SAT(Rn * 2))	Q
	Exception return without stack		SUBS PC, LR, #<imm8>		PC = LR – imm8, CPSR = SPSR(current mode), imm8 range 0-255.	T
Parallel arithmetic	Halfword-wise addition	6	<prefix>ADD16 Rd, Rn, Rm		Rd[31:16] := Rn[31:16] + Rm[31:16], Rd[15:0] := Rn[15:0] + Rm[15:0]	G
	Halfword-wise subtraction	6	<prefix>SUB16 Rd, Rn, Rm		Rd[31:16] := Rn[31:16] – Rm[31:16], Rd[15:0] := Rn[15:0] – Rm[15:0]	G
	Byte-wise addition	6	<prefix>ADD8 Rd, Rn, Rm		Rd[31:24] := Rn[31:24] + Rm[31:24], Rd[23:16] := Rn[23:16] + Rm[23:16], Rd[15:8] := Rn[15:8] + Rm[15:8], Rd[7:0] := Rn[7:0] + Rm[7:0]	G
	Byte-wise subtraction	6	<prefix>SUB8 Rd, Rn, Rm		Rd[31:24] := Rn[31:24] – Rm[31:24], Rd[23:16] := Rn[23:16] – Rm[23:16], Rd[15:8] := Rn[15:8] – Rm[15:8], Rd[7:0] := Rn[7:0] – Rm[7:0]	G
	Halfword-wise exchange, add, subtract	6	<prefix>ASX Rd, Rn, Rm		Rd[31:16] := Rn[31:16] + Rm[15:0], Rd[15:0] := Rn[15:0] – Rm[31:16]	G
	Halfword-wise exchange, subtract, add	6	<prefix>SAX Rd, Rn, Rm		Rd[31:16] := Rn[31:16] – Rm[15:0], Rd[15:0] := Rn[15:0] + Rm[31:16]	G
	Unsigned sum of absolute differences	6	USAD8 Rd, Rm, Rs		Rd := Abs(Rm[31:24] – Rs[31:24]) + Abs(Rm[23:16] – Rs[23:16]) + Abs(Rm[15:8] – Rs[15:8]) + Abs(Rm[7:0] – Rs[7:0])	
	and accumulate	6	USADA8 Rd, Rm, Rs, Rn		Rd := Rn + Abs(Rm[31:24] – Rs[31:24]) + Abs(Rm[23:16] – Rs[23:16]) + Abs(Rm[15:8] – Rs[15:8]) + Abs(Rm[7:0] – Rs[7:0])	
Saturate	Signed saturate word, right shift	6	SSAT Rd, #<sat>, Rm{, ASR <sh>}		Rd := SignedSat((Rm ASR sh), sat). <sat> range 1-32, <sh> range 1-31.	Q, R
	Signed saturate word, left shift	6	SSAT Rd, #<sat>, Rm{, LSL <sh>}		Rd := SignedSat((Rm LSL sh), sat). <sat> range 1-32, <sh> range 0-31.	Q
	Signed saturate two halfwords	6	SSAT16 Rd, #<sat>, Rm		Rd[31:16] := SignedSat(Rm[31:16], sat), Rd[15:0] := SignedSat(Rm[15:0], sat). <sat> range 1-16.	Q
	Unsigned saturate word, right shift	6	USAT Rd, #<sat>, Rm{, ASR <sh>}		Rd := UnsignedSat((Rm ASR sh), sat). <sat> range 0-31, <sh> range 1-31.	Q, R
	Unsigned saturate word, left shift	6	USAT Rd, #<sat>, Rm{, LSL <sh>}		Rd := UnsignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 0-31.	Q
	Unsigned saturate two halfwords	6	USAT16 Rd, #<sat>, Rm		Rd[31:16] := UnsignedSat(Rm[31:16], sat), Rd[15:0] := UnsignedSat(Rm[15:0], sat). <sat> range 0-15.	Q

ARM and Thumb-2 Instruction Set
Quick Reference Card

Operation		§	Assembler	S updates	Action		Notes
Multiply	Multiply		MUL{S} Rd, Rm, Rs	N Z C*	Rd := (Rm * Rs)[31:0]	(If Rm is Rd, S can be used in Thumb-2)	N, S
	and accumulate		MLA{S} Rd, Rm, Rs, Rn	N Z C*	Rd := (Rn + (Rm * Rs))[31:0]		S
	and subtract	T2	MLS Rd, Rm, Rs, Rn		Rd := (Rn – (Rm * Rs))[31:0]		
	unsigned long		UMULL{S} RdLo, RdHi, Rm, Rs	N Z C* V*	RdHi,RdLo := unsigned(Rm * Rs)		S
	unsigned accumulate long		UMLAL{S} RdLo, RdHi, Rm, Rs	N Z C* V*	RdHi,RdLo := unsigned(RdHi,RdLo + Rm * Rs)		S
	unsigned double accumulate long	6	UMAAL RdLo, RdHi, Rm, Rs		RdHi,RdLo := unsigned(RdHi + RdLo + Rm * Rs)		
	Signed multiply long		SMULL{S} RdLo, RdHi, Rm, Rs	N Z C* V*	RdHi,RdLo := signed(Rm * Rs)		S
	and accumulate long		SMLAL{S} RdLo, RdHi, Rm, Rs	N Z C* V*	RdHi,RdLo := signed(RdHi,RdLo + Rm * Rs)		S
	16 * 16 bit	5E	SMULxy Rd, Rm, Rs		Rd := Rm[x] * Rs[y]		
	32 * 16 bit	5E	SMULWy Rd, Rm, Rs		Rd := (Rm * Rs[y])[47:16]		
	16 * 16 bit and accumulate	5E	SMLAxy Rd, Rm, Rs, Rn		Rd := Rn + Rm[x] * Rs[y]		Q
	32 * 16 bit and accumulate	5E	SMLAWy Rd, Rm, Rs, Rn		Rd := Rn + (Rm * Rs[y])[47:16]		Q
	16 * 16 bit and accumulate long	5E	SMLALxy RdLo, RdHi, Rm, Rs		RdHi,RdLo := RdHi,RdLo + Rm[x] * Rs[y]		
	Dual signed multiply, add	6	SMUAD{X} Rd, Rm, Rs		Rd := Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]		Q
	and accumulate	6	SMLAD{X} Rd, Rm, Rs, Rn		Rd := Rn + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]		Q
	and accumulate long	6	SMLALD{X} RdLo, RdHi, Rm, Rs		RdHi,RdLo := RdHi,RdLo + Rm[15:0] * RsX[15:0] + Rm[31:16] * RsX[31:16]		
	Dual signed multiply, subtract	6	SMUSD{X} Rd, Rm, Rs		Rd := Rm[15:0] * RsX[15:0] – Rm[31:16] * RsX[31:16]		Q
	and accumulate	6	SMLSD{X} Rd, Rm, Rs, Rn		Rd := Rn + Rm[15:0] * RsX[15:0] – Rm[31:16] * RsX[31:16]		Q
	and accumulate long	6	SMLSLD{X} RdLo, RdHi, Rm, Rs		RdHi,RdLo := RdHi,RdLo + Rm[15:0] * RsX[15:0] – Rm[31:16] * RsX[31:16]		
	Signed top word multiply	6	SMMUL{R} Rd, Rm, Rs		Rd := (Rm * Rs)[63:32]		
	and accumulate	6	SMMLA{R} Rd, Rm, Rs, Rn		Rd := Rn + (Rm * Rs)[63:32]		
	and subtract	6	SMMLS{R} Rd, Rm, Rs, Rn		Rd := Rn – (Rm * Rs)[63:32]		
	with internal 40-bit accumulate	XS	MIA Ac, Rm, Rs		Ac := Ac + Rm * Rs		
	packed halfword	XS	MIAPH Ac, Rm, Rs		Ac := Ac + Rm[15:0] * Rs[15:0] + Rm[31:16] * Rs[31:16]		
	halfword	XS	MIAxy Ac, Rm, Rs		Ac := Ac + Rm[x] * Rs[y]		
Divide	Signed or Unsigned	RM	<op> Rd, Rn, Rm		Rd := Rn / Rm	<op> is SDIV (signed) or UDIV (unsigned)	
Move data	Move		MOV{S} Rd, <Operand2>	N Z C	Rd := Operand2	See also Shift instructions	N
	NOT		MVN{S} Rd, <Operand2>	N Z C	Rd := 0xFFFFFFFF EOR Operand2		N
	top	T2	MOVT Rd, #<imm16>		Rd[31:16] := imm16, Rd[15:0] unaffected, imm16 range 0-65535		
	wide	T2	MOV Rd, #<imm16>		Rd[15:0] := imm16, Rd[31:16] = 0, imm16 range 0-65535		
	40-bit accumulator to register	XS	MRA RdLo, RdHi, Ac		RdLo := Ac[31:0], RdHi := Ac[39:32]		
	register to 40-bit accumulator	XS	MAR Ac, RdLo, RdHi		Ac[31:0] := RdLo, Ac[39:32] := RdHi		
Shift	Arithmetic shift right		ASR{S} Rd, Rm, <Rs|sh>	N Z C	Rd := ASR(Rm, Rs|sh)	Same as MOV{S} Rd, Rm, ASR <Rs|sh>	N
	Logical shift left		LSL{S} Rd, Rm, <Rs|sh>	N Z C	Rd := LSL(Rm, Rs|sh)	Same as MOV{S} Rd, Rm, LSL <Rs|sh>	N
	Logical shift right		LSR{S} Rd, Rm, <Rs|sh>	N Z C	Rd := LSR(Rm, Rs|sh)	Same as MOV{S} Rd, Rm, LSR <Rs|sh>	N
	Rotate right		ROR{S} Rd, Rm, <Rs|sh>	N Z C	Rd := ROR(Rm, Rs|sh)	Same as MOV{S} Rd, Rm, ROR <Rs|sh>	N
	Rotate right with extend		RRX{S} Rd, Rm	N Z C	Rd := RRX(Rm)	Same as MOV{S} Rd, Rm, RRX	
Count leading zeros		5	CLZ Rd, Rm		Rd := number of leading zeros in Rm		
Compare	Compare		CMP Rn, <Operand2>	N Z C V	Update CPSR flags on Rn – Operand2		N
	negative		CMN Rn, <Operand2>	N Z C V	Update CPSR flags on Rn + Operand2		N
Logical	Test		TST Rn, <Operand2>	N Z C	Update CPSR flags on Rn AND Operand2		N
	Test equivalence		TEQ Rn, <Operand2>	N Z C	Update CPSR flags on Rn EOR Operand2		
	AND		AND{S} Rd, Rn, <Operand2>	N Z C	Rd := Rn AND Operand2		N
	EOR		EOR{S} Rd, Rn, <Operand2>	N Z C	Rd := Rn EOR Operand2		N
	ORR		ORR{S} Rd, Rn, <Operand2>	N Z C	Rd := Rn OR Operand2		N
	ORN	T2	ORN{S} Rd, Rn, <Operand2>	N Z C	Rd := Rn OR NOT Operand2		T
	Bit Clear		BIC{S} Rd, Rn, <Operand2>	N Z C	Rd := Rn AND NOT Operand2		N

ARM and Thumb-2 Instruction Set
Quick Reference Card

Operation		§	Assembler	Action	Notes
Bit field	Bit Field Clear	T2	BFC Rd, #<lsb>, #<width>	Rd[(width+lsb–1):lsb] := 0, other bits of Rd unaffected	
	Bit Field Insert	T2	BFI Rd, Rn, #<lsb>, #<width>	Rd[(width+lsb–1):lsb] := Rn[(width-1):0], other bits of Rd unaffected	
	Signed Bit Field Extract	T2	SBFX Rd, Rn, #<lsb>, #<width>	Rd[(width–1):0] = Rn[(width+lsb–1):lsb], Rd[31:width] = Replicate(Rn[width+lsb–1])	
	Unsigned Bit Field Extract	T2	UBFX Rd, Rn, #<lsb>, #<width>	Rd[(width–1):0] = Rn[(width+lsb–1):lsb], Rd[31:width] = Replicate(0)	
Pack	Pack halfword bottom + top	6	PKHBT Rd, Rn, Rm{, LSL #<sh>}	Rd[15:0] := Rn[15:0], Rd[31:16] := (Rm LSL sh)[31:16]. sh 0-31.	
	Pack halfword top + bottom	6	PKHTB Rd, Rn, Rm{, ASR #<sh>}	Rd[31:16] := Rn[31:16], Rd[15:0] := (Rm ASR sh)[15:0]. sh 1-32.	
Signed extend	Halfword to word	6	SXTH Rd, Rm{, ROR #<sh>}	Rd[31:0] := SignExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.	N
	Two bytes to halfwords	6	SXTB16 Rd, Rm{, ROR #<sh>}	Rd[31:16] := SignExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	
	Byte to word	6	SXTB Rd, Rm{, ROR #<sh>}	Rd[31:0] := SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	N
Unsigned extend	Halfword to word	6	UXTH Rd, Rm{, ROR #<sh>}	Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.	N
	Two bytes to halfwords	6	UXTB16 Rd, Rm{, ROR #<sh>}	Rd[31:16] := ZeroExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	
	Byte to word	6	UXTB Rd, Rm{, ROR #<sh>}	Rd[31:0] := ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	N
Signed extend with add	Halfword to word, add	6	SXTAH Rd, Rn, Rm{, ROR #<sh>}	Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.	
	Two bytes to halfwords, add	6	SXTAB16 Rd, Rn, Rm{, ROR #<sh>}	Rd[31:16] := Rn[31:16] + SignExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := Rn[15:0] + SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	
	Byte to word, add	6	SXTAB Rd, Rn, Rm{, ROR #<sh>}	Rd[31:0] := Rn[31:0] + SignExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	
Unsigned extend with add	Halfword to word, add	6	UXTAH Rd, Rn, Rm{, ROR #<sh>}	Rd[31:0] := Rn[31:0] + ZeroExtend((Rm ROR (8 * sh))[15:0]). sh 0-3.	
	Two bytes to halfwords, add	6	UXTAB16 Rd, Rn, Rm{, ROR #<sh>}	Rd[31:16] := Rn[31:16] + ZeroExtend((Rm ROR (8 * sh))[23:16]), Rd[15:0] := Rn[15:0] + ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	
	Byte to word, add	6	UXTAB Rd, Rn, Rm{, ROR #<sh>}	Rd[31:0] := Rn[31:0] + ZeroExtend((Rm ROR (8 * sh))[7:0]). sh 0-3.	
Reverse	Bits in word	T2	RBIT Rd, Rm	For (i = 0; i < 32; i++) : Rd[i] = Rm[31– i]	
	Bytes in word	6	REV Rd, Rm	Rd[31:24] := Rm[7:0], Rd[23:16] := Rm[15:8], Rd[15:8] := Rm[23:16], Rd[7:0] := Rm[31:24]	N
	Bytes in both halfwords	6	REV16 Rd, Rm	Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:24] := Rm[23:16], Rd[23:16] := Rm[31:24]	N
	Bytes in low halfword, sign extend	6	REVSH Rd, Rm	Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:16] := Rm[7] * &FFFF	N
Select	Select bytes	6	SEL Rd, Rn, Rm	Rd[7:0] := Rn[7:0] if GE[0] = 1, else Rd[7:0] := Rm[7:0] Bits[15:8], [23:16], [31:24] selected similarly by GE[1], GE[2], GE[3]	
If-Then	If-Then	T2	IT{pattern} {cond}	Makes up to four following instructions conditional, according to pattern. pattern is a string of up to three letters. Each letter can be T (Then) or E (Else). The first instruction after IT has condition cond. The following instructions have condition cond if the corresponding letter is T, or the inverse of cond if the corresponding letter is E. See Table **Condition Field** for available condition codes.	T U
Branch	Branch		B <label>	PC := label. label is this instruction ±32MB (T2: ±16MB, T: –252 - +256B)	N, B
	with link		BL <label>	LR := address of next instruction, PC := label. label is this instruction ±32MB (T2: ±16MB).	
	and exchange	4T	BX Rm	PC := Rm. Target is Thumb if Rm[0] is 1, ARM if Rm[0] is 0.	N
	with link and exchange (1)	5T	BLX <label>	LR := address of next instruction, PC := label, Change instruction set. label is this instruction ±32MB (T2: ±16MB).	C
	with link and exchange (2)	5	BLX Rm	LR := address of next instruction, PC := Rm[31:1]. Change to Thumb if Rm[0] is 1, to ARM if Rm[0] is 0.	N
	and change to Jazelle state	5J	BXJ Rm	Change to Jazelle state if available	
	Compare, branch if (non) zero	T2	CB{N}Z Rn,<label>	If Rn {== or !=} 0 then PC := label. label is (this instruction + 4-130).	N T U
	Table Branch Byte	T2	TBB [Rn, Rm]	PC = PC + ZeroExtend(Memory(Rn + Rm, 1) << 1). Branch range 4-512. Rn can be PC.	T U
	Table Branch Halfword	T2	TBH [Rn, Rm, LSL #1]	PC = PC + ZeroExtend(Memory(Rn + Rm << 1, 2) << 1). Branch range 4-131072. Rn can be PC.	T U
Move to or from PSR	PSR to register		MRS Rd, <PSR>	Rd := PSR	
	register to PSR		MSR <PSR>_<fields>, Rm	PSR := Rm (selected bytes only)	
	immediate to PSR		MSR <PSR>_<fields>, #<imm8m>	PSR := immed_8r (selected bytes only)	
Processor state change	Change processor state	6	CPSID <iflags> {, #<p_mode>}	Disable specified interrupts, optional change mode.	U, N
		6	CPSIE <iflags> {, #<p_mode>}	Enable specified interrupts, optional change mode.	U, N
	Change processor mode	6	CPS #<p_mode>		U
	Set endianness	6	SETEND <endianness>	Sets endianness for loads and saves. <endianness> can be BE (Big Endian) or LE (Little Endian).	U, N

ARM Instruction Set
Quick Reference Card

Single data item loads and stores		§	Assembler	Action if <op> is LDR	Action if <op> is STR	Notes
Load or store word, byte or halfword	Immediate offset		<op>{size}{T} Rd, [Rn {, #<offset>}]{!}	Rd := [address, size]	[address, size] := Rd	1, N
	Post-indexed, immediate		<op>{size}{T} Rd, [Rn], #<offset>	Rd := [address, size]	[address, size] := Rd	2
	Register offset		<op>{size} Rd, [Rn, +/-Rm {, <opsh>}]{!}	Rd := [address, size]	[address, size] := Rd	3, N
	Post-indexed, register		<op>{size}{T} Rd, [Rn], +/-Rm {, <opsh>}	Rd := [address, size]	[address, size] := Rd	4
	PC-relative		<op>{size} Rd, <label>	Rd := [label, size]	Not available	5, N
Load or store doubleword	Immediate offset	5E	<op>D Rd1, Rd2, [Rn {, #<offset>}]{!}	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	6, 9
	Post-indexed, immediate	5E	<op>D Rd1, Rd2, [Rn], #<offset>	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	6, 9
	Register offset	5E	<op>D Rd1, Rd2, [Rn, +/-Rm {, <opsh>}]{!}	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	7, 9
	Post-indexed, register	5E	<op>D Rd1, Rd2, [Rn], +/-Rm {, <opsh>}	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	7, 9
	PC-relative	5E	<op>D Rd1, Rd2, <label>	Rd1 := [label], Rd2 := [label + 4]	Not available	8, 9

Preload data or instruction		§ (PLD)	§ (PLI)	Assembler	Action if <op> is PLD	Action if <op> is PLI	Notes
	Immediate offset	5E	7	<op> [Rn {, #<offset>}]	Preload [address, 32] (data)	Preload [address, 32] (instruction)	1, C
	Register offset	5E	7	<op> [Rn, +/-Rm {, <opsh>}]	Preload [address, 32] (data)	Preload [address, 32] (instruction)	3, C
	PC-relative	5E	7	<op> <label>	Preload [label, 32] (data)	Preload [label, 32] (instruction)	5, C

Other memory operations		§	Assembler	Action	Notes
Load multiple	Block data load		LDM{IA|IB|DA|DB} Rn{!}, <reglist-PC>	Load list of registers from [Rn]	N, I
	return (and exchange)		LDM{IA|IB|DA|DB} Rn{!}, <reglist+PC>	Load registers, PC := [address][31:1] (§ 5T: Change to Thumb if [address][0] is 1)	I
	and restore CPSR		LDM{IA|IB|DA|DB} Rn{!}, <reglist+PC>^	Load registers, branch (§ 5T: and exchange), CPSR := SPSR. Exception modes only.	I
	User mode registers		LDM{IA|IB|DA|DB} Rn, <reglist-PC>^	Load list of User mode registers from [Rn]. Privileged modes only.	I
Pop			POP <reglist>	Canonical form of LDM SP!, <reglist>	N
Load exclusive	Semaphore operation	6	LDREX Rd, [Rn]	Rd := [Rn], tag address as exclusive access. Outstanding tag set if not shared address. Rd, Rn not PC.	
	Halfword or Byte	6K	LDREX{H|B} Rd, [Rn]	Rd[15:0] := [Rn] or Rd[7:0] := [Rn], tag address as exclusive access. Outstanding tag set if not shared address. Rd, Rn not PC.	
	Doubleword	6K	LDREXD Rd1, Rd2, [Rn]	Rd1 := [Rn], Rd2 := [Rn+4], tag addresses as exclusive access Outstanding tags set if not shared addresses. Rd1, Rd2, Rn not PC.	9
Store multiple	Push, or Block data store		STM{IA|IB|DA|DB} Rn{!}, <reglist>	Store list of registers to [Rn]	N, I
	User mode registers		STM{IA|IB|DA|DB} Rn{!}, <reglist>^	Store list of User mode registers to [Rn]. Privileged modes only.	I
Push			PUSH <reglist>	Canonical form of STMDB SP!, <reglist>	N
Store exclusive	Semaphore operation	6	STREX Rd, Rm, [Rn]	If allowed, [Rn] := Rm, clear exclusive tag, Rd := 0. Else Rd := 1. Rd, Rm, Rn not PC.	
	Halfword or Byte	6K	STREX{H|B} Rd, Rm, [Rn]	If allowed, [Rn] := Rm[15:0] or [Rn] := Rm[7:0], clear exclusive tag, Rd := 0. Else Rd := 1 Rd, Rm, Rn not PC.	
	Doubleword	6K	STREXD Rd, Rm1, Rm2, [Rn]	If allowed, [Rn] := Rm1, [Rn+4] := Rm2, clear exclusive tags, Rd := 0. Else Rd := 1 Rd, Rm1, Rm2, Rn not PC.	9
Clear exclusive		6K	CLREX	Clear local processor exclusive tag	C

Notes: availability and range of options for Load, Store, and Preload operations

Note	ARM Word, B, D	ARM SB, H, SH	ARM T, BT	Thumb-2 Word, B, SB, H, SH, D	Thumb-2 T, BT, SBT, HT, SHT
1	offset: – 4095 to +4095	offset: –255 to +255	Not available	offset: –255 to +255 if writeback, –255 to +4095 otherwise	offset: 0 to +255, writeback not allowed
2	offset: – 4095 to +4095	offset: –255 to +255	offset: – 4095 to +4095	offset: –255 to +255	Not available
3	Full range of {, <opsh>}	{, <opsh>} not allowed	Not available	<opsh> restricted to LSL #<sh>, <sh> range 0 to 3	Not available
4	Full range of {, <opsh>}	{, <opsh>} not allowed	Full range of {, <opsh>}	Not available	Not available
5	label within +/– 4092 of current instruction	Not available	Not available	label within +/– 4092 of current instruction	Not available
6	offset: –255 to +255	-	-	offset: –1020 to +1020, must be multiple of 4.	-
7	{, <opsh>} not allowed	-	-	Not available	-
8	label within +/– 252 of current instruction	-	-	Not available	-
9	Rd1 even, and not r14, Rd2 == Rd1 + 1.	-	-	Rd1 != PC, Rd2 != PC	-

ARM Instruction Set
Quick Reference Card

Coprocessor operations	§	Assembler		Action	Notes
Data operations		CDP{2} <copr>, <op1>, CRd, CRn, CRm{, <op2>}		Coprocessor defined	C2
Move to ARM register from coprocessor		MRC{2} <copr>, <op1>, Rd, CRn, CRm{, <op2>}		Coprocessor defined	C2
Two ARM register move	5E	MRRC <copr>, <op1>, Rd, Rn, CRm		Coprocessor defined	
Alternative two ARM register move	6	MRRC2 <copr>, <op1>, Rd, Rn, CRm		Coprocessor defined	C
Move to coproc from ARM reg		MCR{2} <copr>, <op1>, Rd, CRn, CRm{, <op2>}		Coprocessor defined	C2
Two ARM register move	5E	MCRR <copr>, <op1>, Rd, Rn, CRm		Coprocessor defined	
Alternative two ARM register move	6	MCRR2 <copr>, <op1>, Rd, Rn, CRm		Coprocessor defined	C
Loads and stores, pre-indexed		<op>{2} <copr>, CRd, [Rn, #+/-<offset8*4>]{!}	op: LDC or STC. offset: multiple of 4 in range 0 to 1020.	Coprocessor defined	C2
Loads and stores, zero offset		<op>{2} <copr>, CRd, [Rn] {, 8-bit copro. option}	op: LDC or STC.	Coprocessor defined	C2
Loads and stores, post-indexed		<op>{2} <copr>, CRd, [Rn], #+/-<offset8*4>	op: LDC or STC. offset: multiple of 4 in range 0 to 1020.	Coprocessor defined	C2

Miscellaneous operations		§	Assembler	Action	Notes
Swap word			SWP Rd, Rm, [Rn]	temp := [Rn], [Rn] := Rm, Rd := temp.	D
Swap byte			SWPB Rd, Rm, [Rn]	temp := ZeroExtend([Rn][7:0]), [Rn][7:0] := Rm[7:0], Rd := temp	D
Store return state		6	SRS{IA|IB|DA|DB} SP{!}, #<p_mode>	[SPm] := LR, [SPm + 4] := CPSR	C, I
Return from exception		6	RFE{IA|IB|DA|DB} Rn{!}	PC := [Rn], CPSR := [Rn + 4]	C, I
Breakpoint		5	BKPT <imm16>	Prefetch abort *or* enter debug state. 16-bit bitfield encoded in instruction.	C, N
Secure Monitor Call		Z	SMC <imm16>	Secure Monitor Call exception. 16-bit bitfield encoded in instruction. Formerly SMI.	
Supervisor Call			SVC <imm24>	Supervisor Call exception. 24-bit bitfield encoded in instruction. Formerly SWI.	N
No operation		6	NOP	None, might not even consume any time.	N
Hints	Debug Hint	7	DBG	Provide hint to debug and related systems.	
	Data Memory Barrier	7	DMB	Ensure the order of observation of memory accesses.	C
	Data Synchronization Barrier	7	DSB	Ensure the completion of memory accesses,	C
	Instruction Synchronization Barrier	7	ISB	Flush processor pipeline and branch prediction logic.	C
	Set event	T2	SEV	Signal event in multiprocessor system. NOP if not implemented.	N
	Wait for event	T2	WFE	Wait for event, IRQ, FIQ, Imprecise abort, or Debug entry request. NOP if not implemented.	N
	Wait for interrupt	T2	WFI	Wait for IRQ, FIQ, Imprecise abort, or Debug entry request. NOP if not implemented.	N
	Yield	T2	YIELD	Yield control to alternative thread. NOP if not implemented.	N

Notes	
A	Not available in Thumb state.
B	Can be conditional in Thumb state without having to be in an IT block.
C	Condition codes are not allowed in ARM state.
C2	The optional 2 is available from ARMv5. It provides an alternative operation. Condition codes are not allowed for the alternative form in ARM state.
D	Deprecated. Use LDREX and STREX instead.
G	Updates the four GE flags in the CPSR based on the results of the individual operations.
I	IA is the default, and is normally omitted.
L	ARM: <imm8m>. 16-bit Thumb: multiple of 4 in range 0-1020. 32-bit Thumb: 0-4095.
N	Some or all forms of this instruction are 16-bit (Narrow) instructions in Thumb-2 code. For details see the *Thumb 16-bit Instruction Set (UAL) Quick Reference Card.*
P	Rn can be the PC in Thumb state in this instruction.
Q	Sets the Q flag if saturation (addition or substraction) or overflow (multiplication) occurs. Read and reset the Q flag using MRS and MSR.
R	<sh> range is 1-32 in the ARM instruction.
S	The S modifier is not available in the Thumb-2 instruction.
T	Not available in ARM state.
U	Not allowed in an IT block. Condition codes not allowed in either ARM or Thumb state.

ARM Instruction Set
Quick Reference Card

ARM architecture versions	
n	ARM architecture version *n* and above
*n*T, *n*J	T or J variants of ARM architecture version *n* and above
5E	ARM v5E, and 6 and above
T2	All Thumb-2 versions of ARM v6 and above
6K	ARMv6K and above for ARM instructions, ARMv7 for Thumb
Z	All Security extension versions of ARMv6 and above
RM	ARMv7-R and ARMv7-M only
XS	XScale coprocessor instruction

Flexible Operand 2	
Immediate value	`#<imm8m>`
Register, optionally shifted by constant (see below)	`Rm {, <opsh>}`
Register, logical shift left by register	`Rm, LSL Rs`
Register, logical shift right by register	`Rm, LSR Rs`
Register, arithmetic shift right by register	`Rm, ASR Rs`
Register, rotate right by register	`Rm, ROR Rs`

Register, optionally shifted by constant		
(No shift)	`Rm`	Same as `Rm, LSL #0`
Logical shift left	`Rm, LSL #<shift>`	Allowed shifts 0-31
Logical shift right	`Rm, LSR #<shift>`	Allowed shifts 1-32
Arithmetic shift right	`Rm, ASR #<shift>`	Allowed shifts 1-32
Rotate right	`Rm, ROR #<shift>`	Allowed shifts 1-31
Rotate right with extend	`Rm, RRX`	

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

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)

All ARM instructions (except those with Note C or Note U) can have any one of these condition codes after the instruction mnemonic (that is, before the first space in the instruction as shown on this card). This condition is encoded in the instruction.

All Thumb-2 instructions (except those with Note U) can have any one of these condition codes after the instruction mnemonic. This condition is encoded in a preceding IT instruction (except in the case of conditional Branch instructions). Condition codes in instructions must match those in the preceding IT instruction.

On processors without Thumb-2, the only Thumb instruction that can have a condition code is `B <label>`.

Processor Modes	
`16`	User
`17`	FIQ Fast Interrupt
`18`	IRQ Interrupt
`19`	Supervisor
`23`	Abort
`27`	Undefined
`31`	System

Prefixes for Parallel Instructions	
`S`	Signed arithmetic modulo 2^8 or 2^{16}, sets CPSR GE bits
`Q`	Signed saturating arithmetic
`SH`	Signed arithmetic, halving results
`U`	Unsigned arithmetic modulo 2^8 or 2^{16}, sets CPSR GE bits
`UQ`	Unsigned saturating arithmetic
`UH`	Unsigned arithmetic, halving results

Document Number

ARM QRC 0001L

Change Log

Issue	Date	Change	Issue	Date	Change
A	June 1995	First Release	B	Sept 1996	Second Release
C	Nov 1998	Third Release	D	Oct 1999	Fourth Release
E	Oct 2000	Fifth Release	F	Sept 2001	Sixth Release
G	Jan 2003	Seventh Release	H	Oct 2003	Eighth Release
I	Dec 2004	Ninth Release	J	May 2005	RVCT 2.2 SP1
K	March 2006	RVCT 3.0	L	March 2007	RVCT 3.1