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.	`<reglist>`	A comma-separated list of registers, enclosed in braces { and }.
`<fields>`	See Table **PSR fields**.	`<reglist-PC>`	As `<reglist>`, must not include the PC.
`<PSR>`	Either CPSR (Current Processor Status Register) or SPSR (Saved Processor Status Register)	`<reglist+PC>`	As `<reglist>`, including the PC.
`C*, V*`	Flag is unpredictable in Architecture v4 and earlier, unchanged in Architecture v5 and later.	`+/-`	+ or –. (+ may be omitted.)
`<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**.	§	See Table **ARM architecture versions**.
		`<iflags>`	Interrupt flags. One or more of a, i, f (abort, interrupt, fast interrupt).
`x, y`	B meaning half-register [15:0], or T meaning [31:16].	`<p_mode>`	See Table **Processor Modes**
`<imm8m>`	ARM: a 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits.	`SPm`	SP for the processor mode specified by `<p_mode>`
	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.	`<lsb>`	Least significant bit of bitfield.
		`<width>`	Width of bitfield. `<width>` + `<lsb>` must be <= 32.
`<prefix>`	See Table **Prefixes for Parallel instructions**	`{X}`	RsX is Rs rotated 16 bits if X present. Otherwise, RsX is Rs.
`{IA|IB|DA|DB}`	Increment After, Increment Before, Decrement After, or Decrement Before.	`{!}`	Updates base register after data transfer if ! present (pre-indexed).
	`IB` and `DA` are not available in Thumb state. If omitted, defaults to `IA`.	`{S}`	Updates condition flags if S present.
`<size>`	B, SB, H, or SH, meaning Byte, Signed Byte, Halfword, and Signed Halfword respectively.	`{T}`	User mode privilege if T present.
	`SB` and `SH` are not available in `STR` instructions.	`{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}	SI	`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}	SI	`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+lsh–1):lsb] := 0, other bits of Rd unaffected	
	Bit Field Insert	T2	`BFI Rd, Rn, #<lsb>, #<width>`	Rd[(width+lsh–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+lsh–1):lsb], Rd[31:width] = Replicate(Rn[width+lsh–1])	
	Unsigned Bit Field Extract	T2	`UBFX Rd, Rn, #<lsb>, #<width>`	Rd[(width–1):0] = Rn[(width+lsh–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