Thumb2 Instruction Set Summary

Move / Add / Subtract		***Operation***	***{S}***	***<op>***	***Notes***
MOV	R_d,<op>	R_d ← <op>	NZC	imm. const. -or- reg{,<shift>}	
ADD	R_d,R_n,<op>	R_d ← R_n + <op>	NZCV		
ADD	R_d,R_n,SP,<op>	R_d ← R_n + SP + <op>	NZCV		
ADC	R_d,R_n,<op>	R_d ← R_n + <op> + C	NZCV		
SUB	R_d,R_n,<op>	R_d ← R_n – <op>	NZCV		
SUB	R_d,SP,<op>	R_d ← SP – <op>	NZCV		
SBC	R_d,R_n,<op>	R_d ← R_n – <op> + C – 1	NZCV		
RSB	R_d,R_n,<op>	R_d ← <op> – R_n	NZCV		
NEG	R_d,R_n	R_d ← –R_n			Always updates: NZCV

Compare Instructions		***Operation***	***{S}***	***<op>***	***Notes***
CMP	R_n,<op>	R_n – <op>	n/a	imm. const. -or- reg{,<shift>}	Always updates: NZCV
CMN	R_n,<op>	R_n + <op>	n/a		Always updates: NZCV
TST	R_n,<op>	R_n & <op>	n/a		Always updates: NZC
TEQ	R_n,<op>	R_n ^ <op>	n/a		Always updates: NZC

Bitwise Instructions		***Operation***	***{S}***	***<op>***	***Notes***
AND	R_d,R_n,<op>	R_d ← R_n & <op>	NZC	imm. const. -or- reg{,<shift>}	
ORR	R_d,R_n,<op>	R_d ← R_n | <op>	NZC		
EOR	R_d,R_n,<op>	R_d ← R_n ^ <op>	NZC		
BIC	R_d,R_n,<op>	R_d ← R_n & ~<op>	NZC		
ORN	R_d,R_n,<op>	R_d ← R_n | ~<op>	NZC		
MVN	R_d,R_n	R_d ← ~R_n	NZC		

Bitfield Instructions		***Operation***	***{S}***	***Notes***
BFC	R_d,#lsb,#width	R_d<bits> ← 0	n/a	
BFI	R_d,R_n,#lsb,#width	R_d<bits> ← R_n<lsb's>	n/a	
SBFX	R_d,R_n,#lsb,#width	R_d ← R_n<bits>	n/a	Sign extends
UBFX	R_d,R_n,#lsb,#width	R_d ← R_n<bits>	n/a	Zero extends

Multiply / Divide		***Operation***	***{S}***	***Notes***
MUL	R_d,R_n,R_{dm}	R_d ← R_n x R_{dm}	NZC	32-bit product; C ← undefined
MLA	R_d,R_n,R_m,R_a	R_d ← (R_n x R_m) + R_a	n/a	32-bit product
MLS	R_d,R_n,R_m,R_a	R_d ← R_a – (R_n x R_m)	n/a	
UMULL	R_{dlo},R_{dhi},R_n,R_m	$R_{dhi}R_{dlo}$ ← R_n x R_m	n/a	Unsigned 64-bit product
UMLAL	R_{dlo},R_{dhi},R_n,R_m	$R_{dhi}R_{dlo}$ ← $R_{dhi}R_{dlo}$ + R_n x R_m	n/a	
SMULL	R_{dlo},R_{dhi},R_n,R_m	$R_{dhi}R_{dlo}$ ← R_n x R_m	n/a	Signed 64-bit product
SMLAL	R_{dlo},R_{dhi},R_n,R_m	$R_{dhi}R_{dlo}$ ← $R_{dhi}R_{dlo}$ + R_n x R_m	n/a	
UDIV	R_d,R_n,R_m	R_d ← R_n / R_m	n/a	Unsigned 32-bit quotient; no remainder
SDIV	R_d,R_n,R_m	R_d ← R_n / R_m	n/a	Signed 32-bit quotient; no remainder

Shifts		***Operation***	***{S}***	***<op>***	***Notes***
ASR	R_d,R_n,<op>	R_d ← R_n >> <op>	NZC	R_m –or- imm	Sign extends
LSL	R_d,R_n,<op>	R_d ← R_n << <op>	NZC	R_m –or- imm	Zero fills
LSR	R_d,R_n,<op>	R_d ← R_n >> <op>	NZC	R_m –or- imm	
ROR	R_d,R_n,<op>	R_d ← R_n >> <op>	NZC	R_m –or- imm	right rotate
RRX	R_d,R_n	R_d ← R_n >> 1	NZC	n/a	right shift, fill w/C

Bits / Bytes / Words		***Operation***	***{S}***	***Notes***
CLZ	R_d,R_n	R_d ← CountZeroes(R_n)	n/a	# leading zeroes (0-32)
RBIT	R_d,R_n	R_d ← RevBits(R_n)	n/a	Reverses bit order
REV	R_d,R_n	R_d ← RevByteOrder(R_n)	n/a	Reverses byte order
REV16	R_d,R_n	R_d ← RevHalfWords(R_n)	n/a	Reverses half words
REVSH	R_d,R_n	R_d ← RevLoHalf(R_n)	n/a	Reverses 2 LSbytes, sign extends
SXTB	R_d,R_n	R_d ← SignedByte(R_n)	n/a	Sign extends, may pre-rotate R_n
SXTH	R_d,R_n	R_d ← SignedHalf(R_n)	n/a	
UXTB	R_d,R_n	R_d ← UnsignedByte(R_n)	n/a	Zero extends, may pre-rotate R_n
UXTH	R_d,R_n	R_d ← UnsignedHalf(R_n)	n/a	

Branch Instructions		***Operation***	***{S}***	***Notes***
B{c}	label	PC ← PC + imm	n/a	'c' is an optional condition
BL	label	PC ← PC + imm; LR ← rtn adr	n/a	Subroutine call
BX	R_n	PC ← R_n	n/a	Used as subroutine return when R_n = LR
CBZ	R_n,label	If R_n=0, PC ← PC + imm	n/a	Cannot append condition code to CBZ
CBNZ	R_n,label	If R_n≠0, PC ← PC + imm	n/a	Cannot append condition code to CBNZ
IT$c_1c_2c_3$	cond	Each c_i is one of T, E, or *empty*	n/a	Controls 1-4 instructions in "IT block"

Literal Pool Instructions		***Operation***	***{S}***	***Notes***
ADR	R_d,label	R_d ← PC + imm	n/a	
LDR	R_d,label	R_d ← mem_{32}[PC+imm]	n/a	
LDRB	R_d,label	R_d ← mem_8[PC+imm]	n/a	Zero fills bits 31..8 of R_d
LDRH	R_d,label	R_d ← mem_{16}[PC+imm]	n/a	Zero fills bits 31..16 of R_d
LDRSB	R_d,label	R_d ← mem_8[PC+imm]	n/a	Sign extends into bits 31..8 of R_d
LDRSH	R_d,label	R_d ← mem_{16}[PC+imm]	n/a	Sign extends into bits 31..16 of R_d

Load/Store Memory		***Operation***	***{S}***	***<mem>***	***Notes***
LDR	R_d,<mem>	R_d ← mem_{32}[address]	n/a	See Memory Access Modes	
LDRB	R_d,<mem>	R_d ← mem_8[address]	n/a		Zero fills
LDRH	R_d,<mem>	R_d ← mem_{16}[address]	n/a		Zero fills
LDRSB	R_d,<mem>	R_d ← mem_8[address]	n/a		Sign extends
LDRSH	R_d,<mem>	R_d ← mem_{16}[address]	n/a		Sign extends
STR	R_d,<mem>	R_d → mem_{32}[address]	n/a		
STRB	R_d,<mem>	R_d → mem_8[address]	n/a		
STRH	R_d,<mem>	R_d → mem_{16}[address]	n/a		
LDRD	R_t,R_{t2},<mem>	R_{t2}.R_t ← mem_{64}[address]	n/a		Addr. Offset must be imm.
STRD	R_t,R_{t2},<mem>	R_{t2}.R_t → mem_{64}[address]	n/a		Addr. Offset must be imm.

Multiple Load/Store		***Operation***	***{S}***	***Notes***
POP	{reg. list}	regs ← mem[SP++]	n/a	Reg. list: Not SP; PC or LR, but not both
PUSH	{reg. list}	regs → mem[−−SP]	n/a	Reg. list may not include SP or PC.

LDMIA	R_n!,<reg. list>	regs ← mem[R_n];
		if !, then R_n += 4 × #regs
			n/a	Same as LDMFD; no R_n update w/out !

STMIA	R_n!,<reg. list>	regs → mem[R_n];
		if !, then R_n += 4 × #regs
			n/a	Same as STMEA; no R_n update w/out !

LDMDB	R_n!,<reg. list>	regs ← mem[R_n – 4 × #regs];
		if !, then R_n −= 4 × #regs
			n/a	Same as LDMEA; no R_n update w/out !

STMDB	R_n!,<reg. list>	regs → mem[R_n – 4 × #regs];
		if !, then R_n −= 4 × #regs
			n/a	Same as STMFD; no R_n update w/out !

Condition Codes:

Any one of these may be appended to any instruction mnemonic when used inside an If-Then-Else (IT) block.
(E.g., "IT NE followed by ADDNE" would add only if Z ≠ 0.) Exceptions: CBZ, CBNZ, CMP, CMN, NEG, TST, or TEQ.

Condition Code	*Meaning*	*Requirements*
EQ	**EQ**ual	Z = 1
NE	**N**ot **E**qual	Z = 0
HS	Unsigned ≥ ("**H**igher or **S**ame")	C = 1 *(Note: Synonym for "CS")*
LO	Unsigned < ("**LO**wer")	C = 0 *(Note: Synonym for "CC")*
HI	Unsigned > ("**HI**gher")	C = 1 && Z = 0
LS	Unsigned ≤ ("**L**ower or **S**ame")	C = 0 || Z = 1
GE	Signed ≥ ("**G**reater than or **E**qual")	N = V
LT	Signed < ("**L**ess **T**han")	N ≠ V
GT	Signed > ("**G**reater **T**han")	Z = 0 && N = V
LE	Signed ≤ ("**L**ess than or **E**qual")	Z = 1 || N ≠ V
CS	**C**arry **S**et	C = 1 *(Note: Synonym for "HS")*
CC	**C**arry **C**lear	C = 0 *(Note: Synonym for "LO")*
MI	**MI**nus/negative	N = 1
PL	**PL**us/positive or zero (non-negative)	N = 0
VS	o**V**erflow **S**et	V = 1
VC	o**V**erflow **C**lear	V = 0
AL	**AL**ways (unconditional)	only used with IT instruction

Shift Codes:

Any of these may be applied to the register option of "<op>" in Move / Add / Subtract, Compare, and Bitwise Groups.

<shift>	*Meaning*	*Notes*
LSL #n	**L**ogical **S**hift **L**eft by n bits	Zero fills; 0 ≤ n ≤ 31
LSR #n	**L**ogical **S**hift **R**ight by n bits	Zero fills; 1 ≤ n ≤ 32
ASR #n	**A**rithmetic **S**hift **R**ight by n bits	Sign extends; 1 ≤ n ≤ 32
ROR #n	**RO**tate **R**ight by n bits	1 ≤ n ≤ 32
RRX	**R**otate **R**ight e**X**tended (with carry) by 1 bit	

Memory Access Modes:

Any of these may be used with Load/Store Memory Instructions.
Exceptions: LDRD and STRD may not use R_m

Memory Access Mode	*Syntax*	*Meaning ("EA" = Effective Address)*	*Example*
Offset addressing	[<R_n>,#imm]	EA ← R_n + imm	[r5,#100]
	[<R_n>,<R_m>]	EA ← R_n + R_m	[r4,r5]
	[<R_n>,<R_m>,LSL #<imm>]	EA ← R_n + (R_m << imm)	[r4,r5,LSL #3]
Pre-indexed addressing	[<R_n>,#imm]!	R_n ← R_n + imm; EA ← R_n	[r5,#100]!
	[<R_n>,<R_m>]!	R_n ← R_n + R_m; EA ← R_n	[r4,r5]!
	[<R_n>,<R_m>,LSL #<imm>]!	R_n ← R_n + (R_m << imm); EA ← R_n	[r4,r5,LSL #3]!
Post-indexed addressing	[<R_n >],#imm	EA ← R_n; R_n ← R_n + imm	[r5],#100
	[<R_n >],<R_m>	EA ← R_n; R_n ← R_n + R_m	[r4],r5
	[<R_n>],<R_m>,LSL #<imm>	EA ← R_n; R_n ← R_n + (R_m << imm)	[r4],r5,LSL #3

Notes: 1. This is only a partial list of the most commonly-used Thumb2 instructions.
2. There are magnitude restrictions on immediate constants; see ARM documentation for more information.