ARM Cortex-M0+ Instruction Set
Quick Reference Card

This card lists all Thumb and Thumb-2 instructions available on Cortex-M0+ processors.
All registers are Lo (R0-R7) except where specified. Hi registers are R8-R15.

Key to Tables

`<loreglist>`	A comma-separated list of Lo registers, enclosed in braces, { and }.		`<loreglist+LR>`	A comma-separated list of Lo registers. plus the LR, enclosed in braces, { and }.
			`<loreglist+PC>`	A comma-separated list of Lo registers. plus the PC, enclosed in braces, { and }.

Operation		Assembler	Updates	Action	Notes
Move	Immediate to Lo	`MOVS Rd, #<imm>`	N Z	Rd := imm	imm range 0-255.
	Lo to Lo	`MOVS Rd, Rm`	N Z	Rd := Rm	Synonym of LSLS Rd, Rm, #0
	Any to Any	`MOV Rd, Rm`		Rd := Rm	Any register to any register.
Add	Immediate 3 bits	`ADDS Rd, Rn, #<imm>`	N Z C V	Rd := Rn + imm	imm range 0-7.
	Immediate 8 bits	`ADDS Rd, Rd, #<imm>`	N Z C V	Rd := Rd + imm	imm range 0-255. May omit second Rd.
	All registers Lo	`ADDS Rd, Rn, Rm`	N Z C V	Rd := Rn + Rm	
	Any to Any	`ADD Rd, Rd, Rm`		Rd := Rd + Rm	Any register to any register, but may not both be PC. May omit second Rd.
	With carry	`ADCS Rd, Rd, Rm`	N Z C V	Rd := Rd + Rm + C-bit	May omit second Rd.
	Value to SP	`ADD SP, SP, #<imm>`		SP := SP + imm	imm range 0-508 (word-aligned). May omit second SP.
	Form address from SP	`ADD Rd, SP, #<imm>`		Rd := SP + imm	imm range 0-1020 (word-aligned). Rd must be Lo. SP may instead be PC.
	Form address from PC	`ADR Rd, <label>`		Rd := label	label range PC to PC+1020 (word-aligned).
Subtract	Lo and Lo	`SUBS Rd, Rn, Rm`	N Z C V	Rd := Rn - Rm	
	Immediate 3 bits	`SUBS Rd, Rn, #<imm>`	N Z C V	Rd := Rn - imm	imm range 0-7.
	Immediate 8 bits	`SUBS Rd, Rd, #<imm>`	N Z C V	Rd := Rd - imm	imm range 0-255.
	With carry	`SBCS Rd, Rd, Rm`	N Z C V	Rd := Rd - Rm - NOT C-bit	
	Value from SP	`SUB SP, SP, #<imm>`		SP := SP - imm	imm range 0-508 (word-aligned).
	Negate	`RSBS Rd, Rn, #0`	N Z C V	Rd := -Rn	Synonym: NEGS Rd, Rn
Multiply	Multiply	`MULS Rd, Rm, Rd`	N Z	Rd := Rm * Rd	May omit second Rd.
Compare		`CMP Rn, Rm`	N Z C V	update APSR flags on Rn – Rm	Rn and Rm may be R0-R14.
	Negative	`CMN Rn, Rm`	N Z C V	update APSR flags on Rn + Rm	
	Immediate	`CMP Rn, #<imm>`	N Z C V	update APSR flags on Rn – imm	imm range 0-255. Rn may be R0-R14.
Logical	AND	`ANDS Rd, Rd, Rm`	N Z	Rd := Rd AND Rm	May omit second Rd.
	Exclusive OR	`EORS Rd, Rd, Rm`	N Z	Rd := Rd EOR Rm	”
	OR	`ORRS Rd, Rd, Rm`	N Z	Rd := Rd OR Rm	”
	Bit clear	`BICS Rd, Rd, Rm`	N Z	Rd := Rd AND NOT Rm	”
	Move NOT	`MVNS Rd, Rd, Rm`	N Z	Rd := NOT Rm	”
	Test bits	`TST Rn, Rm`	N Z	update APSR flags on Rn AND Rm	
Shift/rotate	Logical shift left	`LSLS Rd, Rm, #<shift>`	N Z C*	Rd := Rm << shift	Allowed shifts 0-31. * C flag unaffected if shift is 0.
		`LSLS Rd, Rd, Rs`	N Z C*	Rd := Rd << Rs[7:0]	* C flag unaffected if Rs[7:0] is 0. May omit second Rd.
	Logical shift right	`LSRS Rd, Rm, #<shift>`	N Z C	Rd := Rm >> shift	Allowed shifts 1-32.
		`LSRS Rd, Rd, Rs`	N Z C*	Rd := Rd >> Rs[7:0]	* C flag unaffected if Rs[7:0] is 0. May omit second Rd.
	Arithmetic shift right	`ASRS Rd, Rm, #<shift>`	N Z C	Rd := Rm ASR shift	Allowed shifts 1-32.
		`ASRS Rd, Rd, Rs`	N Z C*	Rd := Rd ASR Rs[7:0]	* C flag unaffected if Rs[7:0] is 0. May omit second Rd.
	Rotate right	`RORS Rd, Rd, Rs`	N Z C*	Rd := Rd ROR Rs[7:0]	* C flag unaffected if Rs[7:0] is 0. May omit second Rd.

ARM Cortex-M0+ Instruction Set
Quick Reference Card

Operation		Assembler	Action	Notes
Load	with immediate offset, word	`LDR Rd, [Rn, #<imm>]`	Rd := [Rn + imm]	imm range 0-124, multiple of 4.
	halfword	`LDRH Rd, [Rn, #<imm>]`	Rd := ZeroExtend([Rn + imm][15:0])	Clears bits 31:16. imm range 0-62, even.
	byte	`LDRB Rd, [Rn, #<imm>]`	Rd := ZeroExtend([Rn + imm][7:0])	Clears bits 31:8. imm range 0-31.
	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, <label>`	Rd := [label]	label range PC to PC+1020 (word-aligned).
	SP-relative	`LDR Rd, [SP, #<imm>]`	Rd := [SP + imm]	imm range 0-1020, multiple of 4.
	Multiple, not including base	`LDM Rn!, <loreglist>`	Loads list of registers (not including Rn)	Always updates base register, Increment After.
	Multiple, including base	`LDM Rn, <loreglist>`	Loads list of registers (including Rn)	Never updates base register, Increment After.
Store	with immediate offset, word	`STR Rd, [Rn, #<imm>]`	[Rn + imm] := Rd	imm range 0-124, multiple of 4.
	halfword	`STRH Rd, [Rn, #<imm>]`	[Rn + imm][15:0] := Rd[15:0]	Ignores Rd[31:16]. imm range 0-62, even.
	byte	`STRB Rd, [Rn, #<imm>]`	[Rn + imm][7:0] := Rd[7:0]	Ignores Rd[31:8]. imm range 0-31.
	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, #<imm>]`	[SP + imm] := Rd	imm range 0-1020, multiple of 4.
	Multiple	`STM Rn!, <loreglist>`	Stores list of registers	Always updates base register, Increment After.
Push/Pop	Push	`PUSH <loreglist>`	Push registers onto full descending stack	
	Push with link	`PUSH <loreglist+LR>`	Push LR and registers onto full descending stack	
	Pop	`POP <loreglist>`	Pop registers from full descending stack	
	Pop and return	`POP <loreglist+PC>`	Pop registers, branch to address loaded to PC	Bit[0] of the value read into PC must be 1, to avoid HardFault.
Branch	Conditional branch	`B{cond} <label>`	If {cond} then PC := label	label must be within – 256 to + 254 bytes of current instruction. See Table: Condition Field.
	Unconditional branch	`B <label>`	PC := label	label must be within ±2KB of current instruction.
	Long branch with link	`BL <label>`	LR := address of next instruction, PC := label	This is a 32-bit instruction.. label must be within ±16MB of current instruction.
	§ Branch and exchange	`BX Rm`	PC := Rm AND 0xFFFFFFFE	Bit[0] address in Rm must be 1, to avoid HardFault.
	§ Branch with link and exchange	`BLX Rm`	LR := address of next instruction, PC := Rm AND 0xFFFFFFFE	Bit[0] address in Rm must be 1, to avoid HardFault.
Extend	Signed, halfword to word	`SXTH Rd, Rm`	Rd[31:0] := SignExtend(Rm[15:0])	
	Signed, byte to word	`SXTB Rd, Rm`	Rd[31:0] := SignExtend(Rm[7:0])	
	Unsigned, halfword to word	`UXTH Rd, Rm`	Rd[31:0] := ZeroExtend(Rm[15:0])	
	Unsigned, byte to word	`UXTB Rd, Rm`	Rd[31:0] := ZeroExtend(Rm[7:0])	
Reverse	Bytes in word	`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]	
	Bytes in both halfwords	`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]	
	Bytes in low halfword, sign extend	`REVSH Rd, Rm`	Rd[15:8] := Rm[7:0], Rd[7:0] := Rm[15:8], Rd[31:16] := Rm[7] * &FFFF	
Processor state change	Supervisor Call	`SVC <immed_8>`	Supervisor Call processor exception	8-bit immediate value encoded in instruction. Formerly SWI.
	Change processor state	`CPSID <iflags>`	Disable specified interrupts	
		`CPSIE <iflags>`	Enable specified interrupts	
	Breakpoint	`BKPT <immed_8>`	Prefetch abort or enter debug state	8-bit immediate value encoded in instruction.

ARM Cortex-M0+ Instruction Set
Quick Reference Card

Operation		Assembler	Action	Notes
Move to or from PSR	PSR to register	`MRS Rd, <spec_reg>`	Rd := spec_reg	<spec_reg> may be one of APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, or CONTROL. When writing APSR or PSR use APSR_nzcvq or PSR_nzcvq.
	Register to PSR	`MSR <spec_reg>, Rm`	spec_reg := Rm	
Nop	No operation	`NOP`	None, might not even consume any time.	
Hint §	Data Memory Barrier	`DMB`	Ensure the order of observation of memory accesses.	
	Data Synchronization Barrier	`DSB`	Ensure the completion of memory accesses.	
	Instruction Synchronization Barrier	`ISB`	Flush processor pipeline and branch prediction logic.	

Condition Field

Mnemonic	Description
EQ	Equal
NE	Not equal
CS / HS	Carry Set / Unsigned higher or same
CC / LO	Carry Clear / Unsigned lower
MI	Negative
PL	Positive or zero
VS	Overflow
VC	No overflow
HI	Unsigned higher
LS	Unsigned lower or same
GE	Signed greater than or equal
LT	Signed less than
GT	Signed greater than
LE	Signed less than or equal
AL	Always. Do not use in B{cond}

§ Cortex-M0+ microcontrollers include some instructions which are not useful. BX and BLX are functional, but they do the same thing as B and BL, with the added risk of a HardFault if Bit[0] is not 1. The SEV, WFE, WFI, and YIELD hint instructions have been omitted from this document, because they function as NOPs on Cortex-M0+ . Unless these instructions are being used for code that may also be used on ARMv7 devices, there is no reason to use them.

Trademark & Copyright Notice

Need and Purpose Notes

This Quick Reference Card is needed because ARM has not published any easily readable and navigatable instruction set reference for the Cortex-M0+ architecture. The online reference manual provides fairly complete information, but it is a hassle to navigate when programming. Assembly programming is already difficult enough as it is, which is why ARM provides its own Quick Reference Cards. Unfortunately, ARM only produces Quick Reference Cards for complete ISAs and not for architectures that omit and include various portions of different ARM ISAs in a rather unpredictable manner, like Cortex-M0+.

Additionally, notes in the official Quick Reference Cards sometimes don't apply to Cortex-M0+, and other times omit information critical to their correct usage on Cortex-M0+ devices. The official cards just don't have room to provide all necessary information about all applicable core architectures, thus there is a significant need for architecture specific Quick Reference Cards.

The primary purpose of this document is to aid in learning and using assembly language as it applies to Cortex-M0+ devices. It does not exhaustively document the architecture itself, and it does not document specific Cortex-M0+ devices. The user will need to learn these things from other sources for this document to be useful.