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></loreglist>	A comma-separated list of Lo registers, enclosed in braces, { and }.		<pre><loreglist+lr></loreglist+lr></pre>	A comma-separated list of Lo registers, plus the LR, enclosed in braces, { and }.		
			<loreglist+pc></loreglist+pc>	A comma-separated list of Lo registers, plus the PC, enclosed in braces, $\{$ and $\}$.		

Move Immediate to Lo Movs Rd , Rd K-imm> N Z Rd := imm Imm range 0-255. Symptomy of ISLS Rd, Rm, #0 Any to Any Mov Rd , Rm N Z Rd := Rm Any register to any register.	
Lot OLo	
Add Immediate 3 bits immediate 3 bits immediate 8 bits ADDS Rd, Rd, # ADDS Rd, Rd, # N Z C V Rd := Rn + imm immediate 8 bits ADDS Rd, Rd, # Immediate 8 bits ADDS Rd, Rd, # N Z C V Rd := Rn + imm immediate 8 bits immediate 8 bits ADDS Rd, Rd, Rm N ADD Rd, SP, # * simm> Porm address from SP ADD Rd, SP, # * simm> Porm address from PC ADD Rd, SP, # simm> Porm address from PC ADD Rd, SP, # simm> Rd := Rd + Rm Any register to any register, but may not both be PC. May of Rd := Rd + Rm ADD Rd, SP, # simm> Rd := Rd + Rm + C-bit Immediate 3 bits Immediate 3 bits Immediate 3 bits Immediate 8 bits SUBS Rd, Rn, Rm N Z C V Rd := Rn - Imm Immediate 8 bits SUBS Rd, Rd, # simm> N Z C V Rd := Rn - Imm Immediate 8 bits SUBS Rd, Rd, Rm N N Z C V Rd := Rd - Imm Immediate 8 bits SUBS Rd, Rd, Rm N N Z C V Rd := Rd - Imm Immediate 8 bits SUBS Rd, Rd, Rm N N Z C V Rd := Rd - Imm Immediate 8 bits SUBS Rd, Rd, Rm N N Z C V Rd := Rd - Rm NOT C-bit Substantial Rd	
Immediate 8 bits ADDS Rd, Rd, # <imm> N Z C V Rd := Rd + imm imm range 0-255. May omit second Rd. </imm>	
Immediate 8 bits ADDS Rd, Rd, # <imm> N Z C V Rd:=Rd + imm imm range 0-255. May omit second Rd. </imm>	
Any to Any With carry ADC Rd, Rd, Rm N Z C V Rd := Rd + Rm Any register to any register, but may not both be PC. May or Value to SP Form address from SP ADD Rd, SP, # <imm> ADD Rd, SP, #<imm> ADD Rd, SP, #<imm> Rd := Rd + Rm Rd := Iabel Imm range 0-1020 (word-aligned). May omit second SP, Imm range 0-1020 (word-aligned). Rd must be Lo. SP may in Imm range 0-1</imm></imm></imm>	
With carry ADCS Rd, Rd, Rm ADD SP, SP, # <iram> ADD SP, SP, #<iram> ADD SP, SP, #<iram> ADD SP, SP, #<iram> ADD SP, SP, #<iram> ADD Rd, SP, SP, #<iram> ADD Rd, Rd, Rd, Rm, Rd SP, SP, SP, SP, SP, SP, SP, SP, SP, SP,</iram></iram></iram></iram></iram></iram></iram></iram></iram></iram></iram></iram></iram></iram></iram></iram></iram></iram></iram></iram></iram></iram></iram>	
Value to SP	y omit second Rd.
Rorm address from SP Form address from PC ADR Rd,	
Form address from PC	
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 := Rn - Imm Imm range 0-7. Immediate 8 bits SUBS Rd, Rd, Rm N Z C V Rd := Rn - Imm Imm range 0-255. With carry Value from SP SUB SP, SP, #<imm> N Z C V Rd := Rn - Imm Imm range 0-255. With carry Value from SP SUB SP, SP, #<imm> N Z C V Rd := Rn - NOT C-bit SP := SP - Imm Imm range 0-508 (word-aligned). Synonym: NEGS Rd, Rn May omit second Rd. RSBS Rd, Rn, #d N Z C V V Rd := Rn - Rm N Z C V V V Rd := Rn - Rm Nd V V V V V V V V V </imm></imm></imm></imm>	ly instead be PC.
Immediate 3 bits SUBS Rd, Rn, # <imm> N Z C V Rd:=Rn-imm imm range 0-7. imm range 0-7. imm range 0-255. imm range 0-508 (word-aligned). SuBS Rd, Rn, #0 N Z C V Rd:=Rd-imm imm range 0-508 (word-aligned). SuBS Rd, Rn, #0 N Z C V Rd:=Rm NOT C-bit Sp:=SP-imm imm range 0-508 (word-aligned). SuBS Rd, Rn, #0 N Z C V Rd:=Rm NOT C-bit Sp:=SP-imm imm range 0-508 (word-aligned). SuBS Rd, Rn NoT NO</imm>	
Immediate 8 bits SUBS Rd, Rn, # <imm> N Z C V Rd:= Rn-imm imm range 0-7. imm range 0-25. imm range 0-25. imm rang</imm>	
With carry Value from SP Value from SP Negate SBCS Rd, Rd, Rm SP, \$F, \$f <imm> SUB SP, SP, \$f<imm> SP, ESP - imm imm range 0-508 (word-aligned). Synonym: NEGS Rd, Rn Multiply Multiply MULS Rd, Rm, Rd N Z C V Rd:= Rn Synonym: NEGS Rd, Rn Compare CMP Rn, Rm CM Rn, Rm N Z C V Update APSR flags on Rn - Rm Update APSR flags on Rn + Rm CMN Rn, Rm Rn and Rm may be R0-R14. Logical AND Exclusive OR OR ORRS Rd, Rd, Rm N Z CN Rd; Rd, Rm N Z Rd; Rd Rd Rd, Rm N Z Rd:= Rd AND Rm Rd:= Rd EOR Rm May omit second Rd. Shift/rotate Logical shift left LSLS Rd, Rm, #<shift> N Z CR Rd:= Rd Rd:= Rd RM = Rd AND Rd:= Rd AND Rm May omit second Rd. Shift/rotate Logical shift right LSLS Rd, Rm, #<shift> N Z CR Rd:= Rd Rd:= Rd AND NOT Rm " " Shift/rotate Logical shift right LSLS Rd, Rm, #<shift> N Z CR Rd:= Rm < Rd:= Rm < Shift Allowed shifts 0-31. *C flag unaffected if shift is 0.</shift></shift></shift></imm></imm>	
Value from SP Negate SUB SP, SP, # <imm> RSBS Rd, Rn, #0 N Z C V Rd:=-Rn Synonym: NEGS Rd, Rn imm range 0-508 (word-aligned). Synonym: NEGS Rd, Rn Multiply Multiply MULS Rd, Rm, Rd N Z C V update APSR flags on Rn - Rm (MR) Rn, Rm Rd := Rm * Rd May omit second Rd. Compare CMP Rn, Rm CMN Rn, Rm CMN Rn, Rm N Z C V update APSR flags on Rn - Rm (MR) update APSR flags on Rn - Rm (MR) update APSR flags on Rn - Imm (MR) update APSR flags</imm>	
Negate	
Multiply Multiply MULS Rd, Rm, Rd N Z Rd := Rm * Rd May omit second Rd. Compare CMP Rn, Rm	
Compare CMP Rn, Rm CMN Rn, Rm CMN Rn, Rm Immediate CMP Rn, # <imm> N Z C V update APSR flags on Rn − Rm update APSR flags on Rn − Rm Immediate Rn and Rm may be R0-R14. Logical AND Exclusive OR OR ORRS Rd, Rd, Rm OR Bit clear Move NOT Test bits EORS Rd, Rd, Rm N Z Rd = Rd AND Rm N Z Rd = Rd AND NOT Rm N Z Rd = Rd = Rd AND Rm N Z Rd = Rd = Rd AND Rm N Z Rd = Rd</imm>	
Negative CMN Rn, Rm 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. </imm>	
Negative	
Logical	
Exclusive OR OR OR ORRS Rd, Rd, Rm N Z Rd := Rd EOR Rm " Rd := Rd FOR Rm " Rd := Rd	
Exclusive OR OR OR OR OR OR SRd, Rd, Rm N Z Rd := Rd EOR Rm " Rd := Rd OR Rm " Rd := Rd OR Rm " Rd := Rd AND NOT Rm Move NOT Move NOT Test bits TST Rn, Rm N Z Rd := Rd AND NOT Rm " Rd := NOT Rm " update APSR flags on Rn AND Rm Shift/rotate Shift/rotate LSLS Rd, Rd, Rs N Z Rd := Rd < NR Rm " Allowed shifts 0-31. * C flag unaffected if shift is 0. * C flag unaffected if Rs[7:0] is 0. May omit second Rd. Logical shift right LSRS Rd, Rm, # <shift> N Z Rd := Rd < NS[7:0] Rd := Rd < Rd := Rd < NS[7:0] Rd := Rd < Rd := Rd</shift>	
Bit clear	
Move NOT Test bits TST Rn, Rm N Z update APSR flags on Rn AND Rm Shift/rotate Logical shift left LSLS Rd, Rd, Rs LSLS Rd, Rd, Rs LSRS Rd, Rm, # <shift> N Z C* Rd:= Rm << shift N Z Rd:= NOT Rm update APSR flags on Rn AND Rm Allowed shifts 0-31. * C flag unaffected if shift is 0. * C flag unaffected if Rs[7:0] is 0. May omit second Rd. LSRS Rd, Rm, #<shift> N Z C* Rd:= Rm >> shift Allowed shifts 1-32.</shift></shift>	
Test bits TST Rn, Rm N Z update APSR flags on Rn AND Rm LSLS Rd, Rm, # <shift> N Z C* Rd:= Rm << shift Rd:= Rd << Rs[7:0] Logical shift right LSRS Rd, Rm, #<shift> N Z C* Rd:= Rm >> shift Allowed shifts 0-31. *C flag unaffected if shift is 0. *C flag unaffected if Rs[7:0] is 0. May omit second Rd. Allowed shifts 1-32.</shift></shift>	
Shift/rotate Logical shift left	
LSLS Rd, Rd, Rs	
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. LSRS Rd, Rm, # <shift> N Z C Rd:= Rm >> shift Allowed shifts 1-32.</shift>	
LSRS Rd, Rd, Rs N 7. C* Rd := Rd >> Rs[7:0] *C flag unaffected if Rs[7:0] is 0. May omit second Rd.	
1. 2 0 1 1	
Arithmetic shift right	
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>]</imm>	Rd := [Rn + imm]	imm range 0-124, multiple of 4.		
	halfword	LDRH Rd, [Rn, # <imm>]</imm>	Rd := ZeroExtend([Rn + imm][15:0])	Clears bits 31:16. imm range 0-62, even.		
	byte	LDRB Rd, [Rn, # <imm>]</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></label>	Rd := [label]	label range PC to PC+1020 (word-aligned).		
	SP-relative	LDR Rd, [SP, # <imm>]</imm>	Rd := [SP + imm]	imm range 0-1020, multiple of 4.		
	Multiple, not including base	LDM Rn!, <loreglist></loreglist>	Loads list of registers (not including Rn)	Always updates base register, Increment After.		
	Multiple, including base	LDM Rn, <loreglist></loreglist>	Loads list of registers (including Rn)	Never updates base register, Increment After.		
Store	with immediate offset, word	STR Rd, [Rn, # <imm>]</imm>	[Rn + imm] := Rd	imm range 0-124, multiple of 4.		
	halfword	STRH Rd, [Rn, # <imm>]</imm>	[Rn + imm][15:0] := Rd[15:0]	Ignores Rd[31:16]. imm range 0-62, even.		
	byte	STRB Rd, [Rn, # <imm>]</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>]</imm>	[SP + imm] := Rd	imm range 0-1020, multiple of 4.		
	Multiple	STM Rn!, <loreglist></loreglist>	Stores list of registers	Always updates base register, Increment After.		
Push/Pop	Push	PUSH <loreglist></loreglist>	Push registers onto full descending stack			
' '	Push with link	PUSH <loreglist+lr></loreglist+lr>	Push LR and registers onto full descending stack			
	Pop	POP <loreglist></loreglist>	Pop registers from full descending stack			
	Pop and return	POP <loreglist+pc></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></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></label>	PC := label	label must be within ±2KB of current instruction.		
	Long branch with link	BL <label></label>	LR := address of next instruction, PC := label	This is a 32-bit instruction label must be within ± 16 MB of current instruction.		
	§ Branch and exchange	BX Rm	PC := Rm AND 0xFFFFFFE	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 0xFFFFFFE	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	Supervisor Call	SVC <immed_8></immed_8>	Supervisor Call processor exception	8-bit immediate value encoded in instruction, Formerly SWI,		
state	Change processor state	CPSID <iflags></iflags>	Disable specified interrupts			
		CPSIE <iflags></iflags>	Enable specified interrupts			
change	Breakpoint	BKPT <immed_8></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	embler Action		Notes		
Move to or from PSR	PSR to register Register to PSR		Rd:=specreg specreg:=Rm		<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.</spec_reg>		
Nop	No operation	NOP	None, might not even consume any time.				
Hint §	Data Memory Barrier Data Synchronization Barrier	DMB DSB	Ensure the order of observation of memory accesses. Ensure the completion of memory accesses.		Condition Field		
	Instruction Synchronization Barrier	ISB	Flush processor pipeline and branch prediction logic.		Mnemonic	Description	

Condition Field Description Mnemonic EQ. Equal NE Not equal CS / HS Carry Set / Unsigned higher or same Carry Clear / Unsigned lower CC / LO MI Negative PLPositive or zero VS Overflow VC No overflow Unsigned higher Н Unsigned lower or same LS GΕ Signed greater than or equal Signed less than LT GT Signed greater than Signed less than or equal LΕ ΑL Always. Do not use in B{cond}

Trademark & Copyright Notice

This document was made from scratch but is deliberately similar to official ARM Quick Reference Cards to maximize readability for those already familiar with them. The author of this card is not affiliated with ARM in any way, this is not an official ARM product, and it has not been reviewed for correctness by ARM.

ARM, Thumb, and Cortex-M0+ are registered trademarks of ARM Limited in the EU and other countries. Any other brands or product names mentioned here may be the trademarks of their respective owners.

Any information in this document is subject to change. The information provided was obtained from the arm-none-eabi version of the GCC and GAS GNU compilers and from ARM's web version of the Cortex-M0+ instruction set manual, with some verification from ARM's `Thumb 16-bit Instruction Set Quick Reference Card` (ARM QRC 0006E) and ARM's `ARM and Thumb-2 Instruction Set Quick Reference Card` (ARM QRC 0001M).

As with ARM's own Quick Reference Cards, this is intended only to assist the reader with the product. Neither ARM Ltd nor the author of this card shall be liable for any loss or damages arising from the use of any information in this card, any information that may have been ommitted, or any incorrect use of the product.

The information provided on this card is copyrighted by ARM Limited. This card is provided as an educational reference material for those who desire to learn to program the Cortex-M0+ architecture in assembly or merely better understand it, and is published under fair-use doctrine as educational material, This card may be freely distributed in electronic and printed form as educational material under fair-use doctrine, in the U.S. and other jurisdictions with applicable copyright laws. This card and the information therein may not be distributed for any other use where fair-use does not apply, except within the license terms set by ARM in the source materials mentioned above.

§ 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,

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.