ARM Cortex-M3/M4 instruction set mini-quickref

Mnemonic	Operands	Action	Flags	Brief description	
Move instructions					
`MOV{S}`	`Rd, Op2`	Rd = Op2	`N,Z,C`	Move	
				`Op2 = #0x000000XY | #0x00XY00XY | #0xXY00XY00 | #0xXYXYXYXY`	
				`Op2 = Rm {, ASR #n}`	Arithmetic Shift Right
				`Rm {, LSL #n}`	Logical Shift Left
				`Rm {, LSR #n}`	Logical shift right n bits, n=1..32
				`Rm {, ROR #n}`	Rotate right n bits, n=1..31
`MVN{S}`	`Rd, Op2`	Rd = ~Op2	`N,Z,C`	Move NOT. Op2, see mov	
`MOVW`	`Rd, #imm16`	Rd[31:0] = imm16	`N,Z,C`	Move 16-bit constant	
`MOVT`	`Rd, #imm16`	Rd[31:16] = imm16	-	Move Top, don't change bottom half-word	
`LDR`	`Rd, =value`	Rd=value	-	Load register with value, immediate or PC-relative	
Arithmetic processing instructions					
`ADD{S}`	`{Rd,} Rn, Op2`	Rd = Rn + Op2	`N,Z,C,V`	Add. Op2, see mov	
`ADC{S}`	`{Rd,} Rn, Op2`	Rd = Rn + Op2 + C	`N,Z,C,V`	Add with Carry. Op2, see mov	
`SUB{S}`	`{Rd,} Rn, Op2`	Rd = Rn - Op2	`N,Z,C,V`	Subtract. Op2, see mov	
`SBC{S}`	`{Rd,} Rn, Op2`	Rd = Rn - Op2 - not C	`N,Z,C,V`	Subtract with Carry. Op2, see mov	
`MUL{S}`	`{Rd,} Rn, Rm`	Rd = Rn * Rm	`N,Z`	Multiply, 32-bit result	
`MLA`	`Rd, Rn, Rm, Ra`	Rd = Ra + (Rn * Rm)	-	Multiply with Accumulate, 32-bit result	
`SDIV`	`{Rd,} Rn, Rm`	Rd = Rn / Rm	-	Signed integer Divide	
`UDIV`	`{Rd,} Rn, Rm`	Rd = Rn / Rm	-	Unsigned integer Divide	
Logical and shift instructions					
`AND{S}`	`{Rd,} Rn, Op2`	Rd = Rn & Op2	`N,Z,C`	Logical AND. Op2, see mov	
`BIC{S}`	`{Rd,} Rn, Op2`	Rd = Rn & (~Op2)	`N,Z,C`	Bit Clear. Op2, see mov	
`ORR{S}`	`{Rd,} Rn, Op2`	Rd = Rn | Op2	`N,Z,C`	Logical OR. Op2, see mov	
`EOR{S}`	`{Rd,} Rn, Op2`	Rd = Rn ^ Op2	`N,Z,C`	Exclusive OR. Op2, see mov	
`LSL{S}`	`Rd, Rm, <Rs|#n>`	Rd = Rm << <Rs|#n>	`N,Z,C`	Logical Shift Left	
`LSR{S}`	`Rd, Rm, <Rs|#n>`	Rd = Rm >> <Rs|#n>	`N,Z,C`	Logical Shift Right	
`ASR{S}`	`Rd, Rm, <Rs|#n>`	Rd = Rm >> <Rs|#n>, Rd[31] = Rm[31]	`N,Z,C`	Arithmetic Shift Right	
`ROR{S}`	`Rd, Rm, <Rs|#n>`	Rd = Rm rotate <Rs|#n>	`N,Z,C`	Rotate Right	

Mnemonic	Operands	Action	Flags	Brief description
Memory access instructions				
LDR{B|SB|H|SH}{T}	Rt, [Rn, {#offset}]	Rt = *(Rn {+ offset})	–	Load Register with word | byte | signed byte | halfword | signed halfword, indirect addressing with immediate offset
LDR{B|SB|H|SH}{T}	Rt, [Rn, #offset]!	Rt = *(Rn {+ offset}), Rn+=offset	–	Load Register, indirect addressing, pre-indexed
LDR{B|SB|H|SH}{T}	Rt, [Rn], #offset	Rt = *(Rn), Rn+=offset	–	Load Register, indirect addressing, post-indexed
STR{B|H}{T}	Rt, [Rn, {#offset}]	*(Rn {+ offset}) = Rt	–	Store Register word | byte | halfword, with immediate offset
STR{B|H}{T}	Rt, [Rn, #offset]!	*(Rn {+ offset}) = Rt, Rn+=offset	–	Store Register word | byte | halfword, pre-indexed
STR{B|H}{T}	Rt, [Rn], #offset	*(Rn) = Rt, Rn+=offset	–	Store Register word | byte | halfword, post-indexed
LDR{B|SB|H|SH}{T}	Rt, [Rn, Rm {, LSL #n}]	Rt = *(Rn + Rm {<<n})	–	Load Register with word | byte | signed byte | halfword | signed halfword, register offset with optional left shift, n=0..3
STR{B|SB|H|SH}{T}	Rt, [Rn, Rm {, LSL #n}]	*(Rn + Rm {<<n}) = Rt	–	Store Register word | byte | halfword, register offset with optional shift, n=0..3
Compare and {conditionnal} branch instructions				
CMP	Rn, Op2	Rn - Op2 -> N,Z,C,V	N,Z,C,V	Compare. Op2, see mov
B{cc}	label	if (cc == true) PC=label	–	Branch {conditionally}

cc	test	cc	test
EQ	Rn == Op2	NE	Rn != Op2
LT (signed)	Rn < Op2	GT (signed)	Rn > Op2
LE (signed)	Rn <= Op2	GE (signed)	Rn >= Op2
LO (unsigned)	Rn < Op2	HI (unsigned)	Rn > Op2
LS (unsigned)	Rn <= Op2	HS (unsigned)	Rn >= Op2

Mnemonic	Operands	Action	Flags	Brief description
Function handling instructions				
BL	label	LR = PC; PC = label	–	Branch with Link
BLX	Rm	LR = PC; PC = Rm	–	Branch indirect with Link
BX	Rm	PC = Rm	–	Branch indirect
POP	reglist	for reg in reglist do reg=*SP++ end	–	Pop registers from descending stack (sp). reglist example: {R0, R2-R5, PC}
PUSH	reglist	for reg in reglist do *(--SP)=reg end	–	Push registers onto descending stack (sp)
Special instructions				
MRS	Rd, spec_reg	Rd = spec_reg	–	Move from Special Register to general Register
MSR	spec_reg, Rm	spec_reg = Rm	N,Z,C,V	Move from general Register to Special Register
CPS<ID|IE>	<i|f>	-	–	Change Processor State, i = PRIMASK register, f = FAULTMASK register, ID = Interrupt Disable, IE = Inetrrupt Enable
SVC	#imm	-	–	Supervisor Call
NOP	–	-	–	No Operation