ARM Thumb-2 Quick Reference

[2013-12-07]

The tables below contain a subset of the available integer instructions. Instructions are 16 or 32 bits, depending on operands. All registers are 32 bits wide.

Memory access instructions

name	operation	syntax	description	example
ldr	load address	`ldr rd, constant`	rd ← constant (macro)	`ldr r0, MADDR`
ldr	load	`ldr rd, [ra, #offs]`[1]	rd ← M[ra+offs]	`ldr r1, [r0, #4]`
str	store	`str rs, [ra, #offs]`[1]	rs → M[ra+offs]	`str r1, [r2, #-4]`
ldrb	load byte	`ldr rd, [ra, #offs]`[1]	rd ← M[ra+offs]	`ldrb r1, [r0, #8]`
strb	store byte	`str rs, [ra, #offs]`[1]	rs → M[ra+offs]	`strb r1, [r0, #8]`
ldr	load preindexed	`ldr rd, [ra, #offs]!`	ra += offs, rd ← M[ra]	`ldr r1, [r0, #4]!`
str	store preindexed	`str rs, [ra, #offs]!`	ra += offs, rs → M[ra]	`str r1, [r0, #4]!`
ldr	load postindexed	`ldr rd, [ra], #offs`	rd ← M[ra], ra += offs	`ldr r1, [r0], #4`
str	store postindexed	`str rs, [ra], #offs`	rs → M[ra], ra += offs	`str r1, [r0], #4`
ldm	load multiple	`ldm rd, {reglist}`[2]	reglist ← M[rd,rd+4, . . .]	`ldm r0, {r1, r3-r5}`
stm	store multiple	`stm rd, {reglist}`[2]	reglist → M[rd,rd+4, . . .]	`stm r0, {r1, r3-r5}`
push	stack push	`push {reglist}`[2]	reglist → stack (macro)	`push {r4-r7, lr}`
pop	stack pop	`pop {reglist}`[2]	reglist ← stack (macro)	`pop {r4-r7, pc}`

[1]: offs: offset = -255 . . . 4095 with superscript 1, otherwise -255 . . . 255

[2]: `reglist` is a comma separated list of individual registers or a range of registers

Arithmetic operations

name	operation	syntax	description	example
add	add	`add rd, ra, rb`	rd ← ra + rb	`add r1, r2, r3`
add	add immediate	`add rd, ra, #imm12`	rd ← ra + imm12	`add r1, r2, #8`
adc	add with carry	`adc rd, ra, rb`	rd ← ra + rb + C	`adc r1, r2, r3`
adds	add and set	`adds rd, ra, rb`	rd ← ra + rb, set N,Z,C,V	`adds r1, r2, r3`
sub[1]	subtract	`sub rd, ra, rb`	rd ← ra - rb	`sub r1, r2, r3`
rsb[1]	reverse subtract	`rsb rd, ra, rb`	rd ← rb - ra	`rsb r1, r2, r3`
cmp	compare	`cmp ra, rb`	N,Z,C,V ← ra - rb	`cmp r1, r2`
cmp	compare immediate	`cmp ra, #imm`	N,Z,C,V ← ra - imm	`cmp r1, #1`
mul	multiply	`mul rd, ra, rb`[2]	rd ← (ra * rb)(L)	`mul r1, r2, r3`
mla	multiply and accumulate	`mla rd, ra, rb, rc`[2]	rd ← (ra * rb + rc)(L)	`mla r1, r2, r3, r4`
sdiv	signed divide	`sdiv rd, ra, rb`	rd ← ra / rb	`sdiv r1, r2, r3`
udiv	unsigned divide	`udiv rd, ra, rb`	rd ← unsigned ra / rb	`udiv r1, r2, r3`

[1]: `sub` has alternate forms like `add` (immediate, `sbc` and `subs`), `rsb` supports immediate and `rsbs`

[2]: returns the 32 least significant bits of the 64-bit product

Logical operations

name	operation	syntax	description	example
and	and	`and rd,ra,rb`[1,2]	rd ← ra ∧ rb	`and r1, r2, r3`
bic	bit clear	`bic rd,ra,rb`[1,2]	rd ← ra ∧ $\overline{rb}$	`bic r1, r2, r3`
orr	or	`orr rd,ra,rb`[1,2]	rd ← ra ∨ rb	`orr r1, r2, r3`
eor	xor	`eor rd,ra,rb`[1,2]	rd ← ra ⊕ rb	`eor r1, r2, r3`
mov	move register	`mov[s] rd,rb`[1,2]	rd ← rb	`mov r1, r2`
mov	move immediate	`mov rd, #imm16`[3]	rd ← imm16	`mov r1, #0`
mvn	move negated	`mvn rd,rb`[1,2]	rd ← $\overline{rb}$	`mvn r1, r2`
lsl	logical shift left	`lsl rd,ra,rb`[4]	rd ← ra sl	`lsl r1, r2, #2`
lsr	logical shift right	`lsr rd,ra,rb`[4]	rd ← ra sr rb	`lsr r1, r2, #2`
asr	arithmetic shift right	`asr rd,ra,rb`[4]	rd ← ra asr rb	`asr r1, r2, #2`
ror	rotate right	`ror rd,ra,rb`[4]	rd ← ra rr rb	`ror r1, r2, #2`

[1]: The instruction has an alternate form with a “s” suffix, setting status bits N,Z,C,V.

[2]: “rb” can be a 8-bit constant `0xab, 0x00ab00ab, 0xab00ab00` or `0xabababab`

[3]: imm16: a constant in the range 0. . . 65536.

[4]: “rb” is a register value rb(7:0) or a constant 1. . . 31

BRANCH INSTRUCTIONS

name	operation	syntax	description	example
b	branch	`b[cc] label`[1]	PC ← PC + *se* label	`bne loop`
bx	branch indirect	`bx rd`	PC ← rd	`bx r1`
bl	branch and link	`bl label`	LR ← PC+4, PC ← PC + *se* label	`bl loop`
blx	branch link indirect	`blx rd`	LR ← PC+4, PC ← rd	`blx r1`
cbz	compare and branch	`cbz ra, label`[2]	ra ≡ 0: PC ← PC + label	`cbz r1, loop`
cbnz	compare and branch	`cbnz ra, label`[2]	ra ≠ 0: PC ← PC + label	`cbnz r1, loop`

[1]: The alternate form bcc branch only if the condition cc is satisfied. The condition might be eq ($= 0$), gt (> 0), lt (< 0), ne ($\neq 0$), ge (≥ 0), and le (≤ 0). The constant *label* is (at least) ± 1 MiB.

[2]: Compares and jumps *without* using the status bits N,Z,C,V. Supports forward branches of 4…130 bytes only.

IF-THEN INSTRUCTION

Format: it*xyz* cc

- Letters *x, y, z* must be **t** (then) or **e** (else)
- 1-4 following instructions with a then-condition cc or an opposite else-condition !cc

Example: `r3 = max(r1, r2)`

```
    cmp r1, r2
    ite ge
    movge r3, r1   @then is ge
    movlt r3, r2   @else is lt
```

ARM REGISTER SET

name	domain	description
r0-r3	Lo	Argument / scratch registers
r4-r7	Lo	Variable (saved) registers
r8-r11	Hi	Variable (saved) registers
r12 or ip	Hi	inter-procedural / scratch register
r13 or sp	Hi	Stack register, holds stack address
r14 or lr	Hi	Link register, to hold pc value on function calls
r15 or pc	Hi	Program counter
APSR(31:28)	*special*	status bits N,Z,C,V

Assembler subprograms conformant to the AAPCS need to save any of r4-r11 on stack and restore on return if used for temporary storage. Subprogram arguments 1-4 goes to r0-r3 and function return is stored in r0.