ARM Reference

1. Data Processing and Branching Instructions

Instruction	Meaning
and{cond}{s} Rd, Rn, op2	Rd ← Rn && op2
eor{cond}{s} Rd, Rn, op2	Rd ← Rn ⊕ op2
sub{cond}{s} Rd, Rn, op2	Rd ← Rn - op2
rsb{cond}{s} Rd, Rn, op2	Rd ← op2 - Rn
add{cond}{s} Rd, Rn, op2	Rd ← Rn + op2
adc{cond}{s} Rd, Rn, op2	Rd ← Rn + op2 + C
sbc{cond}{s} Rd, Rn, op2	Rd ← Rn - op2 + (C - 1)
rsc{cond}{s} Rd, Rn, op2	Rd ← op2 - Rn + (C - 1)
tst{cond} Rn, op2	Rn & op2 PSR ← Flags (Rn & op2)
teq{cond} Rn, op2	Rn ⊕ op2 PSR ← Flags (Rn ⊕ op2)
cmp{cond} Rn, op2	Rn - op2 PSR ← Flags (Rn - op2)
cmn{cond} Rn, op2	op2 - Rn PSR ← Flags (op2 - Rn)
orr{cond}{s} Rd, Rn, op2	Rd ← Rn ‖ op2
mov{cond}{s} Rd, op2	Rd ← op2
bic{cond}{s} Rd, Rn, op2	Rd ← Rn && ∼op2
mvn{cond}{s} Rd, op2	Rd ← ∼op2
mul{cond}{s} Rd, Rm, Rs{cond}{s}	Rd ← Rm × Rs
umull{cond}{s} Rdhi, Rdlow, Rm, Rs	Rdhi:Rdlow ← unsigned(Rm × Rs)
smull{cond}{s} Rdhi, Rdlow, Rm, Rs	Rdhi:Rdlow ← signed(Rm × Rs)
udiv{cond} Rdhi, Rdlow, Rm, Rs	Rd ← unsigned(Rn ÷ Rm)
sdiv{cond} Rdhi, Rdlow, Rm, Rs	Rd ← signed(Rn ÷ Rm)
b{cond} label	PC ← PC + 8 + 4 × Imm
bl{cond} label	PC ← PC + 8 + 4 × Imm LR ← PC + 4
bx{cond} Rm	PC ← Rm
blx{cond} Rm	PC ← Rm LR ← PC + 4

2. Memory-Centric Instructions

Instruction	Meaning
`ldr{cond} Rt, =label`	`Rt` ← `PC + (4 × Imm)`
`ldr{b}{cond} Rt, [Rn]`	`Rt` ← `MEM[Rn]`
`ldr{b}{cond} Rt, [Rn, {-}Rm {, shift}]`	`Rt` ← `MEM[Rn ± shift(Rm)]`
`ldr{b}{cond} Rt, [Rn, {-}Rm {, shift}]!`	`Rn` ← `Rn ± shift(Rm)` then `Rt` ← `MEM[Rn]`
`ldr{b}{cond} Rt, [Rn], {-}Rm {, shift}`	`Rt` ← `MEM[Rn]` then `Rn` ← `Rn ± shift(Rm)`
`ldr{b}{cond} Rt, [Rn, #offset]`	`Rt` ← `MEM[Rn + offset]`
`ldr{b}{cond} Rt, [Rn, #offset]!`	`Rn` ← `Rn + offset` then `Rt` ← `MEM[Rn]`
`ldr{b}{cond} Rt, [Rn], #offset`	`Rt` ← `MEM[Rn]` then `Rn` ← `Rn + offset`
`str{b}{cond} Rt, [Rn]`	`Rt` ← `MEM[Rn]`
`str{b}{cond} Rt, [Rn, {-}Rm {, shift}]`	`MEM[Rn ± shift(Rm)]` ← `Rt`
`str{b}{cond} Rt, [Rn, {-}Rm {, shift}]!`	`Rn` ← `Rn ± shift(Rm)` then `MEM[Rn]` ← `Rt`
`str{b}{cond} Rt, [Rn], {-}Rm {, shift}`	`MEM[Rn]` ← `Rt` then `Rn` ← `Rn ± shift(Rm)`
`str{b}{cond} Rt, [Rn, #offset]`	`MEM[Rn + offset]` ← `Rt`
`str{b}{cond} Rt, [Rn, #offset]!`	`Rn` ← `Rn + offset` then `MEM[Rn]` ← `Rt`
`str{b}{cond} Rt, [Rn], #offset`	`MEM[Rn]` ← `Rt` then `Rn` ← `Rn + offset`
`ldm Rn {!}, {Reglist}`	Use `Rn` as base addr. for memory to load registers, memory address increments by 4 after each read
`ldmib Rn {!}, {Reglist}`	Use `Rn + 4` as base addr. for memory to load registers, memory address increments by 4 after each read
`ldmdb Rn {!}, {Reglist}`	Use `Rn` as base addr. for memory to load registers, memory address decrements by 4 before each read
`ldmda Rn {!}, {Reglist}`	Use `Rn` as base addr. for memory to load registers, memory address decrements by 4 before after read
`str Rn {!}, {Reglist}`	Use `Rn` as base addr. for memory to store registers, memory address increments by 4 after each write
`strib Rn {!}, {Reglist}`	Use `Rn + 4` as base addr. for memory to store registers, memory address increments by 4 after each write
`strdb Rn {!}, {Reglist}`	Use `Rn` as base addr. for memory to store registers, memory address decrements by 4 before each write
`strda Rn {!}, {Reglist}`	Use `Rn` as base addr. for memory to store registers, memory address decrements by 4 before after write
`push {Reglist}`	Push registers in list to stack
`pop {Reglist}`	Pop registers in list from stack

3. Parameters for Instructions

   Any parameter enclosed in {} except `Reglist` is optional and can be ommitted.

   - {`cond`}
     This parameter is for conditional execution. If left out, the instruction will always execute. The following condition codes are available:

Condition Code	Meaning	Flags
`EQ`	Equal	`Z == 1`
`NE`	Not equal	`Z == 0`
`CS`	Carry set	`C == 1`
`CC`	Carry clear	`C == 0`
`MI`	Negative	`N == 1`
`PL`	Positive	`N == 0`
`VS`	Overflow	`V == 1`
`VC`	No overflow	`V == 0`
`HI`	Unsigned greater than	`C == 1 && Z == 0`
`LS`	Unsigned less than or equal	`C == 0 ‖ Z == 1`
`GE`	Signed greater than or equal	`N == V`
`LT`	Signed less than	`N ≠ V`
`GT`	Signed greater than	`Z == 0 && N == V`
`LE`	Signed less than or equal	`Z == 1 ‖ N ≠ V`

   - {`s`}
     This parameter updates the status flags to reflect the result of the operation. If left out, the status flags do not change.
   - `op2`
     This parameter is the flexible operand and may be either an additional register (`Rm`) or an immediate value.
     a . Register
     When using a register for `op2`, an additional modifier may be specified to shift the register. The syntax is `Rm type Rs/#imm` where the type can be `ASR` (arithmetic shift right), `LSL` (logical shift left), `LSR` (logical shift right), `ROR` (rotate right) or `RRX` (rotate right extended with carry). Either the least signficant byte of a register `Rs` or an immediate value from 0 to 31 can be used to specify the number of shifts.
     b . Immediate
     When using an immediate value for `op2`, the value must be unsigned. It can range from 0 to 255 or any unsigned 32-bit number that can be expressed by shifting an 8-bit number left.
   - {`!`}
     This operand will cause the base address register of a memory operation to be updated with the final resulting effective address of the instruction.
   - {`shift`}
     This operand is used in memory instructions and can shift left the register `Rm` 0 to 31 times.

- `#offset`
  This operand is used in memory instructions and provides an offset of ± 4095.
- {`b`}
  This operand is used in memory instructions to designate that a byte is being loaded or stored. If a byte is being loaded from memory, the target register has only the least significant byte overwritten with the value from memory.

4. Register Names

Registers	Names	Use
`R0`	A1	Argument/result/scratch reg. 1
`R1`	A2	Argument/result/scratch reg. 2
`R2`	A3	Argument/result/scratch reg. 3
`R3`	A4	Argument/result/scratch reg. 4
`R4`	V1	Variable reg. 1
`R5`	V2	Variable reg. 2
`R6`	V3	Variable reg. 3
`R7`	V4	Variable reg. 4
`R8`	V5	Variable reg. 5
`R9`	V6	Variable reg. 6
`R10`	V7	Variable reg. 7
`R11`	V8	Variable reg. 8
`R12`	IP	Intra-procedural-call scratch reg.
`R13`	SP	Stack pointer
`R14`	LR	Link reg.
`R15`	PC	Program counter