# ARM assembly language reference card

| | | |
|---|---|---|
| MOV*cd*S | *reg*, *arg* | copy argument (S = set flags) |
| MVN*cd*S | *reg*, *arg* | copy bitwise NOT of argument |
| AND*cd*S | *reg*, *reg*, *arg* | bitwise AND |
| ORR*cd*S | *reg*, *reg*, *arg* | bitwise OR |
| EOR*cd*S | *reg*, *reg*, *arg* | bitwise exclusive-OR |
| BIC*cd*S | *reg*, $reg_a$, $arg_b$ | bitwise $reg_a$ AND (NOT $arg_b$) |
| ADD*cd*S | *reg*, *reg*, *arg* | add |
| SUB*cd*S | *reg*, *reg*, *arg* | subtract |
| RSB*cd*S | *reg*, *reg*, *arg* | subtract reversed arguments |
| ADC*cd*S | *reg*, *reg*, *arg* | add with carry flag |
| SBC*cd*S | *reg*, *reg*, *arg* | subtract with carry flag |
| RSC*cd*S | *reg*, *reg*, *arg* | reverse subtract with carry flag |
| CMP*cd* | *reg*, *arg* | update flags based on subtraction |
| CMN*cd* | *reg*, *arg* | update flags based on addition |
| TST*cd* | *reg*, *arg* | update flags based on bitwise AND |
| TEQ*cd* | *reg*, *arg* | update flags based on bitwise exclusive-OR |

| | | |
|---|---|---|
| B*cd* | $imm_{12}$ | branch to $imm_{12}$ words away |
| BL*cd* | $imm_{12}$ | copy PC to LR, then branch |
| BX*cd* | *reg* | copy *reg* to PC |
| SWI*cd* | $imm_{24}$ | software interrupt |
| LDR*cd*B | *reg*, *mem* | loads word/byte from memory |
| STR*cd*B | *reg*, *mem* | stores word/byte to memory |
| LDM*cdum* | *reg*!, *mreg* | loads into multiple registers |
| STM*cdum* | *reg*!, *mreg* | stores multiple registers |
| SWP*cd*B | $reg_d$, $reg_m$, [$reg_n$] | copies $reg_m$ to memory at $reg_n$, old value at address $reg_n$ to $reg_d$ |

| | | |
|---|---|---|
| MUL*cd*S | $reg_d$, $reg_a$, $reg_b$ | multiply $reg_a$ and $reg_b$, places lower 32 bits into $reg_d$ |
| MLA*cd*S | $reg_d$, $reg_a$, $reg_b$, $reg_c$ | places lower 32 bits of $reg_a \cdot reg_b + reg_c$ into $reg_d$ |
| UMULL*cd*S | $reg_\ell$, $reg_u$, $reg_a$, $reg_b$ | multiply $reg_a$ and $reg_b$, place 64-bit unsigned result into $\{reg_u, reg_\ell\}$ |
| UMLAL*cd*S | $reg_\ell$, $reg_u$, $reg_a$, $reg_b$ | place unsigned $reg_a \cdot reg_b + \{reg_u, reg_\ell\}$ into $\{reg_u, reg_\ell\}$ |
| SMULL*cd*S | $reg_\ell$, $reg_u$, $reg_a$, $reg_b$ | multiply $reg_a$ and $reg_b$, place 64-bit signed result into $\{reg_u, reg_\ell\}$ |
| SMLAL*cd*S | $reg_\ell$, $reg_u$, $reg_a$, $reg_b$ | place signed $reg_a \cdot reg_b + \{reg_u, reg_\ell\}$ into $\{reg_u, reg_\ell\}$ |

*reg*: register

| | |
|---|---|
| R0 to R15 | register according to number |
| SP | register 13 |
| LR | register 14 |
| PC | register 15 |

*um*: update mode

| | |
|---|---|
| IA | increment, starting from *reg* |
| IB | increment, starting from *reg* + 4 |
| DA | decrement, starting from *reg* |
| DB | decrement, starting from *reg* − 4 |

*cd*: condition code

| | |
|---|---|
| AL or omitted | always |
| EQ | equal (zero) |
| NE | nonequal (nonzero) |
| CS | carry set (same as HS) |
| CC | carry clear (same as LO) |
| MI | minus |
| PL | positive or zero |
| VS | overflow set |
| VC | overflow clear |
| HS | unsigned higher or same |
| LO | unsigned lower |
| 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 |

*arg*: right-hand argument

| | |
|---|---|
| #$imm_{8*}$ | immediate (rotated into 8 bits) |
| *reg* | register |
| *reg*, *shift* | register shifted by distance |

*mem*: memory address

| | |
|---|---|
| [*reg*, #±$imm_{12}$] | *reg* offset by constant |
| [*reg*, ±*reg*] | *reg* offset by variable bytes |
| [$reg_a$, ±$reg_b$, *shift*] | $reg_a$ offset by shifted variable $reg_b$† |
| [*reg*, #±$imm_{12}$]! | update *reg* by constant, then access memory |
| [*reg*, ±*reg*]! | update *reg* by variable bytes, access memory |
| [*reg*, ±*reg*, *shift*]! | update *reg* by shifted variable†, access memory |
| [*reg*], #±$imm_{12}$ | access address *reg*, then update *reg* by offset |
| [*reg*], ±*reg* | access address *reg*, then update *reg* by variable |
| [*reg*], ±*reg*, *shift* | access address *reg*, update *reg* by shifted variable† |

† shift distance must be by constant

*shift*: shift register value

| | |
|---|---|
| LSL #$imm_5$ | shift left 0 to 31 |
| LSR #$imm_5$ | logical shift right 1 to 32 |
| ASR #$imm_5$ | arithmetic shift right 1 to 32 |
| ROR #$imm_5$ | rotate right 1 to 31 |
| RRX | rotate carry bit into top bit |
| LSL *reg* | shift left by register |
| LSR *reg* | logical shift right by register |
| ASR *reg* | arithmetic shift right by register |
| ROR *reg* | rotate right by register |