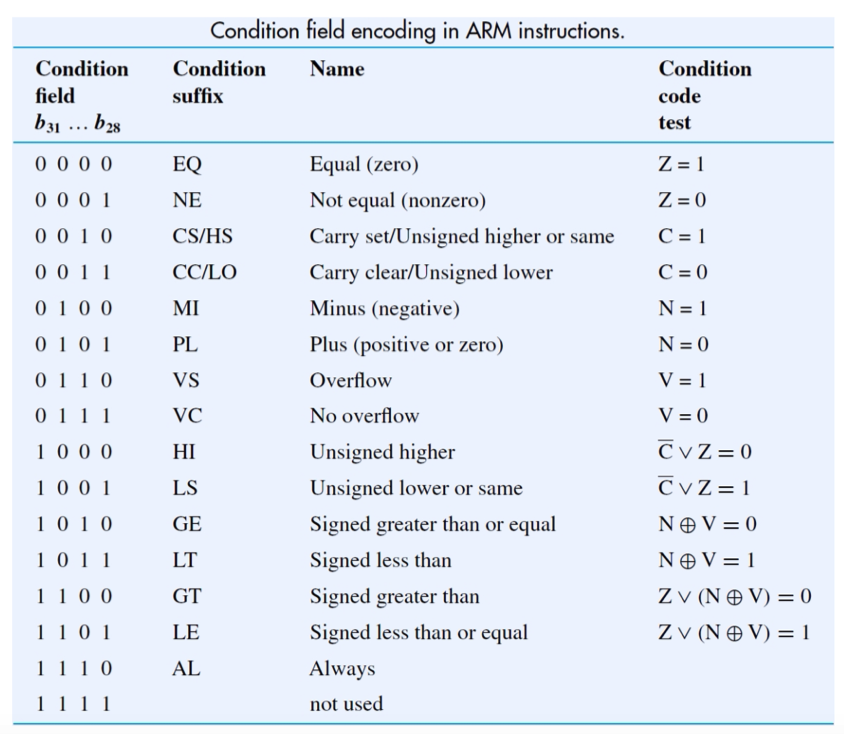
**ARM ISA**

* **Characteristics**
* RISC aspects
  + Instructions have fixed length (32 bits)
  + Only load/store instructions can access memory
  + Arithmetic/logic instructions only operate on registers
* CISC aspects
  + Auto-increment/-decrement and PC-relative addressing modes
  + Condition codes (N, Z, V, C) are used for conditional execution of instructions
  + Multiple registers can be loaded/stored from/to a block of memory
* Memory organization
  + Byte-addressable, 32-bit address space
  + Supports little/big-endian
  + Supports data sizes of words (32), half words (16), bytes (8)
* Registers
  + 16 32-bit registers R0 – R15
  + R15 = PC; R12 = frame pointer (by convention); R13 = stack pointer; R14 = link register
  + CPSR/PSR = status register – stores conditional code flags
    - N = 1 if result is negative; cleared otherwise
    - Z = 1 if result is 0; cleared otherwise
    - V = 1 if overflow occurs; cleared otherwise
    - C = 1 if carry-out occurs; cleared otherwise
* **Directives**
* Label/EQU
  + LABEL: MOV R1, #1 (instruction at address 100)
  + Equivalent to LABEL EQU 100
  + EQU directive tells assembler to replace LABEL with 100
* ORIGIN
  + ORIGIN 200
  + Tells assembler to store subsequent instructions starting at address 200
* RESERVE
  + RESERVE 100
  + Tells assembler to reserve 100 bytes for data – no data will be loaded into these memory locations
* DATAWORD
  + DATAWORD 25
  + Number of entries in the list whose memory is being reserved
* **Note: PC is always 2 instructions ahead of current instruction due to pipelining**
* Memory operations'
  + LDR/STR – load/store words
  + LDRH/STRH – half words
  + LDRB/STRB – bytes
  + LDRSH/LDRSB – sign-extended loads for half words/bytes
  + LDM/STM{FD}{cond} Rn{!}, reglist – multi-word load/store
    - {} means optional
    - FD – full descending, i.e. stack grows from higher to lower address
    - Cond – conditional execution
    - ! – load final address into Rn (i.e. writeback)
    - reglist – one or more registers
  + Use STMFD SP!, {R1, R2, R3} to push register values onto stack
  + Use LDMFD SP!, {R1, R2, R3} to pop register values from stack
* PC-relative addressing
  + LDR Rd, LABEL
    - i.e. Rd ← [[PC] + offset(LABEL)]
  + PC += 8 when LDR executes
* Pre-index addressing
  + LDR/STR Rd, [Rn, #8]{!}
    - i.e. Rd ← [[Rn] + 8] **(accessed w/ offset)**
    - ! enables writeback – additionally increments Rn: Rn ← [Rn] + 8
  + LDR/STR Rd, [Rn, ±Rm, LSL #2]
    - Shifts [Rm] left by 2, then Rd ← [[Rn] ± [Rm]]
    - Only left shift of 0-3 bits allowed
  + Use STR Rs, [SP, #-4]! to push a single register value onto stack
* Post-index addressing
  + LDR/STR Rd, [Rn], #8
    - i.e. Rd ← [[Rn]] **(accessed w/o offset)**
    - Also increments Rn: Rn ← [Rn] + 8
  + LDR/STR Rd, [Rn], ± Rm, LSR #4
    - Rd ← [[Rn]] then shifts [Rm] right by 4, and increments Rn: Rn ← [[Rn] ± [Rm]]
  + Use LDR Rd, [SP], #4 to pop a single register value from stack
* Logical instructions
  + AND, ORR, EOR{S} – bitwise and/or/xor
    - Second operand can be shifted
  + LSL/LSR/ASL/ASR{S} Ri, Rj, #2
    - Ri ← [Rj] shifted by 2 bits
    - Arithmetic shift preserves the sign in the MSB
    - Arithmetic shift/logical shift left – add 0s at LSB, push everything left (multiply by 2)
    - Logical shift right – add 0s at MSB, push everything right (divide by 2)
    - Arithmetic shift right – add 0s or 1s at MSB, depending on the sign
  + ROR Ri, Rj, #2
    - Ri ← [Rj] rotated right by 2 bits
    - Rotate – copy bits from one end to another
    - C (carry) flag still retains the last bit being shifted out
    - E.g. C = 0, R1 = 0110…1011
      * ROR R1, R1, #2: C = 1, R1 = 110110…10
  + RRX Rj, Rj
    - Ri ← [Rj] rotated right using C flag by 1 bit
    - E.g. C = 0, R1 = 0110…1011
      * RLC R1, R1, #2: C = 1, R1 = 00110…101
* Arithmetic instructions
  + ADD/SUB{S} Rd, Rn, Rm
    - Rd ← [Rn] ± [Rm]
    - Rm can be shifted
  + MUL{S} R0, R1, R2
    - R0 ← [R1] \* [R2]
    - Only the least significant 32 bits are stored in R0
  + MLA{S} R0, R4, R5, R6
    - R0 ← [R4] \* [R5] + [R6]
  + SDIV/UDIV{S} R0, R1, R2
    - R0 ← [R1] / [R2]; signed or unsigned
* Move instructions
  + MOV{S} Rd, Rm
    - Rd ← [Rm]
    - Rm can be shifted – equivalent to LSL Rd, Rm, #shift
  + MOV Rd, #value
    - Can only assign 16-bit immediate values
  + MOVT Rd, #value
    - Assigns immediate value to top 16 bits of register
  + MVN Rd, #value/Rm
    - Moves the bit complement of #value/[Rm]
* Test/compare instructions
  + TST Rn, #value/Rm
    - Performs bitwise AND then sets condition code
  + TEQ Rn, #value/Rm
    - Performs bitwise XOR then sets condition code
  + CMP Rn, Rm
    - Performs [Rn] – R[m] then sets condition code
  + CMN Rn, Rm
    - Performs [Rn] + R[m] then sets condition code
  + Rm can all be shifted
  + These instructions always update condition code
* Branch instructions
  + B LOC – branches to LOC if condition is satisfied
  + BL LABEL – store PC in R14 (link register), branch to LABEL
  + BX LR – set PC to address stored in link register, i.e. reset return address
  + CBZ Rn, LABEL – compare & branch on 0; equivalent to:
    - CMP Rn, #0
    - BEQ LABEL
  + CBNZ Rn, LABEL – compare & branch on non-0; equivalent to:
    - CMP Rn, #0
    - BNE LABEL
* NOP – no operation; does nothing
* **Set condition code** 
  + Append S to arithmetic, logic, and move opcodes to update condition code
  + E.g. ADDS, ORRS
* **Check condition code**
  + Append condition code to any opcode to execute only when condition is satisfied
  + E.g. BEQ, LDRGT
* Instructions encoding
  + RISC – instructions are limited to single words
  + 32-bit immediate must be assigned by:
    - Load 16-bit, shift left 16 bits, load 16-bit
    - Or save immediate in memory, then load from memory into register
  + Register-operand format
    - MSB – Rsrc1 – Rsrc2 – Rdst – opcode – LSB
  + Immediate-operand format
    - MSB – Rsrc – Rdst – immediate – opcode – LSB
  + Call format
    - MSB – immediate – opcode – LSB
* Condition codes
* **The assembler**
* Assembler directives – instruct the assembler to perform certain tasks during translation from source program → object program
  + AREA – specifies start of CODE/DATA
  + ENTRY – specifies start of program execution
  + DCD – label and initialize data operands
  + EQU – declare symbolic names for constants
  + RN – ??
* Pseudo-instructions
  + LDR Ri, =X
  + Where X is an address value
* Two-pass assembly
  + Generate symbol table – label → address (position relative of start of program)
  + Go through table, substitute symbol names with addresses
* For multiple files/modules
  + Assemble each module as individual program
  + Linker adds appropriate offsets to reflect where each module is in memory
  + Position-independent code – relative addressing, instructions etc.
  + Position-dependent code – references to external labels