**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
* PC-relative addressing
  + LDR Rd, ITEM
    - i.e. Rd ← [[PC] + offset(ITEM)]
  + PC += 8 when LDR executes
* Pre-index addressing
  + LDR Rd, [Rn, #offset/Rm]
    - i.e. Rd ← [[Rn] + offset/[Rm]]
  + With writeback – add “!” at the end to shift Rn
    - Will additionally perform: Rn ← [Rn] + offset
* Post-index addressing
  + LDR Rd, [Rn], #offset/Rm
    - i.e. Rd ← [[Rn]]
    - Rn ← [Rn] + offset (automatically shift Rn)
* Arithmetic instructions
  + ADD R0, R2, R4
    - R0 ← [R2] + [R4]
  + SUB R0, R1, R2, LSL #4 (left shift operand)
    - R0 ← [R1] + [R2] \* 2^4
  + MLA R0, R4, R5, R6
    - R0 ← [R4] \* [R5] + [R6]
* Move instructions
  + MOV Rd, #value/Rm
    - Rd ← value/[Rm]
  + MVN Rd, #value/Rm
    - Moves the bit complement of #value/[Rm]
* Logical instructions
  + AND, ORR, EOR – bitwise and/or/xor
* Test/compare instructions
  + TST Rn, Rm/#value
    - Performs bitwise AND then sets condition code
  + TEQ Rn, Rm/#value
    - Performs bitwise XOR then sets condition code
  + CMP Rn, Rm
    - [Rn] – R[m] then sets condition code
* Branch instructions
  + B{condition} LOC
    - Branches to LOC if condition code flags satisfy {condition}
    - Look up condition suffixes
* Pushing/popping from stack
  + Op{address\_mode}{cond} Rn{!}, reg\_list
  + Op – lDM or STM
  + Address\_mode
    - FD – increment address before access
    - EA – decrement address after access
  + Cond – conditional execution
  + ! – load final address into Rn
  + reg\_list – one or more registers
* Branch-and-link
  + BL LABEL – store PC in R14 (link register), branch to LABEL
* **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