Introduction to Computing Systems
from bits & gates to C & beyond

Chapter 7

LC-3 Assembly Language

It's hard to write code in 1's & 0's!

- **Assembly language makes it easy to write Machine Language code**
  - each line of assembly language is translated into a single ML instruction
- **A program called the Assembler does the translation and provides useful tools:**
  - use of labels - symbolic names for address locations
  - automatic conversion of binary / hex / decimal
  - pseudo-ops

Assembly Language Instructions

- **Formats**
  - LABEL OPCODE OPERANDS ; COMMENTS
  - LABEL PSEUDO-OPS ; COMMENTS
- **Opcode**
  - Symbolic name for the 4-bit ML opcode
- **Label**
  - Symbolic name for a memory location. It is used to:
    - *indicate the target of a branch instruction, e.g. AGAIN in location 0B*
    - *indicate the location of a stored value or array, e.g. NUMBER and SIX*
- **Comments**
  - intended for humans only: explanation of code, visual display

Pseudo-Ops …

- … are directives to the assembler
  - *they are not translated into ML instructions*
- LC-3 Pseudo-Ops:
  - *.ORIG address* Tells assembler where to locate the program in memory (starting address).
  - *.FILL value* Store value in the this location in memory.
  - *.BLKW n* Set aside a block of n words in memory.
  - *.STRINGZ string* Store the string, one character per word, in memory. Add a word of x0000 after the string.
  - *.END* Marks the end of the source program (not to be confused with the instruction HALT!)
  - *.EXTERNAL* The label so indicated is allocated in another module.

A partial assembly sample

```
        .ORIG   x3000
        AND     R1, R1, #0
        ADD     R1, R1, #10
        LD      R2, Twenty
        LD      R3, Ess
        HALT
Twenty  .FILL     x0014
          .BLKW   2
Ess .FILL      “S” .STRINGZ
  “Hi”
        .BLKW 3
        .END
```

```
x3000:  AND  R1, R1, b1 0 0000
x3001:  ADD  R1, R1, b1 0 1010
x3002:  LD   R2, b0 0000 0010
x3003:  LD   R3, b0 0000 0100
x3004:  TRAP b0010 0101
x3005:  b0000 0000 0001 0100  ; x0014
x3006:
x3007:
x3008:  b0000 0000 0101 0011  ; x0053
x3009:  b0000 0000 0100 1000  ; x0048 = ‘H’
x300A:  b0000 0000 0110 1001  ; x0069 =  ‘i’
x300B:  x0000   ; null terminator
x300C:
x300D:
x300E:
```

The Assembly Process

- **Objective**
  - Translate the AL (Assembly Language) program into ML (Machine Language).
  - Each AL instruction yields one ML instruction word.
  - Interpret pseudo-ops correctly.
- **Problem**
  - An instruction may reference a label.
  - If the label hasn't been encountered yet, the assembler can't form the instruction word
- **Solution**
  - Two-pass assembly

Two-Pass Assembly - 1

- **First Pass - generating the symbol table**
  - Scan each line
  - Keep track of current address
    - *Increment by 1 for each instruction*
    - *Adjust as required for any pseudo-ops*

      (e.g. .FILL or .STRINGZ, etc.)
  - For each label
    - *Enter it into the symbol table*
    - *Allocate to it the current address*
  - Stop when .END is encountered

Symbol Table example

Symbol	Address
Again	x3053
Number	x3057
Six	x3058

```
        ;
        ; Program to multiply a number by  six
        ;
                .ORIG       x3050
x3050           LD      R1, SIX
x3051           LD      R2, NUMBER
x3052           AND     R3, R3, #0
        ;
        ; The inner loop
        ;
x3053   AGAIN   ADD     R3, R3, R2
x3054           ADD     R1, R1, #-1
x3055           BRp     AGAIN
        ;
x3056           HALT
        ;
x3057   NUMBER  .BLKW       1
x3058   SIX     .FILL   x0006
        ;
                .END
```

Two-Pass Assembly - 2

- **Second Pass - generating the ML program**
  - Scan each line again
  - Translate each AL instruction into ML
    - ***Look up symbols in the symbol table instruction***
    - ***Ensure that labels are no more than +256 / -255 lines from instruction***
    - ***Determine operand field for the instruction***
  - Fill memory locations as directed by pseudo-ops
  - Stop when .END is encountered

Assembled code

Symbol	Address
Again	x3053
Number	x3057
Six	x3058

```
x3050    0010 001 0 0000 0111    ; LD R1, SIX
x3051    0010 010 0 0000 0101    ; LD R2, NUMBER
x3052    0101 011 011 1 00000    ; AND R3, R3, #0
x3053    0001 011 011 0 00 010   ; ADD R3, R3, R2
x3054    0001 001 001 1 11111    ; ADD R1, R1, #-1
x3055    0000 001 1 1111 1101    ; BRp AGAIN
x3056    1111 0000 0010 0101     ; HALT
x3057                            ; .BLKW 1
x3058    0000 0000 0000 0110     ; .FILL x0006
```

Object File

- **Each source file is translated into an object file**
  - **a list of ML instructions including the symbol table.**
- **A complete program may include several source and/or object files:**
  - **Source files written in Assembly by the programmer**
  - **Library files provided by the system (OS or other)**
  - **Compiled HLL libraries**
- **The object files must be linked**
  - **One object file will be the “main”**
  - **All cross-referenced labels in symbol tables will be resolved**

The end result …

- **… is the executable image (.exe file)**
  - **this is a file ("image") of the finalized list of ML instructions, with all symbolic references resolved**
  - **it is *loaded* by copying the list into memory, starting at the address specified in the .ORIG directive**
  - **it is *run* by copying the starting address to the PC**