Lecture 33: LC3 Assembly language - Part I

- Low-level language - assembly language

- machine language in ISA for a computer
; file to count occurrences of a character
0011 0000 0001 0110 ; start of file memory address - x3016
;
; actual file to process, one character per memory location
0000 0000 0111 0100 ; x74 t
0000 0000 0110 0101 ; x65 e
0000 0000 0111 0011 ; x73 s
0000 0000 0111 0100 ; x74 t
0000 0000 0000 0100 ; x4 - end-of-file symbol

Assembly language

; Program to count occurrences of a character in a file
.ORIG x3000
;
; Initialization
;
LD R0, CHAR ; R0 gets character to count
LD R3, PTR ; R3 is pointer to characters in the file
; Initialization
;     LD R0, CNTR ; R0 gets character to count
     LD R2, PTR  ; R2 is pointer to characters in the file
     AND R2, R2, #0 ; R2 is counter, initially 0
;
; Test character for end of file
;     LDR R1, R3, #0 ; R1 gets next char to test
     ADD R4, R1, #$4 ; Test for EOF (ASCII #04)
     BNE OUTPUT ; if done, store the output
;
; Test character for match. If a match, increment count.
;     NOT R1, R1
     ADD R2, R1, #1 ; R1 <- -R1
     ADD R1, R1, R0 ; R1 <- R0 - R1
     BNEG NOTMATCH ; if no match, do not increment the counter
     ADD R2, R2, #1 ; if match, increment the counter
;
; Get next character from file.
;
; MOV R2, R3, #1 ; point to next character.
;
; Output the count.
;
; OUTPUT RT R2, RESULT ; store result
;
; Storage for inputs and outputs

PTR .FILL w$0106 ; start of file memory address
CNTR .FILL w$0074 ; 't' - char to count
RESULT .BLK 1 ; place to store result

; END

©2017 Juan Jose Jaramillo. All rights reserved.
Introduction

* Low-level language

* Machine-specific, different processors have different assembly languages

* Each assembly language instruction corresponds to single ISA instruction
LC-3 assembly language instruction

Format: (LABEL) OPCODE OPERANDS (COMMENT)
Format: (LABEL) OP CODE OPERANDS (COMMENT)

* Opcodes: reserved symbols that correspond to LC-3 instructions: ADD, AND, NOT, LD, ST,...

* Operands: separated by comma, written in specific order. There are three types:
1) Registers: R0, R1, ..., R7
   Example: NOT R1, R0

2) Numbers: in decimal (#15) or hexadecimal (xF)
   Example: ADD R3, R3, # -1

3) Labels: symbolic names for address locations
   Example: BRnp LOOP
* Labels: consist of 1 to 20 alphanumeric characters (uppercase, lowercase, decimal digit), starting with letter.
Starting with letter:

Used for two reasons:

1) Memory location is target of branch instruction

2) Memory location is source/destination in a store/load instruction

Example: LD R4, VALUE

\[ R4 \leftarrow M[VALUE] \]
* Comments: intended for humans, ignored by computer. Preceded by semicolon (;), purpose is to explain nonintuitive aspects of the code
Pseudo-ops (assembler directives)

Helpful to assembler during assembly language translation into machine language

* .ORIG: specifies where in memory to place LC-3 program

Example: .ORIG x3100

0011 0000 0000 0000
* .END: tells assembler where program ends.  
   Any characters after .END are ignored.

   Warning: .END does not HALT program

* .FILL: initialize memory location with value

   Example:
   . ORI $16, x3500
   AND R3, R3, #0 ; x3500
   HALT ; x3501
   . FILL x123A ; x3502
   . END
Block of Words

* **BLK N**: sets aside a number of memory locations, does not initialize them. Useful if actual values are not known until runtime.

* **STRING N**: initialize N+1 memory locations with N ASCII characters plus sentinel (x0000)
Example:

```
.OI 6 x3100
.BLKW #2 ; Two locations
.STRINGZ "Hello" ; ASCII representation
```
. BLKW #2 ; Two locations
STRINGZ "Hello" ; ASCII representation
END

For this code, assembler will do the following
starting in address location x3100:

```
x 3100
x 3101
  x 3102  H  x 0048
  x 3103  e
  x 3104  l
  x 3105  l
  x 3106  o
  x 3107  NUL
```

TRAP x 25 ; Halt
PC ← M[x 25] ; Service call to the operating system
<table>
<thead>
<tr>
<th>TRAP code</th>
<th>pseudo-instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>HALT</td>
<td>(TRAP x25)</td>
</tr>
<tr>
<td>IN</td>
<td>(TRAP x23)</td>
</tr>
<tr>
<td>6ETC</td>
<td>(TRAP x20)</td>
</tr>
<tr>
<td>OUT</td>
<td>(TRAP x21)</td>
</tr>
<tr>
<td>PUTS</td>
<td>(TRAP x22)</td>
</tr>
<tr>
<td>PUTSP</td>
<td>(TRAP x24)</td>
</tr>
</tbody>
</table>

- HALT: prompt on display
- GETC: reads a single character from keyboard
- OUT: sends a character from keyboard
- PUTS: sends a single character to display
- PUTSP: sends a single character to display

Stored in RO

Read one character from keyboard

From RO
Assembly process

Converts assembly code into machine code

Performed in two steps:
Performed in two steps:

* Pass 1: create symbol table, which has correspondence of symbols (labels) to memory addresses

* Pass 2: machine code is generated line-by-line, if a label is found, offset is automatically calculated

.asm -> .bin