

# LC3 Assembly Language

Based on slides © McGraw-Hill  
Additional material © 2004/2005 Lewis/Martin  
Modified by Diana Palsetia (2007-2008)

## Assembly: Human-Readable Machine Language

Computers like ones and zeros...

0001110010000110

Humans like readable form ...

**ADD**    **R6, R2, R6**    ; increment index reg.  
*Opcode   Dest Src1 Src2   Comment*

### Assembler

- A program that turns human readable form into machine instructions
- ISA specific
- One assembly instruction translates to one machine instruction

CIT 593

2

## LC-3 Assembly Language Syntax

Each line of a program is one of the following:

- An instruction
- An assembler directive (or pseudo-op)
- A comment

Whitespace (between symbols) and Comments (beginning with “;”) are ignored

An instruction has the following format:

**LABEL: OPCODE OPERANDS ; COMMENTS**

*optional*                      *mandatory*

CIT 593

3

## Opcodes and Operands

### Opcodes

- Reserved symbols that correspond to LC-3 instructions
- Listed in Appendix A
  - E.g. ADD, AND, LD, LDR, ...
  - For BR use lower case
    - n: negative, p: positive and z: zero

### Operands

- Registers -- specified by R0, R1, ..., R7
- Literal/Immediate -- indicated by # (decimal) or x (hex) or b (binary)
  - E.g. “#10” is “xA” is “b1010”
- Label: -- symbolic name of memory location

Opcode, registers and literals are separated by commas

- Number, order, and type correspond to instruction format
- E.g. ADD R1, R1, #3

CIT 593

4

## Labels

### Label

- Followed by colon (:) when declared
  - The textbook does not say this. But for our assembler we use the colon
- Placed at the beginning of the line
- Assigns a symbolic name to the memory address corresponding to line
  - `LOOP: ADD R1,R1,#-1`  
`BRp LOOP`
  - Instead Of  
`ADD R1,R1, #-1`  
`BRp x1FF`
  - Instructions with PCOffset use labels i.e. literal offsets this will not compile in our version of LC3
- Consists of:
  - 1-20 alphanumeric characters
    - Capital or lowercase alphabets or a decimal digit
    - Always starts with a letter of alphabet e.g. `Test1` or `test1`

CIT 593

5

## Comments

### Comment

- Anything after a semicolon (;) is a comment
- Ignored by assembler
- Tips for useful comments:
  - State what each register is/will be holding
  - Use comments to separate pieces of program
  - Explain your approach

CIT 593

6

## Assembler Directives

### Pseudo-operations

- Operations are not part of the ISA
  - More for convenience
- Used by assembler
- Look like instruction, but “opcode” starts with dot

Opcode	Operand	Meaning
<code>.ORIG</code>	address	starting address of program
<code>.END</code>		end of program
<code>.FILL</code>	value	allocate one word, initialize with value
<code>.BLKW</code>	number	allocate multiple words of storage, value unspecified
<code>.STRINGZ</code>	n-character string	allocate n+1 locations, initialize w/characters and null terminator

CIT 593

7

## Assembler Directives (cont..)

### .ORIG

- `.ORIG x3050` – tells the assembler where in memory to place the 1<sup>st</sup> instruction of the LC3 program

### .FILL

- `.FILL x0006` – initializes a memory location with value 6

### .BLKW

- `.BLKW 2` – set aside 2 sequential memory locations
- Useful when the actual value of the operand is not known
- The locations will be initialized with zero

### .STRINGZ

`.ORIG x3010`  
`.STRINGZ "Hello"`

x3010: x0048  
 x3011: x0065  
 x3012: x006C  
 x3013: x006C  
 x3014: x006F  
 x3015: x0000

Null terminated string

CIT 593

8

## Trap Codes

LC-3 assembler provides “pseudo-instructions” for each trap code, so you don’t have to remember them

Code	Equivalent	Description
<b>HALT</b>	TRAP x25	Halt execution and return control to OS
<b>IN</b>	TRAP x23	Print prompt on console, read (and echo) one character from keybd. Character stored in R0.
<b>OUT</b>	TRAP x21	Write one character (in R0) to console.
<b>GETC</b>	TRAP x20	Read one character from keyboard. Character stored in R0.
<b>PUTS</b>	TRAP x22	Write null-terminated string to console. Address of string is in R0.

CIT 593

9

## An Assembly Language Program

```

;
; Program to multiply a number by the constant 6
;
        .ORIG    x3000
        LD       R1, SIX
        AND      R2, R2, #0      ;Clear R2
        ADD      R2, R2, #4      R2 = number = 4
        AND      R3, R3, #0      ;Clear R3. It will
                                   ;contain the product.

; The inner loop
AGAIN:   ADD      R3, R3, R2
        ADD      R1, R1, #-1
        BRp     AGAIN           ;loop until R1 > 0
        HALT                    ;control back to OS

;DATA
SIX:     .FILL    x0006          ;initialize location with value 6
        .END                      ;end of program

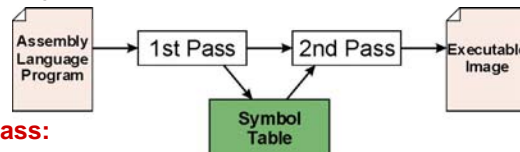
```

CIT 593

10

## Assembly Process

Program that converts assembly language file (.asm) into an executable file (.obj) for the LC-3 machine (simulator)



### First Pass:

- Scan program file
- Find all labels and calculate the corresponding addresses; this is called the symbol table

### Second Pass:

- Convert instructions to machine language, using information from symbol table

CIT 593

11

## First Pass: Constructing the Symbol Table

1. Begin with the **.ORIG** statement, which tells us the address of the first instruction
  - Initialize *location counter* (LC), which keeps track of the current instruction
2. For each non-blank line in the program:
  - a) If line contains a label, put label/LC pair into symbol table
  - b) Increment LC
    - NOTE: If statement is **.BLKW** or **.STRINGZ**, increment LC by the number of words allocated
    - A line with only a comment is considered “blank”
3. Stop when **.END** statement is reached

CIT 593

12

## Assembly Process Example: First Pass

```

.ORG x3000
x3000 AND R2,R2,#0
x3001 LD R3,PTR
x3002 TRAP x23
x3003 LDR R1,R3,#0
x3004 ADD R4,R1,#-4
x3005 TEST: BRz OUTPUT
x3006 NOT R1,R1
x3007 ADD R1,R1,#1
x3008 ADD R1,R1,R0
x3009 BRnp GETCHAR
x300A ADD R2,R2,#1
x300B GETCHAR:ADD R3,R3,#1
x300C LDR R1,R3,#0
x300D BRnzp TEST
x300E OUTPUT: LD R0,ASCII
x300F ADD R0,R0,R2
x3010 TRAP x21
x3011 TRAP x25
x3012 ASCII: .FILL x0030
x3013 PTR: .FILL x4000
.END

```

Symbol	Address
TEST	x3005
GETCHAR	x300B
OUTPUT	x300E
ASCII	x3012
PTR	x3013

CIT 593

13

## Second Pass: Generating Machine Code

For each executable assembly language statement

- Generate the corresponding machine language instruction
- If operand is a label, look up the address from the symbol table

Potential errors:

- Improper number or type of arguments
  - E.g. NOT R1,#7
  - ADD R1,R2
  - ADD R3,R3,NUMBER
- Immediate argument too large
  - E.g. ADD R1,R2,#1023
- Address (associated with label) more than 256 from instruction
  - Can't use PC-relative addressing mode

CIT 593

14

## Assembly Process Example: Second Pass

```

.ORG x3000
x3000 AND R2,R2,#0
x3001 LD R3,PTR
x3002 TRAP x23
x3003 LDR R1,R3,#0
x3004 ADD R4,R1,#-4
x3005 TEST: BRz OUTPUT
x3006 NOT R1,R1
x3007 ADD R1,R1,#1
x3008 ADD R1,R1,R0
x3009 BRnp GETCHAR
x300A ADD R2,R2,#1
x300B GETCHAR:ADD R3,R3,#1
x300C LDR R1,R3,#0
x300D BRnzp TEST
x300E OUTPUT: LD R0,ASCII
x300F ADD R0,R0,R2
x3010 TRAP x21
x3011 TRAP x25
x3012 ASCII: .FILL x0030
x3013 PTR: .FILL x4000
.END

```

```

0101 010 010 1 0000
0010 011 000010001
1111 0000 00100011
.
.

```

Symbol	Address
TEST	x3005
GETCHAR	x300B
OUTPUT	x300E
ASCII	x3012
PTR	x3013

CIT 593

15

## Style Guidelines

Improve the readability of your programs

- Formatting: start labels, opcode, operands in same column
- Use comments to explain what each register does
- Give explanatory comment for most instructions
- Use meaningful symbolic names
- Provide comments between program sections
- Each line must fit on the page -- no wraparound or truncations
  - Long statements split in aesthetically pleasing manner

CIT 593

16