

Chapter 7 Assembly Language

Based on slides © McGraw-Hill
Additional material © 2004/2005 Lewis/Martin
Modified by Diana Palsetia (2006)

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

7-2

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 case are ignored

Comments (beginning with “;”) are also ignored

An instruction has the following format:

```
LABEL OPCODE OPERANDS ; COMMENTS
```

optional *mandatory*

CIT 593

7-3

Opcodes and Operands

Opcodes

- Reserved symbols that correspond to LC-3 instructions
- Listed in Appendix A
 - ex: ADD, AND, LD, LDR, ...

Operands

- Registers -- specified by R0, R1, ..., R7
- Numbers -- indicated by # (decimal) or x (hex) or b (binary)
 - Examples: “#10” is “xA” is “b1010”
- Label -- symbolic name of memory location
- Separated by comma
- Number, order, and type correspond to instruction format

```
➢ ex:  
ADD R1, R1, R3  
ADD R1, R1, #3  
LD R6, NUMBER  
BRz LOOP
```

CIT 593

7-4

Labels

Label

- 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
 

```

 x3006 ADD R1,R1, #-1
 BRp x3006

```
- 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

7-5

## Comments

### Comment

- Anything after a semicolon is a comment
- Ignored by assembler
- Used by humans to document/understand programs
  - Especially if you wrote it a few days ago and want to begin where you left of
- Tips for useful comments:
  - State what each register is/will be holding
  - Use comments to separate pieces of program
  - Explain your approach

CIT 593

7-6

## Assembler Directives

### Pseudo-operations

- Do not refer to operations executed by program
- 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

7-7

## Assembler Directives (cont..)

### `.ORIG`

- `.ORIG x3050` – tells the assembler where in memory to place 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

### `.STRINGZ`

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

X3010: x0048  
 X3011: x0065  
 X3012: x006C  
 X3013: x006C  
 X3014: x006F  
 X3015: X0000

Null terminate the string

CIT 593

7-8

## An Assembly Language Program

```

;
; Program to multiply a number by the constant 6
;
 .ORIG x3050
 LD R1, SIX
 LD R2, NUMBER
 AND R3, R3, #0 ; Clear R3. It will
 ; contain the product.
; The inner loop
;
AGAIN ADD R3, R3, R2
 ADD R1, R1, #-1 ; R1 keeps track of
 BRP AGAIN ; the iteration.
 HALT
;
NUMBER .BLKW 1 ; set aside a location
SIX .FILL x0006 ; initialize location with value 6
;
 .END

```

CIT 593

7-9

## Trap Codes

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

| Code | Equivalent | Description                                                                                     |
|------|------------|-------------------------------------------------------------------------------------------------|
| HALT | TRAP x25   | Halt execution and print message to console.                                                    |
| IN   | TRAP x23   | Print prompt on console, read (and echo) one character from keybd. Character stored in R0[7:0]. |
| OUT  | TRAP x21   | Write one character (in R0[7:0]) to console.                                                    |
| GETC | TRAP x20   | Read one character from keyboard. Character stored in R0[7:0].                                  |
| PUTS | TRAP x22   | Write null-terminated string to console. Address of string is in R0.                            |

CIT 593

7-10

## 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

### Use structured programming constructs

- From chapter 6

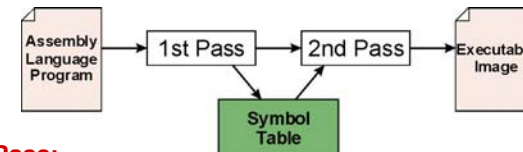
High-level programming style is similar

CIT 593

7-11

## Assembly Process

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

7-12

## 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:

- If line contains a label, put label/LC pair into symbol table
- 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

7-13

## Assembly Process Example: First Pass

```

.ORIG 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

7-14

## 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
  - ex: NOT R1,#7
  - ADD R1,R2
  - ADD R3,R3,NUMBER
- Immediate argument too large
  - ex: ADD R1,R2,#1023
- Address (associated with label) more than 256 from instruction
  - Can't use PC-relative addressing mode

CIT 593

7-15

## Assembly Process Example: Second Pass

```

.ORIG 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

7-16

## LC-3 Assembler

Generates two different output files

### Symbol file (.sym)

- Includes names of labels (also known as symbols)
- Used by simulator to make code easier to read
- A text file of symbol mappings

### Object file (.obj)

- Binary representation of the program

To create the above files in LC3

- Type command: `as filename.asm`

CIT 593

7-17

## Object File Format

LC-3 object file contains

- Starting address (location where program must be loaded), followed by...
- Machine instructions
- (Real-world object file formats can be more complicated)

LC-3 Example

- Beginning of “count character” object file looks like this:

```
0011000000000000 ← .ORIG x3000
0101010010100000 ← AND R2, R2, #0
0010011000010001 ← LD R3, PTR
1111000000100011 ← TRAP x23
.
.
.
```

CIT 593

7-18

## Using Multiple Object Files

An object file is not necessarily a complete program

- System-provided library routines
- Code blocks written by multiple developers

For LC-3 simulator

- Load multiple object files into memory, then start executing at a desired address
  - Command: `load filename.obj`
- System routines, such as keyboard input, are loaded with OS
  - Loaded into “system memory,” below `x3000`
  - User code should be loaded between `x3000` and `xFDFF`
- Each object file includes a starting address
- Be careful not to load overlapping object files

CIT 593

7-19

## Linking and Loading

**Loading** is the process of copying an executable image into memory

- More sophisticated loaders are able to relocate images to fit into available memory
- Must readjust branch targets, load/store addresses

**Linking** is the process of resolving symbols between independent object files

- Suppose we define a symbol in one module, and want to use it in another
- Some notation, such as `.EXTERNAL`, is used to tell assembler that a symbol is defined in another module
- Linker will search symbol tables of other modules to resolve symbols and complete code generation before loading

CIT 593

7-20

## Next Time

### Reading

- Chapter 8

### Lecture

- Chapter 8

### Homework

- Homework 2 assigned.