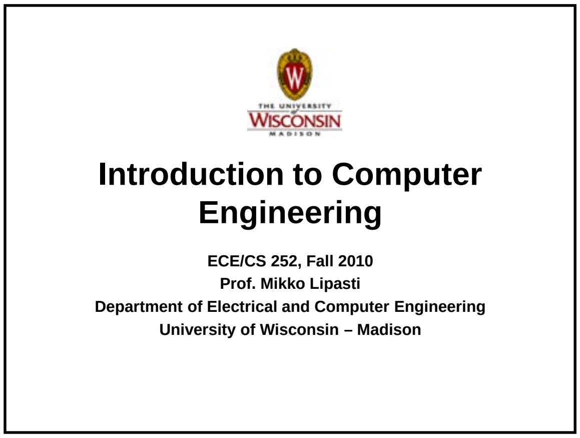
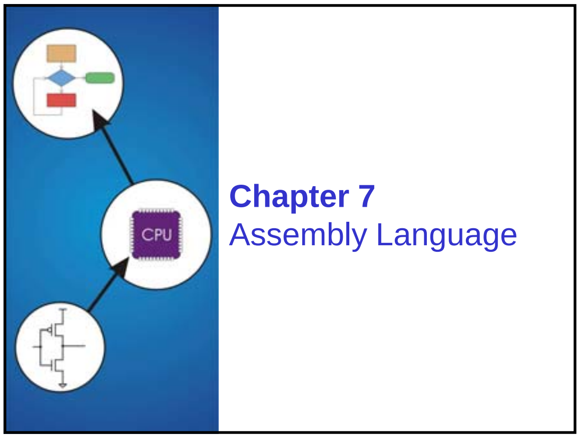
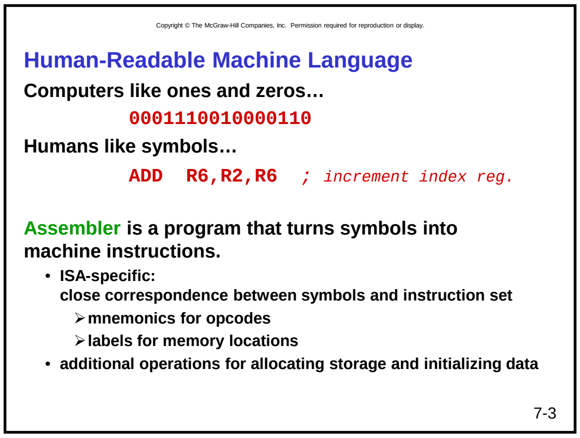


Introduction to Computer Engineering

ECE/CS 252, Fall 2010
Prof. Mikko Lipasti
Department of Electrical and Computer Engineering
University of Wisconsin – Madison



Chapter 7
Assembly Language



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Human-Readable Machine Language

Computers like ones and zeros…

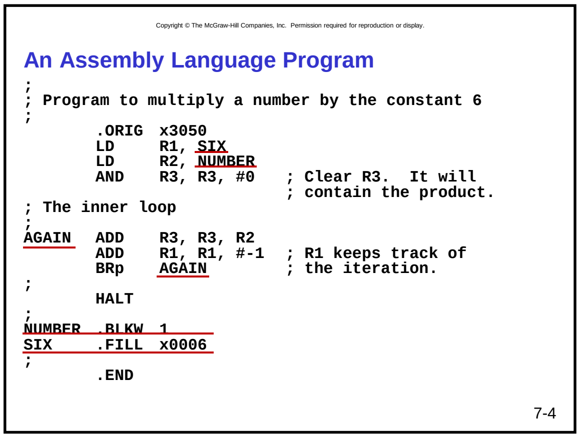
```
0001110010000110
```

Humans like symbols…

```
ADD   R6,R2,R6   ; increment index reg.
```

Assembler is a program that turns symbols into machine instructions.

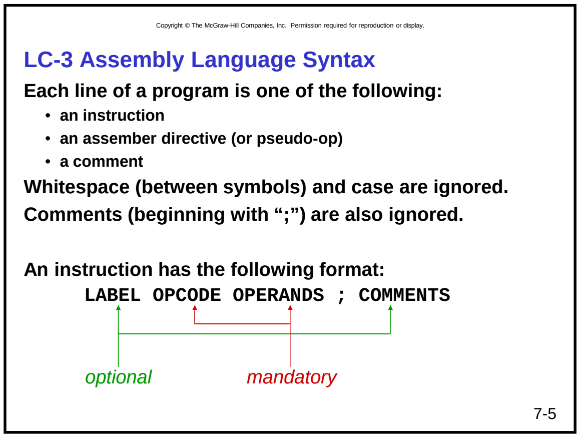
- ISA-specific:
  close correspondence between symbols and instruction set
  - mnemonics for opcodes
  - labels for memory locations
- additional operations for allocating storage and initializing data



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

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
SIX     .FILL  x0006
;
        .END
```



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

LC-3 Assembly Language Syntax

Each line of a program is one of the following:

- an instruction
- an assember directive (or pseudo-op)
- a comment

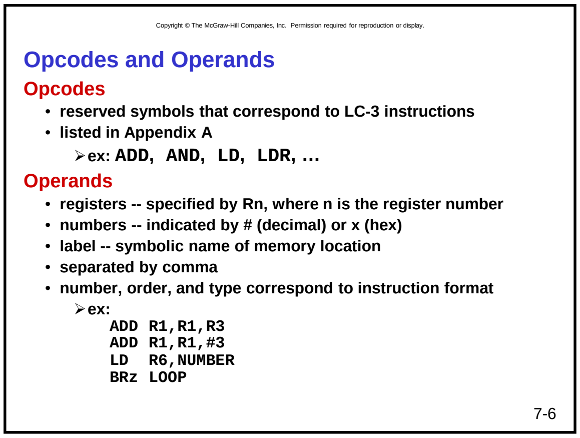
Whitespace (between symbols) and case are ignored.

Comments (beginning with ";") are also ignored.

An instruction has the following format:

```
LABEL OPCODE OPERANDS ; COMMENTS
```

optional *mandatory*



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

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 Rn, where n is the register number
- numbers -- indicated by # (decimal) or x (hex)
- 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
```

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Labels and Comments

Label

- placed at the beginning of the line
- assigns a symbolic name to the address corresponding to line
  - ex:

```
LOOP  ADD  R1,R1,#-1
      BRp  LOOP
```

Comment

- anything after a semicolon is a comment
- ignored by assembler
- used by humans to document/understand programs
- tips for useful comments:
  - avoid restating the obvious, as “decrement R1”
  - provide additional insight, as in “accumulate product in R6”
  - use comments to separate pieces of program

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

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*
`.ORIG`	address	starting address of program
`.END`		end of program
`.BLKW`	n	allocate n words of storage
`.FILL`	n	allocate one word, initialize with value n
`.STRINGZ`	n-character string	allocate n+1 locations, initialize w/characters and null terminator

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

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.

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Style Guidelines

Use the following style guidelines to improve the readability and understandability of your programs:

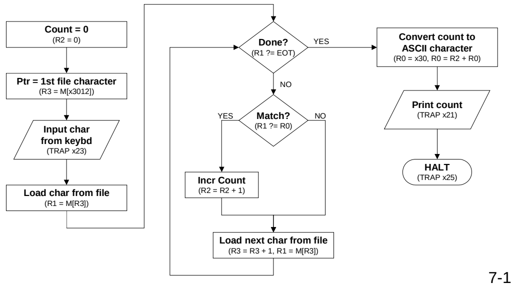
1. **Provide a program header, with author’s name, date, etc., and purpose of program.**
2. **Start labels, opcode, operands, and comments in same column for each line.** (Unless entire line is a comment.)
3. **Use comments to explain what each register does.**
4. **Give explanatory comment for most instructions.**
5. **Use meaningful symbolic names.**
   - **Mixed upper and lower case for readability.**
   - ASCIItoBinary, InputRoutine, SaveR1
6. **Provide comments between program sections.**
7. **Each line must fit on the page -- no wraparound or truncations.**
   - **Long statements split in aesthetically pleasing manner.**

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Sample Program

Count the occurrences of a character in a file.

Remember this?



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Char Count in Assembly Language (1 of 3)

```
;
; Program to count occurrences of a character in a file.
; Character to be input from the keyboard.
; Result to be displayed on the monitor.
; Program only works if no more than 9 occurrences are found.
;
;
; Initialization
;
        .ORIG   x3000
        AND     R2, R2, #0      ; R2 is counter, initially 0
        LD      R3, PTR         ; R3 is pointer to characters
        GETC                    ; R0 gets character input
        LDR     R1, R3, #0      ; R1 gets first character
;
; Test character for end of file
;
TEST    ADD     R4, R1, #-4     ; Test for EOT (ASCII x04)
        BRz     OUTPUT          ; If done, prepare the output
```

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Char Count in Assembly Language (2 of 3)

```
;
; Test character for match.  If a match, increment count.
;
         NOT     R1, R1
         ADD     R1, R1, R0  ; If match, R1 = xFFFF
         NOT     R1, R1      ; If match, R1 = x0000
         BRnp    GETCHAR     ; If no match, do not increment
         ADD     R2, R2, #1
;
; Get next character from file.
;
GETCHAR  ADD     R3, R3, #1  ; Point to next character.
         LDR     R1, R3, #0  ; R1 gets next char to test
         BRnzp   TEST
;
; Output the count.
;
OUTPUT   LD      R0, ASCII   ; Load the ASCII template
         ADD     R0, R0, R2  ; Covert binary count to ASCII
         OUT                 ; ASCII code in R0 is displayed.
         HALT                ; Halt machine
```

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

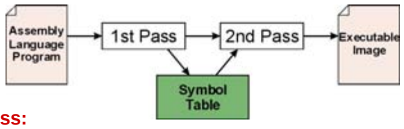
Char Count in Assembly Language (3 of 3)

```
;
; Storage for pointer and ASCII template
;
ASCII    .FILL   x0030
PTR      .FILL   x4000
         .END
```

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Assembly Process

Convert 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**

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

First Pass: Constructing the Symbol Table

1. **Find 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-empty line in the program:**
   - **a) If line contains a label, add label and LC to symbol table.**
   - **b) Increment LC.**
     - **– NOTE: If statement is `.BLKW` or `.STRINGZ`, increment LC by the number of words allocated.**
3. **Stop when `.END` statement is reached.**

NOTE: A line that contains only a comment is considered an empty line.

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Practice

Construct the symbol table for the program in Figure 7.1 (Slides 7-11 through 7-13).

LC: x3013

Symbol	Address
TEST	x3004
GETCHAR	x300B
OUTPUT	x300E
ASCII	x3012
PTR	x3013

```
➡                  .ORIG   x3000
                   AND     R2, R2, #0
                   LD      R3, PTR
                   GETC
                   LDR     R1, R3, #0
➡  TEST            ADD     R4, R1, #-4
                   BRz     OUTPUT
                   NOT     R1, R1
                   ADD     R1, R1, R0
                   NOT     R1, R1
                   BRnp    GETCHAR
                   ADD     R2, R2, #1
➡  GETCHAR         ADD     R3, R3, #1
                   LDR     R1, R3, #0
                   BRnzp   TEST
➡  OUTPUT          LD      R0, ASCII
                   ADD     R0, R0, R2
                   OUT
                   HALT
➡  ASCII           .FILL   x0030
➡  PTR             .FILL   x4000
                   .END
```

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Second Pass: Generating Machine Language

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 problems:

- **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**

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Practice

Using the symbol table constructed earlier, translate these statements into LC-3 machine language.

@x3001+1, PTR is x3013, so offset is x11

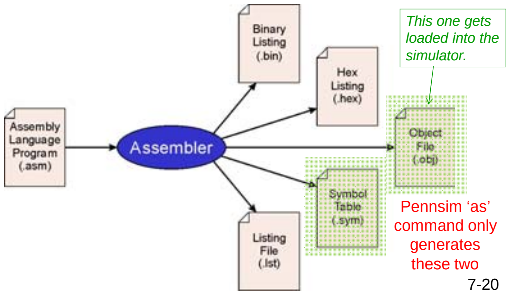
Statement	Machine Language
`LD   R3,PTR`	0010 011 000010001
`ADD  R4,R1,#-4`	0001 100 001 1 11100
`LDR  R1,R3,#0`	0110 001 011 000000
`BRnp GETCHAR`	0000 101 000000001

@x3009+1, GETCHAR is x300B, so offset is x1

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

LC-3 Assembler

Using “lc3as” (Unix) or LC3Edit (Windows), generates several different output files.



This one gets loaded into the simulator.

Pennsim ‘as’ command only generates these two

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Object File Format

LC-3 object file contains

- **Starting address (location where program must be loaded), followed by…**
- **Machine instructions**

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
        .
        .
        .
```

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Multiple Object Files

An object file is not necessarily a complete program.

- **system-provided library routines**
- **code blocks written by multiple developers**

For Pennsim LC-3 simulator, can load multiple object files into memory, then start executing at a desired address.

- **system routines, such as keyboard input, are in lc3os.obj (assembled from lc3os.asm), provided for you**
  - ➢ **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**

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

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 (.ORIG is not fixed)**
- **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**
- **this is not supported in Pennsim**

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Summary

Assembly: “Human readable” machine language

Opcodes, operands, labels, comments, directives

Assembly process

- **Pass 1: symbol table**
- **Pass 2: machine code generation**

Object files

Linking and loading