Lecture 21 10/22/20

Wednesday, October 21, 2020 12:16 PM

TA Help Session during PS Week of 10/19.

Program #2 Due 10/27

Today:

Quiz #3 Feedback

Program #2 Q&A

Chapter 5 - Instruction Flow Charts PattPatelCh05_Instr_2020.pptx

Chapter 7 Assemblers

Human-Readable Machine Language

Computers like ones and zeros…

```
0001110010000110
```


Humans like symbols…

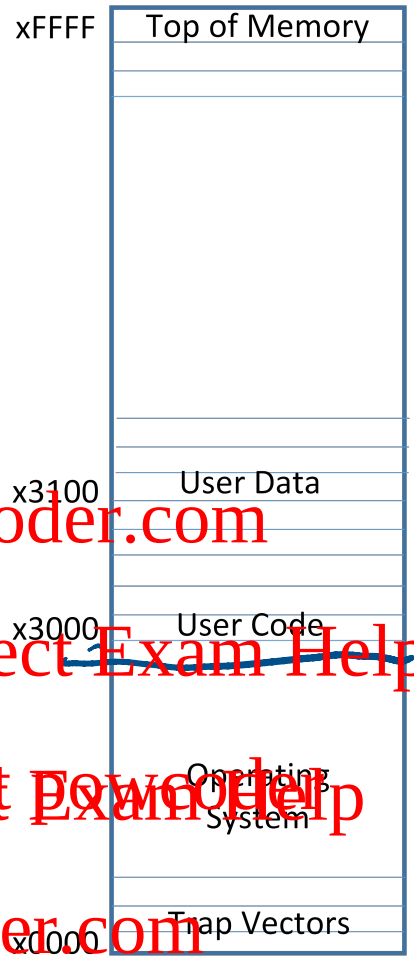
```
ADD   R6,R2,R6   ; increment index reg.
```

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**

Common Memory Usage

16-bit Memory Addressing



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
```

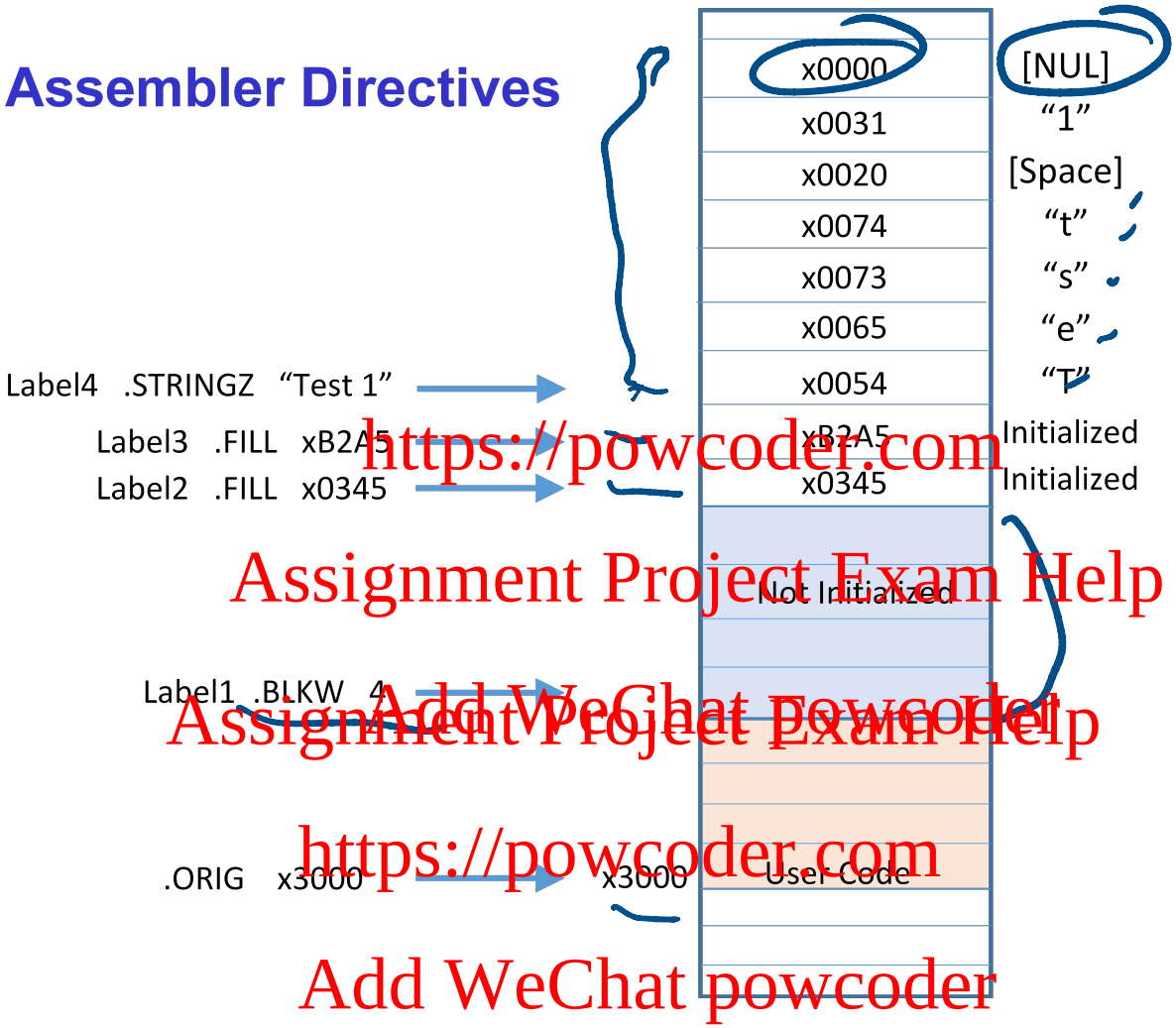

Assembler Directives

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

Assembler Directives



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].

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.

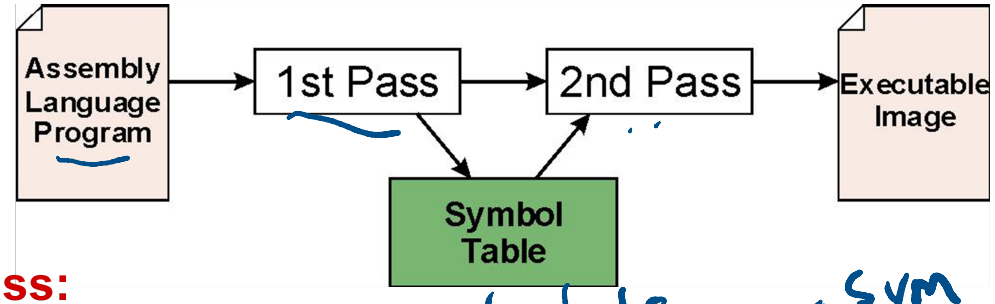
Style Guidelines

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 (max 80 chars) -- no wraparound or truncations.**
   - **Long statements split in aesthetically pleasing manner.**

Assembly Process

Convert assembly language file (.asm) into an executable file (.obj) for the LC-3 simulator.



First Pass:

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

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.

Second Pass: Generating Machine Language

For each executable assembly language statement, generate the corresponding machine language instruction.

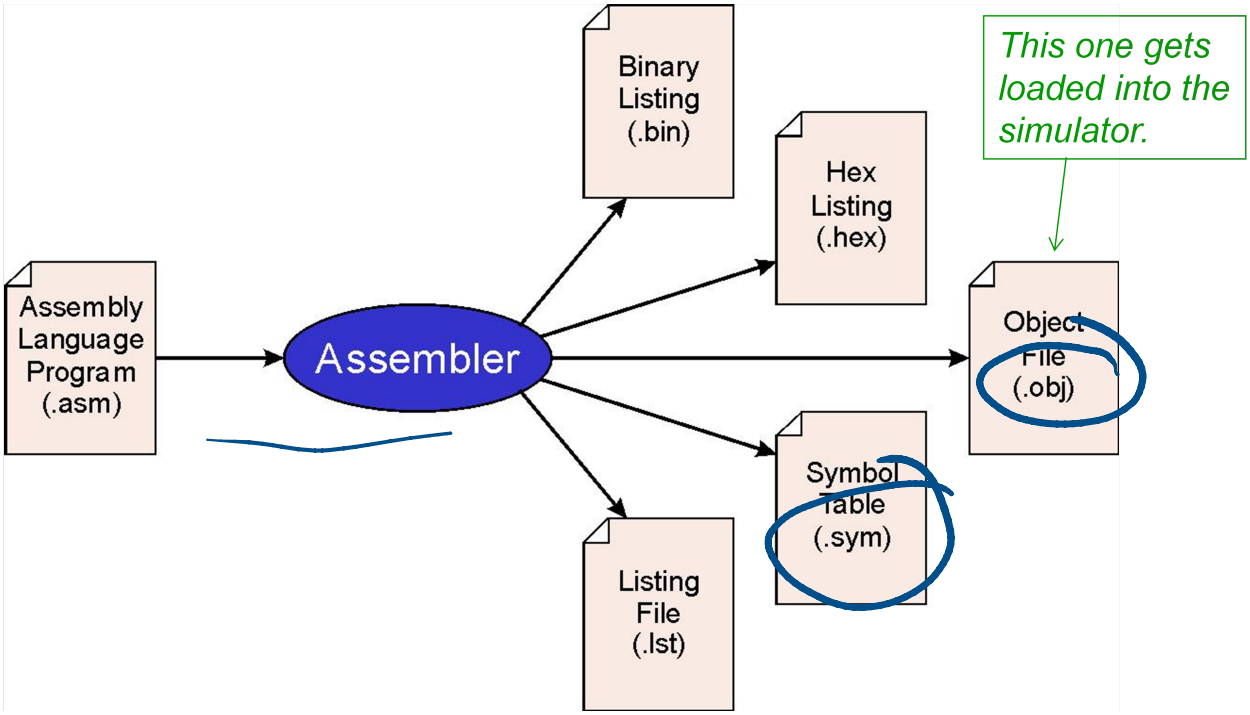
- **If operand is a label, look up the address from the symbol table.**

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**

LC-3 Assembler

Using "assemble" (Unix) or LC3Edit (Windows), generates several different output files.



Object File Format

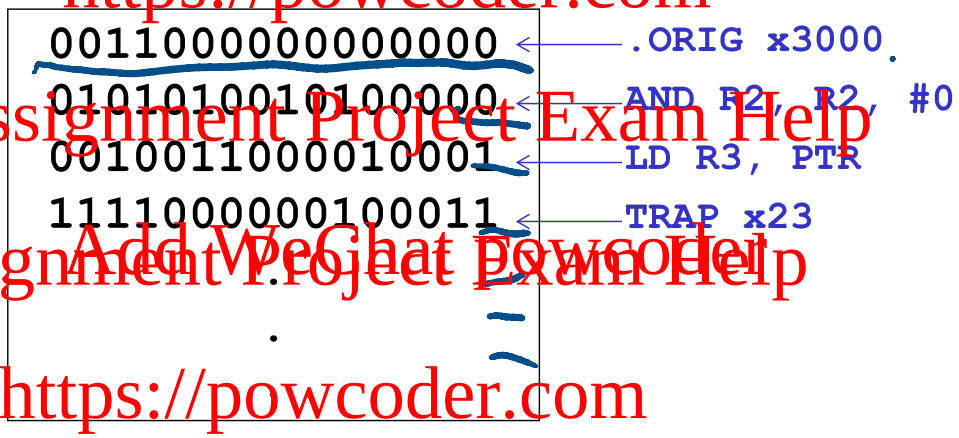
.obj

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:



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,
can load multiple object files into memory,
then start executing at a desired address.

- system routines, such as keyboard input, are loaded automatically
  - 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

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 (part of PennSim) will search symbol tables of other modules to resolve symbols and complete code generation before loading.

Linking and Loading

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

Add a new assembler directive:

- **.EXTERNAL LabelName**
- **Assembler leaves a hole in symbol table for LabelName**
- **Fill in bogus value when LabelName used by instructions.**
- **Make a list of all instructions that use LabelName**
- **Run a separate linker process on all object files.**
- **Resolves (fills in the values for) external labels.**
- **Rewrites machine language instructions that use such labels.**

Linking and Loading

Other programming languages call these external variables "globals"

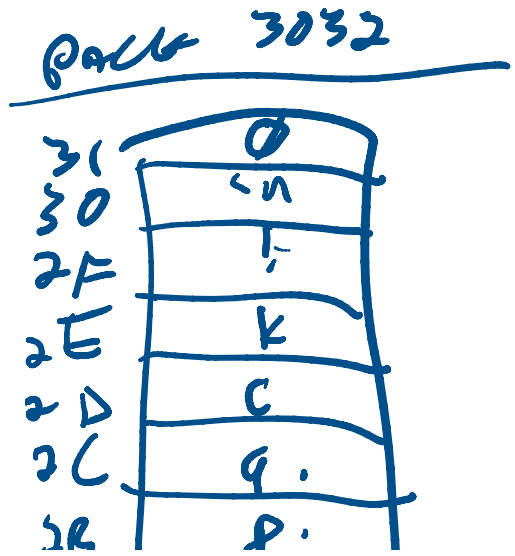
```
; Code Section Alpha      alpha.asm
;
            .ORIG x3000
GO_A        AND R0, R0, #0      ; Clear R0
            ADD R0, R0, #12     ; Put #12 into R0
            ADD R3, R0, #0      ; Move to R3
            ADD R0, R0, R0      ; Double R0
            PUTC                ; Display R0
DATA_A      .FILL x0035         ; Data word
            .EXTERNAL DATA_A
            .END
```

```
; Code Section Beta       beta.asm
            .ORIG x3100
            .EXTERNAL DATA_A
GO_B        AND R0, R0, #0      ; Clear R0
            LD R0, DATA_A       ; Load DATA_A to R0
            PUTC                ; Display R0
            .END
```


Sample Creation of Symbol Table

For the following assembly program generate the Symbol Table.
There may be extra rows in the table that are not required.

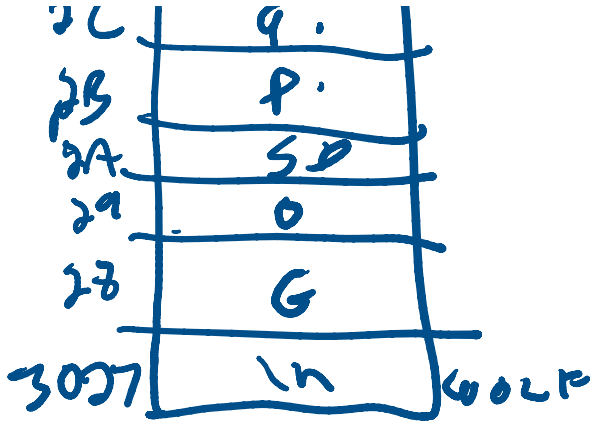
```
      .ORIG x3020
START  LEA R0, WOLF   ; start here
       PUTS
       LDI R3, STATE
SPEAK  PUTS
XX1    .FILL x20
      .FILL x30
       .FILL x40
WOLF   .STRINGZ "\nGo Pack!\n"
PACK   ST R1, D1
       ST R2, D2
       ST R3, D3
       ST R7, D7
       ADD R1, R0, #0
WHITE  LD R3, D3
       LDR R2, R1, #0
```



```
      ADD R1, R0, #0
WHITE LD R3, D3
      LDR R2, R1, #0
    HALT

D1  .BLKW 4
STATE .FILL x3157

    .END
```



Label	Address
START	3020
SPEAK	3023
XX1	3024
WOLF	3027
PACK	3032
WHITE	3037
D1	303A
STATE	303E

chp7_inclass_fall_2020_sol.docx

chp7_code_count_fall_2020_sol.docx