中国科学技术大学
University of Science and Technology of China

计算系统概论A
Introduction to Computing Systems
(CS1002A.03)

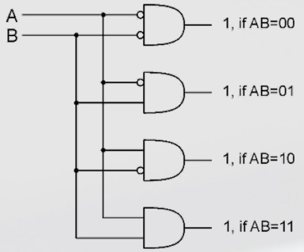
Chapter 7
Assembly Language Program

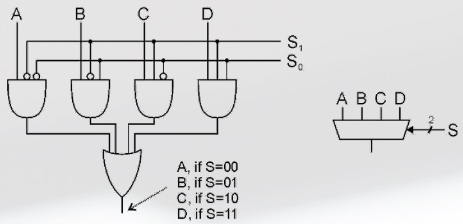
陈俊仕
cjuns@ustc.edu.cn
2023 Fall

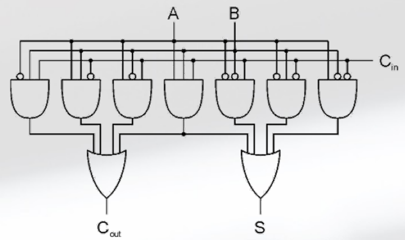
计算机科学与技术学院
School of Computer Science and Technology

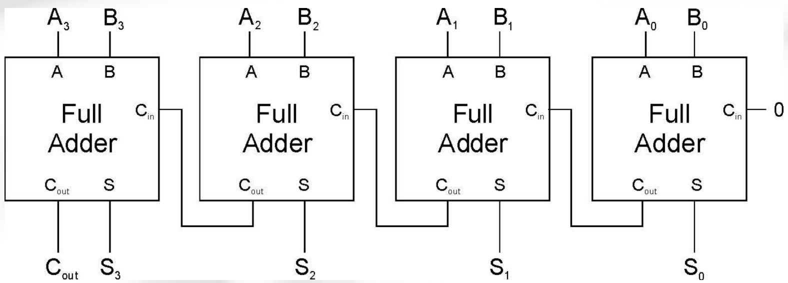
Outline

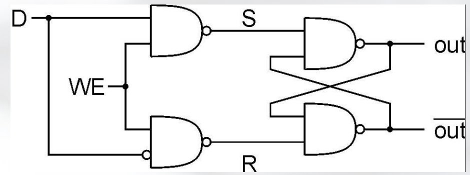
Review: The Transistor & Basic Logical Structure

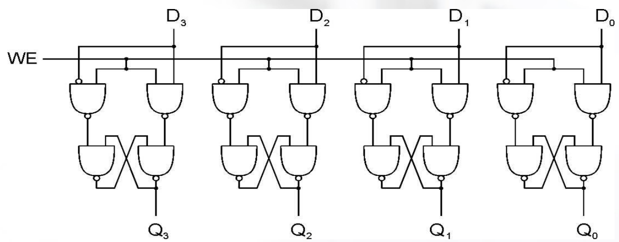


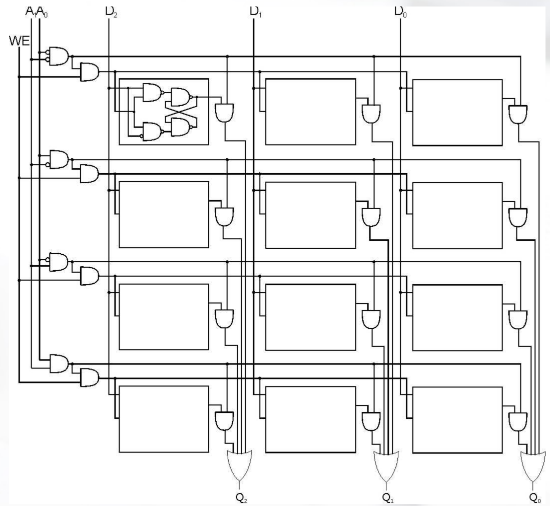




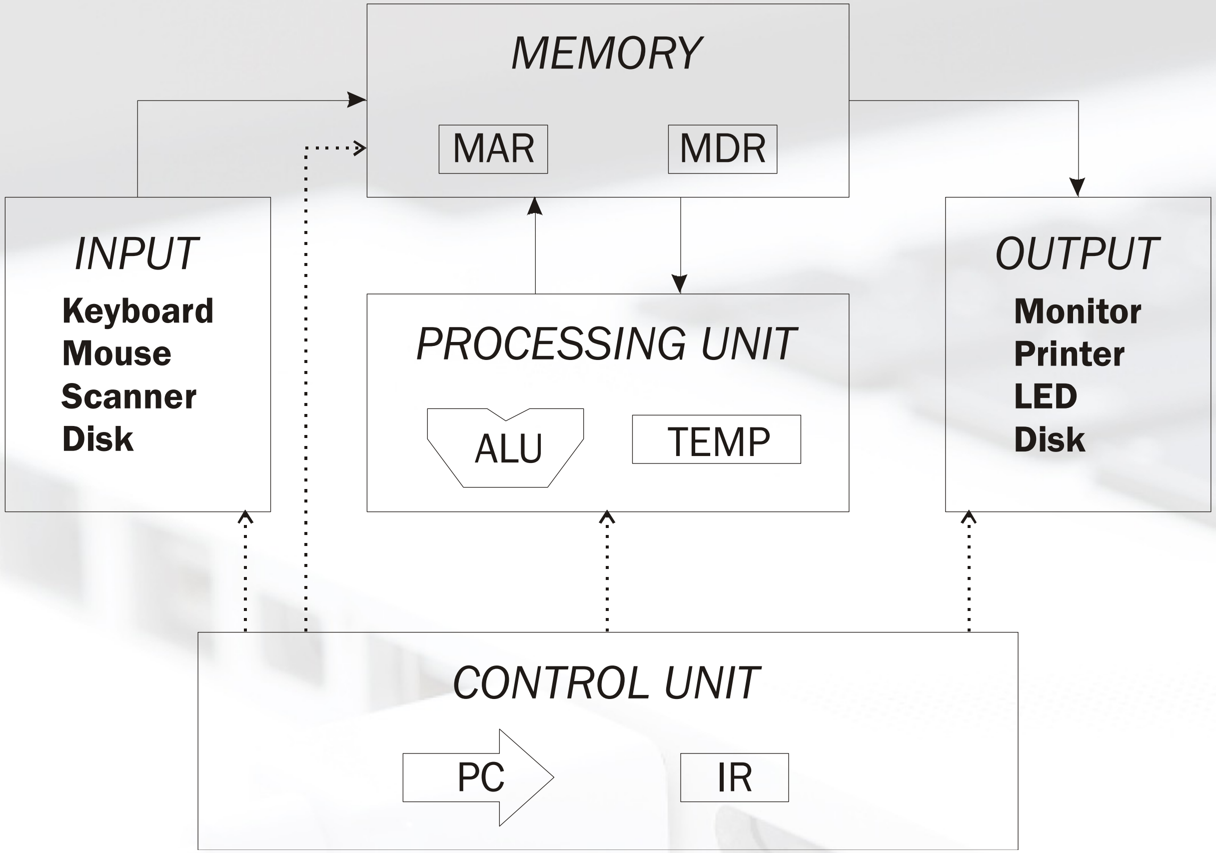






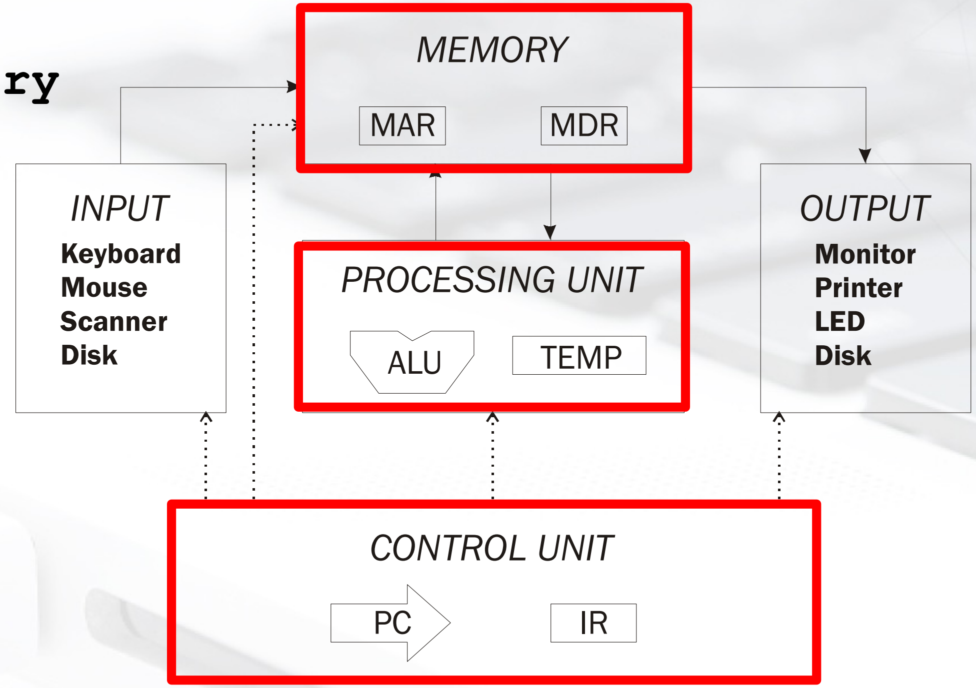


Review: Von Neumann Model



Review: Von Neumann Model

- So far, we've learned how to:
  - compute with values in registers
  - load data from memory to registers
  - store data from registers to memory

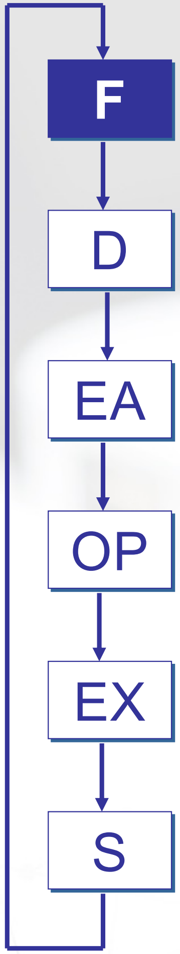


Review: The ISA

	15–12	11–9	8–6	5	4–3	2–0
ADD+	0001	DR	SR1	0	00	SR2
ADD+	0001	DR	SR1	1	imm5	
AND+	0101	DR	SR1	0	00	SR2
AND+	0101	DR	SR1	1	imm5	
BR	0000	n z p	PCoffset9			
JMP	1100	000	BaseR	000000		
JSR	0100	1	PCoffset11			
JSRR	0100	0 00	BaseR	000000		
LD+	0010	DR	PCoffset9			
LDI+	1010	DR	PCoffset9			

LDR+	0110	DR	BaseR	offset6
LEA	1110	DR	PCoffset9	
NOT+	1001	DR	SR	111111
RET	1100	000	111	000000
RTI	1000	000000000000		
ST	0011	SR	PCoffset9	
STI	1011	SR	PCoffset9	
STR	0111	SR	BaseR	offset6
TRAP	1111	0000	trapvect8	
reserved	1101			

Review: The State Machine(Turing Machine equivalent)



702 appendix C The Microarchitecture of the LC-3

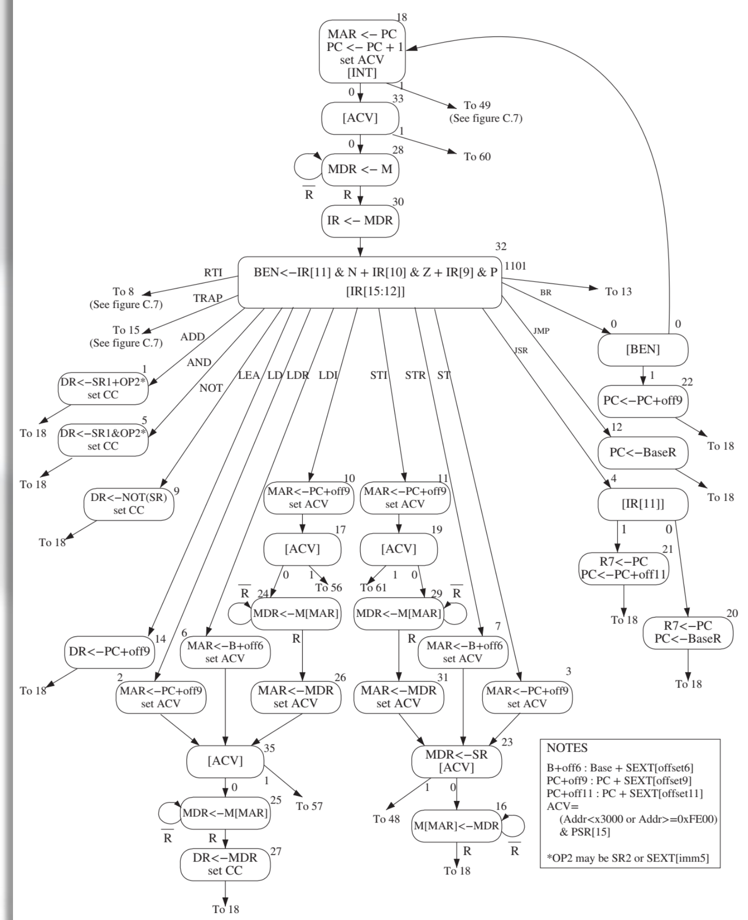


Figure C.2 A state machine for the LC-3.

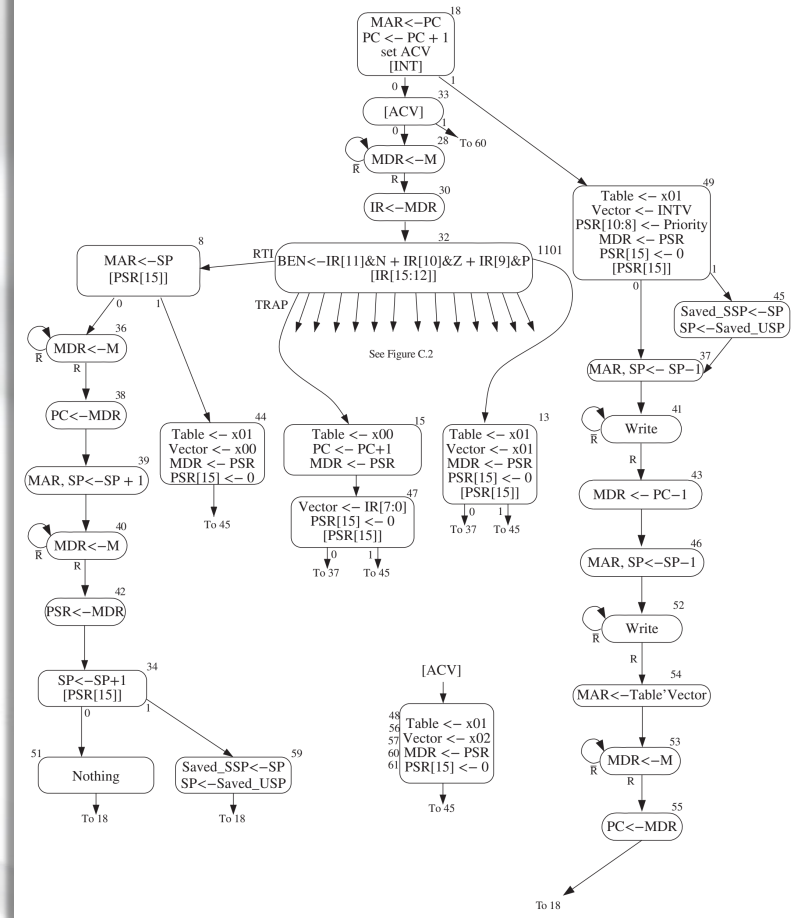


Figure C.7 LC-3 state machine showing interrupt control.

Review: The Data Path(von Neumann Model)

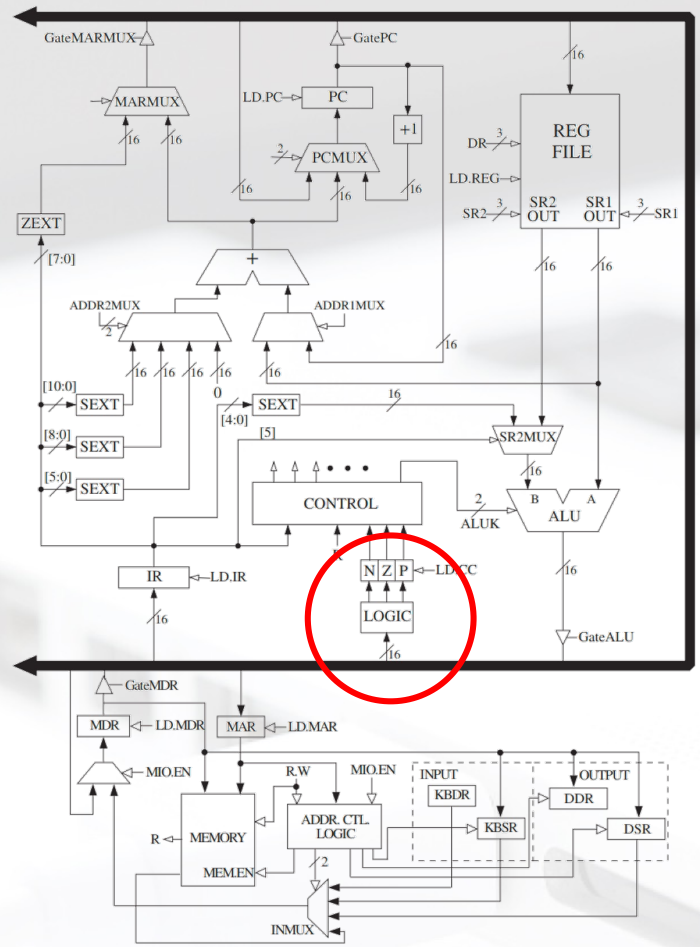
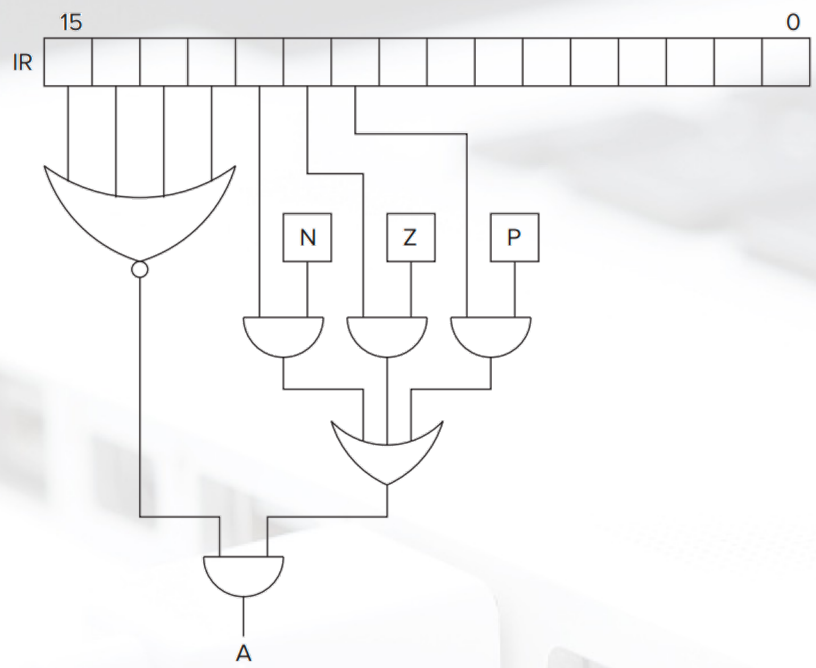


Figure C.3 The LC-3 data path.

Exercise

5.40 The following logic diagram shows part of the control structure of the LC-3 machine. What is the purpose of the signal labeled A?

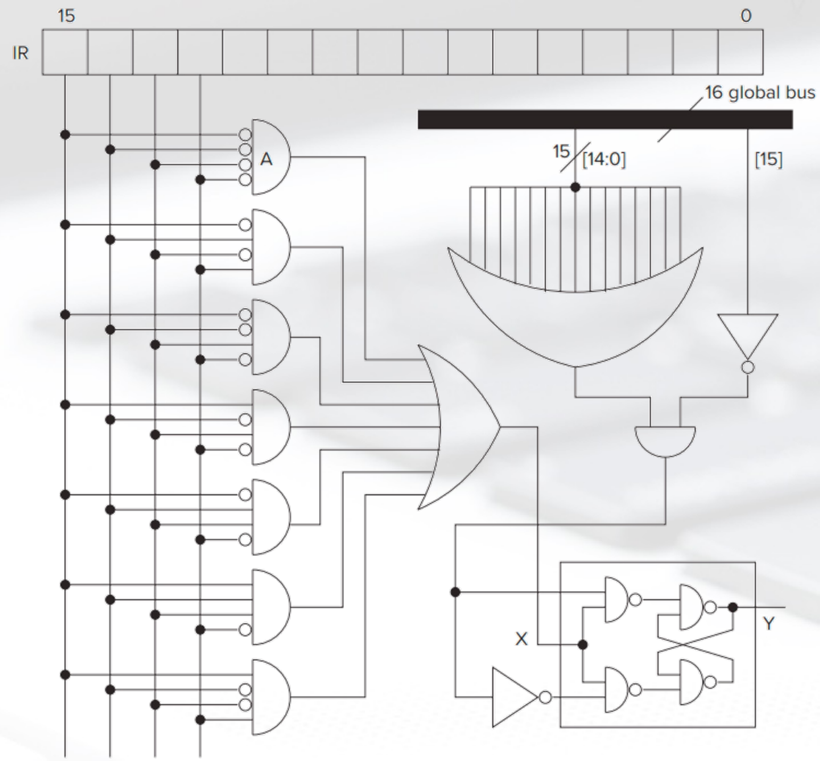


Exercise

5.41 A part of the implementation of the LC-3 architecture is shown in the following diagram.

a. What information does Y provide?

b. The signal X is the control signal that gates the gated D latch. Is there an error in the logic that produces X?



Exercise

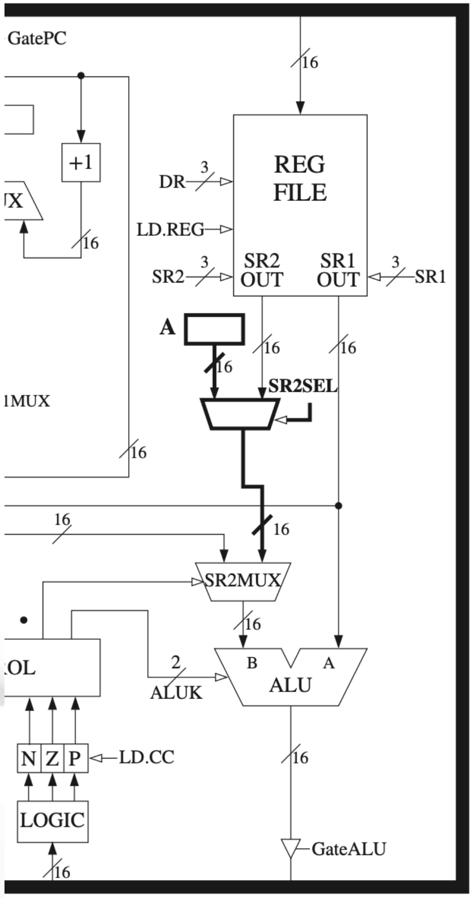
★5.51 An aggressive young engineer decides to build and sell the LC-3 but is told that if he wants to succeed, he really needs a SUBTRACT instruction. Given the unused opcode 1101, he decides to specify the SUBTRACT instruction as follows:

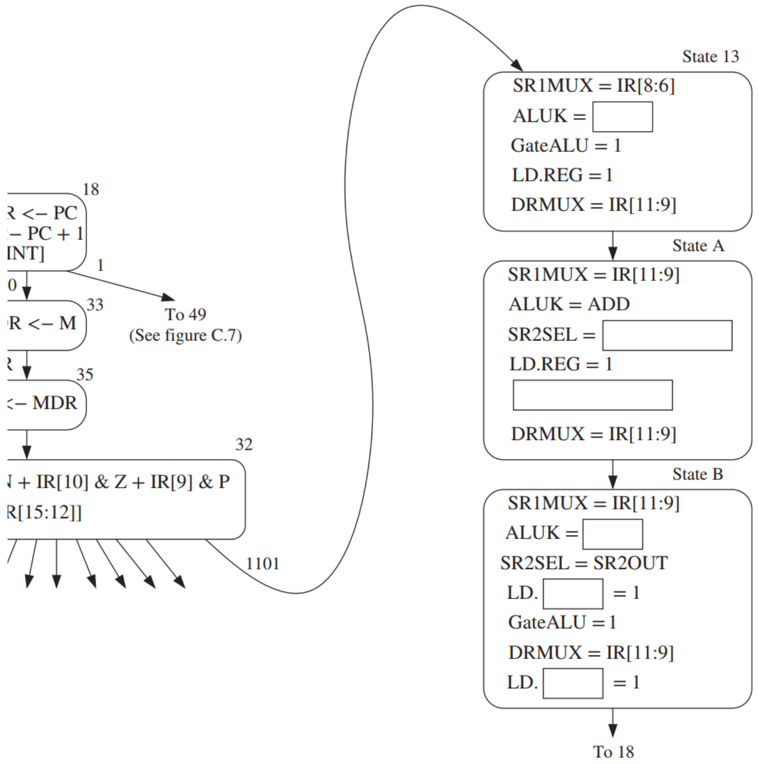
15–12	11–9	8–6	5–3	2–0
1101	DR	SR1	000	SR2

The instruction is defined as: DR ← SR2 - SR1, and the condition codes are set. **Assume DR, SR1, and SR2 are all different registers.**
To accomplish this, the engineer needs to add three states to the state machine and a mux and register A to the data path. The modified state machine is shown below, and the modified data path is shown on the next page. The mux is controlled by a new control signal SR2SEL, which selects one of its two sources.

SR2SEL/1: SR2OUT, REGISTER_A

Your job:
For the state machine shown below, fill in the empty boxes with the control signals that are needed in order to implement the SUBTRACT instruction.
For the data path, fill in the value in register A.





Outline

1 Review

2 **Assembly Language Overview**

3 Assembly Process

4 Summary

A LC-3 Program

X4101	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0
X4102	0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0
X4103	1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 1
X4104	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0
X4105	0 0 0 1 1 0 0 0 0 1 1 1 1 1 0 0
X4106	0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0
X4107	1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1
X4108	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1
X4109	0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0
X410A	0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1
X410B	0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 1
X410C	0 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1
X410D	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0
X410E	0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 0
X410F	0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0
X4110	0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
X4101	1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1
X4102	1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1
X4103	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
X4104	0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0

X8001	0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0
X8002	0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0
X8003	0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1
X8004	0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0
X8005	0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0
X8006	0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1
X8007	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
X8008	0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1
X8009	0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0
X800A	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
X800B	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
X800C	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
X800D	0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0
X800E	0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1
X800F	0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0
X8010	0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0
X8011	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
X8012	0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
X8013	0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0
X8014	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
X8015	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
X8016	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
X8017	0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0
X8018	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

Human-Readable Machine Language

- Computers like ones and zeros…

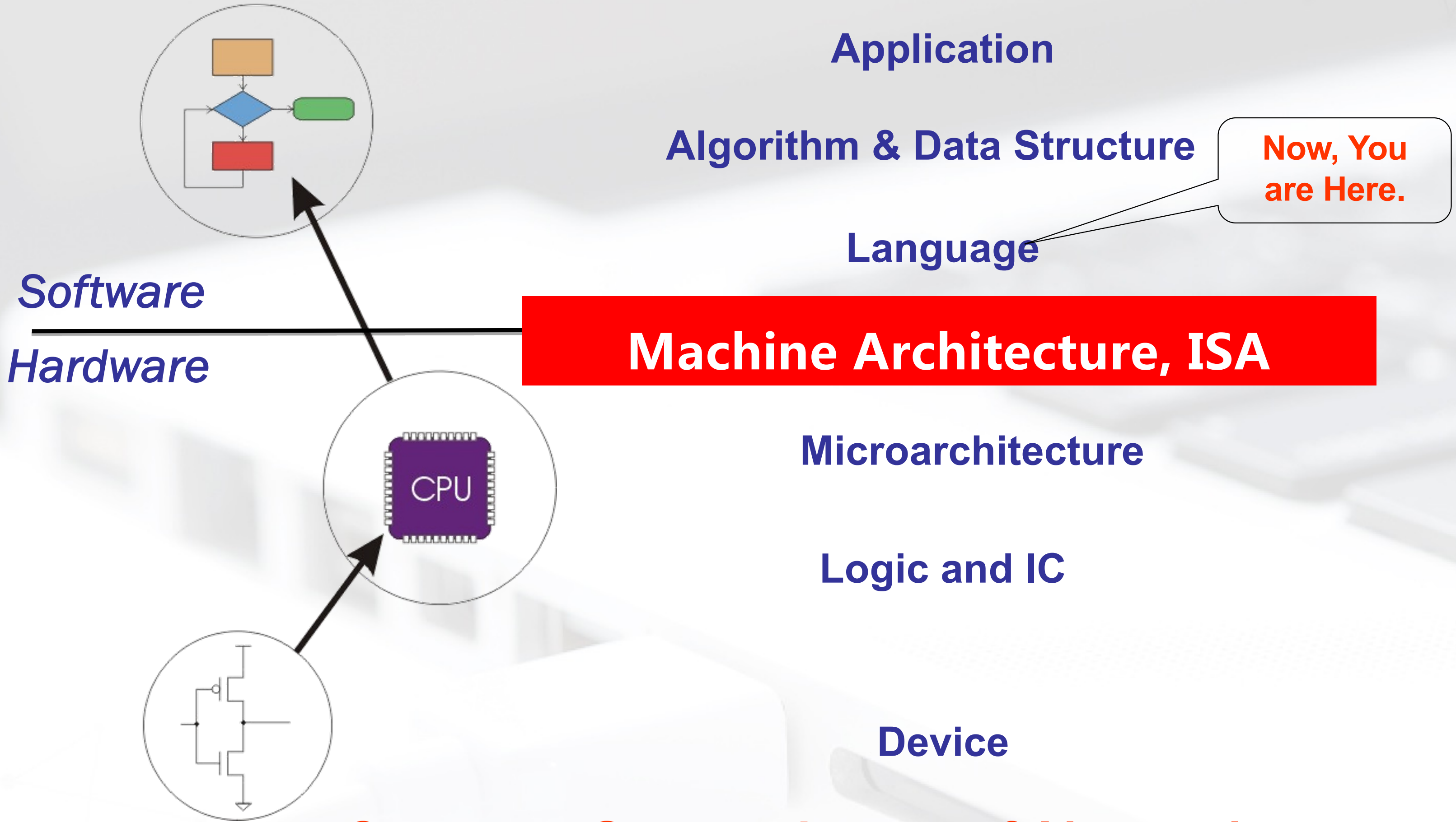
  `0001110010000110`

- Humans like symbols…

```
ADD   R6,R2,R6   ; increment index reg.
or
C = a + b;
```

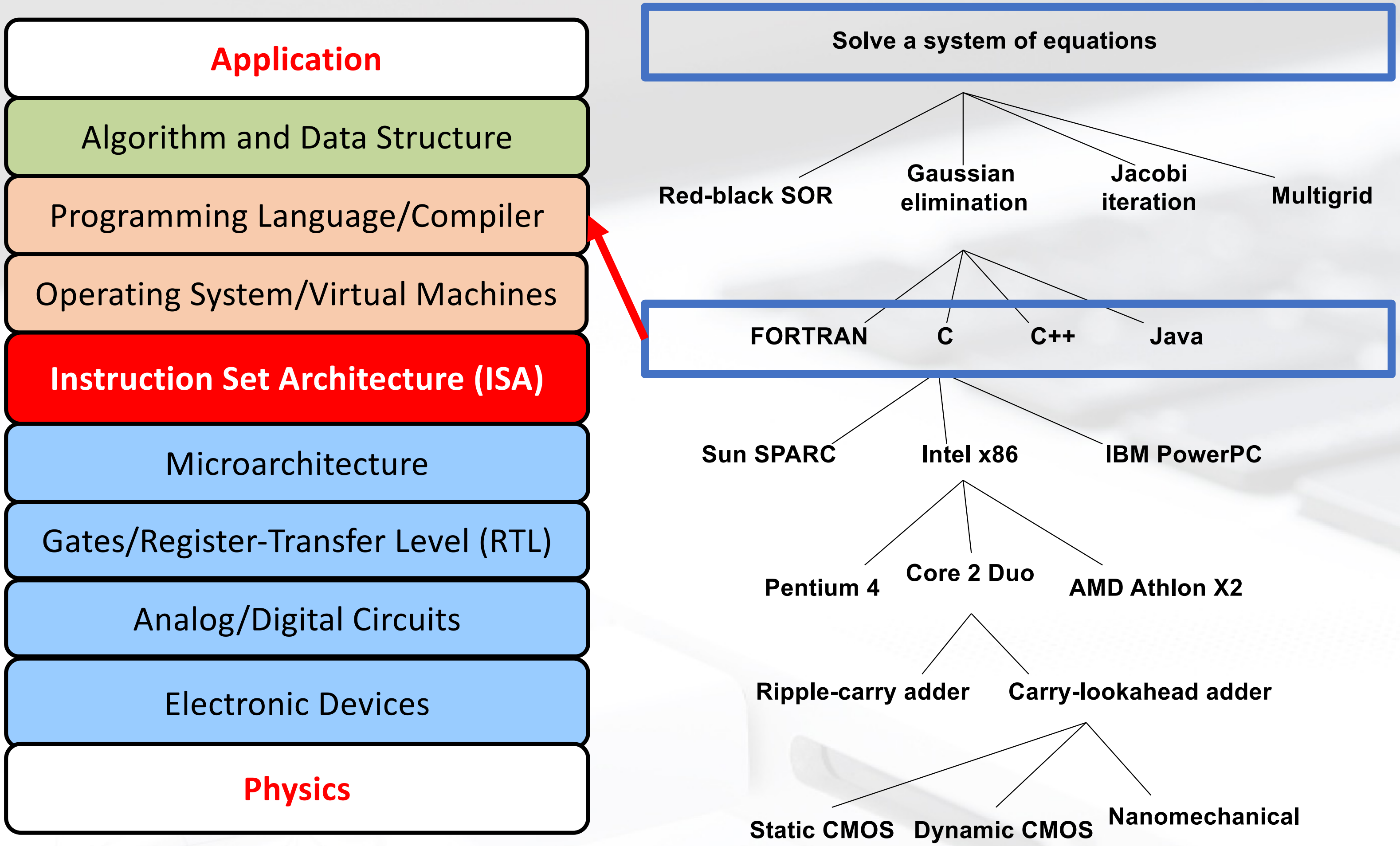
- **Assembler** is a program that turns symbols into machine instructions.
  - ISA-specific: close correspondence between symbols and instruction set
    - — mnemonics for opcodes
    - — labels for memory locations
  - additional operations for allocating storage and initializing data

Great Idea #4: Software and Hardware Co-design



Computer System: Layers of Abstraction

Great Idea #3: Abstraction Helps Us Manage Complexity



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
```

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
```

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) or b (binary)
- 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
```

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

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`	value	allocate one word, initialize with a value
`.STRINGZ`	n-character string	allocate n+1 locations, initialize w/characters and null terminator

Example

```
        .ORIG   X3010
HELLO   .STRINGZ  " Hello, World! "
```

```
x3010: x0048
x3011: x0065
x3012: x006C
x3013: x006C
x3014: x006F
x3015: x002C
x3016: x0020
x3017: x0057
x3018: x006F
x3019: x0072
x301A: x006C
x301B: x0064
x301C: x0021
x301D: x0000
```

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.

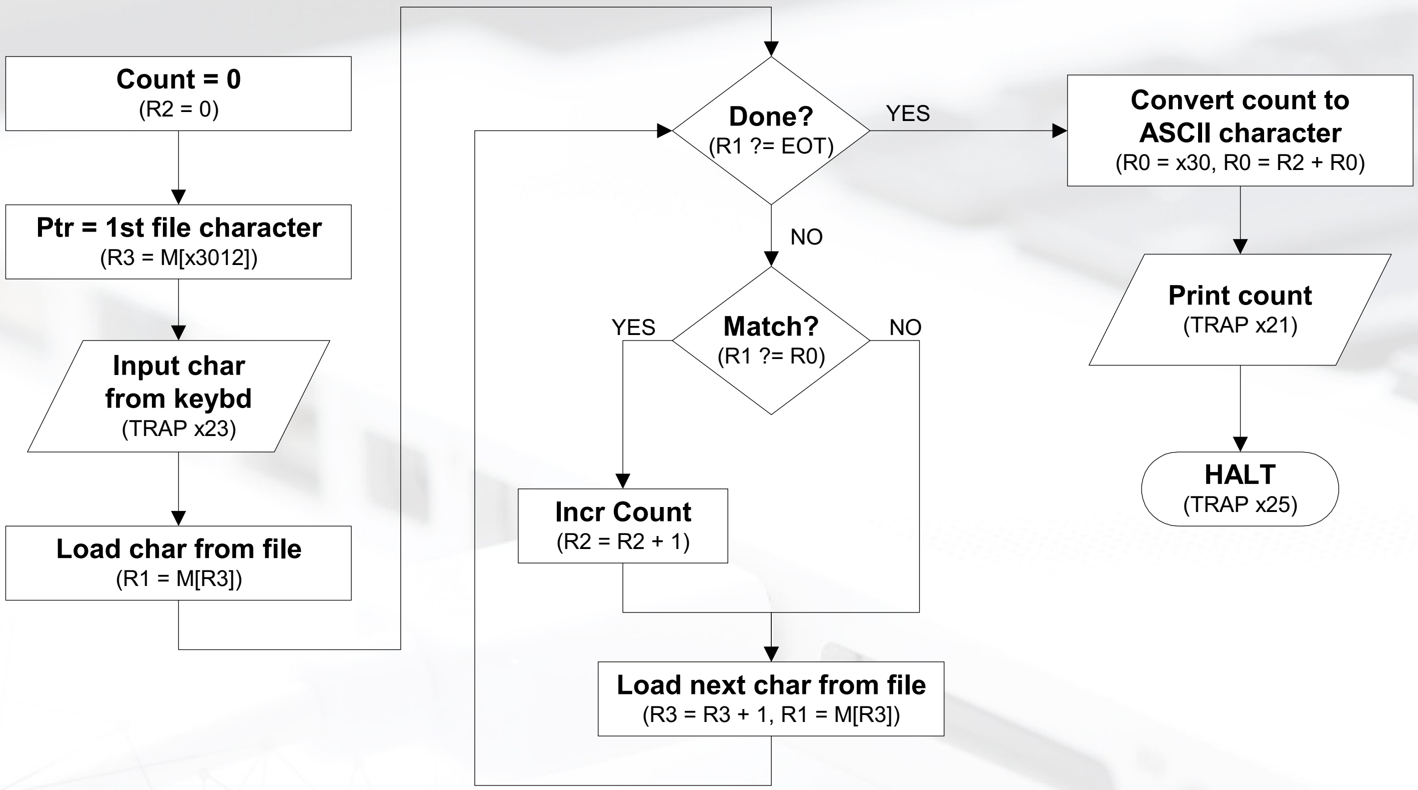
Style Guidelines

- 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.

Sample Program

Remember this?

- Count the occurrences of a character in a file.



Program (1 of 2)

Address	Instruction	Comments
x3000	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	R2 ← 0 (counter) AND R2,R2, #0
x3001	0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0	R3 ← M[x3012] (ptr) LD R3, x3012 (LD R3, PTR)
x3002	1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 1	Input to R0 (TRAP x23) TRAP x23 (GETC)
x3003	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	R1 ← M[R3] LDR R1, R3, #0
x3004	0 0 0 1 1 0 0 0 0 1 1 1 1 1 0 0	R4 ← R1 – 4 (EOT) ADD R4,R1, #-4
x3005	0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0	If Z, goto x300E BRz x300E (BRz OUTPUT)
x3006	1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1	R1 ← NOT R1 NOT R1,R1
x3007	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	R1 ← R1 + 1 ADD R1,R1,#1
X3008	0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0	R1 ← R1 + R0 ADD R1,R1,R0
x3009	0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1	If N or P, goto x300B BRnp x300B (BRnp GETCHAR)

Program (2 of 2)

Address	Instruction	Comments
x300A	0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 1	R2 ← R2 + 1 ADD R2,R2,#1
x300B	0 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1	R3 ← R3 + 1 ADD R3,R3,#1
x300C	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	R1 ← M[R3] LDR R1,R3,#0
x300D	0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 0	Goto x3004 BRnzp x3004 (BRnzp TEST)
x300E	0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0	R0 ← M[x3013] LD R0,x3013 (LD R0, ASCII)
x300F	0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0	R0 ← R0 + R2 ADD R0,R0,R2
x3010	1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1	Print R0 TRAP x21 (OUT)
x3011	1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1	HALT TRAP x25 (HALT)
X3012	1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	Starting Address of File (X9000)
x3013	0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0	ASCII x30 (‘0’)

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                    ; TRAP x23
                                ; 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
```

Char Count in Assembly Language (2 of 3)

```
;
; Test character for match.  If a match, increment count.
;
        NOT   R1, R1
        ADD   R1, R1, #1  ; R1 <-- -R1
        ADD   R1, R1, R0  ; R1 <-- R0 - R1. if R1=0, a match!
        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               ; TRAP x21
                          ; ASCII code in R0 is displayed.
        HALT              ; TRAP x25, Halt machine
```

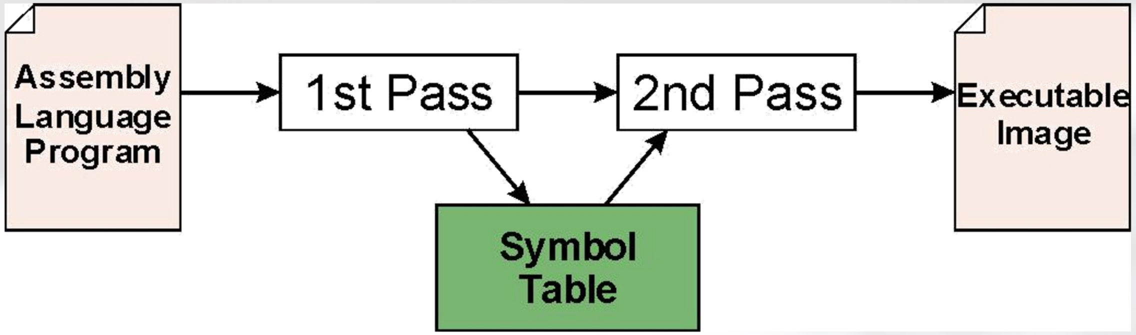
Char Count in Assembly Language (3 of 3)

```
;
; Storage for pointer and ASCII template
;
ASCII .FILL   x0030       ; ASCII code of number '0'
PTR   .FILL   x9000       ; pointer to the first character
      .END
```

Outline

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

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.**

Practice

■ Construct the symbol table for the program in Figure 7.2

```
; Initialization
;
            .ORIG   x3000
            AND     R2,R2,#0
            LD      R3,PTR
            TRAP    x23
            LDR.    R1,R3,#0
;
; Test char. For end of file
;
TEST        ADD     R4,R1,#-4
            BRz     OUTPUT
;
; Test char. For match.
;
            NOT     R1,R1
            ADD     R1,R1,#1
            ADD     R1,R1,R0
            BRnp    GETCHAR
            ADD     R2,R2,#1
;
; Get next char. from the file
;
GETCHAR     ADD     R3,R3,#1
            LDR     R1,R3,#0
            BRnp    TEST
;
; Output the count
;
OUTPUT      LD      R0,ASCII
            ADD     R0,R0,R2
            TRAP    x21
            TRAP    x25
;
; Storage for pointer and ASCII temp.
;
ASCII       .FILL   x0030
PTR         .FILL   x4000
            .END
```

Practice

- Construct the symbol table for the program in Figure 7.2

Symbol	Address
TEST	X3004
GETCHAR	X300B
OUTPUT	X300E
ASCII	X3012
PTR	X3013

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) not on the same page`**
    - — can't use direct addressing mode

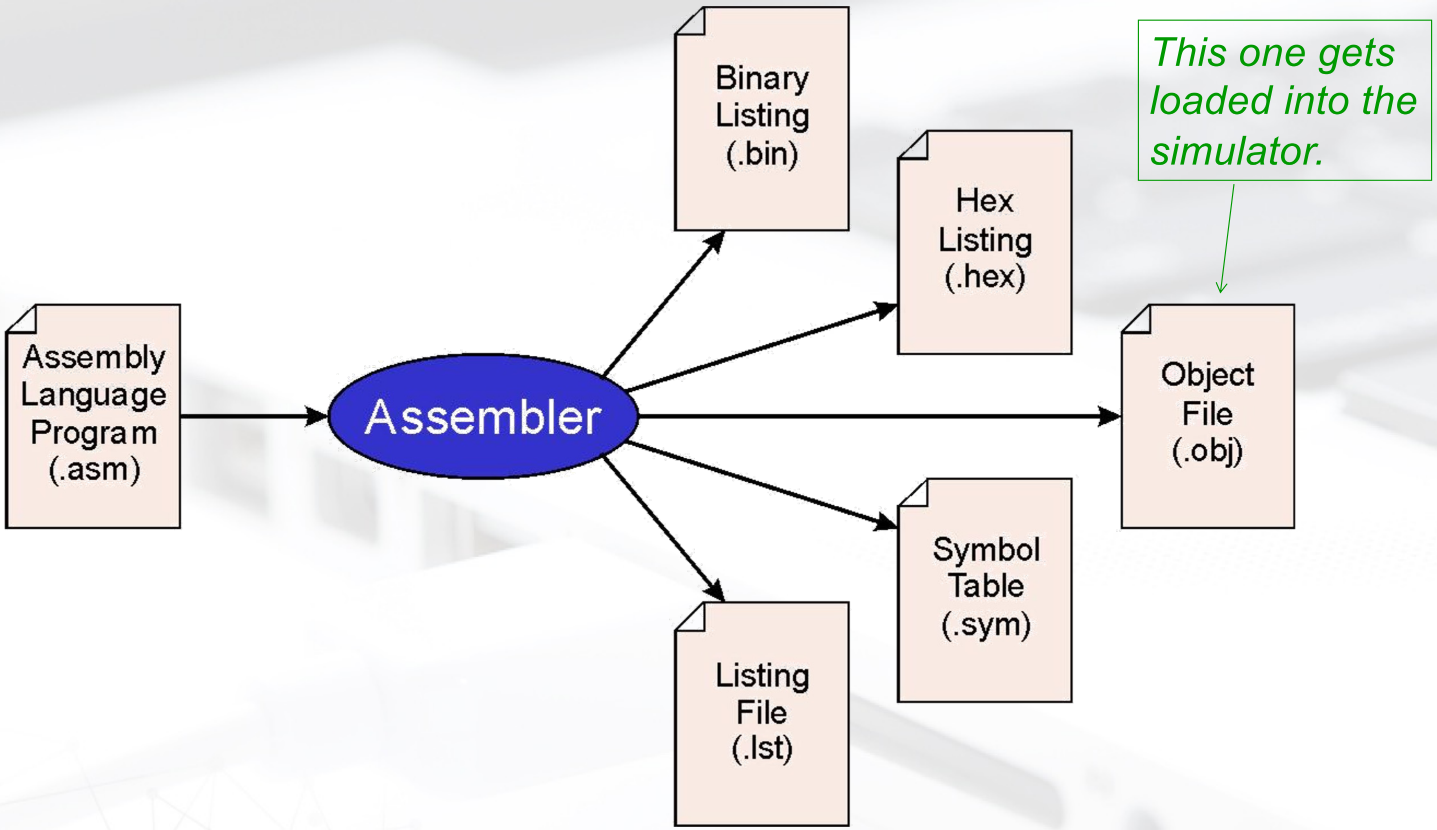
Practice

- **Using the symbol table constructed earlier, translate these statements into LC-3 machine language.**
- ***(Assume all addresses are on the current page.)***

Statement	Machine Language
`LD   R3,PTR`	
`ADD  R4,R1,#-4`	
`LDR  R1,R3,#0`	
`BRnp GETCHAR`	

LC-3 Assembler

- Using "assemble" (Unix) or LC3 Edit (Windows), generates several different output files.



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:

Object file	Assembly
0011000000000000	.ORIG x3000
0101010010100000	AND R2, R2, #0
0010011000010100	LD R3, PTR
1111000000100011	TRAP x23
.	
.	
.	

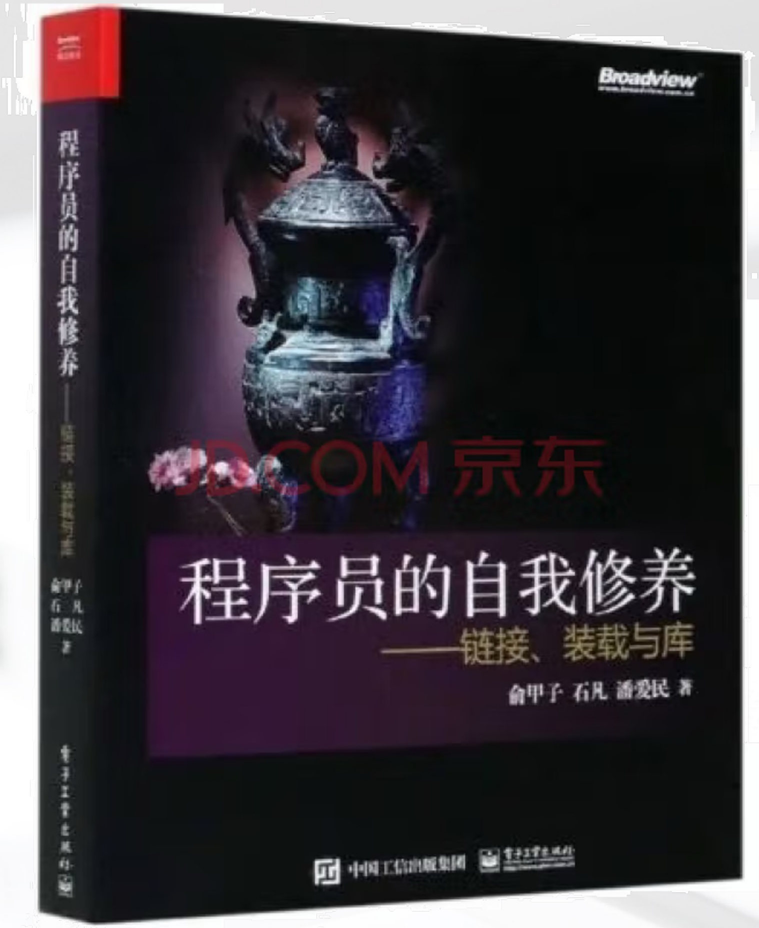
Multiple Object Files

- An object file is not necessarily a complete program.
  - `system-provided library routines`
  - `code blocks written by multiple developers`
- For LC-3, 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 x1000
    - by convention, user code should be loaded between x3000 and xCFFF
  - `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 will search symbol tables of other modules to resolve symbols and complete code generation before loading

Linking and Loading



Outline

Summary: Assembly Language

High Level Language Program (e.g., C)

Compiler

Assembly Language Program (e.g., MIPS)

Assembler

Machine Language Program (MIPS)

Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)

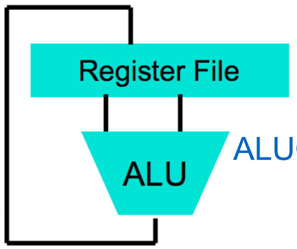
```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Now, You are Here.

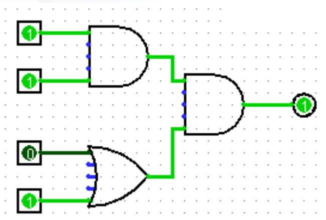
```
lw    $t0, 0($2)
lw    $t1, 4($2)
sw    $t1, 0($2)
sw    $t0, 4($2)
```

Anything can be represented as a *number*, i.e., data or instructions

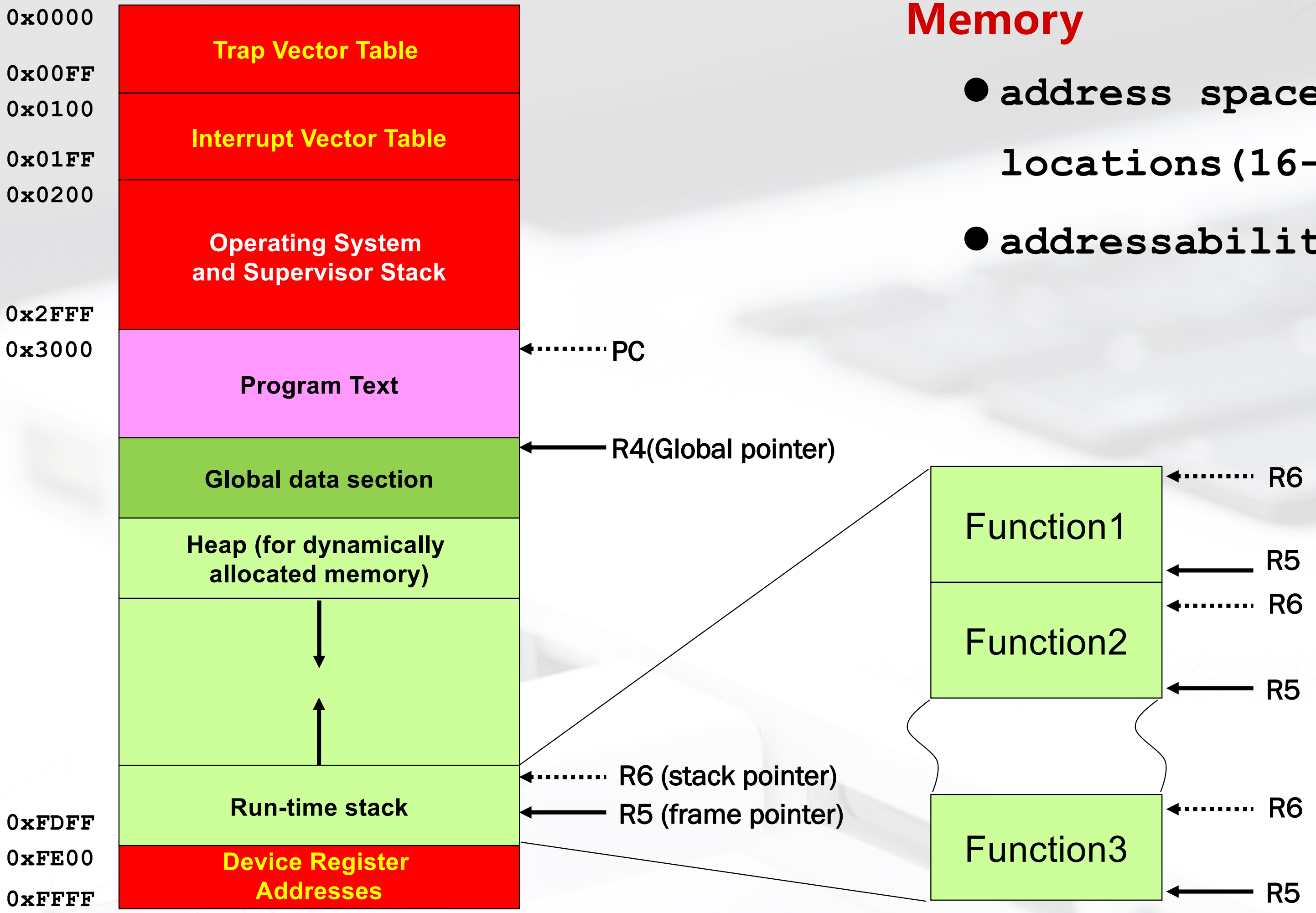
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```



ALUOP[0:3] <= InstReg[9:11] & MASK



Memory map of the LC-3



Memory

- address space: 2^{16} locations (16-bit addresses)
- addressability: 16 bits

7.2 An LC-3 assembly language program contains the instruction:

```
ASCII   LD  R1, ASCII
```

The symbol table entry for ASCII is x4F08. If this instruction is executed during the running of the program, what will be contained in R1 immediately after the instruction is executed?