

Midterm Review

Fall 2006

CIT 593

R-1

Chapter 1

- **Universality**
 - All computers can compute the same thing
 - Concept invented by Alan Turing
- **Layered Abstraction**
 - We can build very complex systems from simple components

CIT 593

R-2

Chp 1: Universality

In theory

- Computers can compute anything that's possible to compute
- Given enough *memory* and *time*

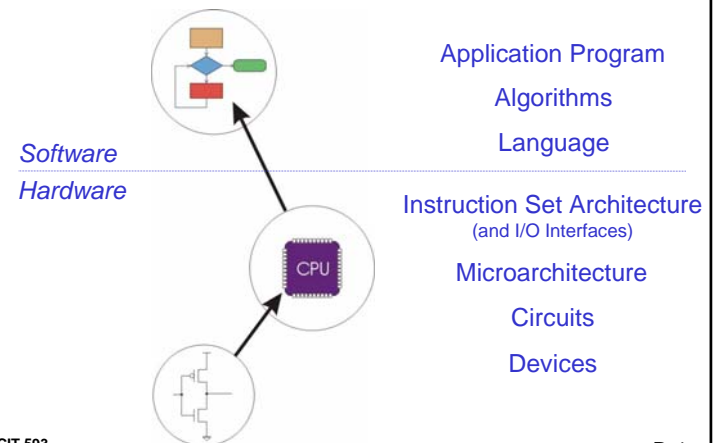
In practice

- Solving *real* problems requires computing under constraints
- Time
 - Weather forecast, next frame of animation, ...
- Cost
 - Cell phone, automotive engine controller, ...
- Power
 - Cell phone, handheld video game, ...

CIT 593

R-3

Chp 1: Layers of Abstraction



CIT 593

R-4

Chapter 2

Binary System

- **Basic unit of information - the *binary digit*, or *bit***
- **Has two states: 0 and 1**
- **3+ state values require multiple bits**
 - A collection of **two** bits has **four** possible states:
00, 01, 10, 11
 - A collection of **three** bits has **eight** possible states:
000, 001, 010, 011, 100, 101, 110, 111
 - A collection of **n** bits has **2^n** possible states

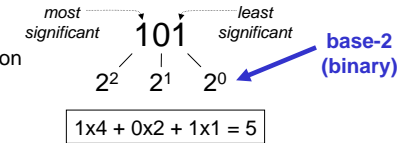
CIT 593

R-5

Chp 2: Integer Representation

Unsigned

- Weighted positional notation
- Values from 0 to $2^n - 1$



Signed

- Assign "half" to positive integers (1 through $\sim 2^{n-1}$) and "half" to negative ($\sim -2^{n-1}$ through -1)
- Three types: Signed, 1's comp, 2's comp

CIT 593

R-6

Chp 2: 2's Complement

2's complement

- **If number is positive or zero**
 - Normal binary representation, zeroes in upper bit(s)
- **If number is negative**
 - Start with positive number
 - Flip every bit (*i.e.*, take the one's complement)
 - Then add one

$$\begin{array}{r}
 00101 \quad (5) \\
 11010 \quad (1's \text{ comp}) \\
 + \quad \quad 1 \\
 \hline
 11011 \quad (-5)
 \end{array}$$

CIT 593

R-7

Chp 2: Binary \longleftrightarrow Decimal

$$\begin{aligned}
 X &= 00100111_{\text{two}} \\
 &= 2^5 + 2^2 + 2^1 + 2^0 = 32 + 4 + 2 + 1 \\
 &= 39_{\text{ten}}
 \end{aligned}$$

$$\begin{aligned}
 X &= 104_{\text{ten}} \\
 104 - 64 &= 40 \quad \text{bit 6} \\
 40 - 32 &= 8 \quad \text{bit 5} \\
 8 - 8 &= 0 \quad \text{bit 3}
 \end{aligned}$$

$$X = 01101000_{\text{two}}$$

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

CIT 593

R-8

Chp2: Sign Extension

- **To get correct results**

- Must represent numbers with same number of bits
- Use the MSB as the sign bit

<u>4-bit</u>	<u>8-bit</u>
0100 (4)	00000100 (still 4)
1100 (-4)	11111100 (still -4)

CIT 593

R-9

Chp 2: Add/Sub

Addition

```

000010111 (23)
+000000111 (7)
-----
000011110 (30)
    
```

Subtraction

```

01101000 (104)
+11110000 (-16)
-----
01011000 (88)
    
```

Overflow

1. Signs of both numbers are the same, and Sign of sum is different
2. If Positive number is subtracted from a Negative number, result is positive and vice versa

CIT 593

R-10

Chp2 : Logical Operations

- **Operations on logical TRUE or FALSE**

- Two states: TRUE=1, FALSE=0

A	B	A AND B	A	B	A OR B	A	NOT A
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

- **View n -bit number as a collection of n logical values**

- Operation applied to each bit independently

CIT 593

R-11

Chp 2: Hexadecimal Notation

- **Hexadecimal (base-16) numbers**

- Fewer digits: four bits per hex digit
- Less error prone: easy to corrupt long string of 1's & 0's

Binary	Hex	Decimal	Binary	Hex	Decimal
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	A	10
0011	3	3	1011	B	11
0100	4	4	1100	C	12
0101	5	5	1101	D	13
0110	6	6	1110	E	14
0111	7	7	1111	F	15

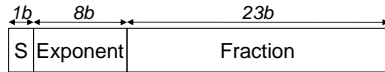
CIT 593

R-12

Chp 2: Floating-Point

IEEE 754 Floating-Point Standard (32-bits)

- Need to represent *F* (fraction), *E* (exponent), and sign
- Can represent very small and large values



$$N = -1^S \times 1.\text{fraction} \times 2^{\text{exponent}-127}, 1 \leq \text{exponent} \leq 254$$

$$N = -1^S \times 0.\text{fraction} \times 2^{-126}, \text{exponent} = 0$$

CIT 593

R-13

Chp2: Characters & Strings

ASCII: Maps 128 characters to 7-bit code.

- Both printable and non-printable (ESC, DEL, ...)
characters

Text strings

- Sequence of characters, terminated with NULL (0)
- Typically, no hardware support
 - In LC3, .STRINGZ is not an instruction

CIT 593

R-14

Chapter 3

•Combinational Structures/Circuits

- Always gives the same output for a given set of inputs
- Do not store any information
- Examples: adder, subtracter and multiplexer (mux)

•Sequential Structures/Circuits

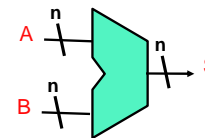
- Always store information
- Example: memory, register file

CIT 593

R-15

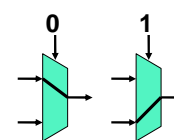
Chp3: Combinational Structure

• Adder

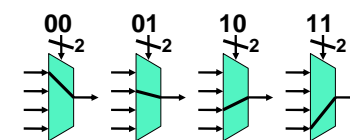


• Mux

2-to-1 Mux



4-to-1 Mux



CIT 593

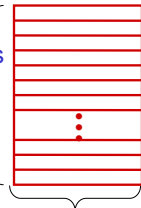
R-16

Chp 3: Sequential Structure

Memory (logical view)

- Holds information (data or instructions)
- a logical k by m array of stored bits

Address Space:
number of locations
(usually a power of 2)
 $k = 2^n$
locations



Addressability:
number of bits per location

CIT 593

R-17

Chp3: Memory Size

- Binary Systems notation

- 1 Byte = $2^3 = 8$ bits
- 1 Kilo Byte (KB) = 1024 bytes = 2^{10} Bytes = $2^{10} \times 2^3 = 2^{13}$ bits
- 1 Mega Byte (MB) = 1024 KB = 2^{20} Bytes = $2^{20} \times 2^3 = 2^{23}$ bits
- 1 Giga Byte (GB) = 1024 MB = 2^{30} Bytes = $2^{30} \times 2^3 = 2^{33}$ bits

- Example: 16 MB memory = $2^4 \times 2^{20} = 2^{24}$ bytes

- $2^{24} = 16777216$ (~ 16 million) unique locations, each holds a 8-bit value

- Different size of data storage

- We can also have each memory location hold 16-bit or 32-bit depending on the machine's need
- Example: LC3 has $2^{16} \times 16 (= 2^4)$ memory = $2^{17} \times 2^3 = 2^{17}$ bytes

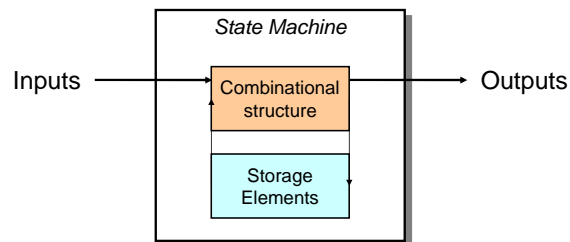
CIT 593

R-18

Chp3: State Machine

Another type of sequential structure

- Combines combinational structure with storage (memory element)
- "Remembers" state (old stored information), and changes output (and state) based on inputs and current state



CIT 593

R-19

Chapter 4

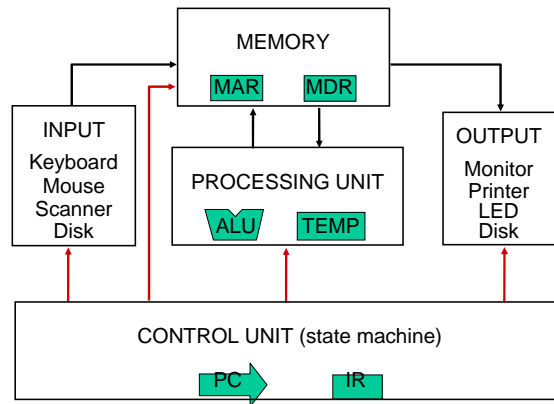
To perform a task, computer is given

- Program: Consists of a set of instructions that the computer must to do complete the task
- Instructions: Smallest piece of work specified (fundamental unit)
 - Either carried completely or not at all (this is because future instructions can be dependent on the previous)

CIT 593

R-20

Chp 4: Von Neumann Model



CIT 593

R-21

Chp 4: ISA

• Instruction Set Architecture (ISA)

- Computer's instructions
 - ADD, NOT, SUB
- Instruction Format
 - Which bits are opcodes, which are operands
- Instruction Behaviour
 - How instructions perform
- Other
 - addressing modes, interrupt and exception handling, and I/O
 - Memory Organization, Register File

CIT 593

R-22

Chp 4: Instruction Processing

- **Fetch**
 - Get the next instruction
- **Decode**
 - Identifies OPCODE, SRC and DEST Reg
- **Evaluate Address**
 - Used by Load, Stores, Branch, JSR
- **Operand Fetch**
 - Read the content of the register
- **Execute**
 - Arithmetic: Add/Sub, logical: AND, OR
- **Store**
 - Write to a Register or Memory Location



CIT 593

R-23

Chp 4: Instructions that change PC

• Increment of PC during FETCH

- FETCH phase always increments PC by 1 instruction

• We can also skip instructions:

- programming constructs that change the sequence of instruction execution like *If-then, loops* etc.
 - E.g: *If (Overall points equals 90) then Grade equals A*
- **Branches** are conditional
 - Change the PC only if some condition is true
e.g., the contents of a register is zero
- **Jumps** are unconditional ->Always change the PC

Which other ?

Trap Routines, Interrupt Routines and Regular Subroutines

CIT 593

R-24

Chapter 5

LC3 Overview

• Memory and Register File

➤ 2^{16} x 16 and 8 registers

• Opcodes

➤ 16 opcodes ([15:12] of instruction = $2^4 = 16$ possible values)

➤ Types of instructions:

- **Operate** instructions: ADD, AND, NOT
- **Data movement** instructions: LD, LDI, LDR, LEA, ST, STR, STI
- **Control** instructions: BR, JUMP, TRAP JSR, JSRR, RET, RTI

• Addressing Modes

➤ How is the location of an operand (data to acted upon) specified?

➤ Non-memory addresses: *register*, *immediate (literal)*

➤ Memory addresses: *base+offset*, *PC-relative*, *indirect*

• Data Types

➤ 16-bit 2's complement integer

CIT 593

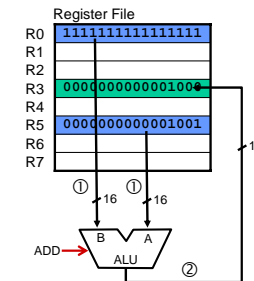
R-25

Chp 5: ADD (Register format)

— this zero means "register mode"

ADD 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 1 DR SR1 0 0 0 SR2

IR ADD R3 R5 R0
0001011101000000

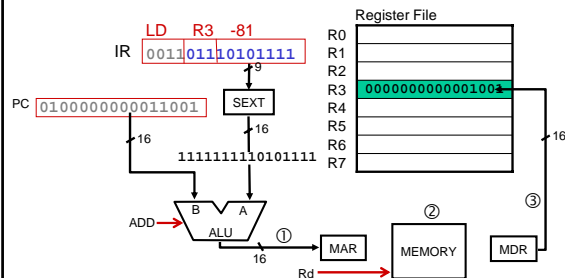


CIT 593

R-26

Chp 5: LD (PC-Relative)

LD 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 0 DR PCOffset9



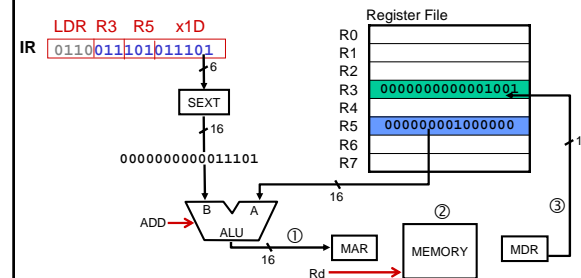
Example: LD: R1 ← M[PC+SXT(IR[8:0])]

CIT 593

R-27

Chp 5: LDR (Base+Offset)

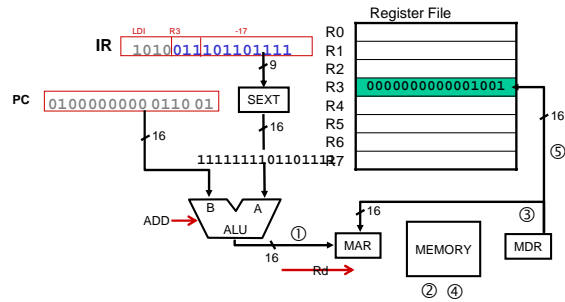
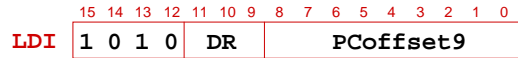
LDR 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 0 DR BaseR offset6



CIT 593

R-28

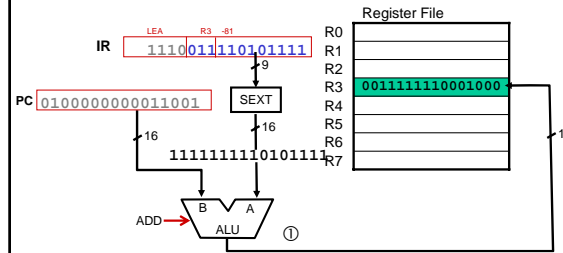
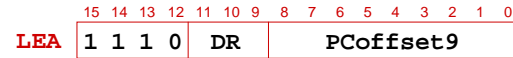
Chp 5: LDI (Indirect)



CIT 593

R-29

Chp 5: LEA



CIT 593

R-30

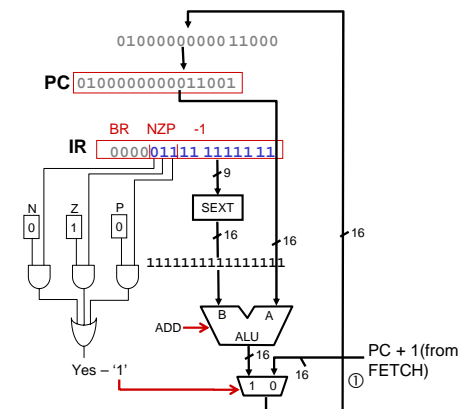
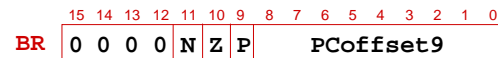
Chp 5: Control Instructions

- LC-3 has three 1-bit **condition code** registers
N -- negative
Z -- zero
P -- positive (greater than zero)
- **Set/cleared by instructions that store value to register**
 ➤ e.g., ADD, AND, NOT, LD, LDR, LDI, LEA, *not* ST
- **Exactly one will be set at all times**
 ➤ Based on the last instruction that altered a register

CIT 593

R-31

Chp 5: BR



CIT 593

R-32

Chp 5: Instruction Summary

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	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													
LD*	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															

CIT 593

R-33

Chapter 7

- Computers like ones and zeros...

0001110010000110

- Humans like readable form ...Assembly Language

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

R-34

Chp 7: 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

R-35

Chp 7: Assembler Directives

• Pseudo-operations

- Do not refer to operations executed by program
- Used by assembler
- Look like instruction, but "opcode" starts with dot

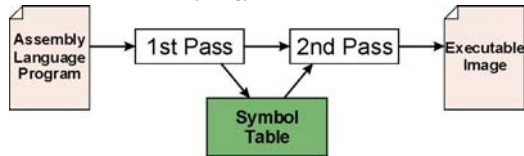
Opcode	Operand	Meaning
.ORIG	address	starting address of program
.END		end of program
.FILL	value	allocate one word, initialize with value
.BLKW	number	allocate multiple words of storage, value unspecified
.STRINGZ	n-character string	allocate n+1 locations, initialize w/characters and null terminator

CIT 593

R-36

Chp 7: 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

R-37

Chapter 8

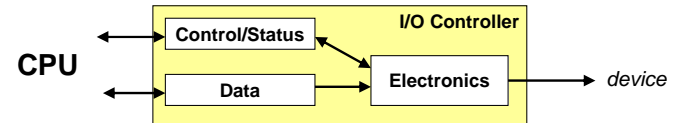
I/O Basics

• Control/Status Registers

- CPU tells device what to do -- write to control register
- CPU checks whether task is done -- read status register

• Data Registers

- CPU transfers data to/from device



• Device electronics

- Performs actual operation
 - Pixels to screen, bits to/from disk, characters from keyboard

CIT 593

R-38

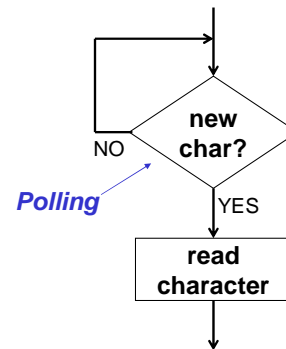
Chp 8: Programming Interface

- How are device registers **identified**?
 - Memory-mapped vs. special instructions
- How is **timing** of transfer managed?
 - Asynchronous vs. synchronous
- Who **controls transfer**?
 - CPU (polling) vs. device (interrupts)

CIT 593

R-39

Chp 8: Basic Input Routine



```
POLL    LDI  R0, KBSRPtr
        BRzp POLL
        LDI  R0, KBDPPtr
        ...
KBSRPtr .FILL xFE00
KBDPPtr .FILL xFE02
```

CIT 593

R-40

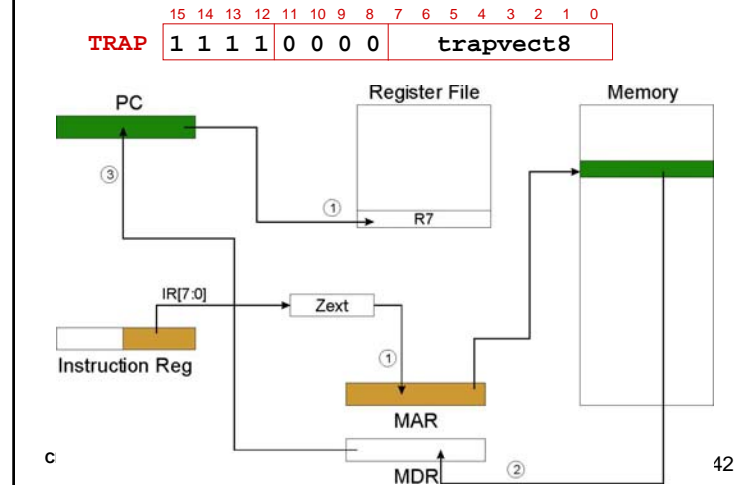
Chp 8: Role of the Operating System

- In real systems, only the operating system (OS) does I/O
 - “Normal” programs ask the OS to perform I/O on its behalf
- Hardware prevents non-operating system code from
 - Accessing I/O registers
 - Operating system code and data
 - Accessing the code and data of other programs
- Why?
 - Protect programs from themselves
 - Protect programs from each other
 - Multi-user environments

CIT 593

R-41

Chapter 9



c

42

Chp 9: Need for saving & restoring reg

```

AGAIN    LEA   R3, Block      ; Init. to first loc.
         LD    R6, ASCII     ; Char->digit template
         LD    R7, COUNT     ; Init. to 10
         TRAP x23           ; Get char
         ADD  R0, R0, R6     ; Convert to number
         STR  R0, R3, #0     ; Store number
         ADD  R3, R3, #1     ; Incr pointer
         ADD  R7, R7, -1     ; Decr counter
         BRp  AGAIN         ; More?
         BRnzp NEXT_TASK
ASCII    .FILL xFFD0        ; Negative of x0030
COUNT  .FILL #10
Block   .BLKW #10
    
```

What's wrong with this code?

CIT 593

R-43

Chp 9: Saving and Restoring Registers

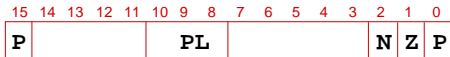
- Called routine ⇒ “callee-save”
 - Before start, save registers that will be altered
(unless altered value is desired by calling program!)
 - Before return, restore those same registers
 - Values are saved by storing them in memory
- Calling routine ⇒ “caller-save”
 - If register value needed later, save register destroyed by own instructions or by called routines (if known)
 - Save R7 before TRAP
 - Save R0 before TRAP x23 (input character)
 - Or avoid using those registers altogether
- LC-3: By convention, callee-saved when possible

CIT 593

R-44

Chp 9: Privilege

- **Goal: Isolation**
 - OS performs I/O (in traps)
 - Application can't perform I/O directly
- **Privilege: Processor modes**
 - Encoded in 15th bit of processor status register (PSR)
 - Privileged (supervisor – PSR[15] = 1)
 - Unprivileged (user – PSR[15] = 0)

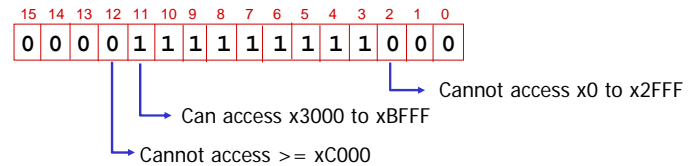


CIT 593

R-45

Chp 9: Supervisor Mode Versus User Mode

- **Supervisor mode**
 - Program has access to resources not available to user programs
 - LC-3: Memory
- **User mode in LC-3**
 - Memory access is limited by memory protection register (MPR)
 - Each MPR bit corresponds to 4K memory segment
 - 1 indicates that users can access memory in this segment

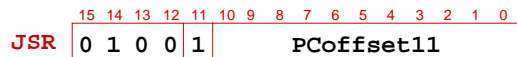


CIT 593

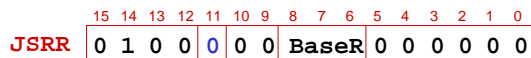
R-46

Chp 9: Subroutine

- **JSR/JSRR** – saves the return address in R7 and computes the the starting address of the subroutine and loads it into PC
- **JSR**
 - PC-relative mode (just like PC-relative LD/ST)
 - Target address of the subroutine is incremented PC + offset



- **JSRR**
 - Base addressing mode
 - Target address of the subroutine is obtained from Base Register



CIT 593

R-47

Chp 9: Using Subroutines

- **Programmer must know**
 - **Address:** or at least a label that will be bound to its address
 - **Function:** what it does
 - NOTE: The programmer does not need to know *how* the subroutine works, but what changes are visible in the machine's state after the routine has run
 - **Arguments:** what they are and where they are placed
 - **Return values:** what they are and where they are placed

CIT 593

R-48

Chp 9: Linking and Loading

- **Linking** is the process of resolving symbols between independent object files
 - 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
- **Loading** is the process of copying an executable (machine code) image into memory

CIT 593

R-49

Chapter 10

Stack

A last-in first-out (LIFO) storage structure

- The first thing you put in is the last thing you take out
- The last thing you put in is the first thing you take out
- Not like an array, where you can access any item based on an index

Two main operations

PUSH: add an item to the stack

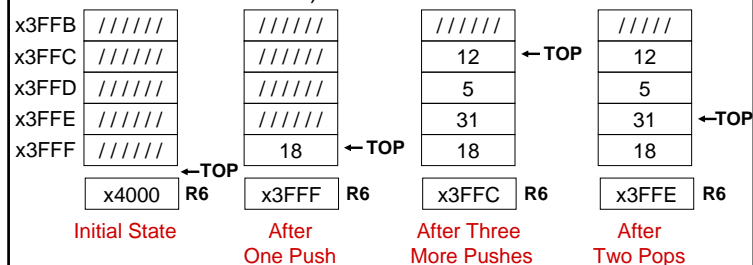
POP: remove an item from the stack

CIT 593

R-50

Chp 10: Software Stack

- Assume section x3FFFB to x3FFF in mem used for stack
- Data items don't move in memory, just our idea about where TOP of the stack is (a register is used to keep track of the location)



By convention, in LC3 R6 holds the Top of Stack (TOS) pointer

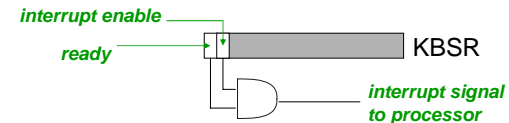
CIT 593

R-51

Chp 10: Interrupt Signal Generation

How is interrupt indicated?

- Control register has "Interrupt Enable" bit
- Must be set for interrupt to be generated



Who decides that processor can be allowed to interrupt?

- Processor can choose to ignore Interrupt (INT) signal
- Why? Example: may not want to be interrupted while a critical program is being executing or already in a middle of another interrupt that has higher priority.

CIT 593

R-52

Chp 10: Handling Interrupts

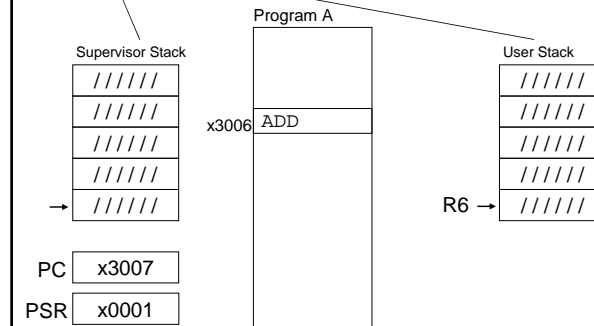
- Examines INT (interrupt) signal just before starting FETCH phase
 - If INT=1, don't fetch next instruction
 - Instead
 - Save state (PC, PSR (privilege and CCs)) on Supervisor stack
 - Update PSR (set privilege bit i.e. PSR[15] = 1)
- | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P | | | | | | PL | | | | | | N | Z | P | |
- Index INTV into IVT to get start address of ISR (put in PC)
 - After service routine**
 - RTI (Return from Interrupt) instruction restores PSR and PC from Supervisor stack
 - Processor only checks between STORE and FETCH phases -- Why?

CIT 593

R-53

Chp 10: Example of Interrupt

Different sections in memory

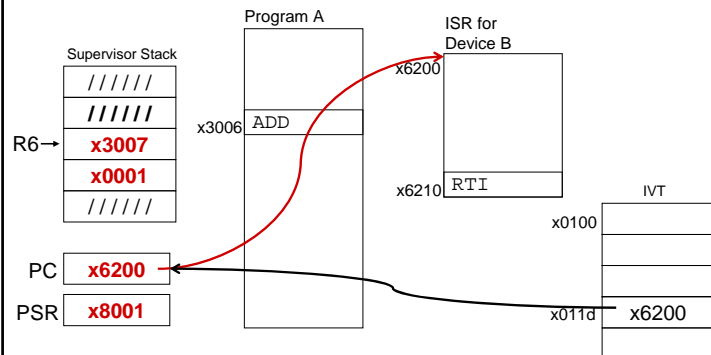


Executing ADD at location x3006 when Device B interrupts (INTV = x1d)

CIT 593

R-54

Chp 10: Example (contd..)

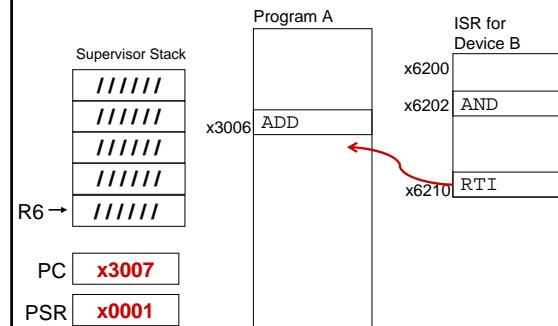


Push PC and PSR onto stack, set privilege bit, find Device B (INTV=x1d) service routine address in IVT, and transfer control

CIT 593

R-55

Chp 10: Example (contd..)



Execute RTI at x6210; pop PSR and PC from stack; continue Program A as if nothing happened!

CIT 593

R-56