ECE 220

Lecture x0000 - 01/20

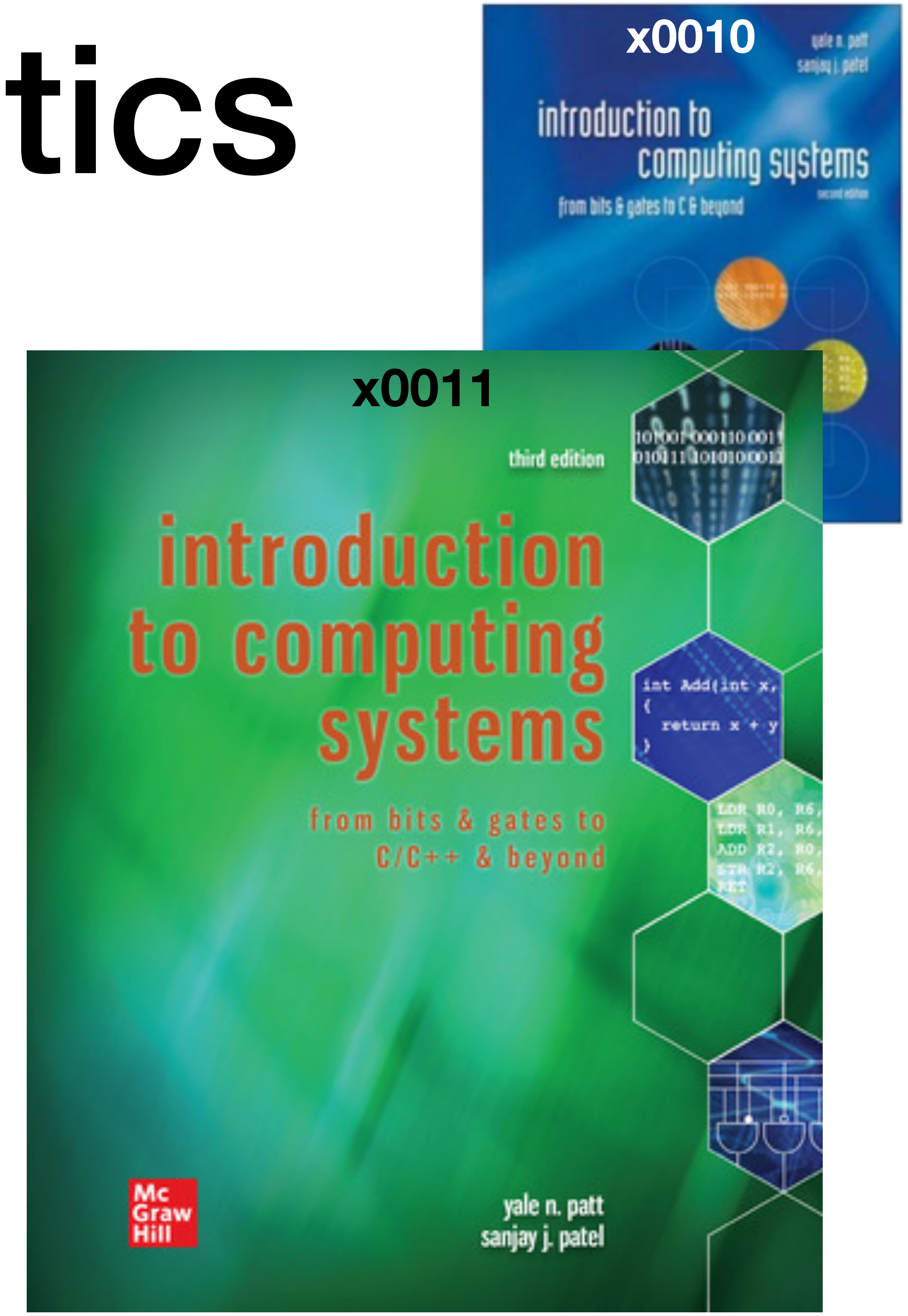
Slides based on material by: Yuting Chen, Yih-Chun Hu & Ujjal Bhowmik

Course logistics

- Lectures: Tuesdays & Thursday
  - Three sections offered by different instructors
  - This one (1500, BL3), Prof. Yuting Chen (1100, BL2) and Prof. Hu (1230, BL).
- Labs: Fridays
  - Starts on the hour, every hour from 0800 hrs until 1750 hrs
- Office hours: Schedule TBD, will be posted to website

Course logistics

- [Course Website](#) (and syllabus)
- **Grading**: Gradescope + autograder
- **Discussions**: EdStem
- **Quizzes**: CBTF
- **Machine problems** (MPs): Github
- **Textbook**: Patt & Patel (3rd Ed)



Course logistics

- **MPs**: 12 in total, lowest dropped (except MP 12)
- **Quizzes** (in-person in CBTF): 6 total, lowest dropped
- **Exams** (in-person, on-paper): 02/26 and 04/02
- **Labs**: make up points lost on MPs

Group	Weight
Labs	0%
Machine Problems	15%
Midterms	40%
Final Exam	25%
Quizzes	20%
Total	**100%**

Syllabus review

Quick recap of ECE 120

Lesson objectives

- Recall Von Neumann model of computation
- Recall LC3 basics
    - Number of GPRs, types of commands, addressing, etc.
    - Write a small LC3 program
- Understand memory mapped I/O
    - Handshaking procedures
    - Implementation using LC3

Computation

Von Neumann model

- Five major components:
    1.
    2.
    3.
    4.
    5.

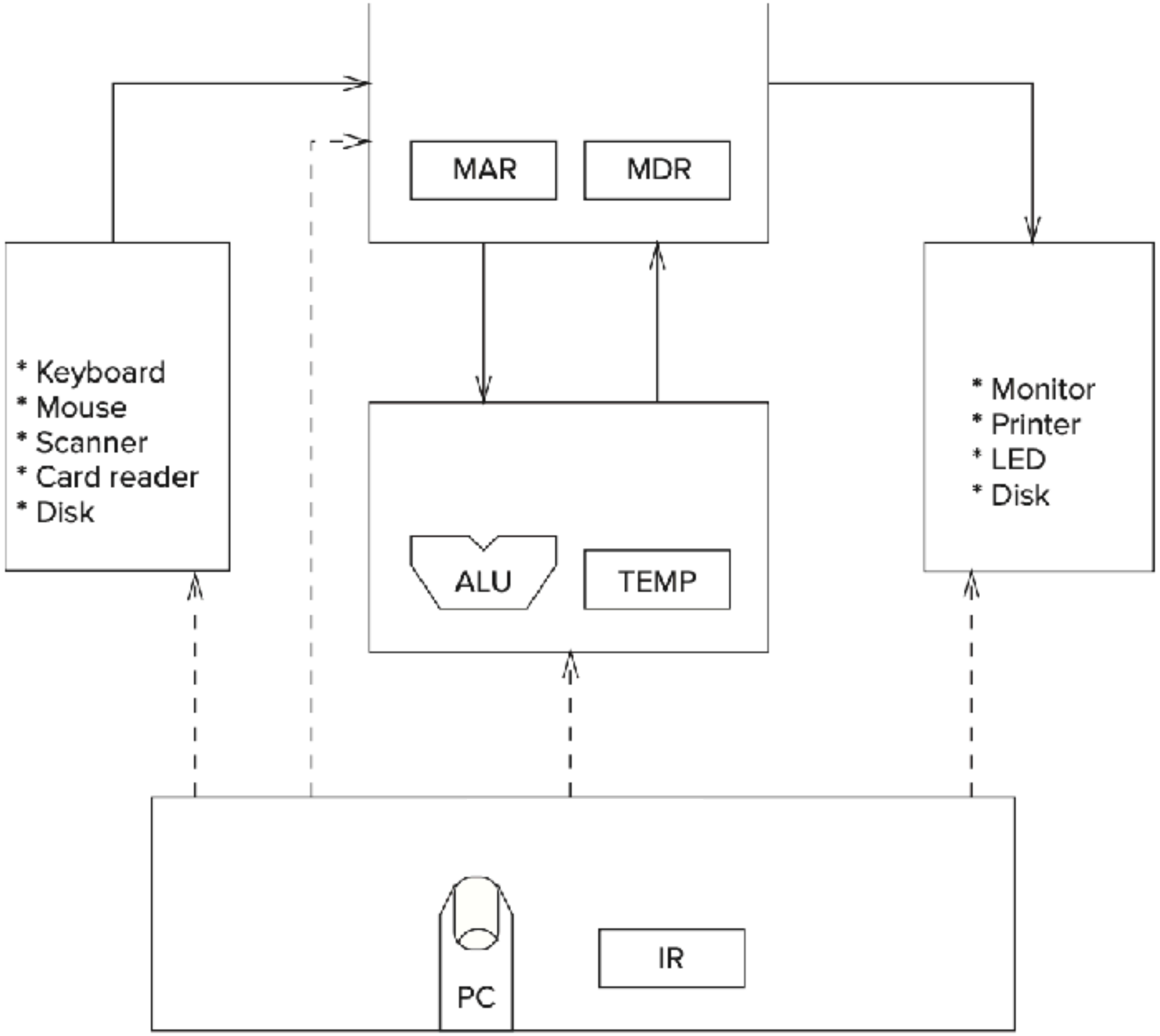


Figure 4.1 - P&P 3rd Ed.

LC3 Review

- Eight GPRs - denoted ____________________.
- Data type: 16-bit 2's complement integers
- Addressing: Locations ________________ contain 16 bits each.
- Addressing modes:
  - Immediate, register relative, PC-relative, base + offset, indirect

LC3 - Review

Addressing modes

- **PC relative**, the address is calculated by adding an offset to the incremented program counter, PC.
- **Register relative**, address is read from a register.
- **Indirect**, address is read ***from*** a memory location whose address is calculated by adding an offset to the incremented program counter.
- **Load effective address** (LEA), address is calculated by adding an offset to the incremented program counter. The address itself (not its value) is stored in a register.

LC3 - Review

Addressing modes

Sign-extend (SX), by replicating the most significant bit as many times as necessary to extend to the word size of 16 bits.

Opcode	Name	Assembly	Operation
LD	Load	LD DR, label	dr = mem[pc + SX(offset9)]
LDR	Load Register	LDR DR, BaseR, offset6	dr = mem[baseR + SX(offset6)]
LDI	Load Indirect	LDI DR, label	dr = mem[mem[pc + SX(offset9)]]
LEA	Load Eff. Addr.	LEA DR, target	dr = pc + SX(offset9)
ST	Store	ST SR, label	mem[pc + SX(offset9)] = sr
STR	Store Register	STR SR, BaseR, offset6	mem[baseR + SX(offset6)] = sr
STI	Store Indirect	STI SR, label	mem[mem[pc + SX(offset9)]] = sr

LC3 - Review

Instruction set

Instruction	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			
LDI$^+$	1010	DR	PCoffset9			
LDR$^+$	0110	DR	BaseR	offset6		
LEA	1110	DR	PCoffset9			

Set condition codes

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

Figure A.2 - P&P 3rd Ed.

Operation | Data movement | Program flow

Exercise

```
.ORIG x3000
 LD  R1, LABEL
 LDI R2, LABEL
 LDR R3, R2, #1
 LEA R4, LABEL
 LABEL .FILL x4001
.END
```

What are the values of `R1`, `R2`, `R3` & `R4` at each step?

Assume

```
; x4001 x6001
; ......
; x6001 x7001
; x6002 x7002
```

Exercise

- Write a program to perform the multiplication 5 x 4.
  - Need a way to store 5 and 4 as arguments
  - There is no multiplication operation
    - So have to repeat addition

```
.ORIG x3000
; R0 - output, init to 0
; R1 - multiplicand 1, init to 5
; R2 - loop counter, init to multiplicand 2
```

LC3 - Review

Pseudo-ops

- Looks like instruction but the "opcode" starts with a dot.
- Assembler instructions/directives that make our lives easier.

Opcode	*Operand*	*Meaning*
.ORIG	address	Starting address of program
.END		End of program
.BLKW	n	Allocate n words of storage
.STRINGZ	n-character string	Allocate n+1 locations, initialize with characters and null terminator

Textbook v2 vs. v3

- What is different in v3 compared to v2?
  - `LEA` no longer sets condition codes
  - `TRAP` instructions do not store linkage in `R7`

  This probably doesn't mean much to you right now …

- What does that mean for you?
  - Do MPS on EWS machines!
  - Practice for the quiz on the online simulator: https://courses.grainger.illinois.edu/ece220/sp2020/lc3web/index.html

Memory mapped I/O

- How do we communicate with the computer?
- **Memory-mapped I/O**: Hardware devices (i.e. their registers) are *treated* the same as the computer's main memory and addressable the same way
  - Memory of *peripherals* **is** *physically separate* from main memory
- Alternative: Port mapped I/O (older paradigm, requires having more specialized instructions)
- In LC3: `KBDR,KBSR,DSR,DDR` are used for [K]eyboard and [D]isplay respectively.

LC3 - Input/Output (IO)

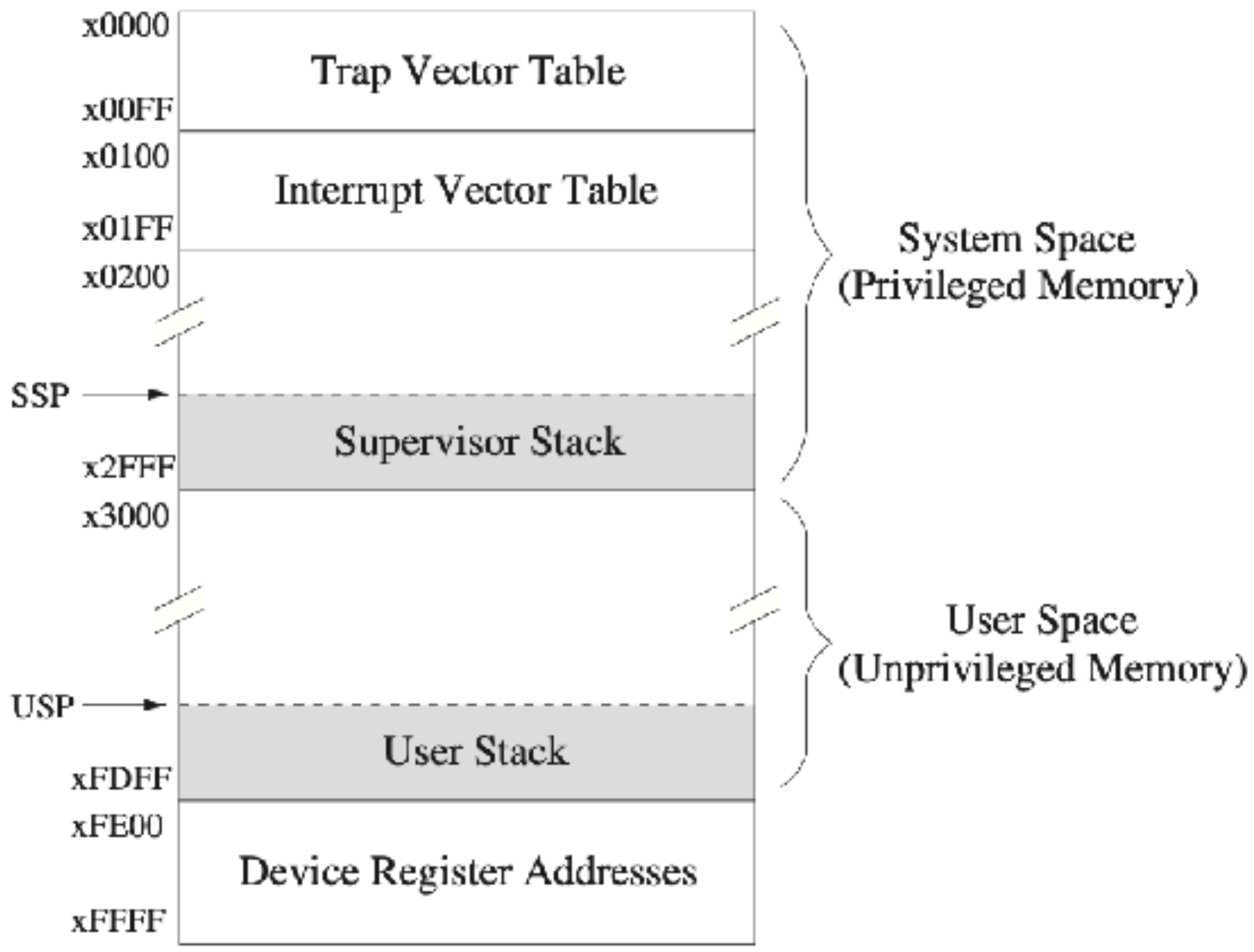
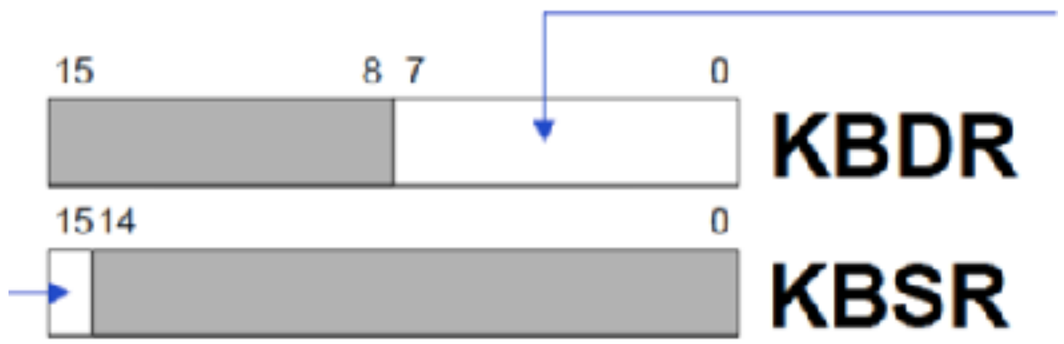
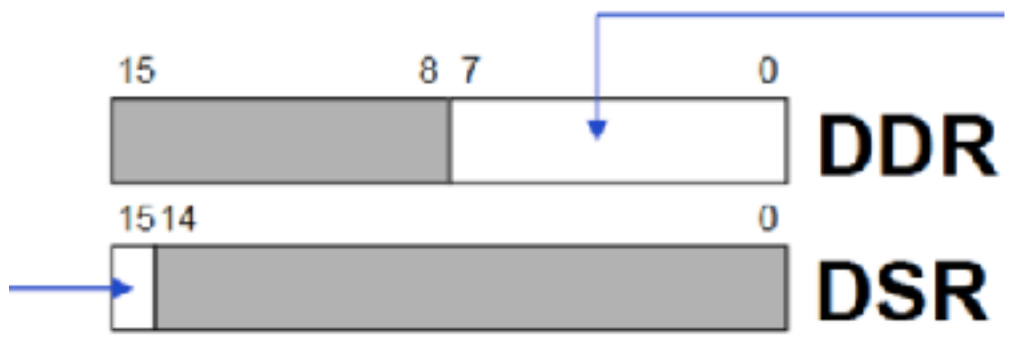


Figure A.1 - P&P 3rd Ed.

Address	I/O Register Name	I/O Register Function
xFE00	Keyboard status register (KBSR)	The ready bit (bit[15]) indicates if the keyboard has received a new character
xFE02	Keyboard data register (KBDR)	Bits [7:0] contain the last character typed on the keyboard
xFE04	Display status register (DSR)	The ready bit (bit[15]) indicates if the display device is ready to receive another character to print on the screen
xFE06	Display data register (DDR)	A character written in bits [7:0] will be displayed displayed on the screen

LC3 - Input/Output (IO)





Address	I/O Register Name	I/O Register Function
xFE00	Keyboard status register (KBSR)	The ready bit (bit[15]) indicates if the keyboard has received a new character
xFE02	Keyboard data register (KBDR)	Bits [7:0] contain the last character typed on the keyboard
xFE04	Display status register (DSR)	The ready bit (bit[15]) indicates if the display device is ready to receive another character to print on the screen
xFE06	Display data register (DDR)	A character written in bits [7:0] will be displayed displayed on the screen

LC3 - Input from keyboard

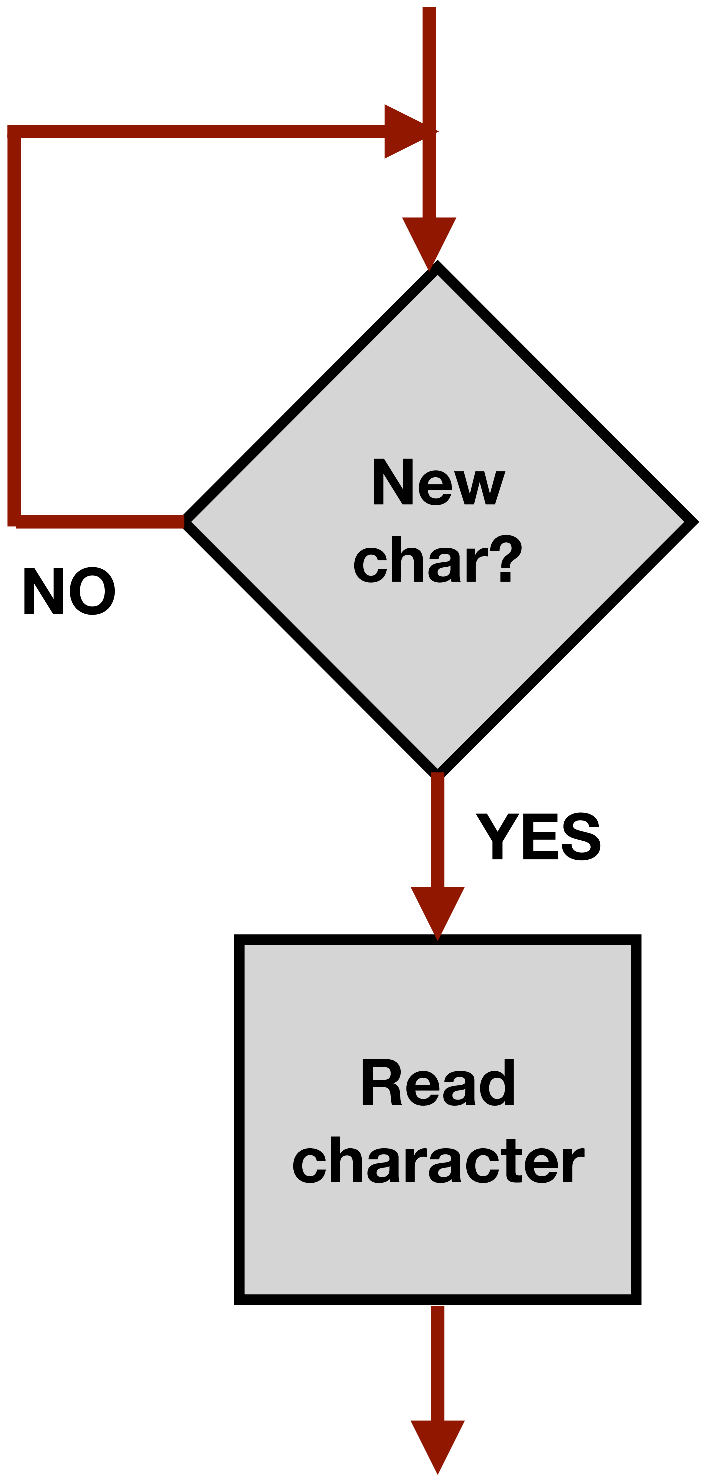
Basic routine

Handshaking is performed using `KBSR` & `KBDR`

- When user presses a key
  - Its ASCII code is placed in `KBDR[0:7]`
  - `KBSR[15]` is set to 1 *(ready bit)*
  - Keyboard is disabled, i.e., any further keypress is ignored
- When `KBDR` is read by CPU
  - `KBSR[15]` is set to 0
  - Keyboard is enabled

LC3 - Input from keyboard

Basic routine




```
.ORIG x3000
;Create a loop to
check KBSR

;If ready bit unset
loop again

;If ready bit set,
read KBDR into R0

KBSR .FILL xFE00
KBDR .FILL xFE02
```

```
.ORIG x3000
KPOLL LDI R1, KBSR
  BRzp KPOLL
  LDI R0, KBDR
;…
;…
HALT

KBSR .FILL xFE04
KBDR .FILL xFE06
.END
```

LC3 - Display to console

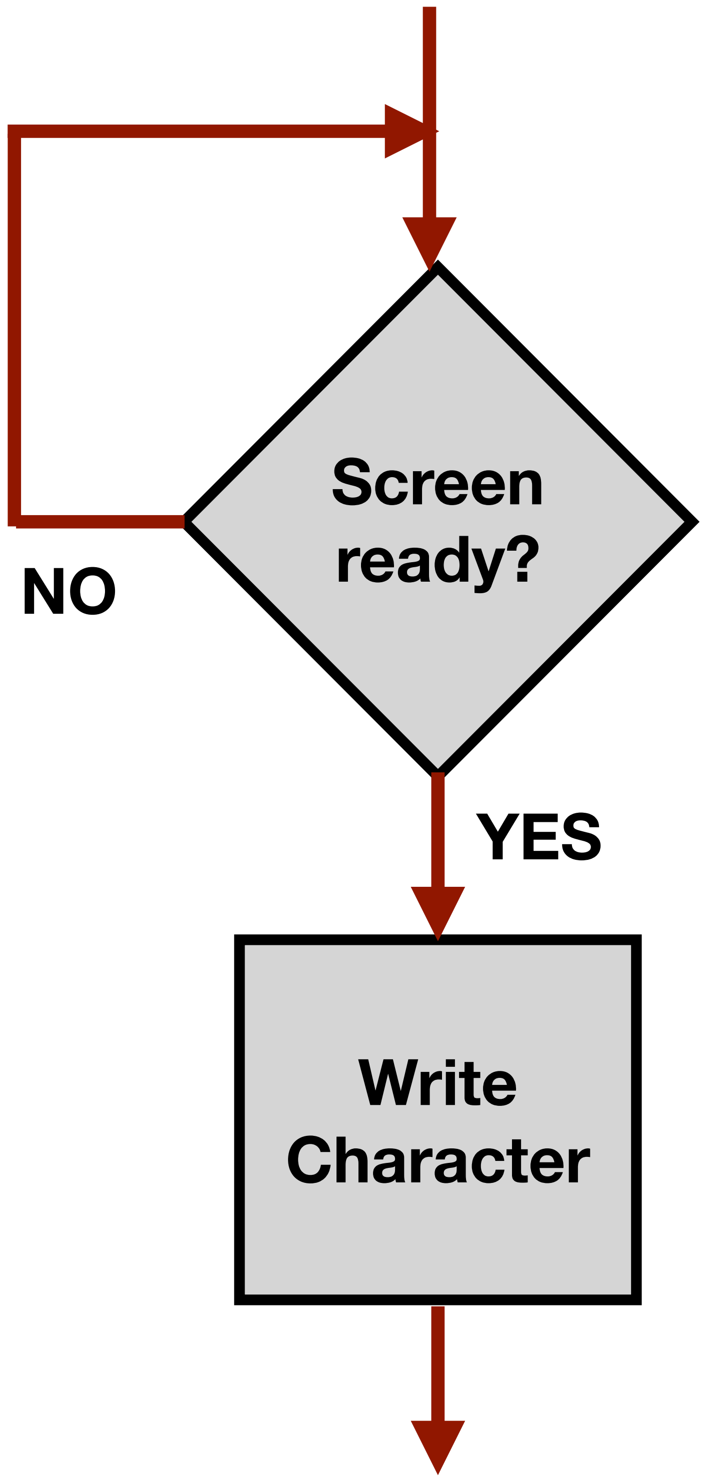
Basic routine

Handshaking is performed using `DSR` & `DDR`

- When display is ready to present a character
  - `DSR[15]` is set to 1 *(ready bit)*
- When a new character is written to `DDR`
  - `DSR[15]` is set to 0
  - Any other chars written to `DDR` are ignored
  - `DDR[7:0]` is displayed

LC3 - Display to console

Basic routine




```
.ORIG x3000
;Create a loop to
check DSR

;If ready bit unset
loop again

;If ready bit set,
write R0 into DDR

DSR .FILL xFE04
DDR .FILL xFE06
```

```
.ORIG x3000
DPOLL LDI R1, DSR
  BRzp DPOLL
  STI R0, DDR
;…
;…
HALT

DSR .FILL xFE00
DDR .FILL xFE02
.END
```

Exercise

- Write a program to display "ECE 220 is fun!" to the console. You can use the pseudo-op `.STRINGZ` to store string to memory. Do not use `TRAP` codes (*if you know what they are*).

Issues?

- Consider "echo" routine:

```
KPOLL   LDI     R1, KBSR
        BRzp    KPOLL
        LDI     R0, KBDR
```

```
DPOLL   LDI     R1, DSR
        BRzp    DPOLL
        STI     R0, DDR
```

```
        BRnzp   NEXT_TASK
KBSR    .FILL   xFE00
KBDR    .FILL   xFE02
DSR     .FILL   xFE04
DDR     .FILL   xFE06
```

- Reading & writing from keyboard or display is common task
  - Inefficient to keep repeating this code
  - Need to free up `R1` and `R0` for use whenever blocks run
    - Save/restore current values before/after these blocks run

Recap from last time

- Consider "echo" routine:

```
;SAVE R0, R1
KPOLL   LDI     R1, KBSR
        BRzp    KPOLL
        LDI     R0, KBDR
;RESTORE R0, R1
```

```
;SAVE R0, R1
DPOLL   LDI     R1, DSR
        BRzp    DPOLL
        STI     R0, DDR
;RESTORE R0, R1
```

```
        BRnzp   NEXT_TASK
KBSR    .FILL   xFE00
KBDR    .FILL   xFE02
DSR     .FILL   xFE04
DDR     .FILL   xFE06
```

- Reading & writing from keyboard or display is common task
  - Inefficient to keep repeating this code
  - Need to free up `R1` and `R0` for use whenever blocks run
    - Save/restore current values before/after these blocks run

Repeating code

- Consider $f(x) = x^4 + 4x^3 + 3x^2 + 2x + 1$
- Evaluate $f(2)$
  - How many multiplications?

Repeating code

- Consider $f(x) = x^4 + 4x^3 + 3x^2 + 2x + 1$
- Evaluate $f(2)$
  - How many multiplications?
- Suppose we wish to evaluate $f(x)$ for many values of x
  - Why? E.g. [Newton-Raphson] method for finding roots of $f(x)$

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Issues?

- Limited amount of GPRs - polling display & keyboard uses up two of them

- Code often repeated - inefficient to keep inserting same code over & over again

- Human error - keeping track of registers & having direct access to hardware registers is recipe for unforced errors & bugs

Solution?

- Subroutines & repeated code
  - Also called *functions*
- TRAP routines
- More next time ...