University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 220: Computer Systems and Programming

Instructor: Ujjal Kumar Bhowmik
Section: BL3, 2:00-3:20PM, ECEB 2017
Office Hours: 5-6PM, Monday, ECEB 2054
Course Website:
https://courses.grainger.illinois.edu/ece220/fa2025/

ECE220: Introduction to Computing

Instructors	T/R	Section:
Prof. Chen	12:30 p.m.	BL
Prof. Abraham	11:00 a.m.	BL2
Prof. Bhowmik	2:00 p.m.	BL3

ECE220 – Course Objectives

- Understand the low-level concepts such as I/O, subroutine, stack in LC-3
- Understand the basic data organization such as array, pointer, functions, recursion, simple data structure, linked-list, tree and how they are laid out in memory.
- Be able to write C program to accomplish simple task and be familiar with testing and debugging of simple programs.
- Understand the basics of C++

Course Overview:

- Programming Studio (or Labs) on Fridays (10 makeup pts/lab worksheet towards MPs, except MP12)
- MPs: due every Thursday by 10:00 PM (100 pts each, late penalty 2pts/hour) – 15%
- Quizzes: 6 programming quizzes on CBTF (50-minute), lowest score dropped – 20%
- Exams: 2 midterms and a final Exam (paper format) – 40% + 25%
- Textbook: Patt & Patel, Introduction to Computing Systems: from bits to gates to C/C++ and beyond, 2nd or 3rd Edition.
- Academic Integrity

Grading Policy

Grading Mechanics:

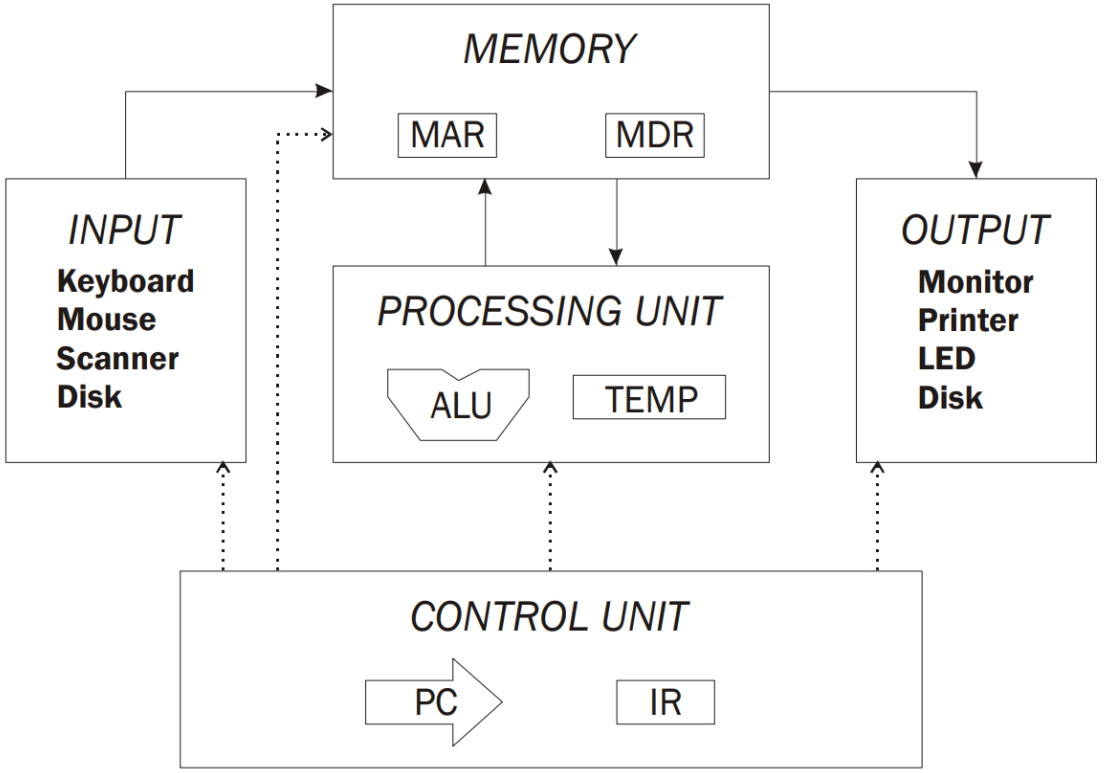
- **Programming Assignments (12 MPs. Lowest grade dropped, excluding MP12):** 15%
- **Quizzes (6 quizzes, lowest grade dropped)**: 20%
- **Midterm 1:** 20%
- **Midterm 2**: 20%
- **Final Exam:** 25%

Course Logistics & Tools

- Course Web page: course info, MP write-up, exam info, etc.
- Github: MP/LAB release and submission
- Gradescope: lab worksheets
- Campuswire: discussion board
- CBTF: Quiz proctoring
- Resources: CARE, counseling center, DRES

LC3 Review – Von Neumann Model

1. Memory
2. Processing Unit
3. Input
4. Output
5. Control Unit



LC3 Instruction Set Architecture (ISA)

Instruction Set

Data Types: 16-bit 2's complement integers

Addressing Modes (how the location of operand is specified):

Non-memory addresses – immediate (part of instruction), register

Memory address – PC-relative, base+offset, indirect

Opcodes (4-bit, bits 12-15 used to specify the opcode):

Operate instructions: ADD, AND, NOT

Data movement instructions: LD, LDI, LDR, LEA, ST, STR, STI

Control instructions: BR, JSR/JSRR, JMP, RET, TRAP, RTI

Condition codes: N (negative), Z (zero), P (positive)

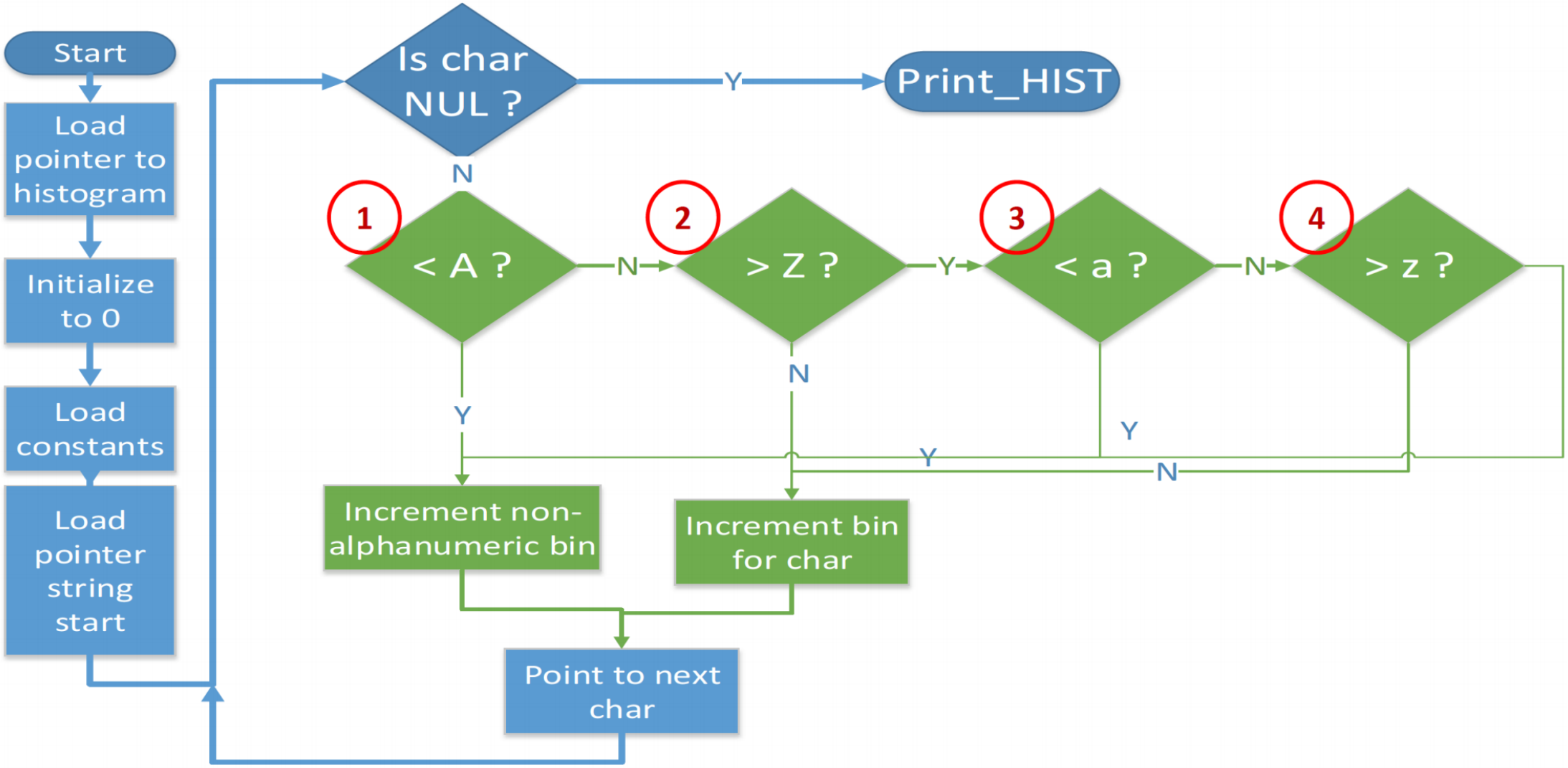
Review: LD, LDI, LDR, and LEA

```
.ORIG x3000
LD      R6, LABEL
LDI     R6, LABEL
LDR     R2, R6, #1
LEA     R2, LABEL
LABEL .FILL x4000
.END

; Assume the following
; Address       Data
; x4000         x5000
;    …             …
; x5000         x6000
; x5001         x6001
```

Review:

- Initialize a register to zero:
- Copy the content of one register to another:
- Perform Subtractions (8-3):
- Perform Multiplication (5x4):

MP1 – Printing a Histogram



Display the content of an LC3 register in HEX

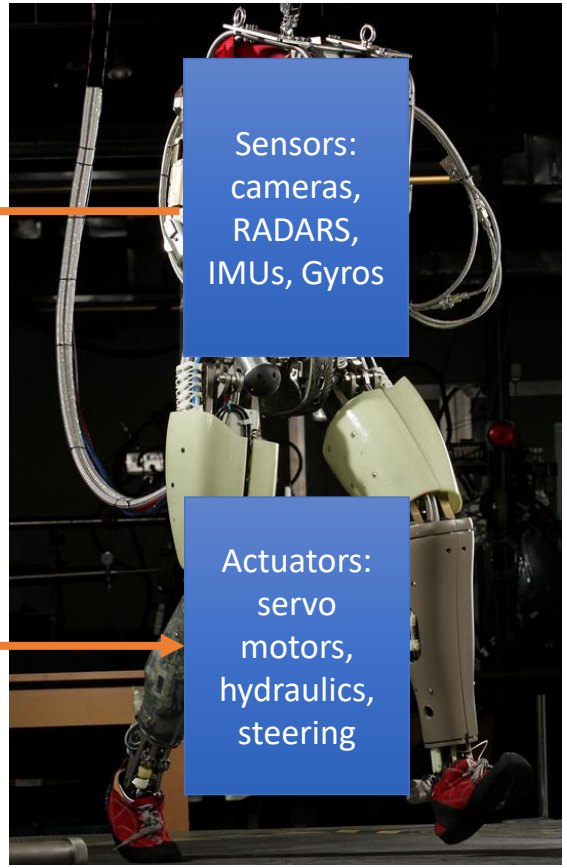
Outline

- Section 9.2 (3rd Ed.), 8.1-8.4 (2nd Ed.) of Patt and Patel
- I/O principles
- Input from keyboard
- Output to monitor

- Key concepts
  - Memory mapped I/O
  - Asynchronous and synchronous communication

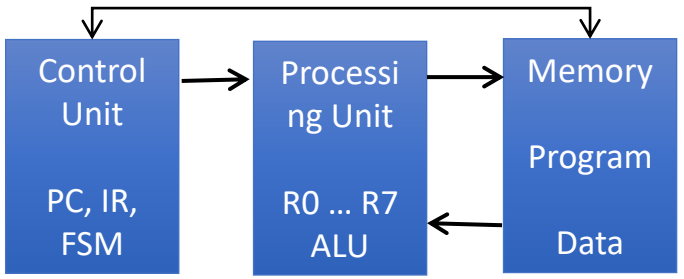
Humanoid Robot

I/O with the physical world



CPU & Memory
Controlling bipedal gait, Image processing, obstacle detecting, path planning, control, …

CPU and Memory



I/O and Basics of Interface Design

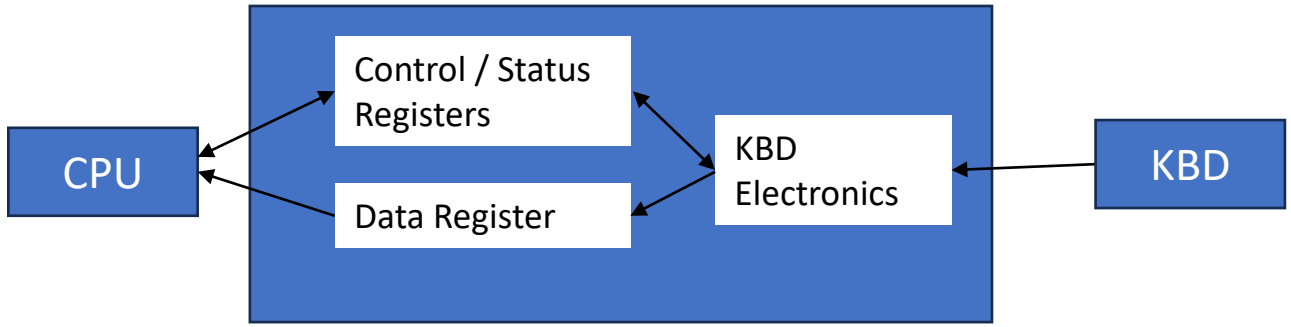
I/O is for interfacing the physical world and the digital world.

- Producer of data (sensors) is working much more slowly than consumer of that data (processor/program)
- We need to account for ***asynchronous*** operation
- We will use a simple consumer/producer handshake

For LC3 we just need to consider…. Input/output?

I/O Device Controller

Keyboard Interfacing:



Control/Status Register:

CPU tells the device what to do: **write** to control register (X)

CPU checks whether a new key is pressed: **read** the status register

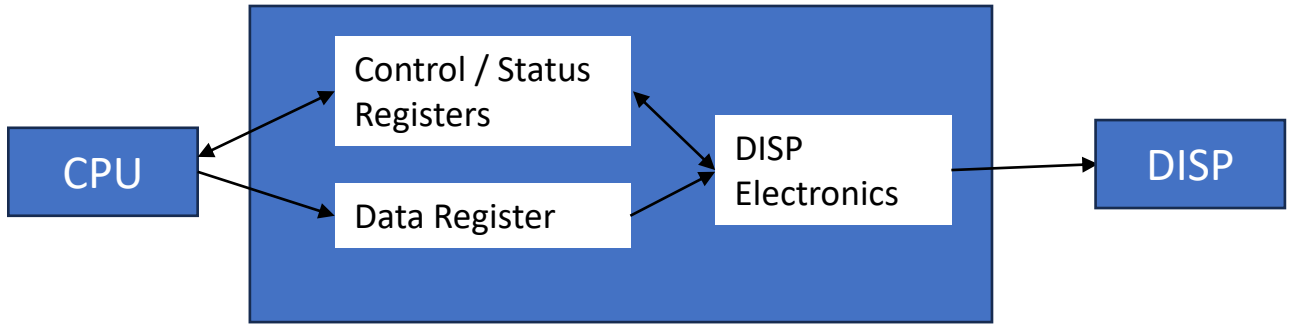
Data Register:

CPU reads the ASCII value of the key pressed

KBD Electronics: Performs actual operation (character from keyboard)

I/O Device Controller

Display Interfacing:



Control/Status Register:

CPU tells the device what to do: **write** to control register (X)

CPU checks whether a last character is displayed: **read** the status register

Data Register:

CPU sends the ASCII value of the character to be displayed

DISP Electronics: Performs actual operation (character to the screen)

Memory-Mapped I/O

- **Assign a memory address to each device register**
  - I/O device registers are mapped to set of addresses that are allocated to I/O device registers rather than to actual memory locations.
- **Use data movement instructions (load/store) for control and data transfer**

LC-3 Input and Output Device Registers

KBDR - store **ASCII value** of character entered from **keyboard**

KBSR - let processor know a new value is entered

DDR - store **ASCII value** of character to be displayed on **monitor**

DSR - let processor know a new value is ready to be displayed

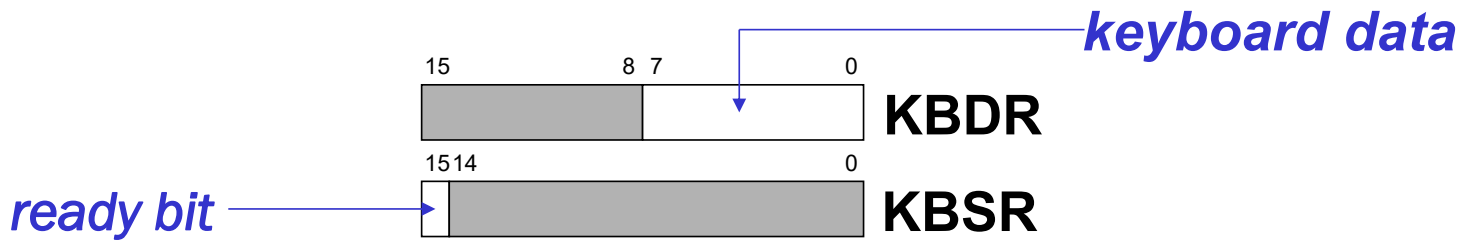
LC3 Memory: Memory mapped device registers

Address	Contents	Comments
x0000		;system space
…		
x3000		; user space
		; programs
		; and data
…		
xFE00	KBSR	; Device registers maps
xFE02	KBDR	
xFE04	DSR	
xFE06	DDR	
…		
xFFFF		

These are the memory addresses to which the device registers (KBDR, etc.) are mapped

The device registers physically are separate circuits from the memory

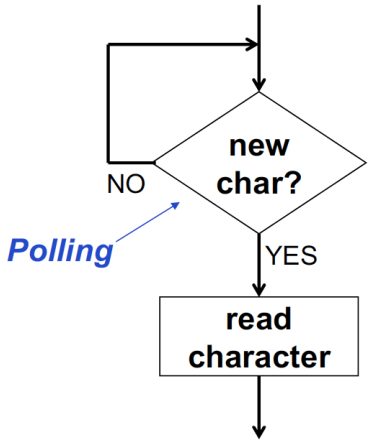
Handshaking using KBDR and KBSR



- When a char is typed by user in the keyboard
  - 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

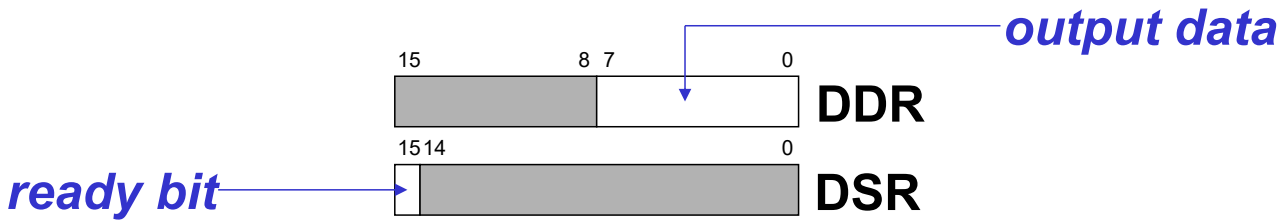
This is part of the keyboard Hardware.

LC-3 Basic Instructions to Read from the Keyboard




```
.ORIG x3000
;set up a loop to check ready bit in KBSR


;branch to the beginning if there is no KB input


;otherwise, load data from KBDR to R0



HALT
KBSR_ADDR .FILL xFE00
KBDR_ADDR .FILL xFE02
.END
```

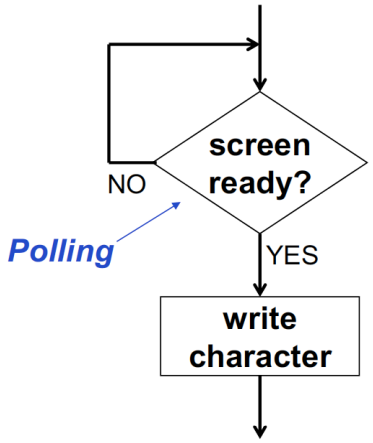
Handshaking using DDR and DSR



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

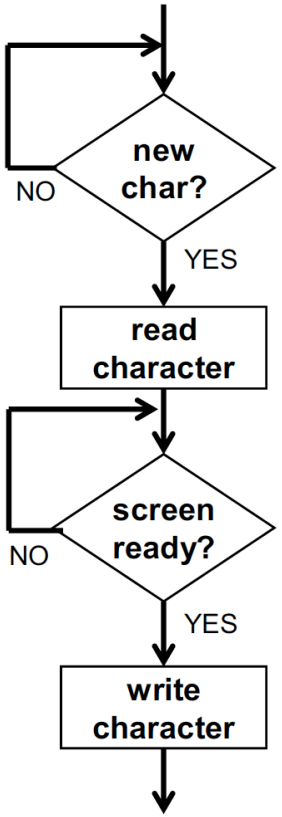
This is part of the display hardware.

Use LC3 LOAD/STORE Instructions to Display a Character to the Monitor




```
.ORIG x3000
;set up a loop to check ready bit in DSR


;branch to the beginning if display is
;not ready for new data


;otherwise, store data from R0 to DDR


HALT
DSR_ADDR .FILL xFE04
DDR_ADDR .FILL xFE06
.EN
```

Write code for ECHO (read a char and display it)

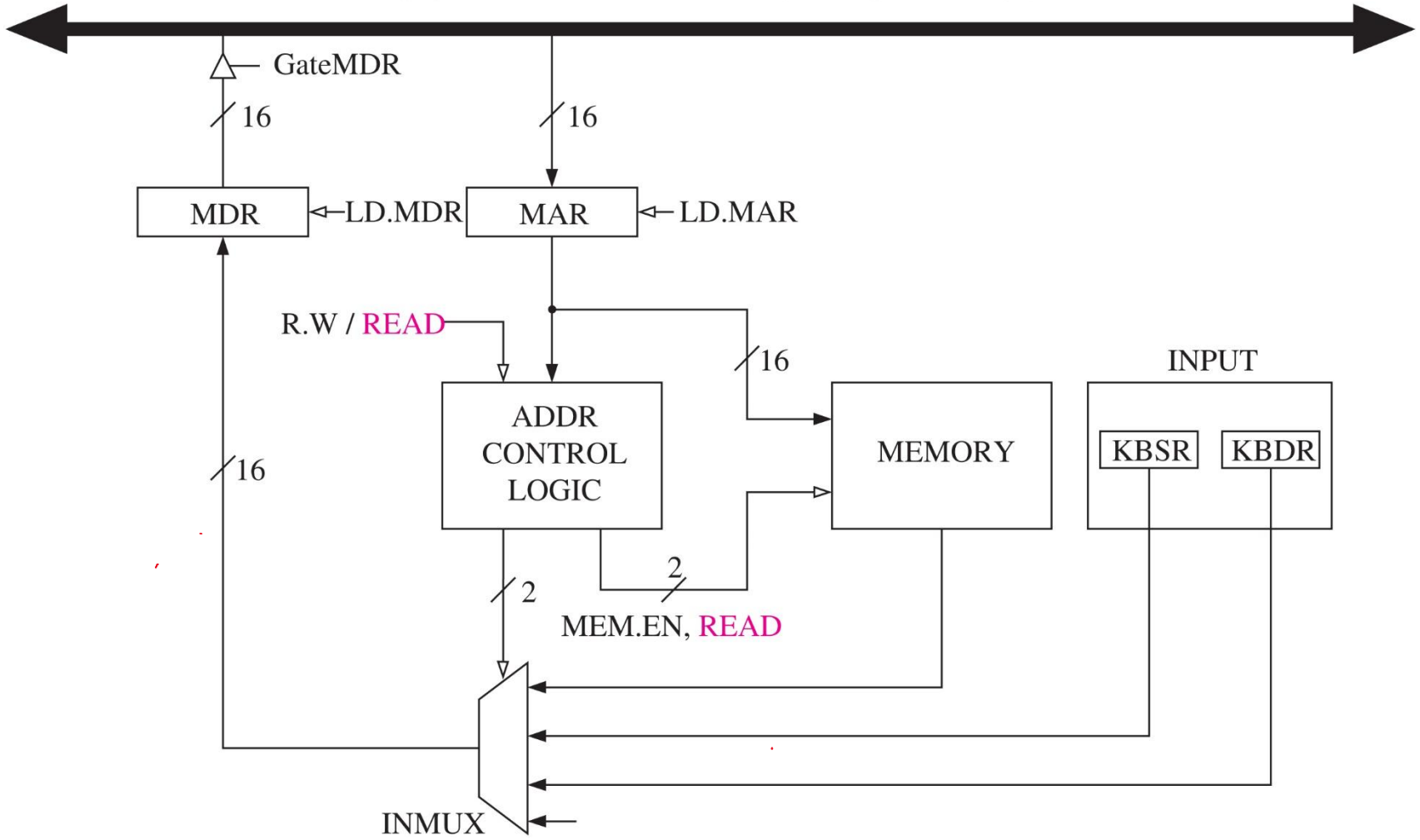



```
.ORIG x3000


HALT
KBSR_ADDR .FILL xFE00
KBDR_ADDR .FILL xFE02
DSR_ADDR .FILL xFE04
DDR_ADDR .FILL xFE06
.END
```

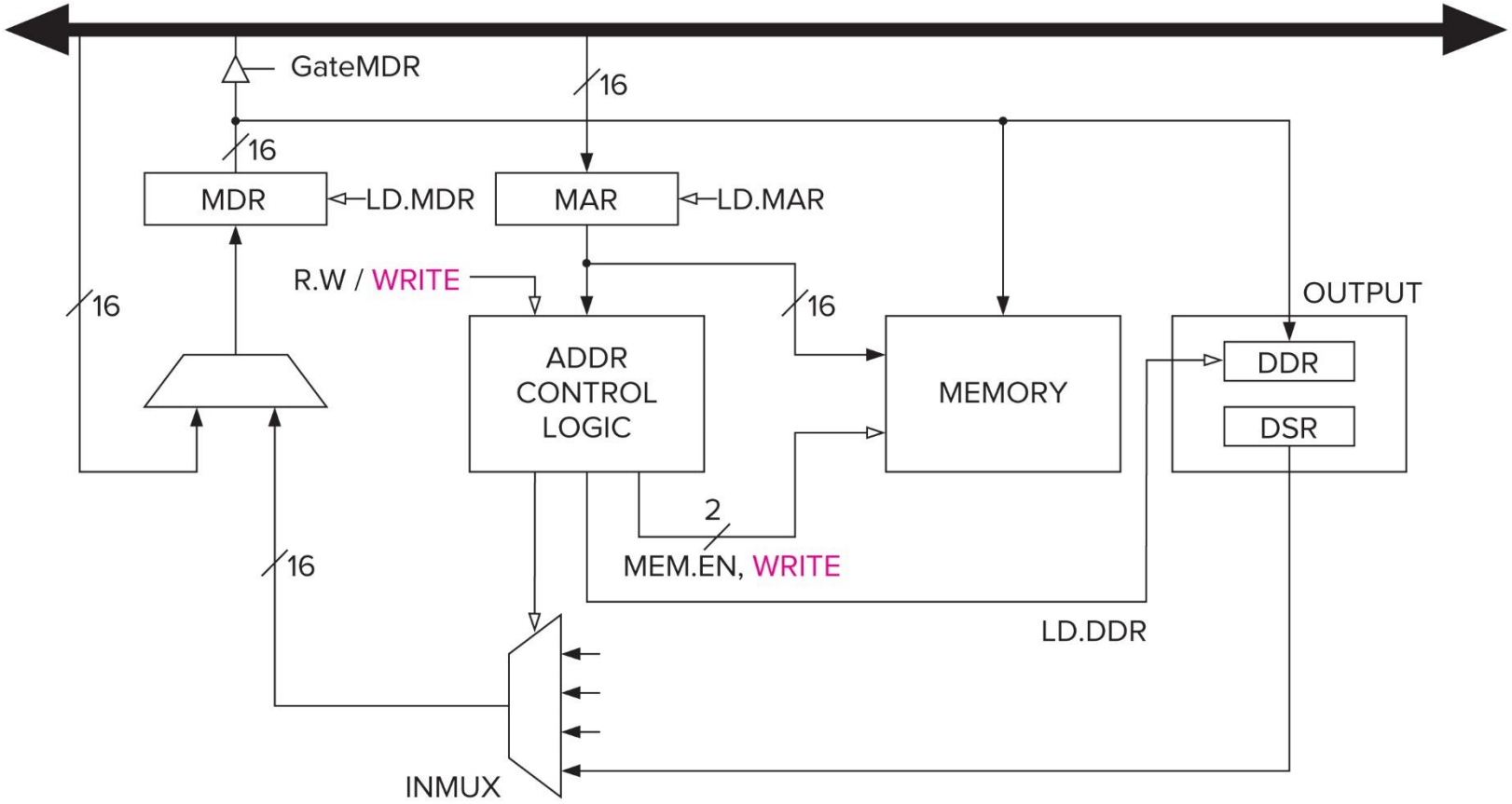
What does this code do?

```
.ORIG x3000

KPOLL   LDI R1, KBSR_ADDR
        BRzp KPOLL
        LDI R0, KBDR_ADDR
DPOLL   LDI R1, DSR_ADDR
        BRzp DPOLL
        STI R0, DDR_ADDR
HALT

KBSR_ADDR .FILL xFE00
KBDR_ADDR .FILL xFE02
DSR_ADDR  .FILL xFE04
DDR_ADDR  .FILL xFE06
.END
```

Simplified Memory-Mapped Input

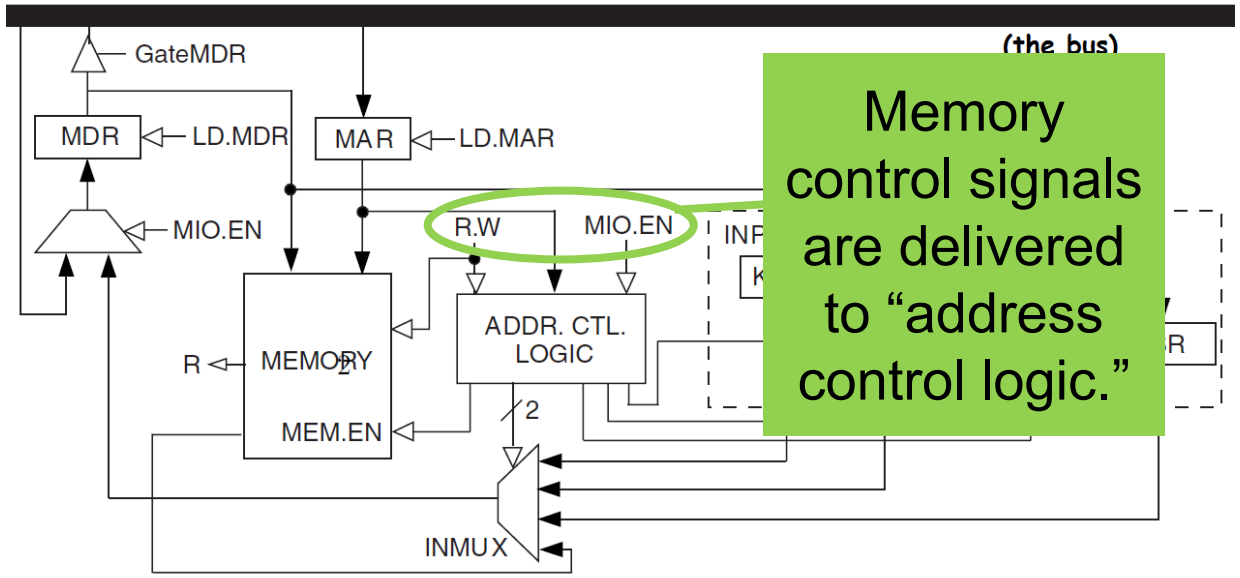


Simplified Memory-Mapped Output (monitor)

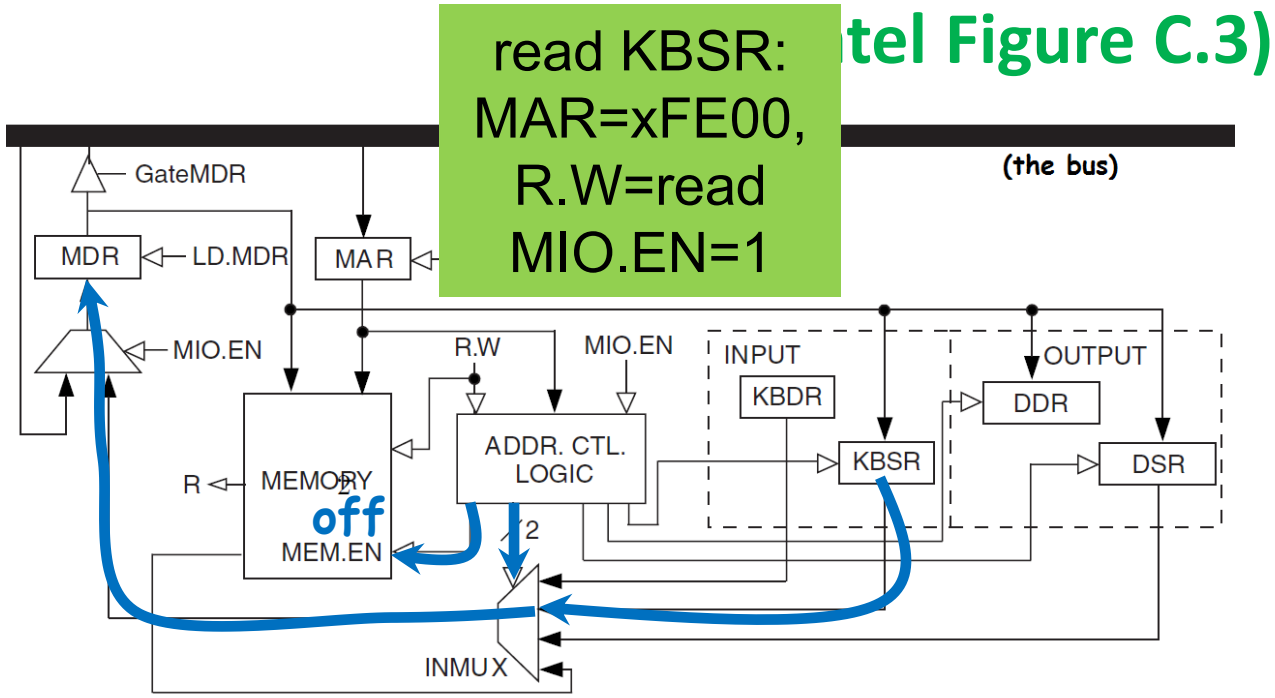


P&P Appendix C Describes I/O Memory Mapping

(Patt and Patel Figure C.3)

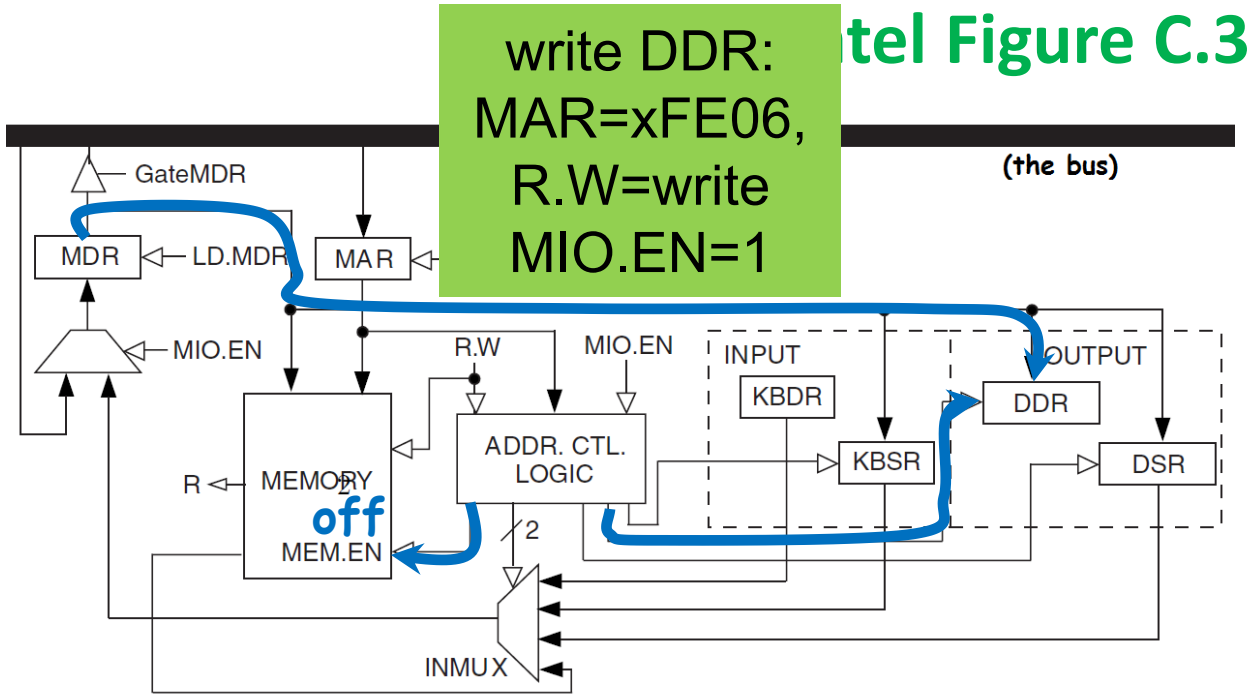


Example: Reading the KBSR



Example: Writing the DDR

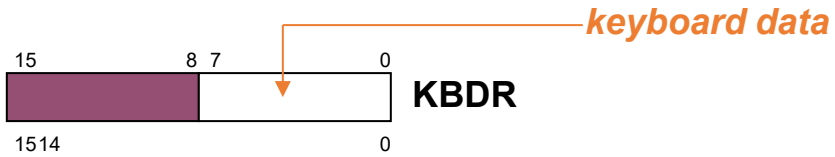
write DDR:
MAR=xFE06,
R.W=write
MIO.EN=1



Exercises

- Write code for ECHO (read a char and display it)
- Write code for PUTS (display a stored string)
- Write a more sophisticated input function using command prompt.

Reading Input (first attempt)

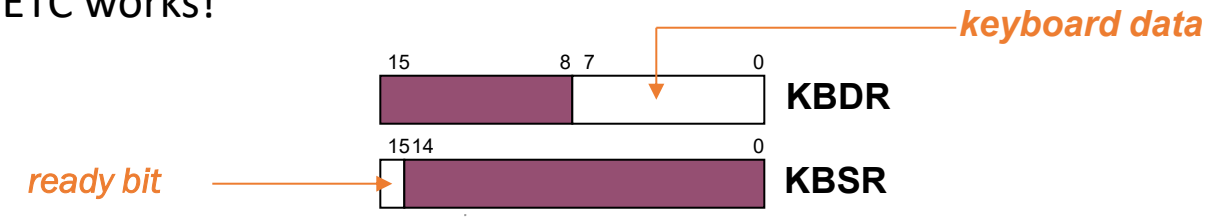



```
START           LDI     R1, KBDRAdd     ; Read from KBD
                ….
                BRnzp   START
KBDRAdd         .FILL      xFE02        ; Address of KBDR
```

Does this work?

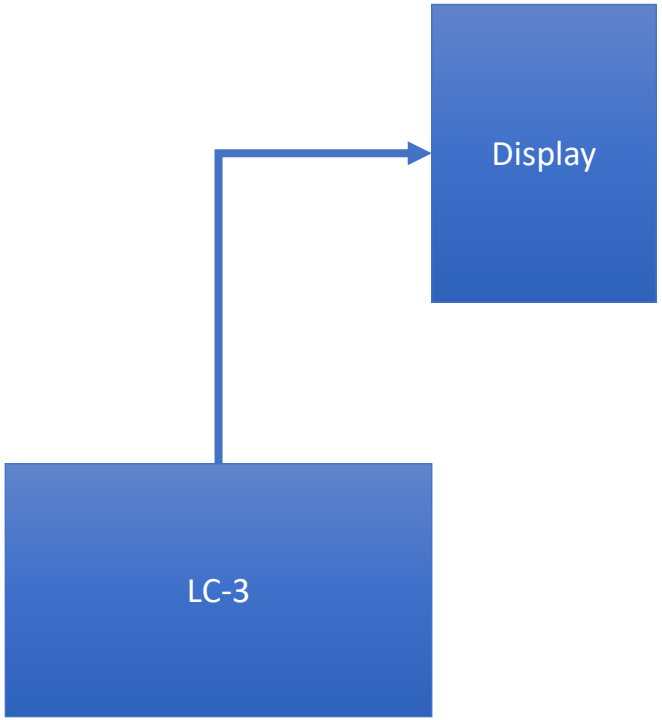
Reading Input the right way

This is how TRAP x20 = GETC works!




```
START           LDI     R1, KBSR_ADDR   ; Test for
                BRzp    START           ; character input
                LDI     R0, KBDR_ADDR
                BRnzp   NEXT_TASK       ; Go to the next task
                ...
KBSR_ADDR       .FILL     xFE00         ; Address of KBSR
KBDR_ADDR       .FILL     xFE02         ; Address of KBDR
```

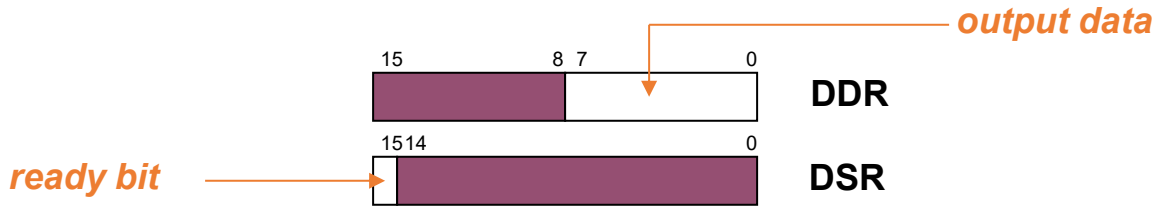
I/O Layout



- How to connect a display to LC3?

Writing TRAP x21

This is how TRAP x21 = OUT works!




```
START           LDI     R1, DSR_ADDR    ; Test for
                BRzp    START           ; character input
                STI     R0, DDR_ADDR
                BRnzp   NEXT_TASK       ; Go to the next task
                ...
DSR_ADDR        .FILL     xFE04         ; Address of DSR
DDR_ADDR        .FILL     xFE06         ; Address of DDR
```

Summary of concepts

- Memory mapped I/O (extra hardware for flexibility and convenience of programming)
- Asynchrony
- Polling