Computer Architecture and Assembly Language

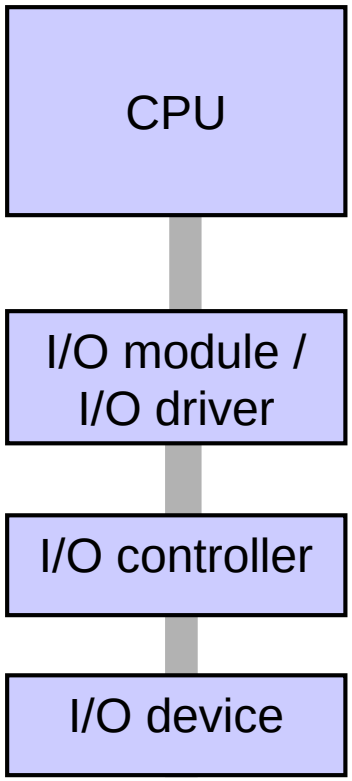
Practical Session 11

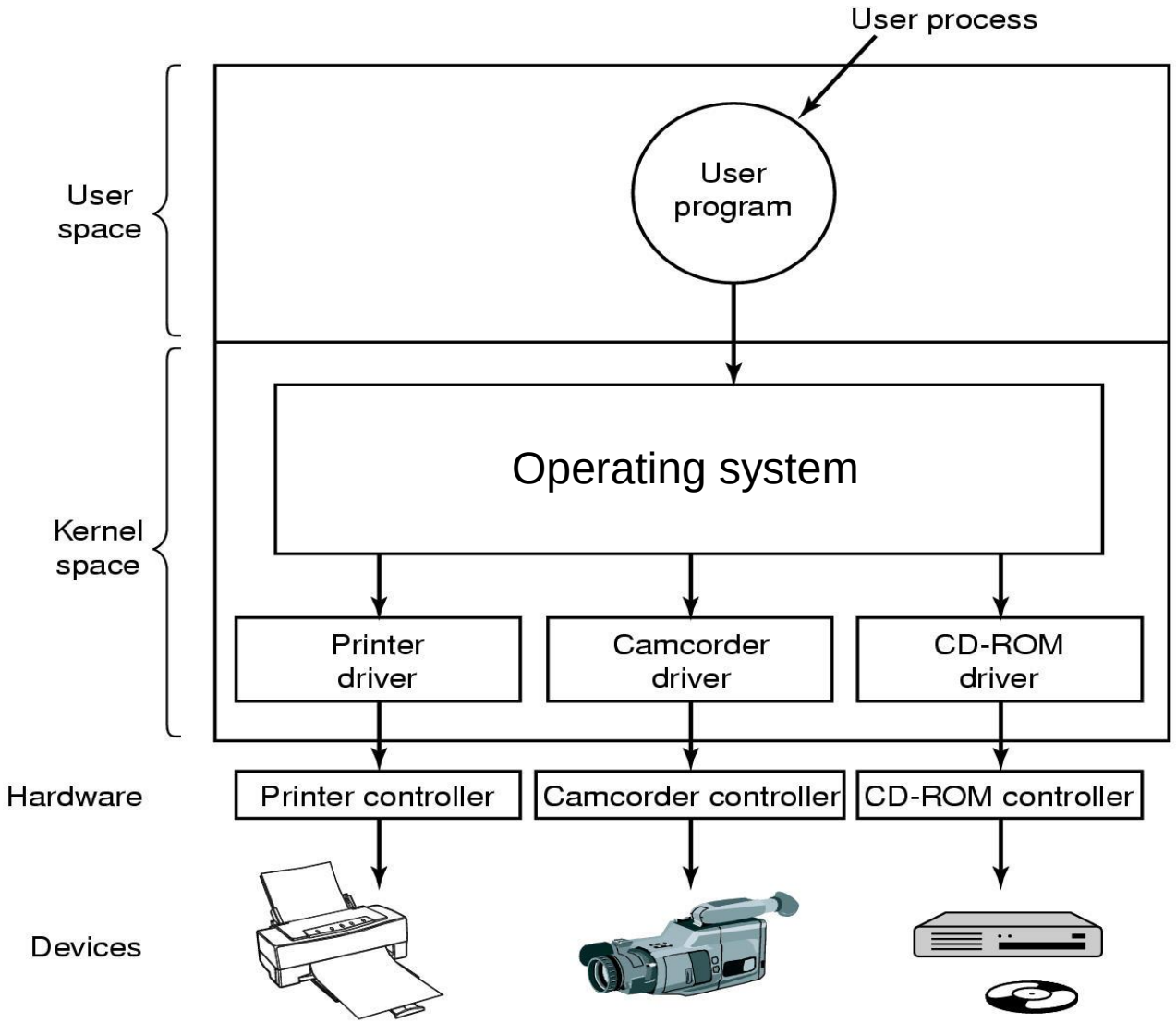
Input &Output (I/O)

CPU-Memory-I/O Architecture

Input / Output (I/O) devices are used to interact with "outside world"

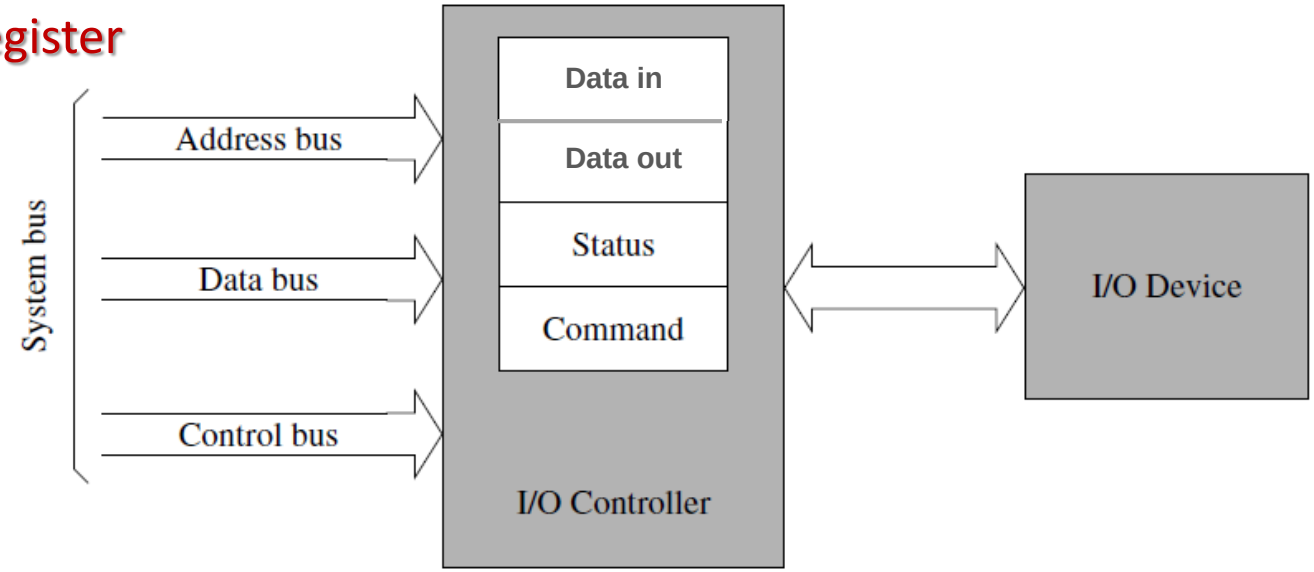
- mouse, screen, disks, printer …





I/O Controller

- Typically I/O Controller has 3 internal registers:
  - Data register
  - Status register
  - Command register



Example: character output

- Check **status** register of I/O controller
  - e.g. busy, idle, offline
- Put **data** into data in register
- Put command into **command** register
  - e.g. “send the character in data register to printer”

Programmed I/O

- Software controls I/O operations
  - slow ☹
  - busy CPU wait ☹
  - simple ☺

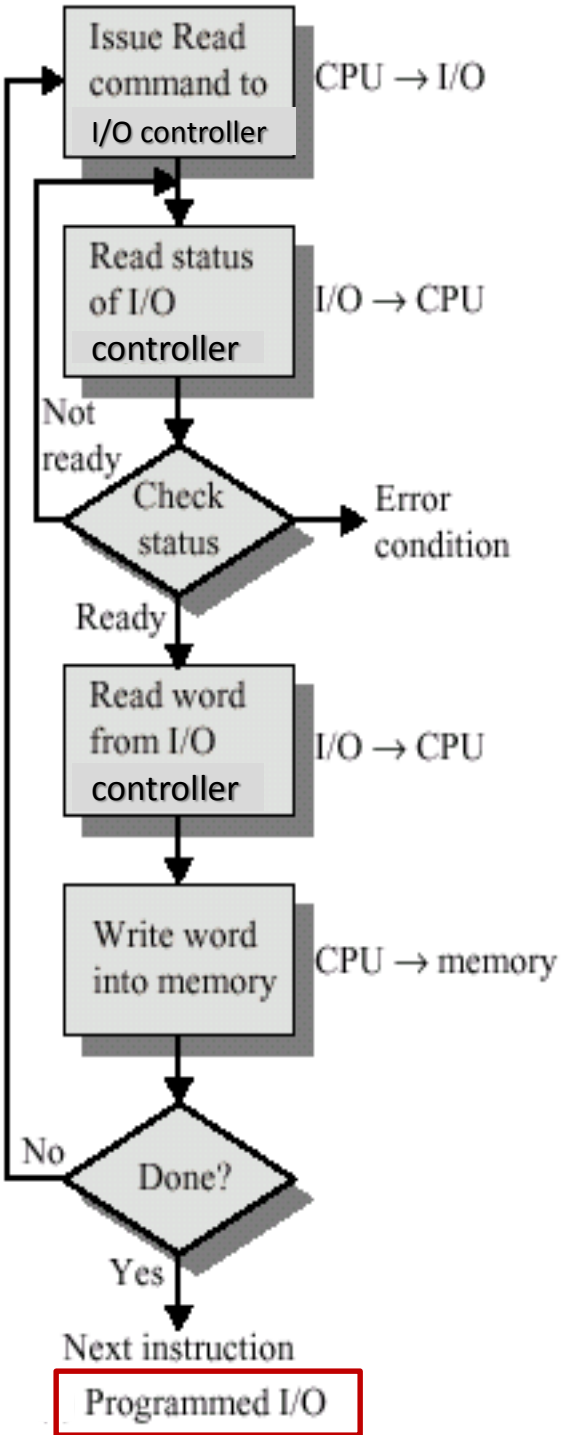
Interrupt-driven I/O

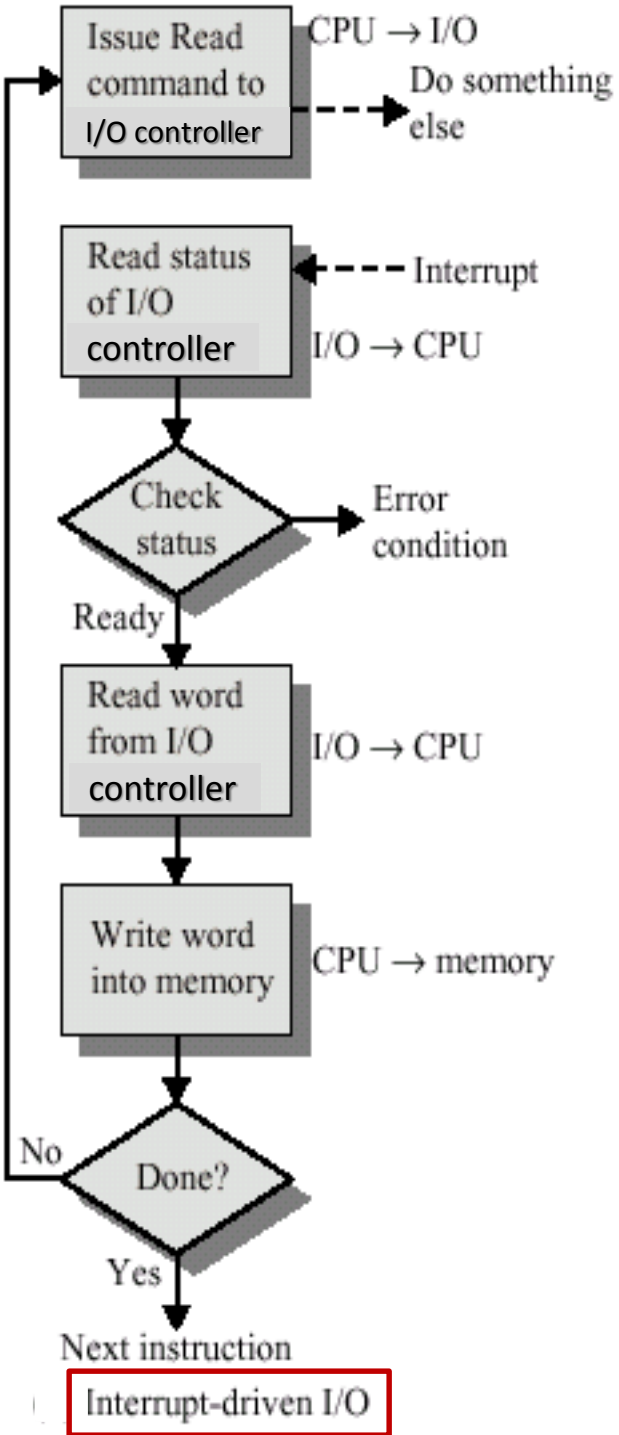
- Device includes a signal to interrupt CPU (interrupt is produced by hardware)
- When an interrupt occurs (and is accepted), a special routine executes to service the interrupt
  - no CPU busy wait ☺

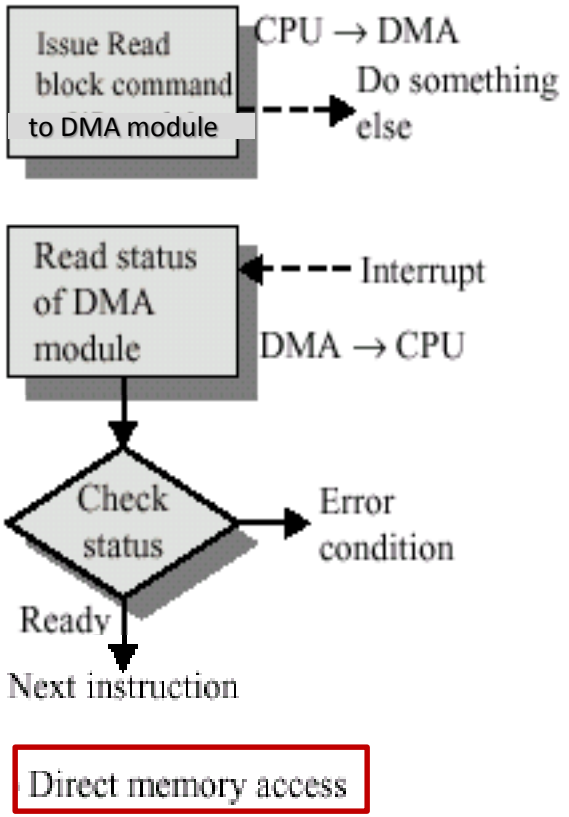
Direct memory access (DMA) I/O

DMA Controller

- CPU transfer information to DMA controller (DMAC)
  - location of data on device
  - location of data in memory
  - size of block to transfer
  - direction of transfer
- When device is ready, DMAC takes control of the system buses


Issue Read command to I/O controller
CPU → I/O
Read status of I/O controller
I/O → CPU
Not ready
Check status
Error condition
Ready
Read word from I/O controller
I/O → CPU
Write word into memory
CPU → memory
No
Done?
Yes
Next instruction
Programmed I/O


Issue Read command to I/O controller
CPU → I/O
Do something else
Read status of I/O controller
Interrupt
I/O → CPU
Check status
Error condition
Ready
Read word from I/O controller
I/O → CPU
Write word into memory
CPU → memory
No
Done?
Yes
Next instruction
Interrupt-driven I/O


Issue Read block command to DMA module
CPU → DMA
Do something else
Read status of DMA module
Interrupt
DMA → CPU
Check status
Error condition
Ready
Next instruction
Direct memory access

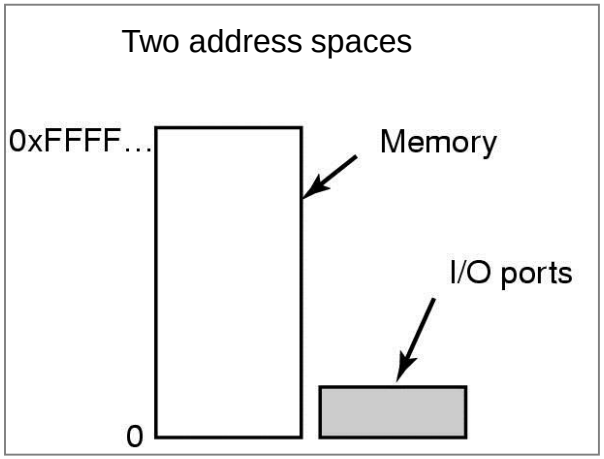
I/O Ports

- I/O port is the address of I/O controller register
- Two kinds of mapping:

Isolated I/O

– separate address spaces

– special commands for I/O



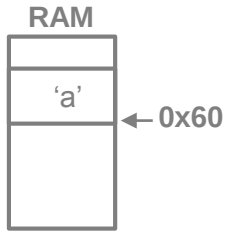
x86 provides **64 KB** of isolated I/O address space

I/O ports **is not physical memory**, it is virtual mapping of I/O controllers' registers and to their addresses

- **in** instruction is used to **read data** from I/O port:
  - **in register, port address** (direct address)
  - **in register, DX** (indirect address)
- **out** instruction is used to **write data** to I/O port:
  - **out port address, register** (direct address)
  - **out DX, register** (indirect address)
- register must be AL, AX, or EAX

Example:

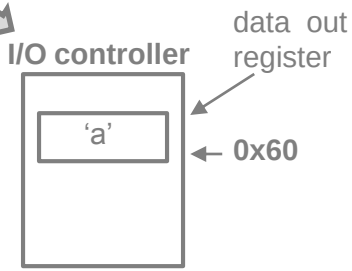
mov byte [0x60], 'a'



mov AL, 'a'
out 0x60, AL

I/O ports mapping table

address	I/O controller
0x60	Screen device
…	…



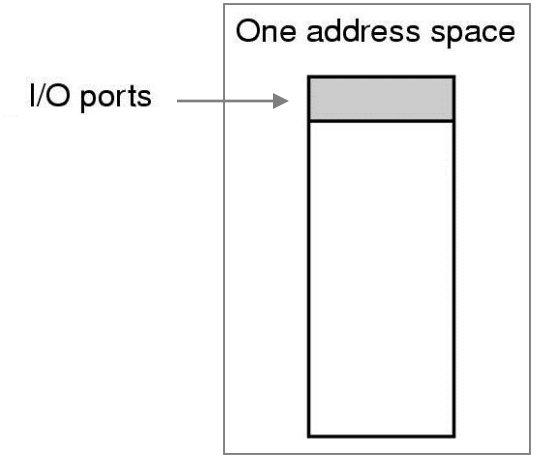
I/O Ports

- I/O port is the address of I/O controller register
- Two kinds of mapping:

Memory mapped I/O

– no special commands for I/O

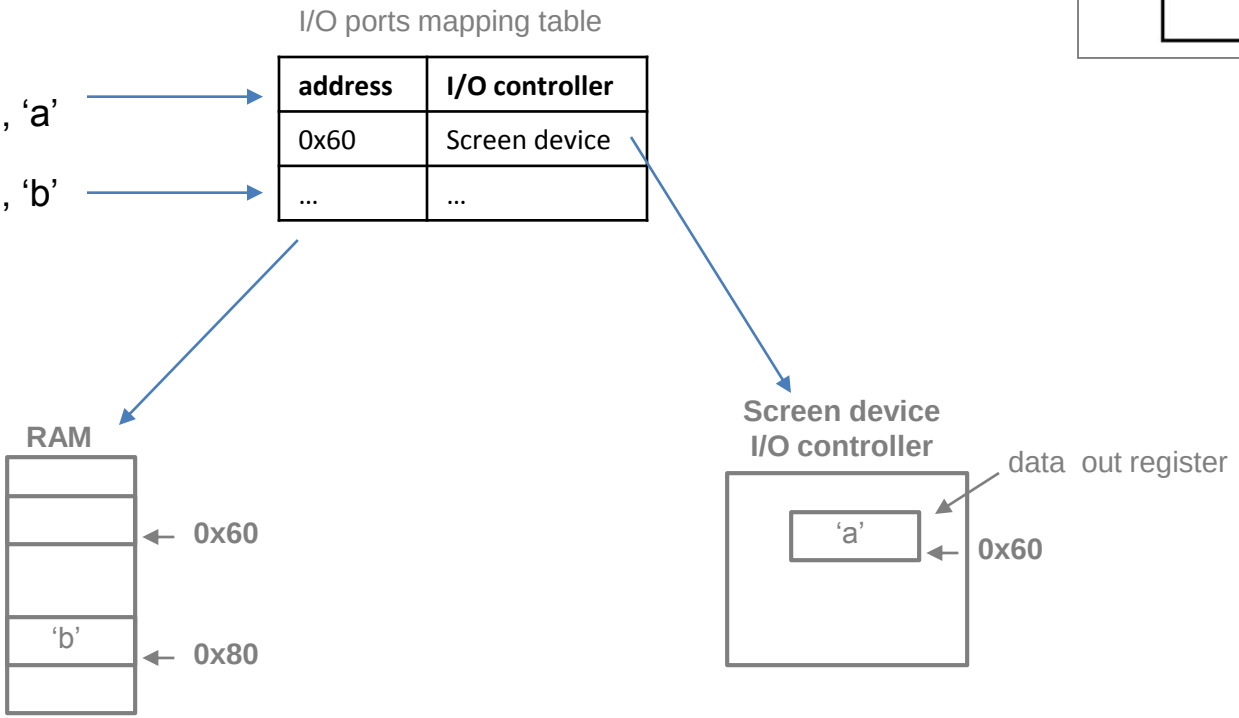
– use regular memory read/write commands



Example:

mov byte [0x60], ‘a’

mov byte [0x80], ‘b’



Example – How to Write a Simple Keyboard Driver

- Use PA input port register at address 60h
  - PA7 = 0 if a key is pressed
  - PA7 = 1 if a key is released
  - PA0–PA6 = key scan code

PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0

- Use busy wait loop
  - wait until a key is pressed (i.e. until PA7 = 0)
  - read scan code of the key
  - wait until a key is released (i.e. until PA7 = 1)

- Pressing ESC key terminates the program

Example – How to Write a Simple Keyboard Driver

```
section .data
ESC_KEY EQU 0x1B                ; ASCII code for ESC key
KB_DATA  EQU 0x60               ; port PA

section .text
global _start
_start:

key_down_loop:                  ; loop until a key is pressed i.e., until PA7 = 0
    in AL, KB_DATA              ; read keyboard status & scan code
    test AL, 0x80               ; PA7 = 0? (0x80=10000000b)
    jnz key_down_loop           ; if not, loop back

and AL,0x7F                     ; (0x7F=01111111b) – isolate scan code

..translate scan code to ASCII code in AL..

cmp AL, ESC_KEY                 ; ESC key - terminate program
je done

key_up_loop:                    ; loop until a key is released i.e., until PA7 = 1
    in AL, KB_DATA              ; read keyboard status & scan code
    test AL, 0x80               ; PA7 = 1? (0x80=10000000b)
    jz key_up_loop              ; if not, loop back (busy wait)

jmp key_down_loop

done:
 ..exit program..
```

TEST instruction performs a bitwise AND on two operands. The flags SF, ZF, PF are modified while the result of AND is discarded.