

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



EE213 COMPUTER ORGANIZATION AND ASSEMBLY LANGUAGE

Spring 2018

INTERRUPTS AND RELATED CONCEPTS

OUTLINES (COVERAGE FROM TEXT AND REFERENCE BOOK)

- Introduction
- Memory Mapped Vs Port Mapped I/O (17.5)
- IN and OUT Instructions (17.5.1)
- Interrupts
- Interrupt Handling (17.4)
- Interrupt Numbers (Appendix D)

INTRODUCTION

- We have seen instructions to:
 - Transfer information between the processor and the memory.
- However, Input/Output operations which transfer data from the processor or memory to and from the real world are essential.
- Generally, the rate of transfer from any input device to the processor, or from the processor to any output device is likely to be slower than the speed of a processor.
 - it is necessary to create mechanisms to synchronize the data transfer between them.

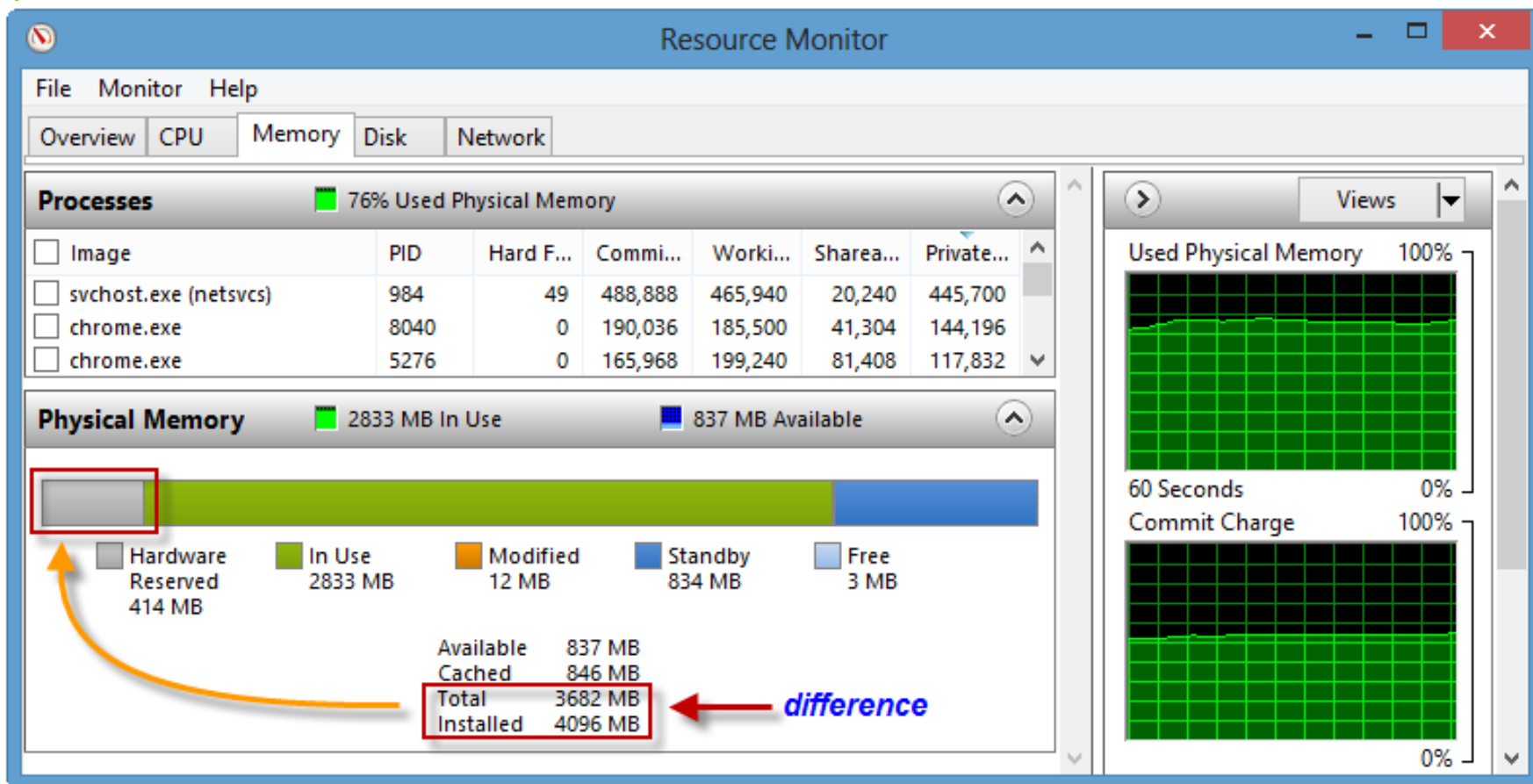
ACCESSING I/O DEVICES (MEMORY-MAPPED VS PORT-MAPPED I/O)

- Microprocessors normally use two methods to connect external devices: **memory mapped** or **port mapped I/O**.
- **Memory-Mapped I/O**
 - I/O devices and the program memory may share the same address space.
 - Any machine instruction that can access memory can be used to transfer data to or from an I/O device. (e.g. **MOV** instruction).
 - A program can write data to a particular memory address, and the data is transferred to the output device.
 - Similarly, data can be read from an input device by copying data from a predefined memory address.

PORT MAPPED I/O

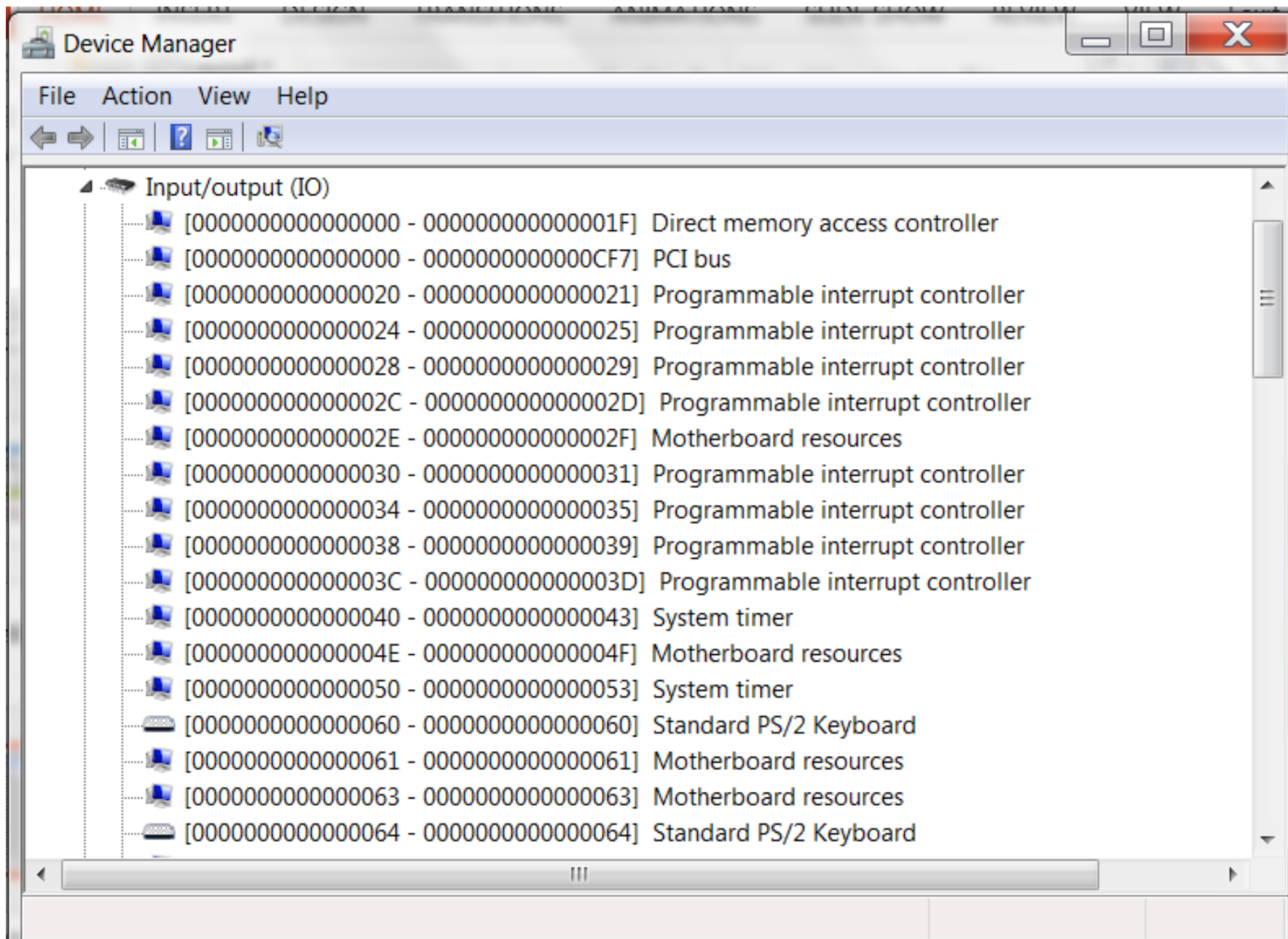
- I/O devices and the program memory have different address spaces.
- Port mapped I/O uses a separate, dedicated address space.
- Special instructions to transfer data to and from I/O devices
 - accessed via a dedicated set of microprocessor instructions.
- This is usually accomplished by having a different set of signal lines to indicate a memory access versus a port access.

Memory-mapped IO	Port-mapped IO
Same address space to address memory and I/O devices	Different address spaces for memory and I/O devices
Access to the I/O devices using regular instructions	Uses a special class of CPU instructions to access I/O devices
Most widely used I/O method	x86 Intel microprocessors - IN and OUT instructions



I/O PORTS

- Alternatively referred to as **I/O address**, **I/O ports**, and **I/O port address**, the **input/output port** is what allows the software drivers to communicate with hardware devices on your computer.
 - In your computer there are 65,535 ports that are numbered from 0000h to FFFFh.




IN AND OUT INSTRUCTIONS

- The **IN** instruction inputs a byte, word, or doubleword from a port.
- Conversely, the **OUT** instruction outputs a value to a port.

IN *accumulator, port*
OUT *port, accumulator*

- *Port* may be a constant in the range **0** to **FFh**, or it may be a value in DX between **0** and **FFFFh**.
- *Accumulator* must be AL for 8-bit transfers, AX for 16-bit transfers, and EAX for 32-bit transfers.



```
in al,3Ch  
out 3Ch,al
```

```
mov dx, portNumber  
in ax,dx  
out dx,ax
```

```
in eax,dx  
out dx,eax
```

PROGRAMMED I/O (KEYBOARD ATTACHED TO 03F8H)

```
MOV ESI, 0
MOV ECX, 100
MOV DX, 3F8h
L1: IN AL, DX
MOV SampleArray[ESI], AL
INC ESI
LOOP L1
```

POLLING

INTERRUPTS

- When accessing I/O devices **Program-controlled I/O** is the approach where processor repeatedly monitors the status of device, it does not perform any useful tasks.
 - Processor polls the I/O device.
- An alternate approach would be for the I/O device to alert the processor when it becomes ready.
- Do so by sending a hardware signal called an **interrupt** to the processor.
 - At least one of the bus control lines, called an interrupt request line is dedicated for this purpose.
 - Processor can perform other useful tasks while it is waiting for the device to be ready

POLLED I/O VS INTERRUPT DRIVEN I/O

- With **Polled I/O** processor continually checks I/O devices to see it is ready for data transfer.
 - Inefficient
- **Interrupt driven I/O** – I/O device interrupts processor when it is ready for data transfer.
 - Processor can do other jobs while waiting for last data transfer to complete.

TYPES OF INTERRUPTS

(YTHA YU, CHARLES MARUT ASM BOOK)

1. Hardware Interrupts (Interrupts)

- Used by devices to communicate that they require attention.
- hardware interrupt is generated by the Intel 8259 *Programmable Interrupt Controller* (PIC), which signals the CPU to suspend execution of the current program and execute an interrupt service routine.

2. Software Interrupts (Traps)

- Software interrupts are used by programs to request system services.
- Software interrupt occurs when a program calls an interrupt routine using the **INT** instruction.

INTERRUPTS HANDLING

- An interrupt alerts the processor for an immediate response.
- The processor responds by suspending its current activities, saving its state, and executing a function called an ***interrupt handler*** (or an interrupt service routine, ISR) to deal with the event.
- An **interrupt handler** or **interrupt service routine (ISR)** is the function that is executed by OS in response to a specific **interrupt**.
- **Interrupt numbers** are used by devices to identify their own service routines.
 - Not every interrupt number has a corresponding interrupt routine
 - One interrupt number may have multiple functions, identified by function number. (e.g. INT 21h, see Appendix-D).



- Interrupt handler (interrupt service routine) – performs common I/O tasks

- can be called as functions
- can be activated by hardware events

- Examples:

- video output handle
- critical error handler
- keyboard handler
- divide by zero handler
- Ctrl-Break handler
- serial port I/O

WHAT HAPPENS WHEN INTERRUPT OCCURS

1. Processor received Interrupt
 2. Save context (return address, registers, flags, etc.) and JMP to ISR
 3. Interrupt Service Routing (ISR) or Interrupt Handler is execute
 4. IRET instruction is last in the ISR and it returns controls to the main program.
- The transfer to an interrupt routine is similar to a procedure call.
 - Interrupt-service routine and the program that it interrupts may belong to different users
 - Before branching to the interrupt-service routine, information such as condition code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored

- The 8086 does not generate the interrupt routine's address directly from the Interrupt number.
 - Doing so would mean that a particular interrupt routine must *be placed in* exactly the same location
- This means that the routine may appear anywhere in the memory, therefore, its address, called **interrupt vector**, is stored in a predefined memory location.
- All interrupt vectors are placed in an **interrupt vector table**.

Table 17-2 Interrupt Vector Table Example.

Interrupt Number	Offset	Interrupt Vectors
00-03	0000	02C1:5186 0070:0C67 0DAD:2C1B 0070:0C67
04-07	0010	0070:0C67 F000:FF54 F000:837B F000:837B
08-0B	0020	0D70:022C 0DAD:2BAD 0070:0325 0070:039F
0C-0F	0030	0070:0419 0070:0493 0070:050D 0070:0C67
10-13	0040	C000:0CD7 F000:F84D F000:F841 0070:237D

- Each entry in the table (interrupt vector) is a 32-bit segment-offset address that points to one of the existing service routines.

- Thus an interrupt handler is executed if:
 1. An application program executes **INT** instruction. Or
 2. when a hardware device sends a signal to the Programmable Interrupt Controller chip.
- Occasionally, programs must disable hardware interrupts (**Interrupt Disabling**) when performing sensitive operations on segment registers and the stack.
- **Interrupt Control Instructions:** The **CLI** (*clear interrupt flag*) instruction disables interrupts, and the **STI** (*set interrupt flag*) instruction enables interrupts.

IRQ LEVELS (INTERRUPT REQUEST LEVEL)

- Interrupts can be triggered by a number of different devices on a PC.
- Each device has a priority, based on its *interrupt request level* (IRQ).
 - Level 0 has the highest priority, and level 15 has the lowest.
- Prevents low-priority interrupt from interrupting a high-priority interrupt.

IRQ	Interrupt Number	Description
0	8	System timer (18.2 times/second)
1	9	Keyboard
2	0Ah	Programmable Interrupt Controller
3	0Bh	COM2 (serial port 2)
4	0Ch	COM1 (serial port 1)
5	0Dh	LPT2 (parallel port 2)
6	0Eh	Floppy disk controller
7	0Fh	LPT1 (parallel port 1)
8	70h	CMOS real-time clock
9	71h	(Redirected to INT 0Ah)
10	72h	(Available) sound card
11	73h	(Available) SCSI card
12	74h	PS/2 mouse
13	75h	Math coprocessor
14	76h	Hard disk controller
15	77h	(Available)

E.G. KEYBOARD INTERRUPT

- When a key is pressed, the **8259 PIC** sends an **INTR** signal to the CPU, passing it the interrupt number.
- if external interrupts are not currently disabled, the CPU does the following, in sequence:
 1. Pushes the Flags register on the stack.
 2. Clears the Interrupt flag, preventing any other hardware interrupts.
 3. Pushes the current CS and IP on the stack.
 4. Locates the interrupt vector table entry for INT 9 and places this address in CS and IP.



- Next, the BIOS routine for INT 9 executes, and it does the following in sequence:

1. Re-enables hardware interrupts so the system timer is not affected.
2. Inputs a scan code from the keyboard port, attempts to convert it to an ASCII character, or assigns an ASCII code equal to zero. It then stores the scan code and ASCII code in the keyboard buffer.
3. Executes an IRET (interrupt return) instruction, which pops IP, CS, and the Flags register off the stack.
 - Control returns to the program that was executing when the interrupt occurred

Mnemonic	Meaning	Format	Operation	Flags Affected
CLI	Clear interrupt flag	CLI	$0 \rightarrow (IF)$	IF
STI	Set interrupt flag	STI	$1 \rightarrow (IF)$	IF
INT n	Type n software interrupt	INT n	$(Flags) \rightarrow ((SP) - 2)$ $0 \rightarrow TF, IF$ $(CS) \rightarrow ((SP) - 4)$ $(2 + 4 \cdot n) \rightarrow (CS)$ $(IP) \rightarrow ((SP) - 6)$ $(4 \cdot n) \rightarrow (IP)$	TF, IF
IRET	Interrupt return	IRET	$((SP)) \rightarrow (IP)$ $((SP) + 2) \rightarrow (CS)$ $((SP) + 4) \rightarrow (Flags)$ $(SP) + 6 \rightarrow (SP)$	All