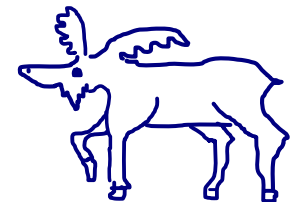


Lecture 27

Polling and Interrupt

Byung-gi Kim
School of Computing
Soongsil University



6. Storage and Other I/O Topics

6.1 Introduction

6.2 Dependability, Reliability, and Availability

6.3 Disk Storage

6.4 Flash Storage

6.5 Connecting Processors, Memory, and I/O Devices

6.6 Interfacing I/O Devices to the Processor, Memory, and Operating System

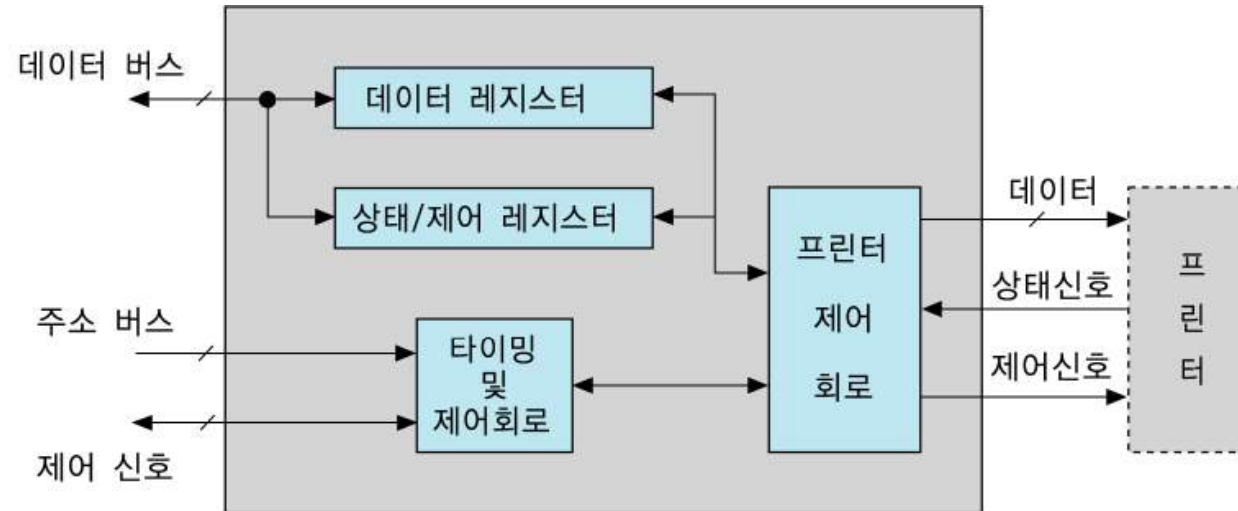
6.7 I/O Performance Measures: Examples from Disk and File Systems

6.8 Designing an I/O System

6.9 Parallelism and I/O: RAID

6.10 Real Stuff: Sun Fire x4150 Server

Giving Commands to I/O Devices



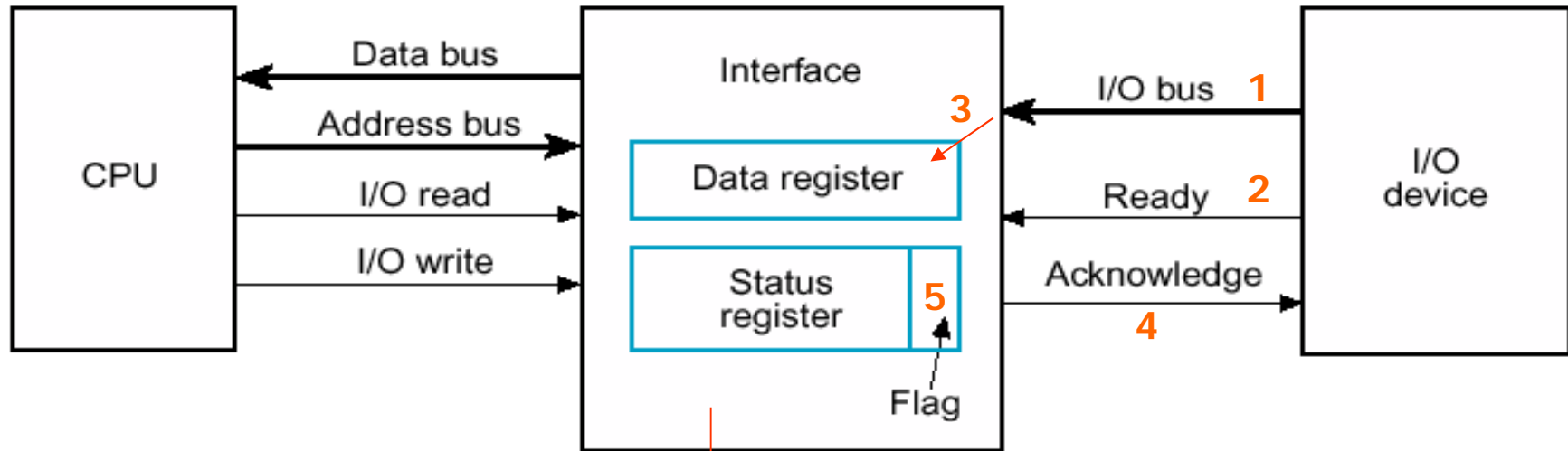
- **Status register**
 - ❖ Done bit and Error bit
- **Data register**
 - ❖ Data to be printed
- **Operations of processor**
 - ❖ Must wait until the done bit set by the printer
 - ❖ Must check the error bit

Communicating with the Processor

■ Polling

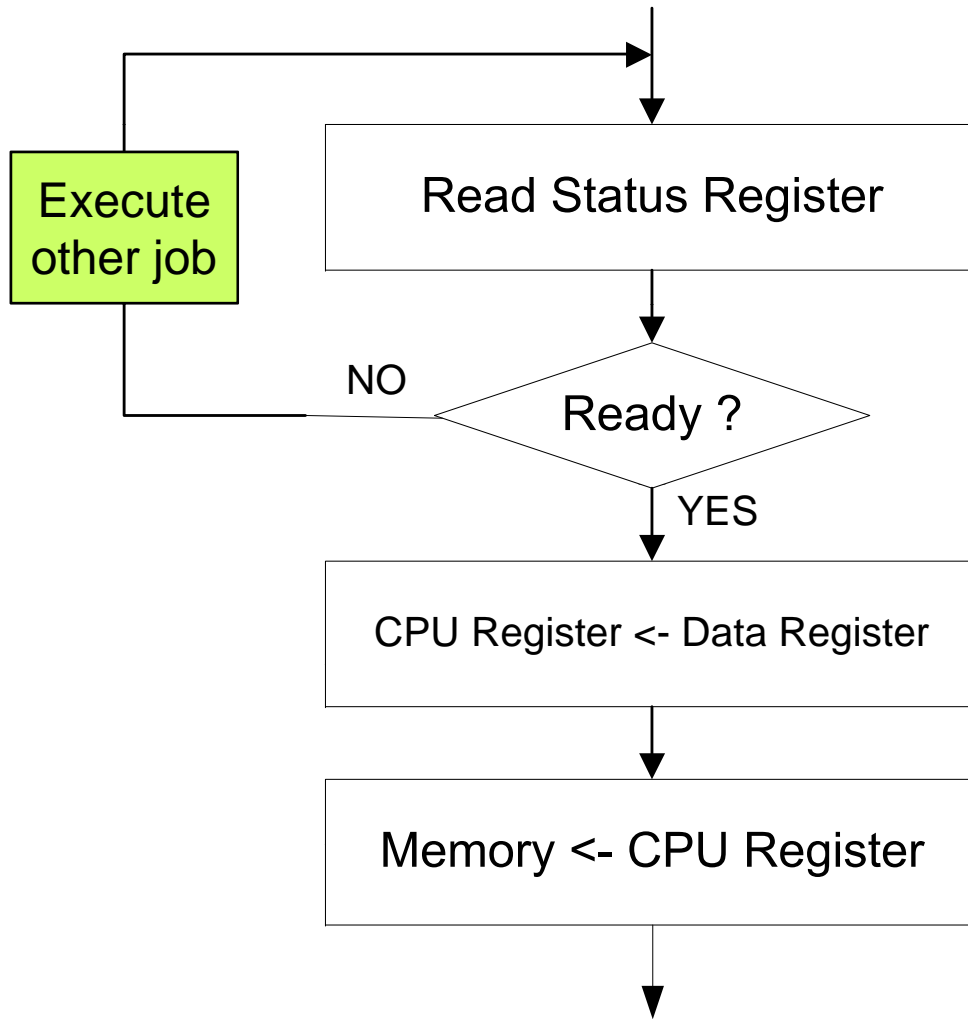
- ❖ aka () I/O or program-controlled transfer
- ❖ The process of periodically checking status bits to see if it is time for the next I/O operation
- ❖ I/O device simply puts the information in a () register.
- ❖ The simplest way for an I/O device to communicate with the processor
- ❖ The processor is totally in control and does all the work.
- ❖ Waste a lot of processor time
 - ◆ Because the processors are much faster than I/O devices
 - ◆ Read the Status register many times, only to find that the device has not yet completed I/O operation.
 - ◆ idle loop, busy waiting, spinlock

Example Programmed Input

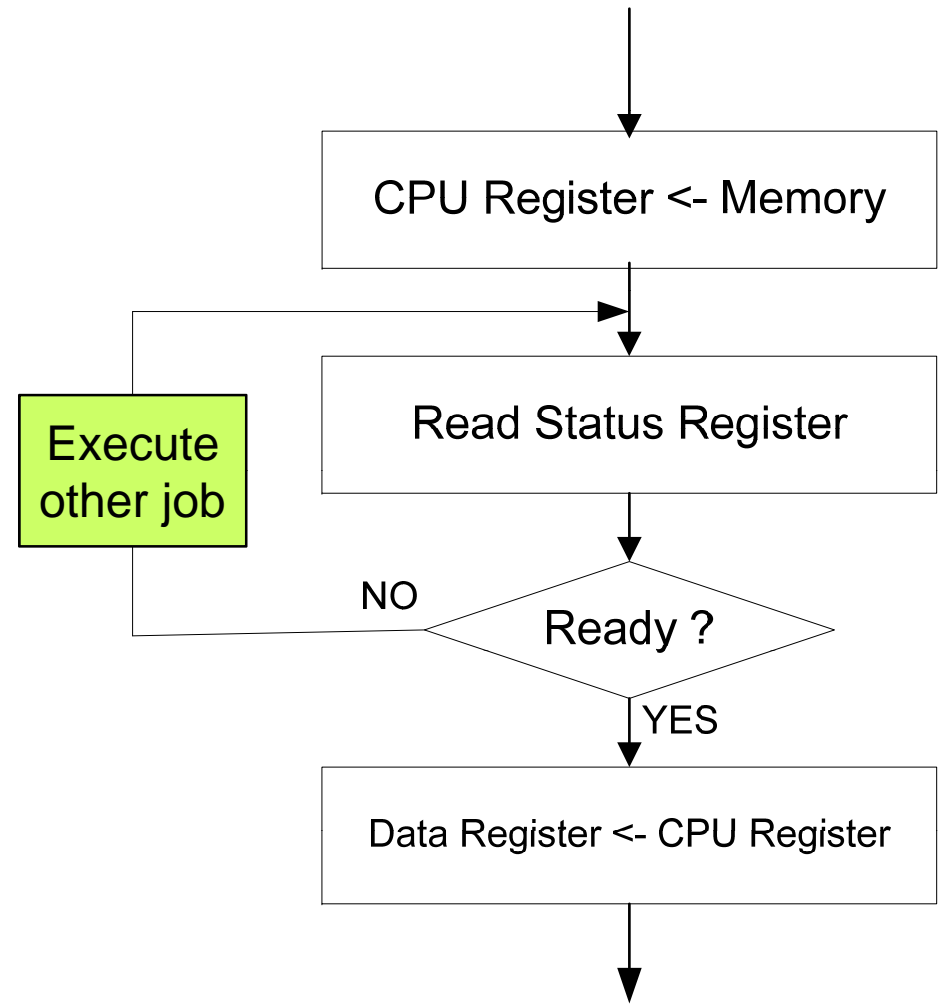


**CPU must check the value of the flag
- wasting time**

Programmed Input



Programmed Output



Interrupt-Driven I/O

■ I/O interrupts

- ❖ Alleviate overhead in polling interface
- ❖ Using interrupts to notify the processor when an I/O device requires attention from the processor
- ❖ Do not waste processor time

■ Differences of I/O interrupts from exceptions

- 1) () with respect to the instruction execution
 - ◆ Is not associated with any ()
- 2) Need more information
 - ◆ Such as () and ()
 - ◆ Different priorities of the devices

Communicating Information to Processor

- Such as identity of the interrupting device

(cf) Lecture 19 Exceptions

1. Vectored interrupt

- ❖ Send vector address

2. Non-vectored interrupt

- ❖ Send status field to place in the Cause register

Interrupt Priority Levels

■ **Interrupt priority**

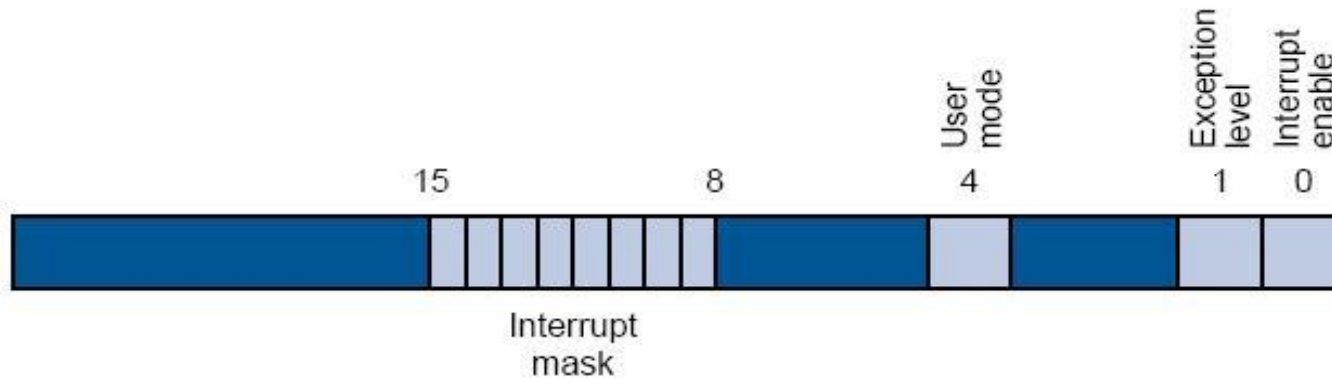
- ❖ Indicate the order in which the processor should process interrupts
- ❖ 4 to 6 levels in UNIX
- ❖ Typically I/O interrupts have lower priority than internal exceptions.
- ❖ MIPS has 6 hardware and 2 software interrupt levels.

■ **Status register**

- ❖ Determines who can interrupt the computer
- ❖ Interrupt enable & interrupt mask
- ❖ User mode: 1 when in user mode, 0 when in kernel mode
- ❖ Exception level: 1 after an exception occurs

Key Registers

- Status register



- Cause register

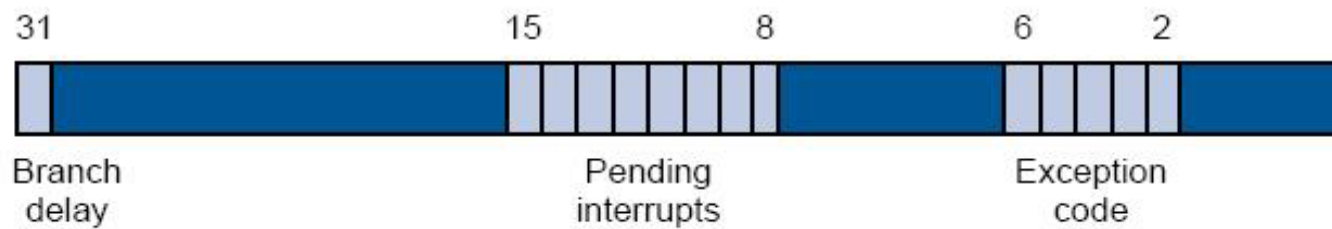
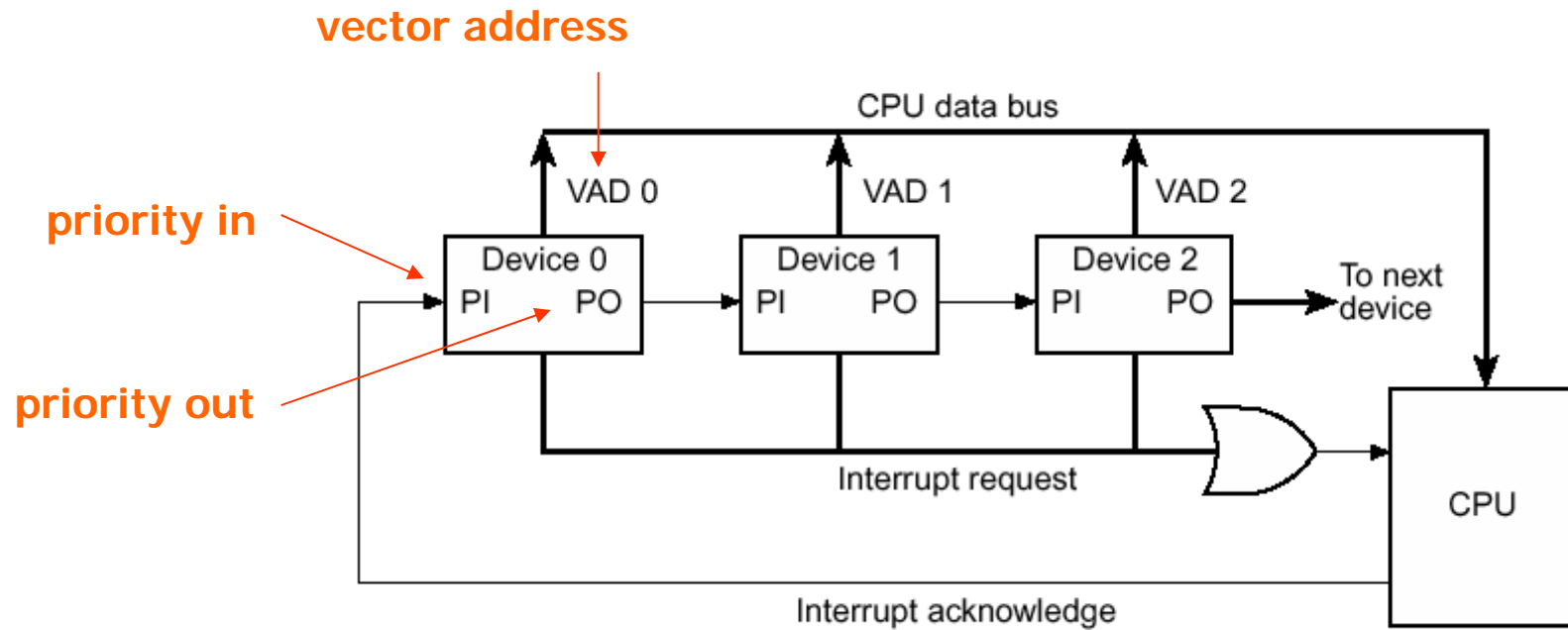
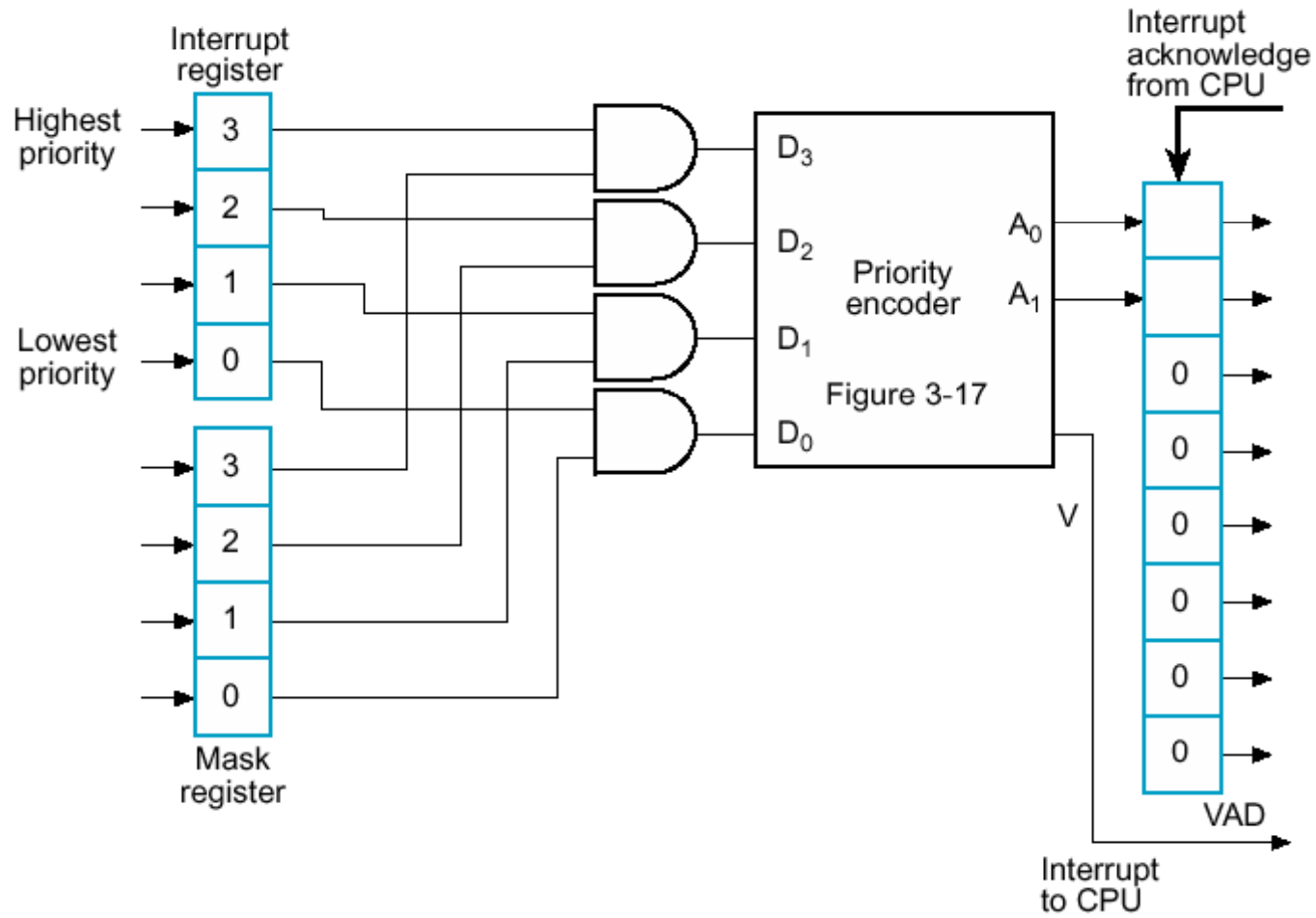


Figure 6.11

Daisy-chain Priority Interrupt



Parallel Priority Interrupt



Interrupt Handling in MIPS

- **Exception handler**

- ❖ $8000\ 0180_{\text{hex}}$ in the kernel space
- ❖ Examines the exception's cause and jumps to an appropriate point in the operating system

- **Cause of exception**

Number	Name	Cause of exception
0	Int	interrupt (hardware)
4	AdEL	address error exception (load or instruction fetch)
5	AdES	address error exception (store)
6	IBE	bus error on instruction fetch
7	DBE	bus error on data load or store
8	Sys	syscall exception
9	Bp	breakpoint exception
10	RI	reserved instruction exception
11	CpU	coprocessor unimplemented
12	Ov	arithmetic overflow exception
13	Tr	trap
15	FPE	floating point

Interrupt Handling

1. Logically AND the () field and the () field.
2. Select the highest priority interrupt.
3. Save the interrupt () .
4. Change the interrupt mask.
 - ❖ Disable all interrupts of equal or lower priority.
5. Save the () .
6. () the interrupt enable bit.
 - ❖ Allow higher-priority interrupts.
7. Call the appropriate interrupt routine.
8. () the interrupt enable bit and restore () .

Transferring the Data between a Device and Memory

1. Polling-based transfer (= programmed I/O)

- ❖ Periodical check of device status by CPU

```
while (not ready) get_status_of_the_device;  
load_data_from_I/O_device;  
store_the_data_into_memory;
```

- ❖ Best with lower-bandwidth devices
- ❖ More interested in reducing the cost of the device controller and interface than in providing a high-bandwidth transfer
- ❖ Put burden of moving data and managing the transfer on the processor

2. Interrupt Driven Transfer

- **Common characteristics with programmed I/O**

- ❖ () still transfers data in small number of bytes.
- ❖ Best with lower-bandwidth devices
- ❖ More interested in reducing the cost

- **Difference**

- ❖ () informs the processor when ready
- ❖ Relieving the processor from having to wait for every I/O

- **I/O operation**

- ❖ OS simply works on other tasks while data is being read from or written to the device.
- ❖ On interrupts, OS reads the status to check for errors.
- ❖ If none, OS transfers data.
- ❖ When I/O completed, OS can inform the program.