Chapter 7
Input/Output

2020.5
Howon Kim

- **정보보호 및 지능형 IoT연구실 - http://infosec.pusan.ac.kr**
- **부산대 지능형융합보안대학원 - http://aisec.pusan.ac.kr**

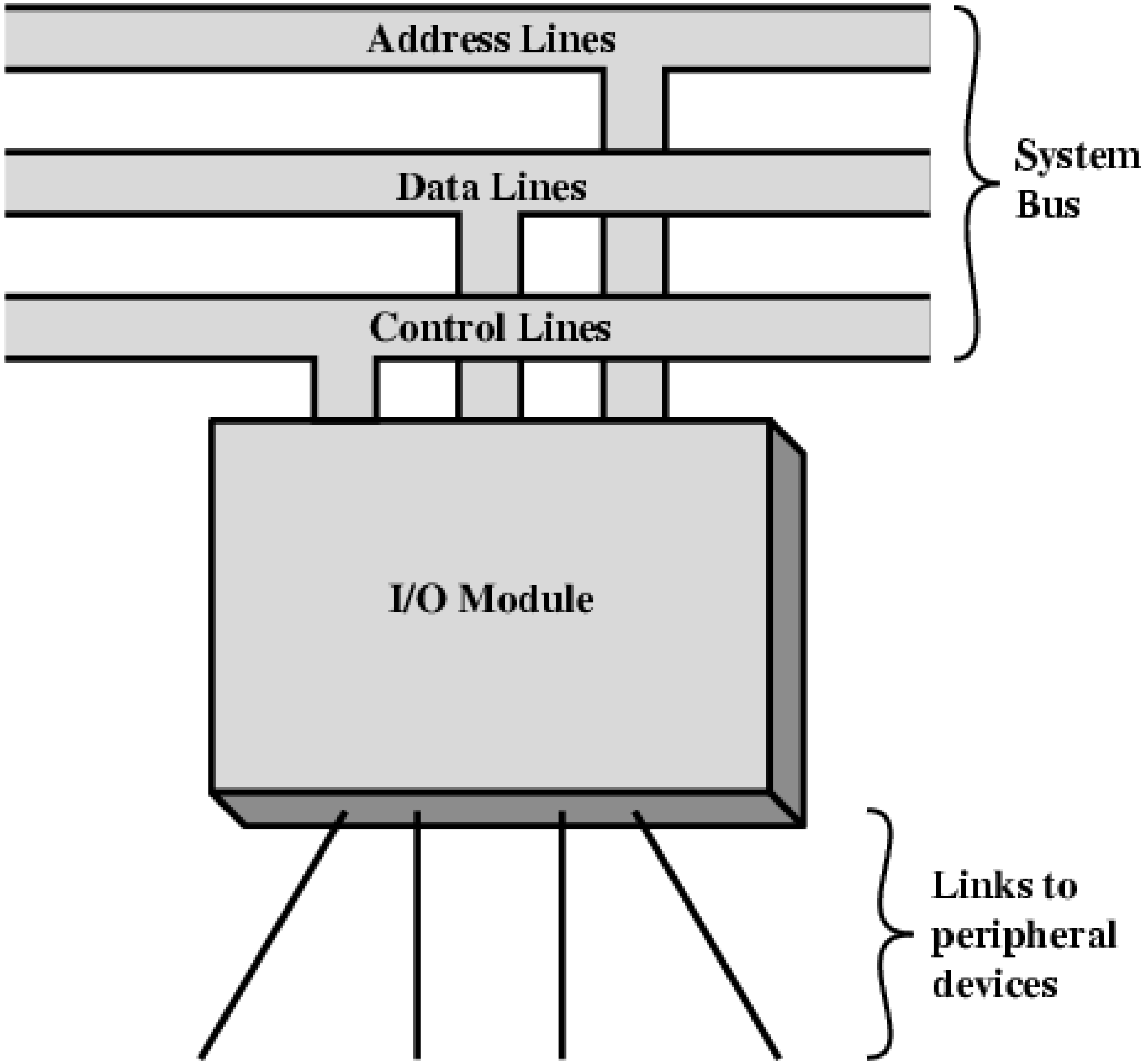
Input/Output Problems

- Wide variety of peripherals
  - Delivering different amounts of data
  - At different speeds
  - In different formats
- All slower than CPU and RAM
- Need I/O modules

Input/Output Module

- Interface to CPU and Memory
- Interface to one or more peripherals

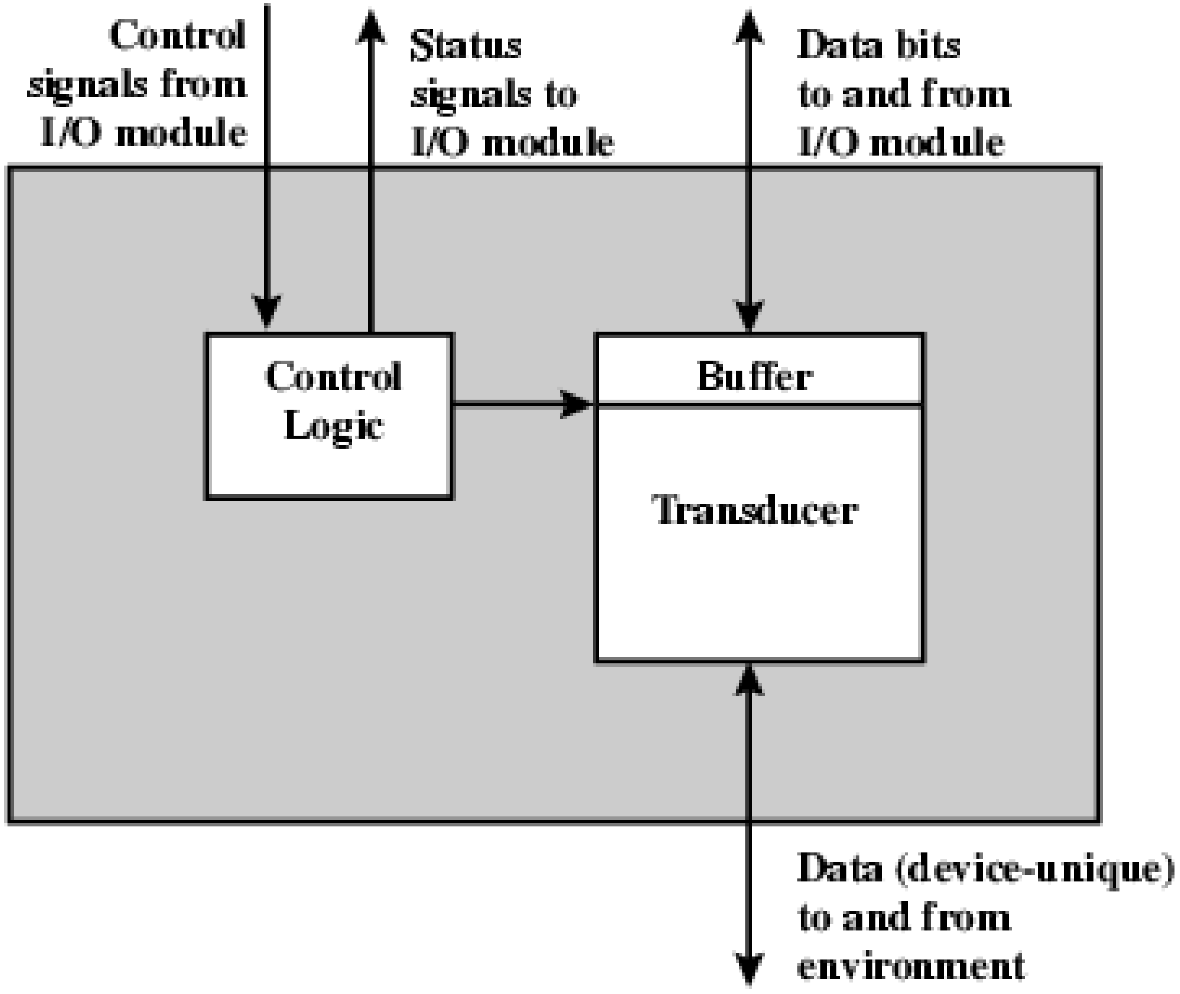
Generic Model of I/O Module



External Devices

- Human readable
  - Screen, printer, keyboard
- Machine readable
  - Monitoring and control
- Communication
  - Modem
  - Network Interface Card (NIC)

External Device Block Diagram


Control signals from I/O module
Status signals to I/O module
Data bits to and from I/O module
Control Logic
Buffer
Transducer
Data (device-unique) to and from environment

The major functions for an I/O module fall into the following categories:

Control and timing

- Coordinates the flow of traffic between internal resources and external devices

Processor communication

- Involves command decoding, data, status reporting, address recognition

Device communication

- Involves commands, status information, and data

Data buffering

- Performs the needed buffering operation to balance device and memory speeds

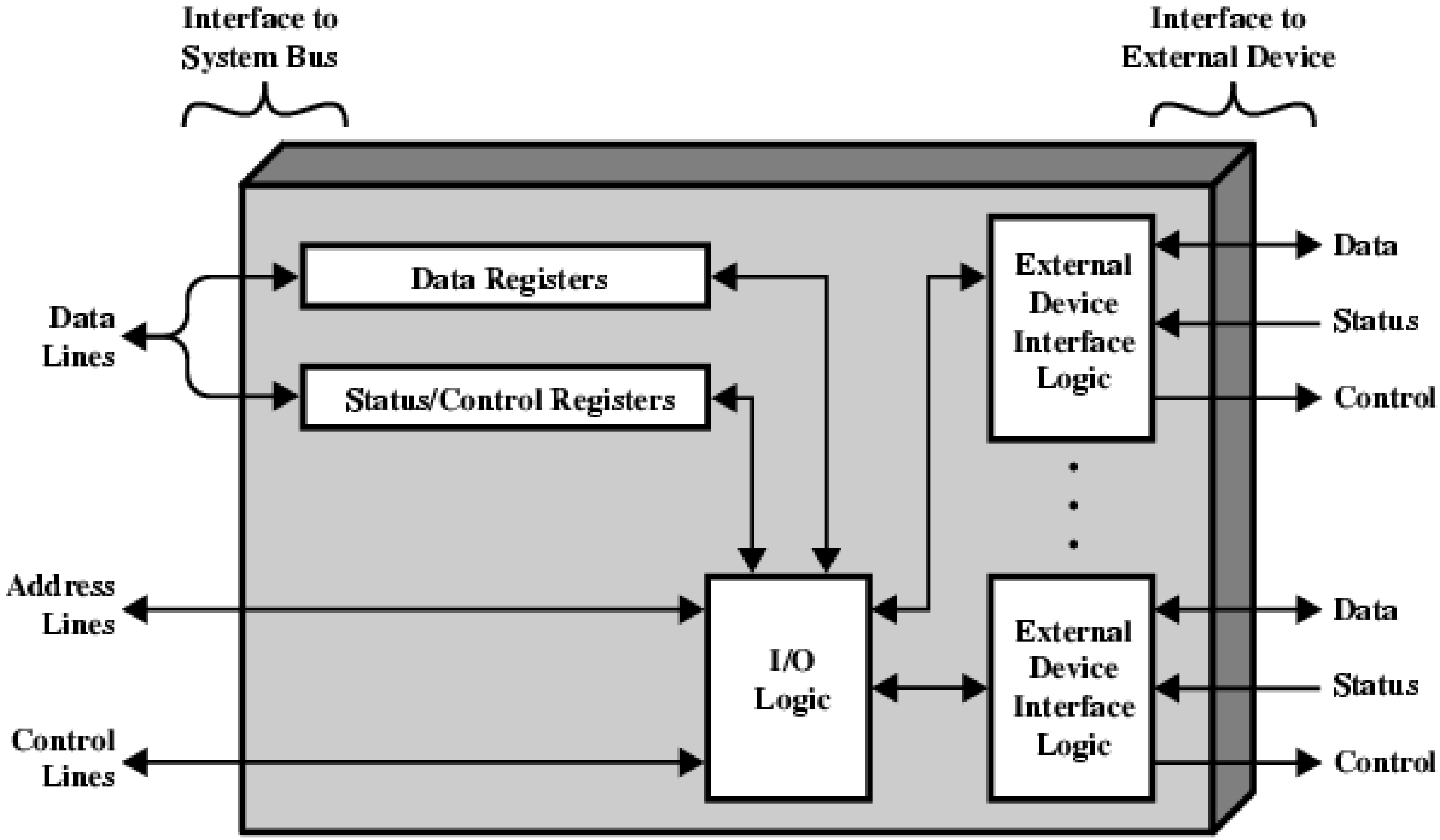
Error detection

- Detects and reports transmission errors

I/O Steps

- CPU checks I/O module device status
- I/O module returns status
- If ready, CPU requests data transfer
- I/O module gets data from device
- I/O module transfers data to CPU
  - Variations for output, DMA, etc.

I/O Module Diagram



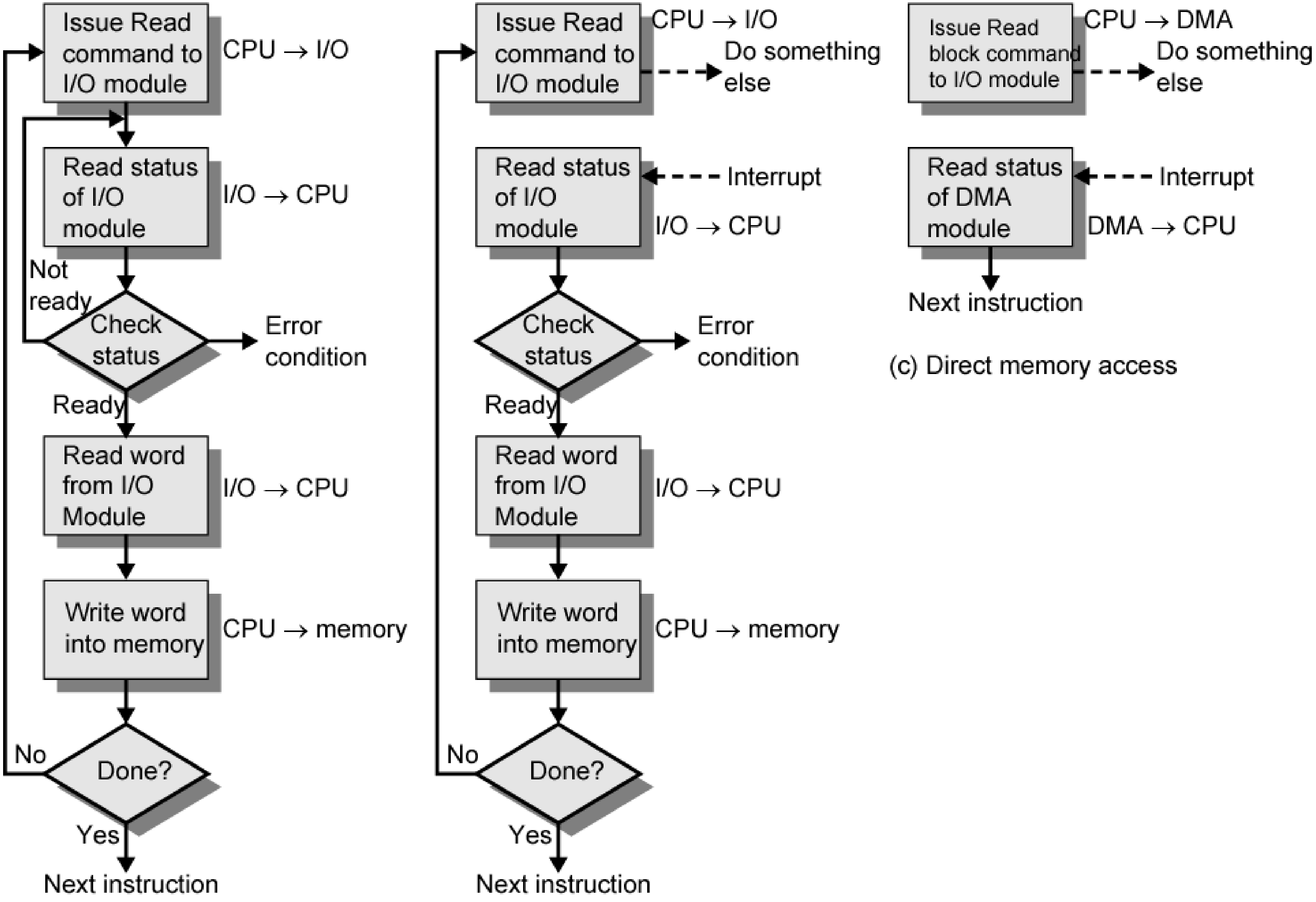
I/O Module Decisions

- I/O module **may hide the details of** timing, formats and the electromechanics of an external device so that processor can function in terms of simple read and write commands, and possibly open and close file commands.
- In its simplest form, the I/O module can take much of the work of controlling a device (e.g., rewind a tape) instead of the processor
- An I/O module that takes on most of the detailed processing burden, presenting a high-level interface to the processor, is usually referred to as an *I/O channel* or *I/O processor.*
- Support multiple or single device
- Also O/S decisions
  - e.g. Unix treats everything it can as a file
  - /dev/tty0, /dev/rdsk/c0s0d1, etc.

Input Output Techniques

- Three techniques are possible for I/O operations:
- Programmed I/O
  - Data are exchanged between the processor and the I/O module
  - Processor executes a program that gives it direct control of the I/O operation
  - When the processor issues a command it must wait until the I/O operation is complete
  - If the processor is faster than the I/O module this is wasteful of processor time
- Interrupt-driven I/O
  - Processor issues an I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work
- Direct memory access (DMA)
  - The I/O module and main memory exchange data directly without processor involvement

Three Techniques for Input of a Block of Data



(a) Programmed I/O

(b) Interrupt-driven I/O

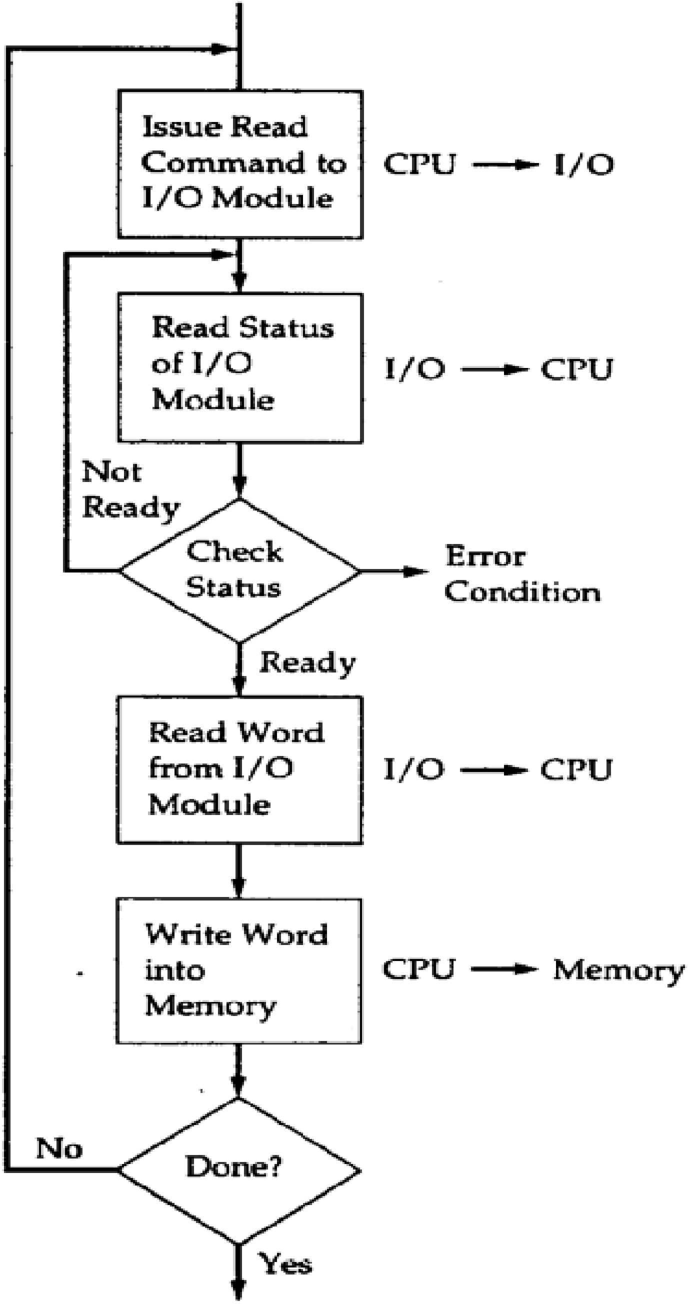
(c) Direct memory access

Programmed I/O

- CPU has direct control over I/O
  - Sensing status
  - Read/write commands
  - Transferring data
- CPU waits for I/O module to complete operation
- Wastes CPU time

Programmed I/O - detail

- CPU requests I/O operation
- I/O module performs operation
- I/O module sets status bits
- CPU checks status bits periodically
- I/O module does not inform CPU directly
- I/O module does not interrupt CPU
- CPU may wait or come back later



I/O Commands

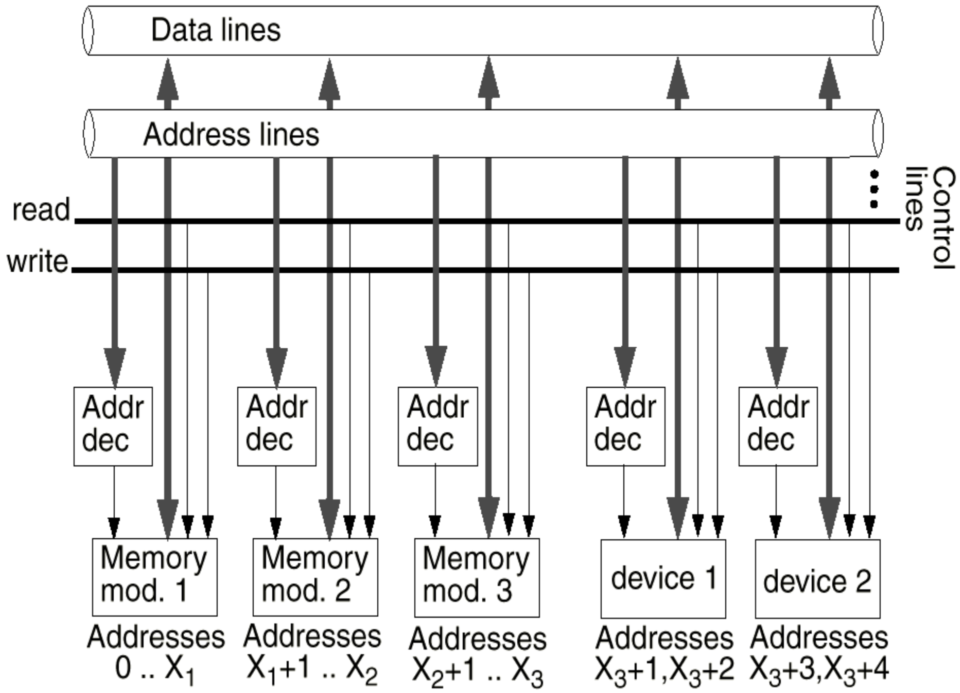
- CPU issues address
  - Identifies module (& device if >1 per module)
- CPU issues command
  - Control - telling module what to do
    - e.g. spin up disk
  - Test - check status
    - e.g. power? Error?
  - Read/Write
    - Module transfers data via buffer from/to device

Addressing I/O Devices

- Under programmed I/O data transfer is very like memory access (CPU viewpoint)
- Each device given unique identifier
- CPU commands contain identifier (address)

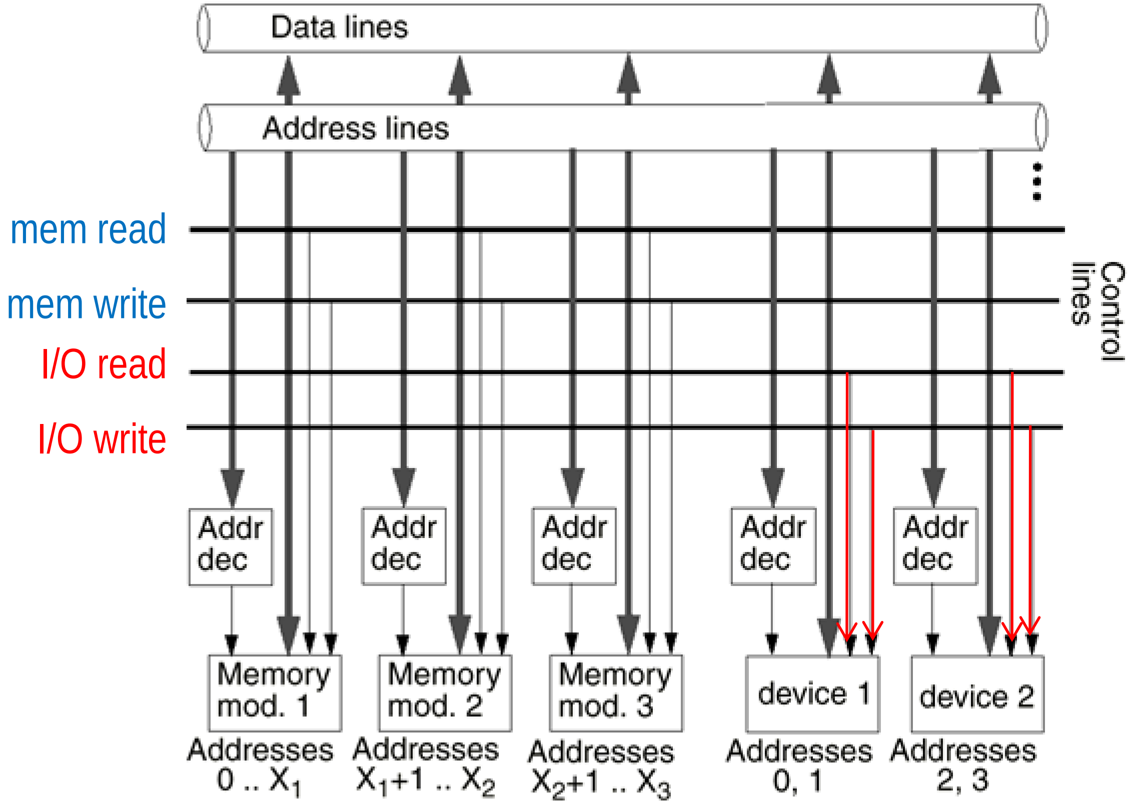
I/O Mapping

- Memory mapped I/O
  - Devices and memory share an address space
  - I/O looks just like memory read/write
  - No special commands for I/O
    - Large selection of memory access commands available

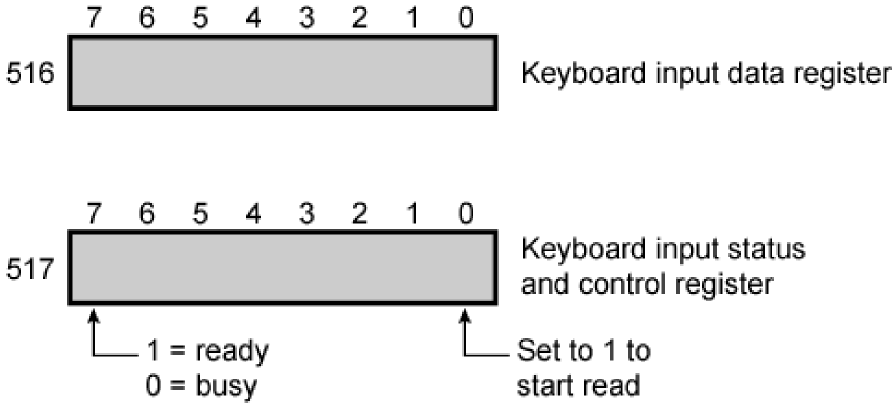


I/O Mapping

- Isolated I/O
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for I/O
    - Limited set



Memory Mapped and Isolated I/O



// set to 1 to start read the Keyboard input data reg.

ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load AC	"1"	Load accumulator
	Store AC	517	Initiate keyboard read
202	Load AC	517	Get status byte
	Branch if Sign = 0	202	Loop until ready
	Load AC	516	Load data byte

// Not ready !

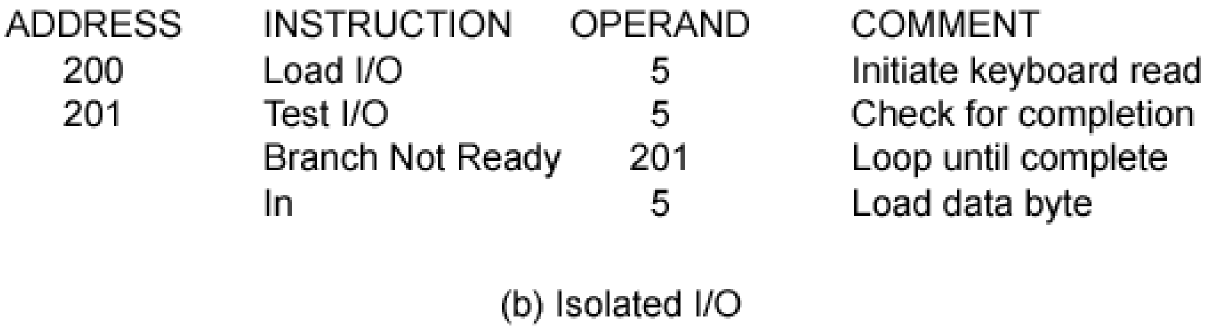
// ready ! Read the address 516

// that is, read the keyboard input

(a) Memory-mapped I/O

Memory Mapped and Isolated I/O

Assumption: I/O port # 5 is the port # of the keyboard input & control



ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load I/O	5	Initiate keyboard read
201	Test I/O	5	Check for completion
	Branch Not Ready	201	Loop until complete
	In	5	Load data byte

(b) Isolated I/O

I/O Mapping

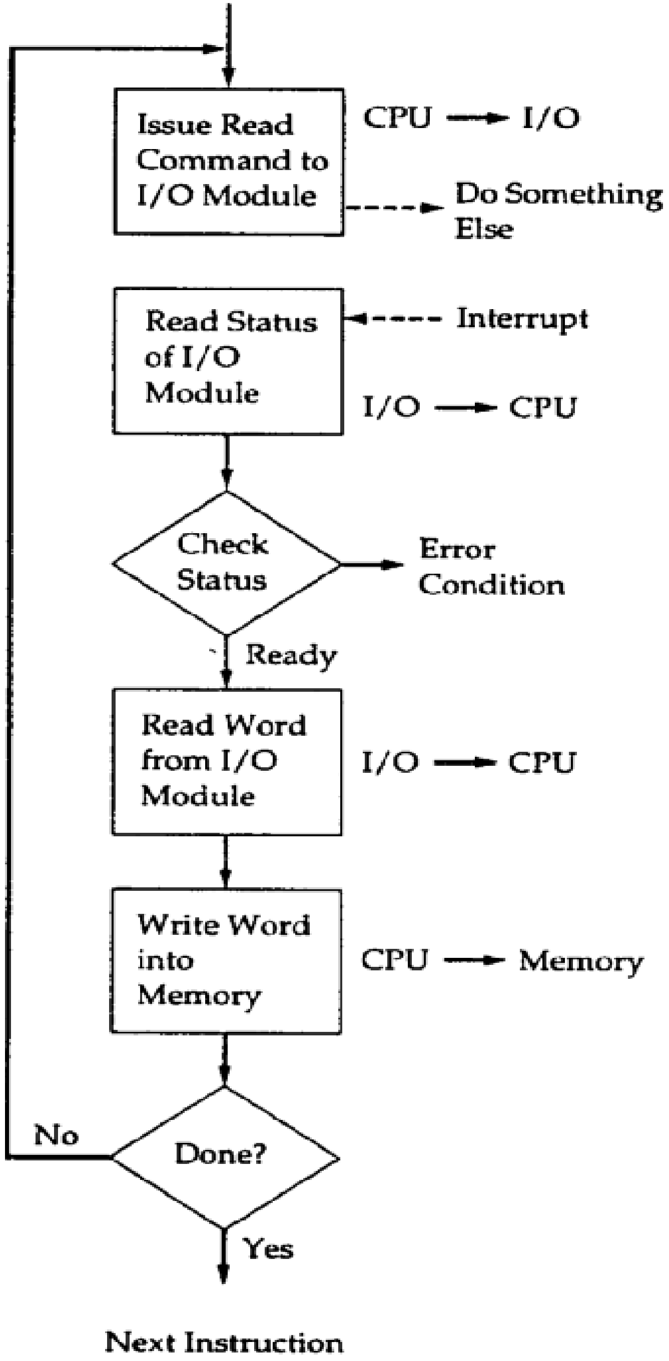
- Comparison
  - Advantage of memory mapped I/O: More efficient programming
  - Disadvantage of memory mapped I/O: Valuable memory space used up

Interrupt Driven I/O

- Overcomes CPU waiting
- No repeated CPU checking of device
- I/O module interrupts when ready

Interrupt Driven I/O Basic Operation

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data



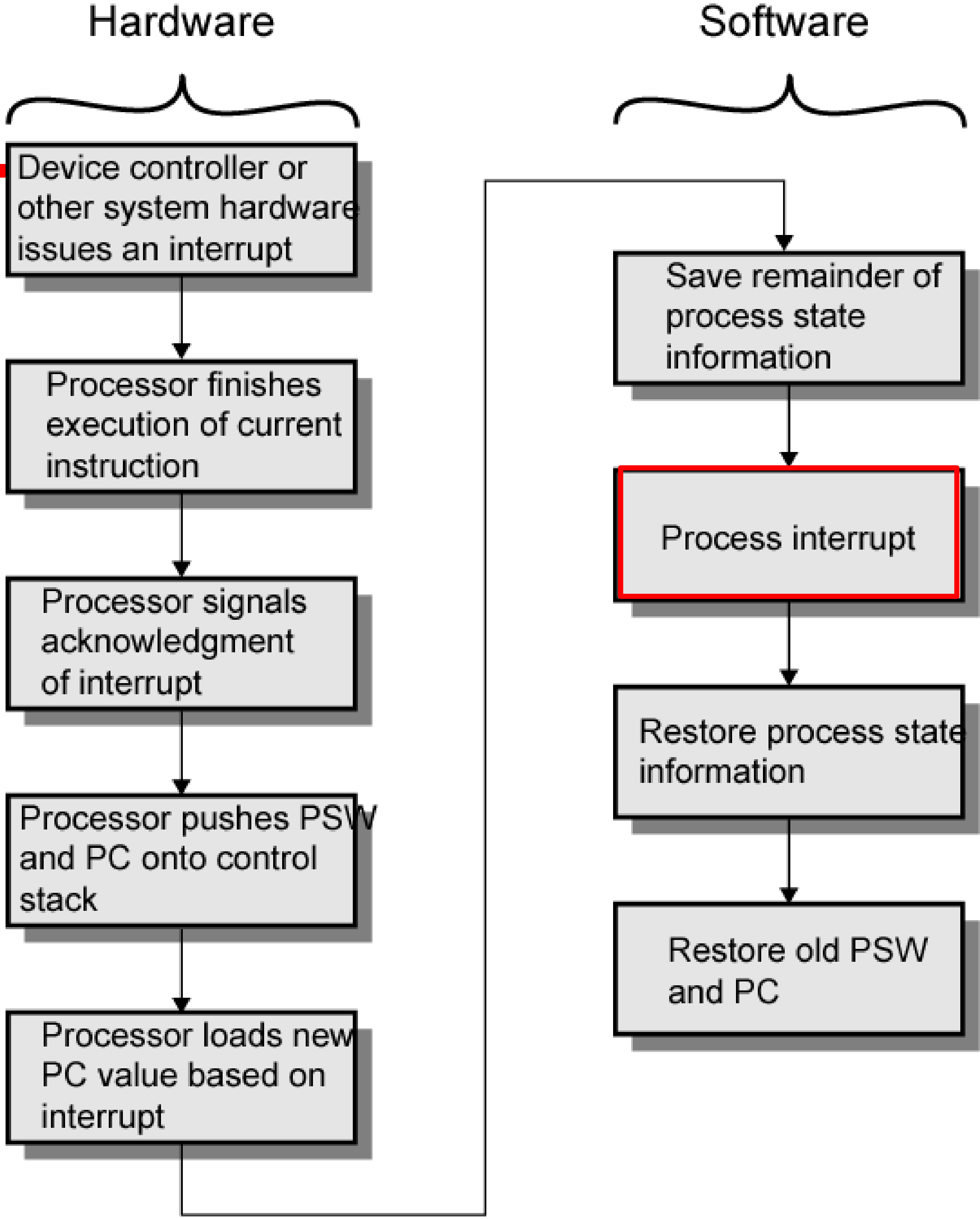
I/O Module Viewpoint

- Receive read command
- Read data from associated peripheral
- Signal interrupt to processor when data is ready (i.e., in data register)
- Wait until data requested by processor
  - Place data on data bus
  - Ready for another I/O operation

CPU Viewpoint

- Issue read command
- Do other work
- Check for interrupt at end of each instruction cycle
- If interrupted:-
    - Save context (registers)
    - Process interrupt
        - Fetch data & store

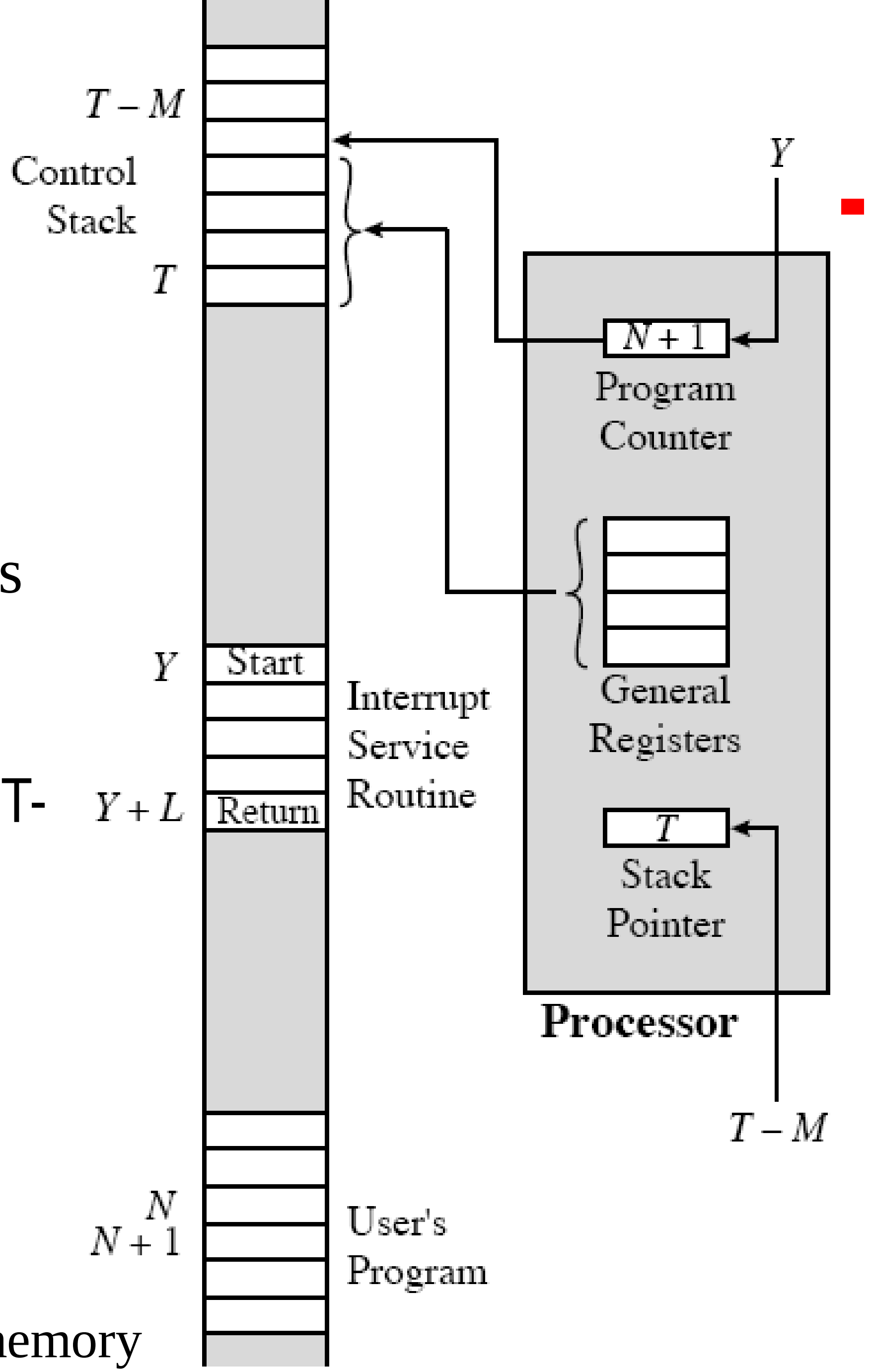
Interrupt Processing


Hardware
Software
Device controller or other system hardware issues an interrupt
Processor finishes execution of current instruction
Processor signals acknowledgment of interrupt
Processor pushes PSW and PC onto control stack
Processor loads new PC value based on interrupt
Save remainder of process state information
Process interrupt
Restore process state information
Restore old PSW and PC

PSW (Program Status Word)

Changes in Memory and Registers for an Interrupt

(a) Interrupt occurs after instruction at location N



(1) Interrupt occurred when user’s program N executed

(2) the contents of PC(N+1) & GRs are saved to the stack (address: T-M)

(3) At this time, the stack pointer is updated to the “T-M”

(4) The PC is updated to the “Y”

(5) ISR will be executed...

Changes in Memory and Registers for an Interrupt

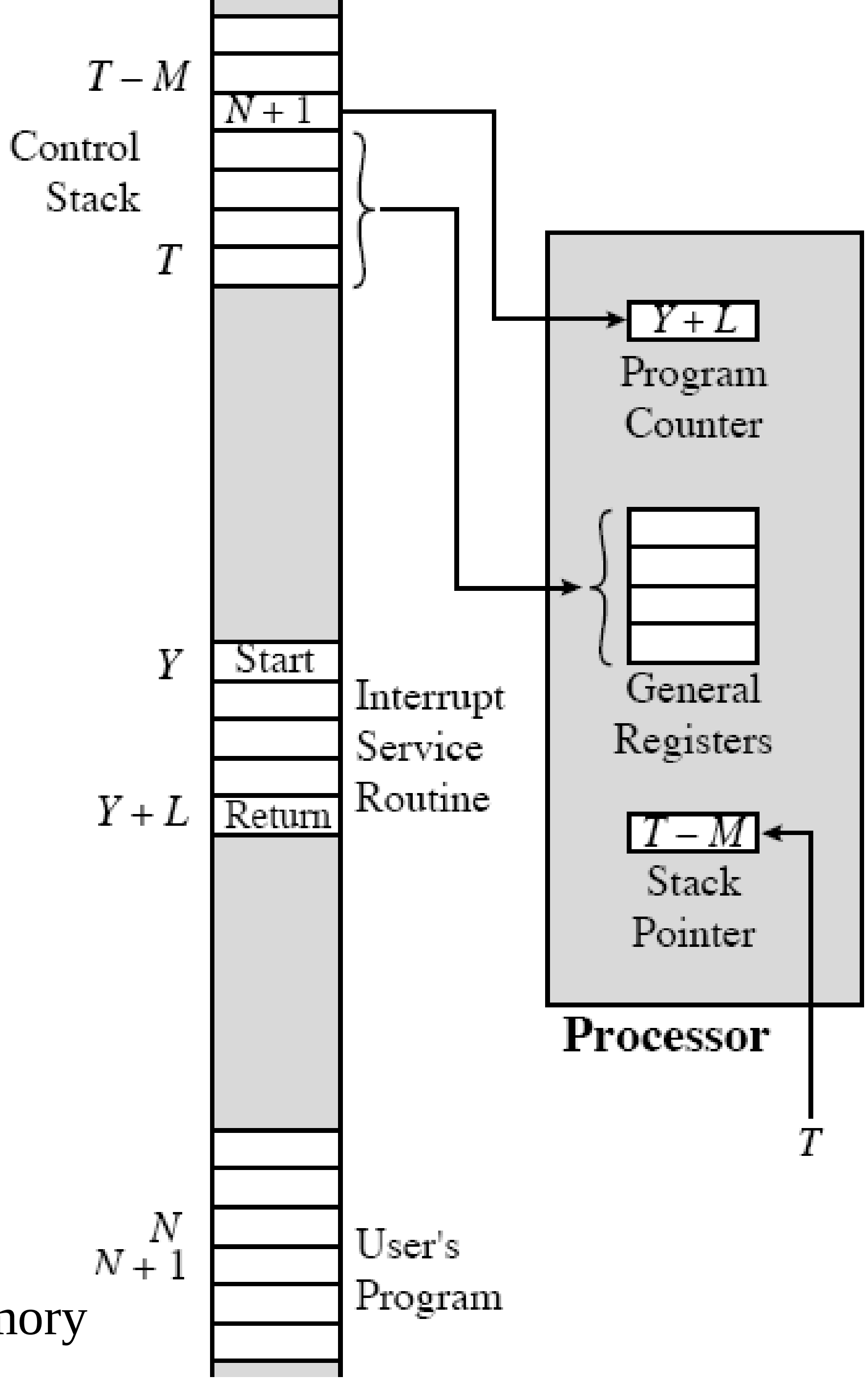
(b) Return from Interrupt

(1) ISR was done

(2) The saved PC & GRs will be restored

(3) The stack pointer also will set to "T"

(4) Now the user's program (address N+1) will be executed ...



Design Issues

- Among the multiple numbers of I/O modules, when an interrupt occurred, how can we identify the module issuing the interrupt?
- How do you deal with multiple interrupts?
  - i.e. an interrupt handler being interrupted

Identifying Interrupting Module (1)

- Different line for each module
  - Pin constraint limits number of devices
- Software poll
  - CPU asks each module in turn
    - Since each module has its own addressable status register, it will respond the asking
  - Slow

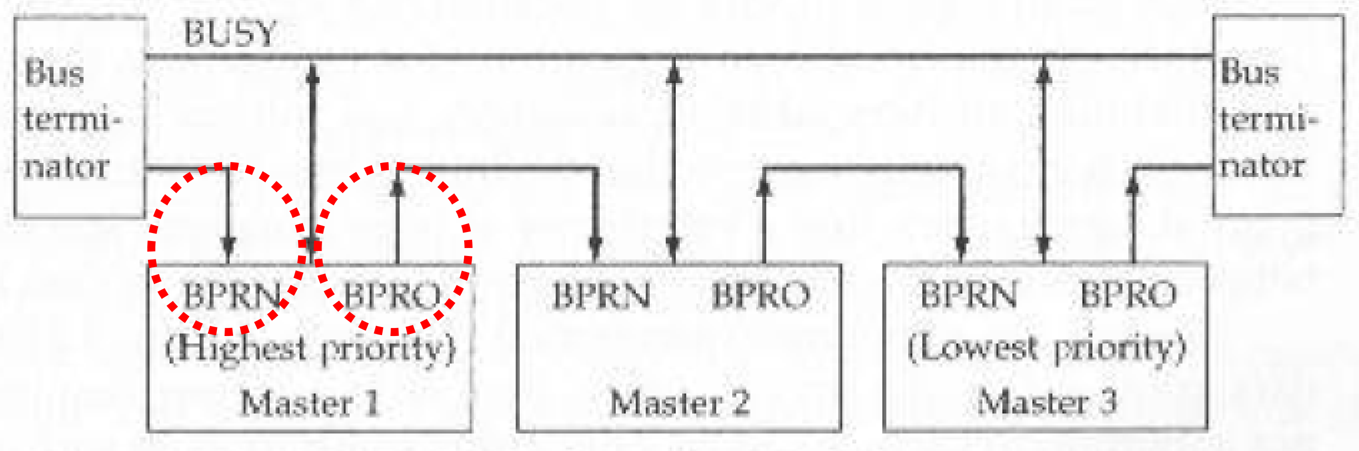
Identifying Interrupting Module (2)

- Daisy Chain or Hardware poll
  - Interrupt Acknowledge sent down a chain
  - Module which is responsible places its own vector on this chained bus
  - CPU can identify handler routine based on the received vector value

< source: wikipedia >

Identifying Interrupting Module (2)

- Daisy Chain or Hardware poll (cont'd)
  - The left-most agent in the diagram receives a constant "bus priority in(BPRN)" signal indicating that no higher-priority agent desires the bus.
  - If the agent does not wish the bus, it asserts its "bus priority out(BPRO)" line



Identifying Interrupting Module (3)

- Bus Master
  - Module must claim the bus before it can raise interrupt
  - So only one module can send the interrupt signal at some instant
  - e.g. PCI & SCSI

Multiple Interrupts

- If different line for each module
  - Each interrupt line has a priority
  - Higher priority lines can interrupt lower priority lines
- If software polling
  - Poling order according to priority
- If daisy chaining
  - Devices chained based on priority
- If bus mastering
  - Only current master can interrupt

Example - PC Bus

- 80x86 has one interrupt line
- 8086 based systems use one 8259A interrupt controller
- 8259A has 8 interrupt lines

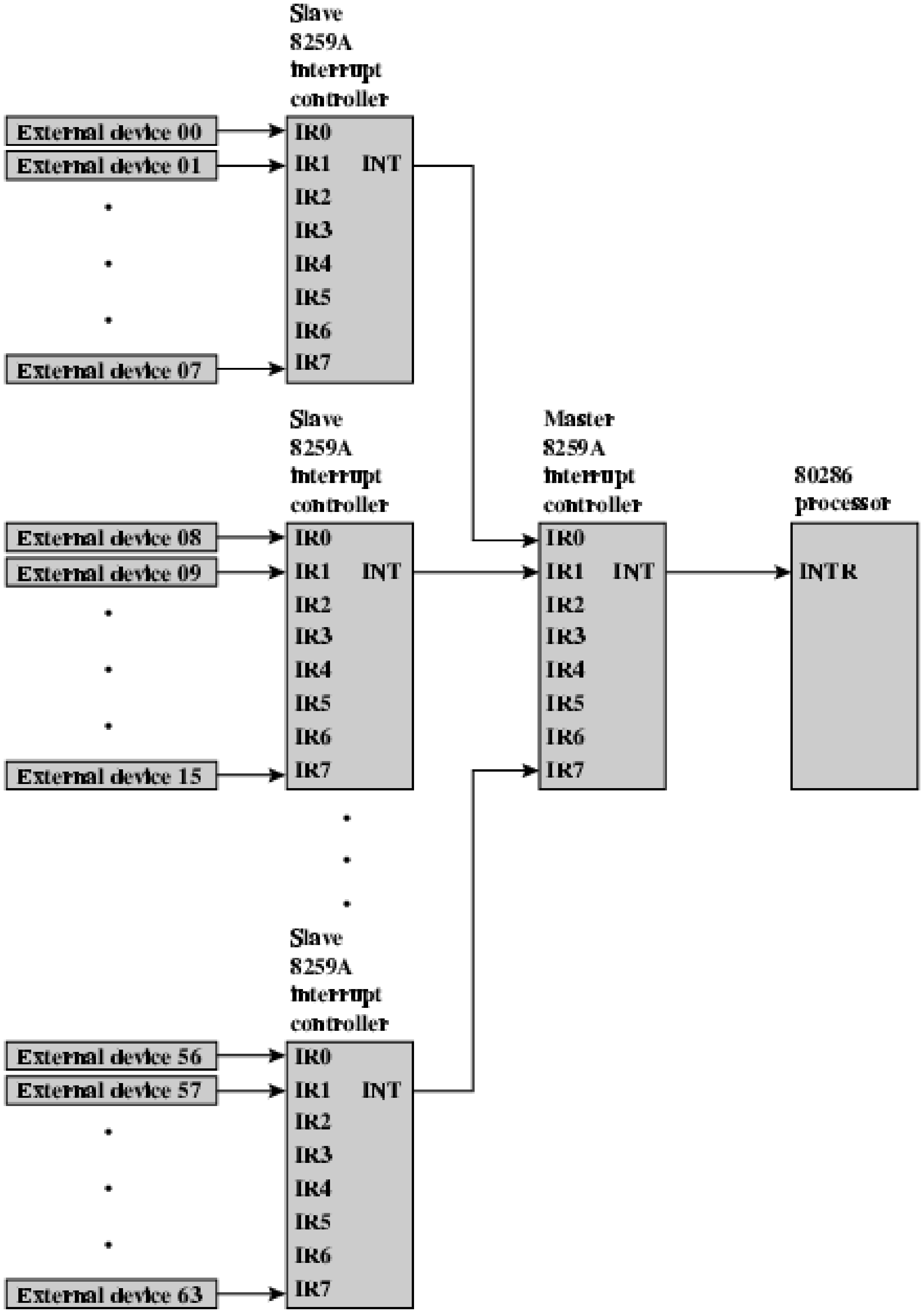
Sequence of Events

- 8259A accepts interrupts
- 8259A determines priority
- 8259A signals 8086 (raises INTR line)
- CPU Acknowledges
- 8259A puts correct vector on data bus
- CPU processes interrupt

ISA Bus Interrupt System

- ISA bus chains two 8259As together
- Link is via interrupt 2
- Gives 15 lines
  - 16 lines less one for link
- IRQ 9 is used to re-route anything trying to use IRQ 2
  - Backwards compatibility
- Incorporated in chip set

82C59A Interrupt Controller



Direct Memory Access

- Interrupt driven and programmed I/O require active CPU intervention
  - Transfer rate is limited
  - CPU is tied up
- Inefficiency in block transfer
- Solution to transfer large volume of data ➔ DMA

DMA Function

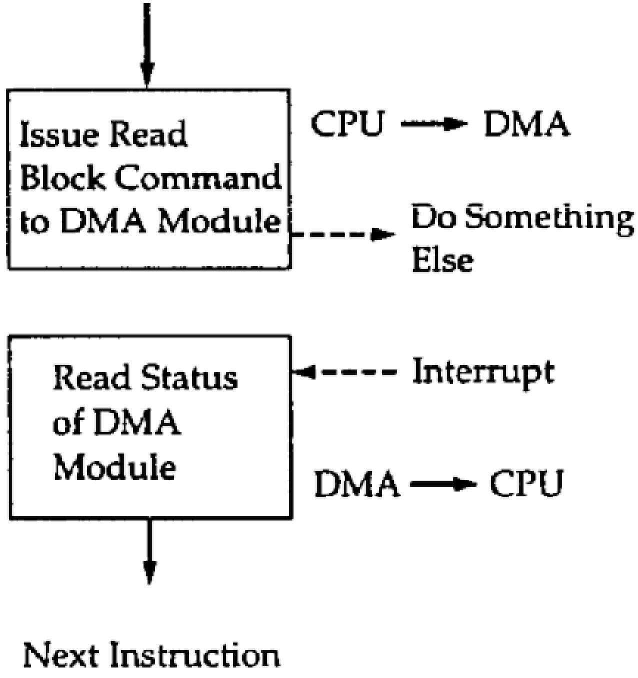
- Additional Module (hardware, DMA module) on bus
- DMA module takes over from CPU for I/O
  - DMA module uses bus only when CPU does not

  or

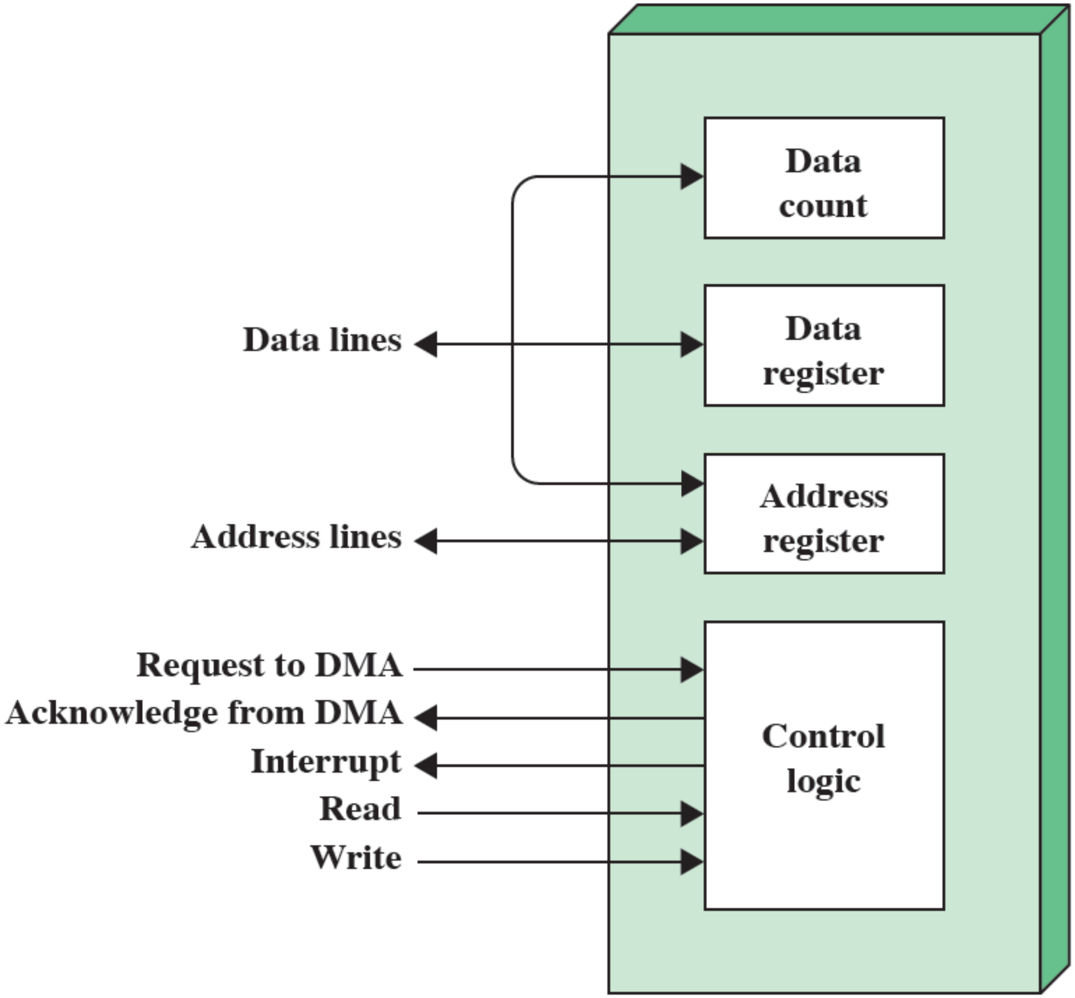
  - DMA module forces CPU to suspend temporarily – *cycle stealing (refer to Appendix in this file)*

DMA Operation

- CPU tells DMA controller
  - Read/Write
  - Device address
  - Starting address of memory block for data
  - Amount of data to be transferred
- CPU carries on with other work
- DMA controller deals with transfer
- DMA controller sends interrupt when finished


Issue Read Block Command to DMA Module
CPU → DMA
Do Something Else
Read Status of DMA Module
Interrupt
DMA → CPU
Next Instruction

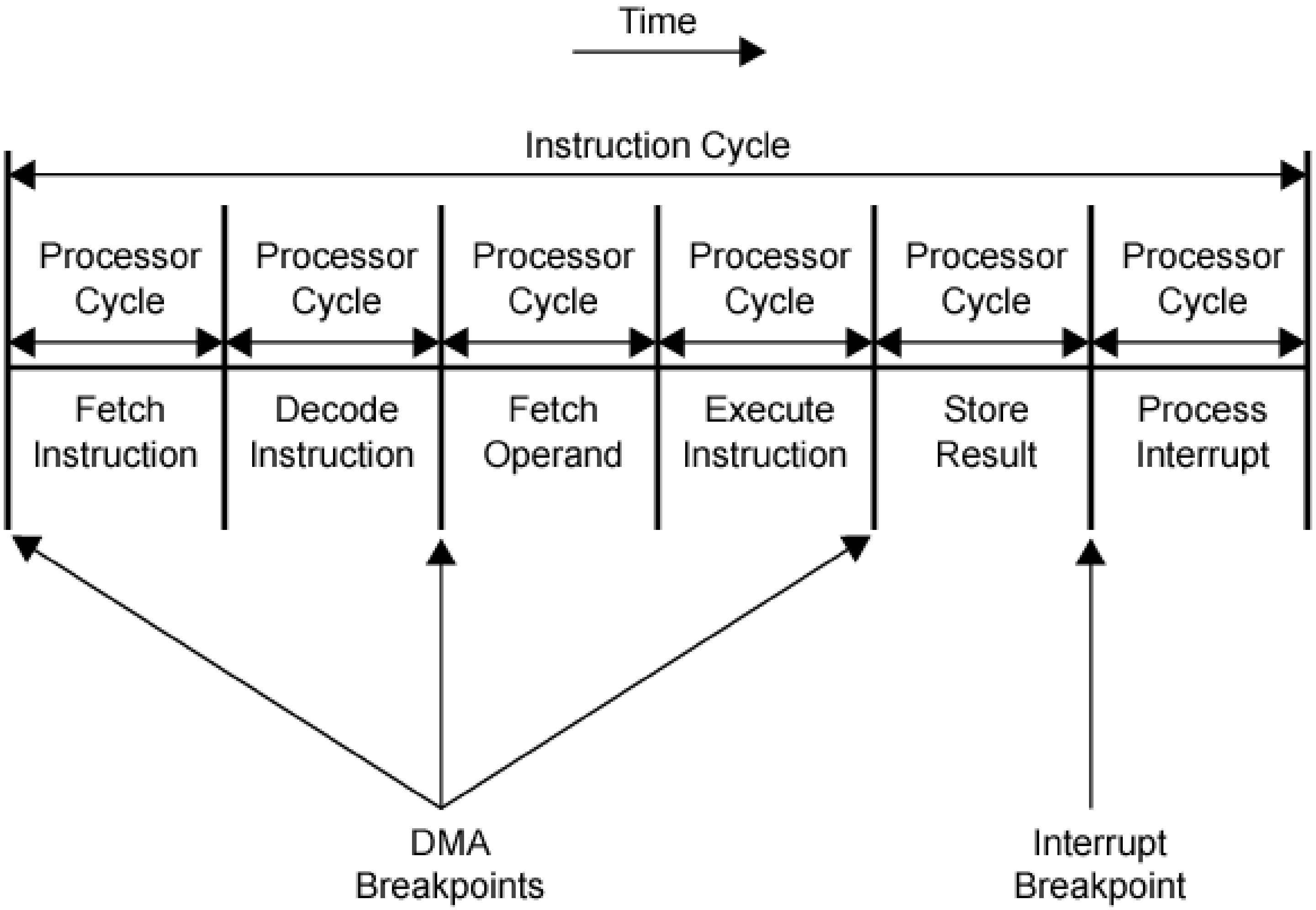
Typical DMA Module Diagram



DMA Transfer Cycle Stealing

- DMA controller takes over bus for a cycle
- Transfer of one word of data
- Not an interrupt
  - CPU does not switch context
- CPU suspended just before it accesses bus
  - i.e. before an operand or data fetch or a data write
- Slows down CPU but not as much as CPU doing transfer
- *(refer to Appendix in this file)*

DMA and Interrupt Breakpoints During an Instruction Cycle

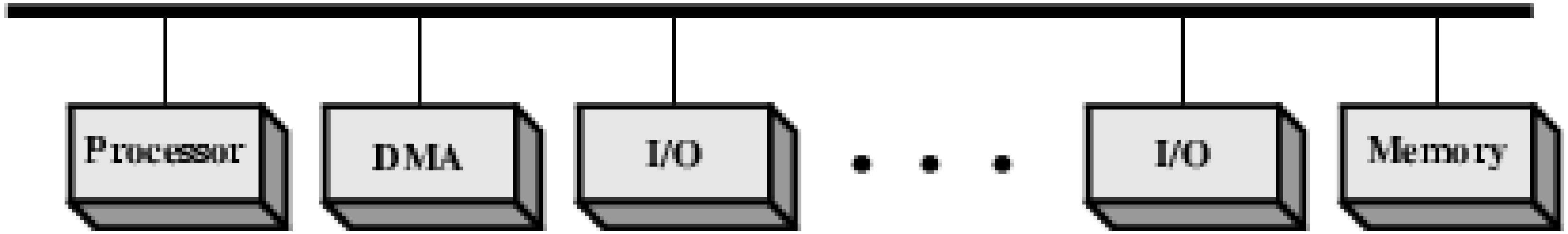


DMA동작을 위해 memory access 필요
이때 bus를 사용하므로 processor의 bus 사용은 supspended됨
(Fetch, Store는 bus사용하는 processor의 cycle)

Aside

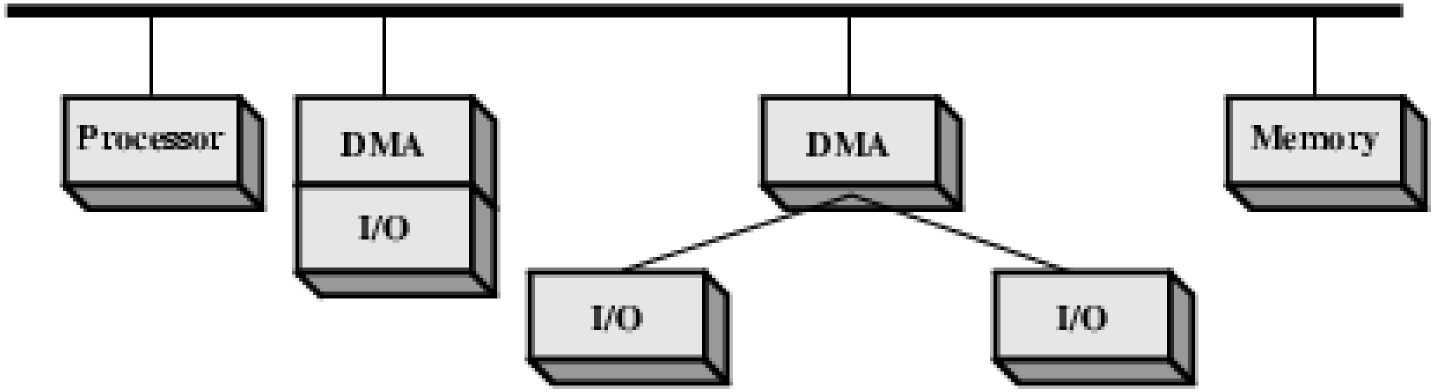
- What effect does caching memory have on DMA?
- What about on board cache?
- Hint: how much are the system buses available?

DMA Configurations (1)



- Single Bus, Detached DMA controller
- Each transfer uses bus twice
  - I/O to DMA then DMA to memory
- CPU is suspended twice

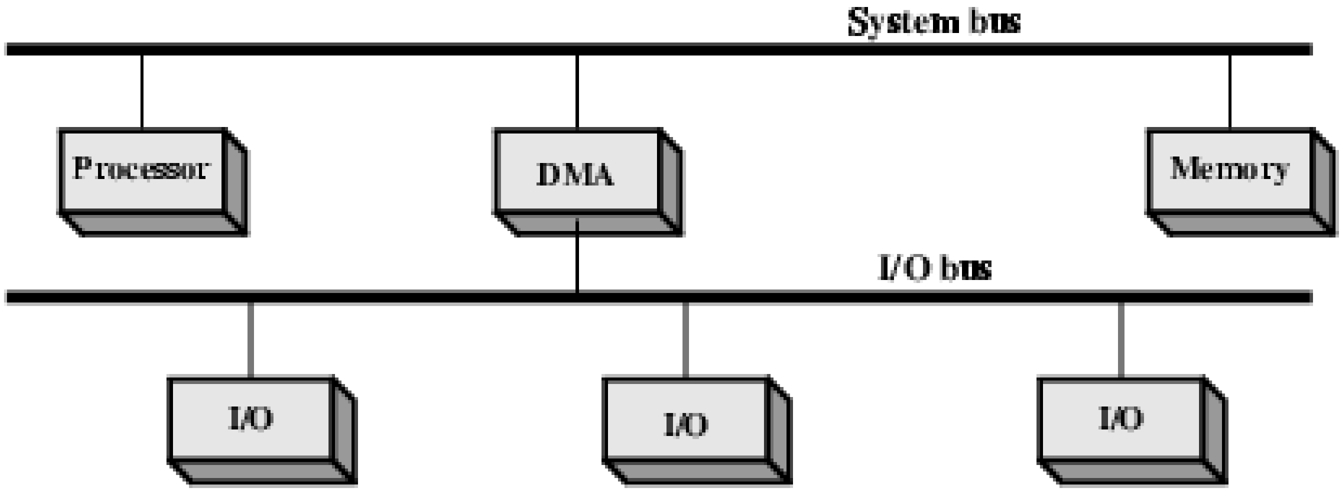
DMA Configurations (2)



(b) Single-bus, Integrated DMA-I/O

- Single Bus, Integrated DMA controller
- Controller may support >1 device
- Each transfer uses bus once
  - —DMA to memory
- CPU is suspended once

DMA Configurations (3)



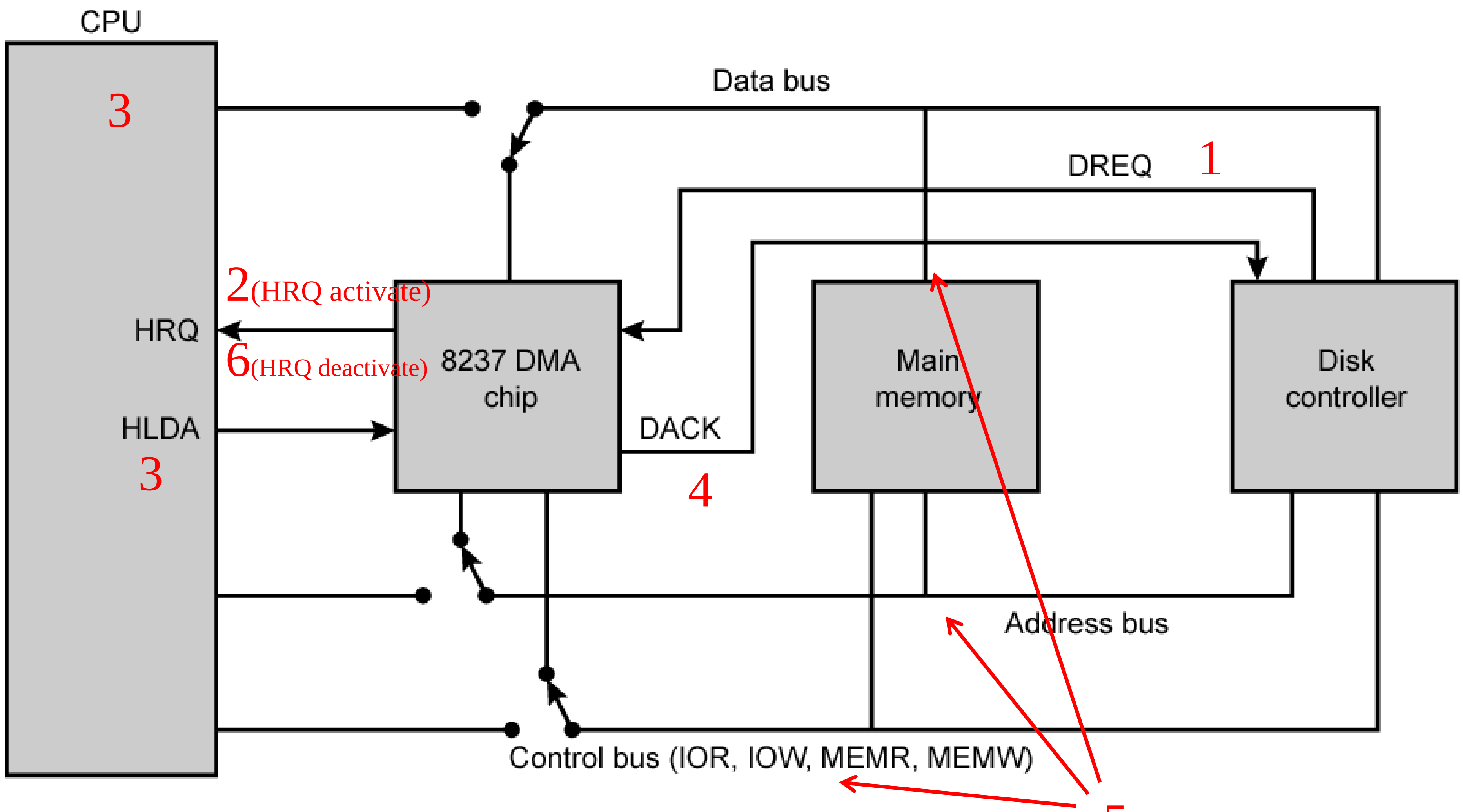
(c) I/O bus

- Separate I/O Bus
- Bus supports all DMA enabled devices
- Each transfer uses bus once
    - —DMA to memory
- CPU is suspended once

Intel 8237A DMA Controller

- Interfaces to 80x86 family and DRAM
- When DMA module needs buses it sends HOLD signal to processor
- CPU responds HLDA (hold acknowledge)
  - DMA module can use buses
- E.g. transfer data from memory to disk
  1. Device requests service of DMA by pulling DREQ (DMA request) high
  2. DMA puts high on HRQ (hold request),
  3. CPU finishes present bus cycle (not necessarily present instruction) and puts high on HDLA (hold acknowledge). HOLD remains active for duration of DMA
  4. DMA activates DACK (DMA acknowledge), telling device to start transfer
  5. DMA starts transfer by putting address of first byte on address bus and activating MEMR; it then activates IOW to write to peripheral. DMA decrements counter and increments address pointer. Repeat until count reaches zero
  6. DMA deactivates HRQ, giving bus back to CPU

8237 DMA Usage of Systems Bus



DACK = DMA acknowledge
DREQ = DMA request
HLDA = HOLD acknowledge
HRQ = HOLD request

Fly-By

- When DMA is using buses, processor is in idle
- Processor is using bus, DMA is in idle
- The fly-by DMA controller
  - Data does not pass through and is not stored in DMA chip
    - DMA only between I/O port and memory
    - Not between two I/O ports or two memory locations
  - Can do memory to memory via register
  - 8237 contains four DMA channels
    - Programmed independently
    - Any one active
    - Numbered 0, 1, 2, and 3

Q&A

Direct Cache Access (DCA)

- DMA is not able to scale to meet the increased demand due to dramatic increases in data rates for network I/O
- Demand is coming primarily from the widespread deployment of 10-Gbps and 100-Gbps Ethernet switches to handle massive amounts of data transfer to and from database servers and other high-performance systems
- Another source of traffic comes from Wi-Fi in the gigabit range
- Network Wi-Fi devices that handle 3.2 Gbps and 6.76 Gbps are becoming widely available and producing demand on enterprise systems

Evolution of the I/O Function

1. The CPU directly controls a peripheral device.
2. A controller or I/O module is added. The CPU uses programmed I/O without interrupts.
3. Same configuration as in step 2 is used, but now interrupts are employed. The CPU need not spend time waiting for an I/O operation to be performed, thus increasing efficiency.
4. The I/O module is given direct access to memory via DMA. It can now move a block of data to or from memory without involving the CPU, except at the beginning and end of the transfer.
5. The I/O module is enhanced to become a processor in its own right, with a specialized instruction set tailored for I/O
6. The I/O module has a local memory of its own and is, in fact, a computer in its own right. With this architecture a large set of I/O devices can be controlled with minimal CPU involvement.

Universal Serial Bus (USB)

- Widely used for peripheral connections
- Is the default interface for slower speed devices
- Commonly used high-speed I/O
- Has gone through multiple generations
  - USB 1.0
    - Defined a *Low Speed* data rate of 1.5 Mbps and a *Full Speed* rate of 12 Mbps
  - USB 2.0
    - Provides a data rate of 480 Mbps
  - USB 3.0
    - Higher speed bus called *SuperSpeed* in parallel with the USB 2.0 bus
    - Signaling speed of *SuperSpeed* is 5 Gbps, but due to signaling overhead the usable data rate is up to 4 Gbps
  - USB 3.1
    - Includes a faster transfer mode called *SuperSpeed+*
    - This transfer mode achieves a signaling rate of 10 Gbps and a theoretical usable data rate of 9.7 Gbps
- Is controlled by a root host controller which attaches to devices to create a local network with a hierarchical tree topology

FireWire Serial Bus

- Was developed as an alternative to small computer system interface (SCSI) to be used on smaller systems, such as personal computers, workstations, and servers
- Objective was to meet the increasing demands for high I/O rates while avoiding the bulky and expensive I/O channel technologies developed for mainframe and supercomputer systems
- IEEE standard 1394, for a High Performance Serial Bus
- Uses a daisy chain configuration, with up to 63 devices connected off a single port
- 1022 FireWire buses can be interconnected using bridges
- Provides for hot plugging which makes it possible to connect and disconnect peripherals without having to power the computer system down or reconfigure the system
- Provides for automatic configuration
- No terminations and the system automatically performs a configuration function to assign addresses

SCSI

- Small Computer System Interface
- A once common standard for connecting peripheral devices to small and medium-sized computers
- Has lost popularity to USB and FireWire in smaller systems
- High-speed versions remain popular for mass memory support on enterprise systems
- Physical organization is a shared bus, which can support up to 16 or 32 devices, depending on the generation of the standard
  - The bus provides for parallel transmission rather than serial, with a bus width of 16 bits on earlier generations and 32 bits on later generations
  - Speeds range from 5 Mbps on the original SCSI-1 specification to 160 Mbps on SCSI-3 U3

Thunderbolt

- Most recent and fastest peripheral connection technology to become available for general-purpose use
- Developed by Intel with collaboration from Apple
- The technology combines data, video, audio, and power into a single high-speed connection for peripherals such as hard drives, RAID arrays, video-capture boxes, and network interfaces
- Provides up to 10 Gbps throughput in each direction and up to 10 Watts of power to connected peripherals

InfiniBand

- I/O specification aimed at the high-end server market
- First version was released in early 2001
- Heavily relied on by IBM zEnterprise series of mainframes
- Standard describes an architecture and specifications for data flow among processors and intelligent I/O devices
- Has become a popular interface for storage area networking and other large storage configurations
- Enables servers, remote storage, and other network devices to be attached in a central fabric of switches and links
- The switch-based architecture can connect up to 64,000 servers, storage systems, and networking devices

PCI Express

- High-speed bus system for connecting peripherals of a wide variety of types and speeds

SATA

- Serial Advanced Technology Attachment
- An interface for disk storage systems
- Provides data rates of up to 6 Gbps, with a maximum per device of 300 Mbps
- Widely used in desktop computers and in industrial and embedded applications

Ethernet

- Predominant wired networking technology
- Has evolved to support data rates up to 100 Gbps and distances from a few meters to tens of km
- Has become essential for supporting personal computers, workstations, servers, and massive data storage devices in organizations large and small
- Began as an experimental bus-based 3-Mbps system
- Has moved from bus-based to switch-based
  - Data rate has periodically increased by an order of magnitude
  - There is a central switch with all of the devices connected directly to the switch

  - Ethernet systems are currently available at speeds up to 100 Gbps

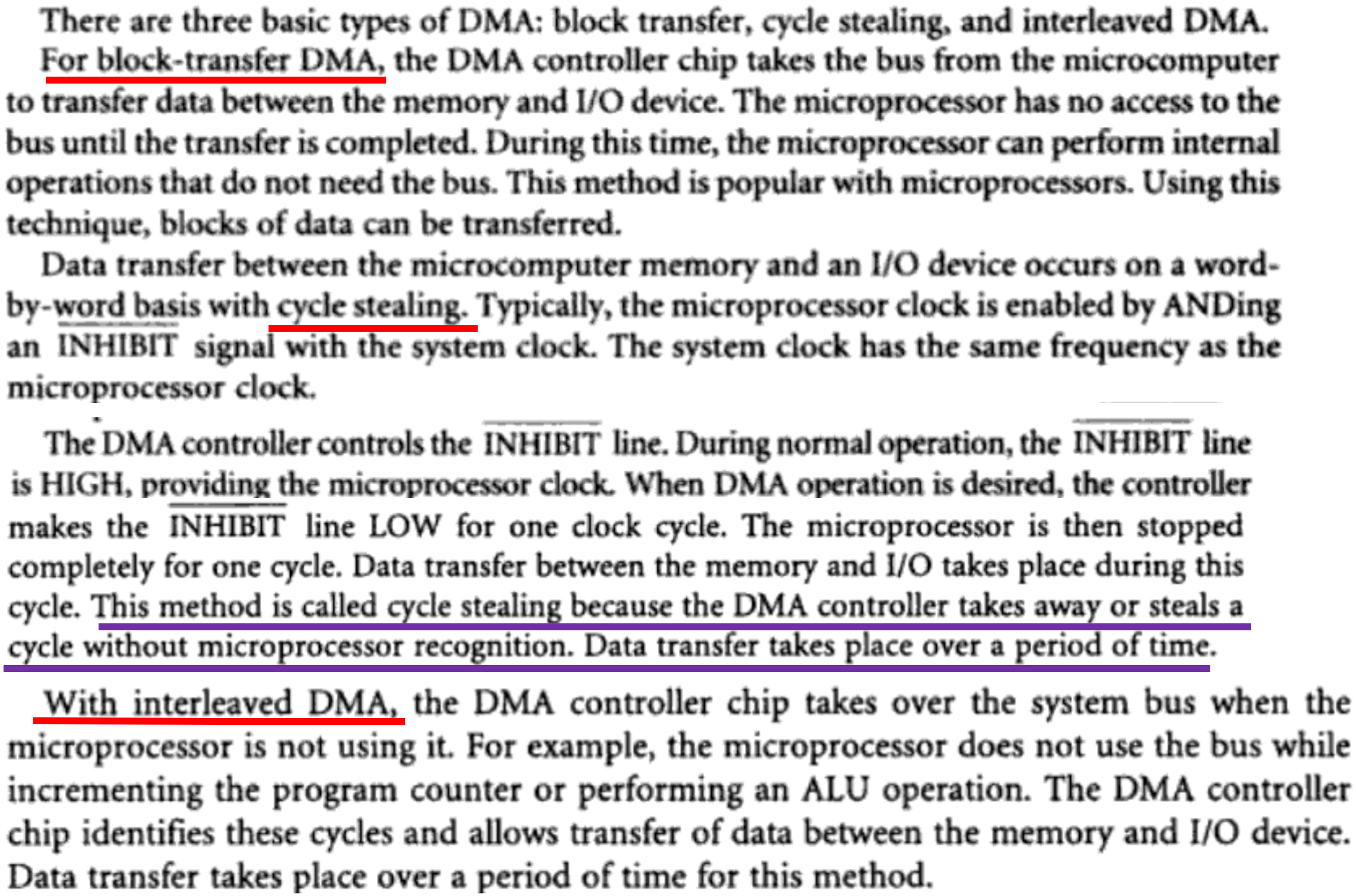
Wi-Fi

- Is the predominant wireless Internet access technology
- Now connects computers, tablets, smart phones, and other electronic devices such as video cameras TVs and thermostats
- In the enterprise has become an essential means of enhancing worker productivity and network effectiveness
- Public hotspots have expanded dramatically to provide free Internet access in most public places
- As the technology of antennas, wireless transmission techniques, and wireless protocol design has evolved, the IEEE 802.11 committee has been able to introduce standards for new versions of Wi-Fi at higher speeds

— Current version is 802.11ac (2014) with a maximum data rate of 3.2 Gbps

Appendix 1: Three Types of DMA

- 3 Types of DMA : Block transfer, Cycle Stealing, Interleaved DMA



There are three basic types of DMA: block transfer, cycle stealing, and interleaved DMA. For block-transfer DMA, the DMA controller chip takes the bus from the microcomputer to transfer data between the memory and I/O device. The microprocessor has no access to the bus until the transfer is completed. During this time, the microprocessor can perform internal operations that do not need the bus. This method is popular with microprocessors. Using this technique, blocks of data can be transferred.

Data transfer between the microcomputer memory and an I/O device occurs on a word-by-word basis with cycle stealing. Typically, the microprocessor clock is enabled by ANDing an $\overline{\text{INHIBIT}}$ signal with the system clock. The system clock has the same frequency as the microprocessor clock.

The DMA controller controls the $\overline{\text{INHIBIT}}$ line. During normal operation, the $\overline{\text{INHIBIT}}$ line is HIGH, providing the microprocessor clock. When DMA operation is desired, the controller makes the $\overline{\text{INHIBIT}}$ line LOW for one clock cycle. The microprocessor is then stopped completely for one cycle. Data transfer between the memory and I/O takes place during this cycle. This method is called cycle stealing because the DMA controller takes away or steals a cycle without microprocessor recognition. Data transfer takes place over a period of time.

With interleaved DMA, the DMA controller chip takes over the system bus when the microprocessor is not using it. For example, the microprocessor does not use the bus while incrementing the program counter or performing an ALU operation. The DMA controller chip identifies these cycles and allows transfer of data between the memory and I/O device. Data transfer takes place over a period of time for this method.

인용) Microprocessors and Microcomputer-Based System Design by Mohamed Rafiquzzaman, p.32