Computer Organization & Architecture

Chapter 3 – Basic Input/Output

Zhang Yang 张杨

cszyang@scut.edu.cn

Autumn 2025

Content of this lecture

- 3.1 Accessing I/O Devices
  - I/O Devices
  - I/O Device Interface
  - Program controlled I/O
  - Summary

I/O Devices (1)

- Link to the outside world
- Types
  - Human readable
    - Monitor
    - Printer
  - Machine readable
    - Disk
    - CD ROM
    - Scanner
  - Communication
    - Modem
    - Network Interface Card (NIC)

I/O Devices (2)

- Examples of I/O Devices

Device	Input/Output	Date Rate (Kbytes/s)
Keyboard	Input	0.01
Mouse	Input	0.02
Voice input (microphone)	Input	0.02
Scanner	Input	200
Voice output (speaker)	Output	0.5
Dot-matrix printer	Output	1
Laser printer	Output	100
Graphics display	Output	30,000
Local area network	Input/output	200 – 20,000
Optical disk	Storage (I/O)	500
Magnetic tape	Storage (I/O)	2,000
Magnetic disk	Storage (I/O)	2,000

Accessing I/O Devices (1)

- Computer system components communicate through an interconnection network.
- Address space and memory access concepts from preceding chapter also apply here.
- Locations associated with I/O devices are accessed with Load and Store instructions.
- Locations implemented as I/O registers within same address space → memory-mapped I/O

Accessing I/O Devices (2)

- Figure 3.1

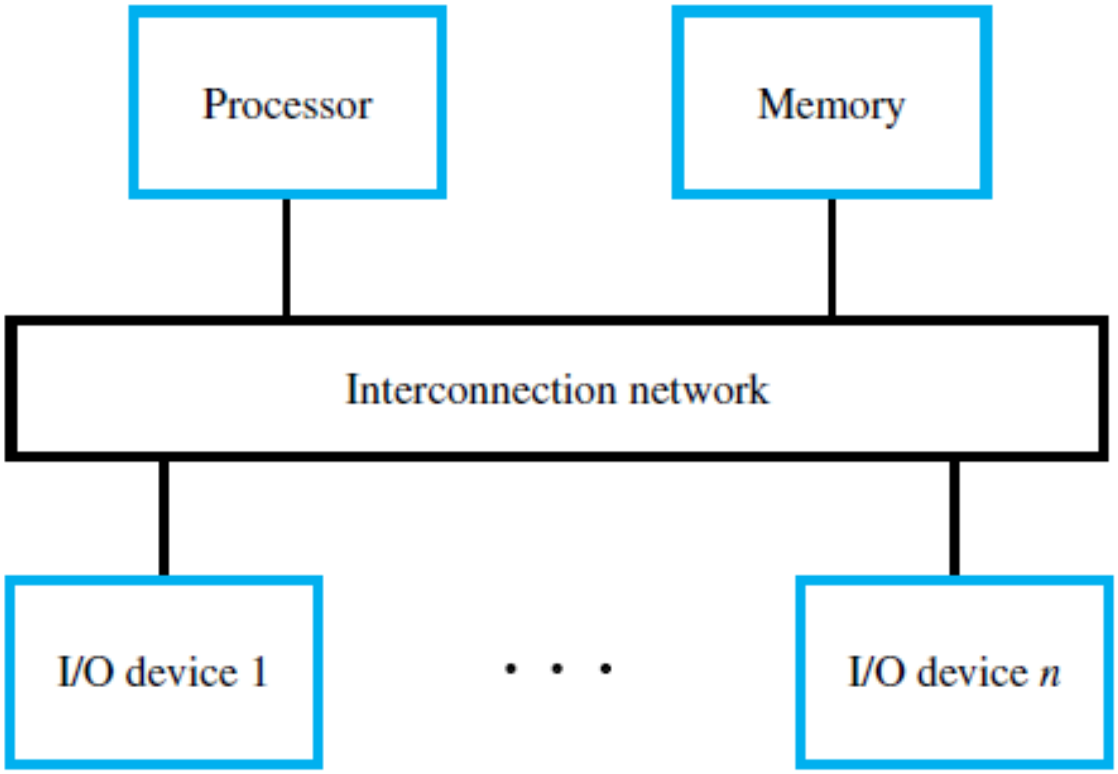


Figure 3.1 A computer system.

Accessing I/O Devices (3)

- 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 (interfaces)
- Input/Output Module (Interface)
  - Interface to CPU and Memory
  - Interface to one or more peripherals

I/O Device Interface (1)

- Constituents of I/O Interface
  - The address decoder, the data and status registers, and the control circuitry constitute the device's interface circuit.

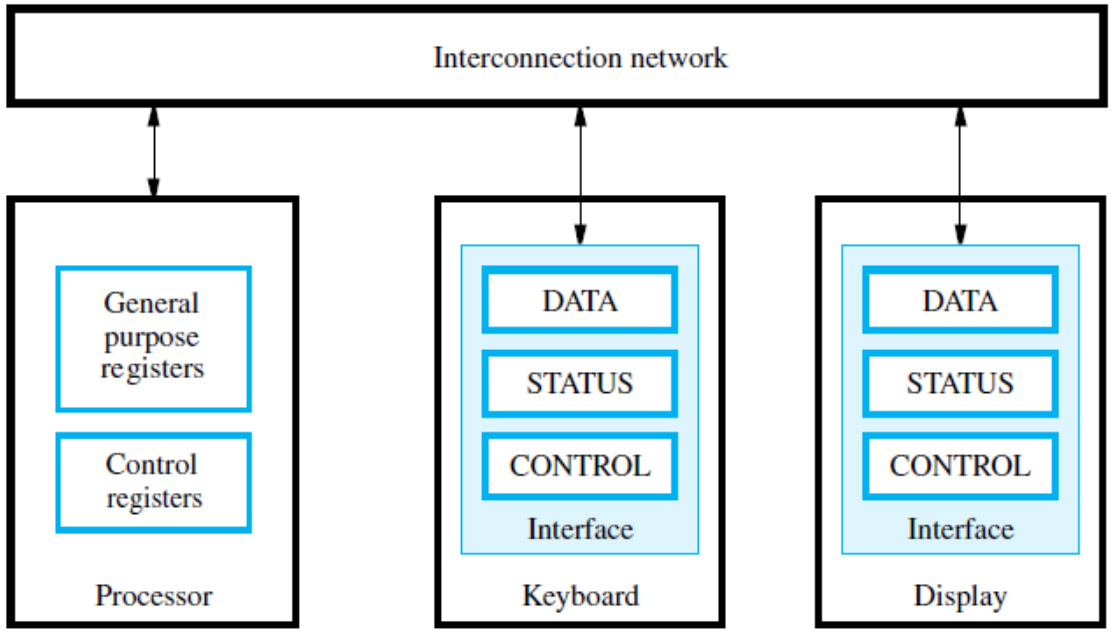


Figure 3.2 The connection for processor, keyboard, and display.

I/O Device Interface (2)

- Functions of interface
  - Device communication
    - Control: Determine whether device will send or receive data.
    - Data: Actual data transferred
    - Status: Device is ready, Data is available for transfer or error.
  - Data buffering
    - Must handle varying data rates from memory and I/O device.
    - Buffer data so that no one gets tied down.

I/O Device Interface (3)

- Functions of interface (ctd.)
  - Error detection
    - Report to CPU mechanical failure e.g. paper jam
    - Report data transmission error
  - Control and Signaling
    - Co-ordinates data transfer based on communication method with the processor.
  - Communication with CPU
    - Programmed a.k.a Polling
    - Interrupt driven
    - Direct Memory Access (DMA)

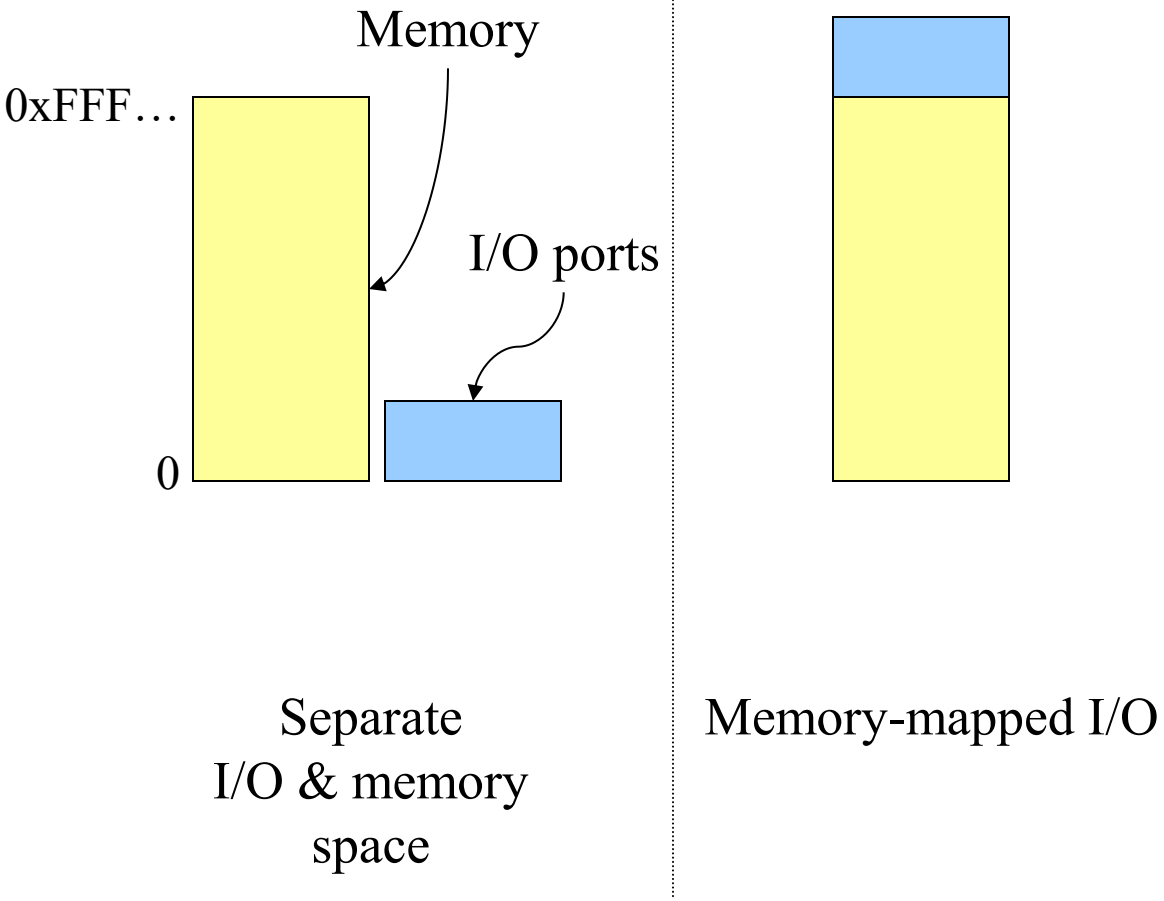
I/O Device Interface (4)

- Addressing Modes of I/O Devices(Interface)
  - Memory-mapped I/O (e.g., MIPS)
    - I/O devices and the memory share the same address space.
    - Some memory address values are used to refer to peripheral device interface registers.
    - I/O looks just like memory read/write.
    - Example Load R2, DATAIN
      - DATAIN is the address of the input buffer associated with the keyboard.
    - Advantage
      - Any machine instruction that can access memory can be used to transfer data to or from an I/O device.

I/O Device Interface (5)

- Addressing Modes of I/O Devices(Interface) (ctd.)
  - Memory-mapped I/O (ctd.)
    - Disadvantage
      - Valuable memory address space is used up.
  - Isolated I/O (e.g., Pentium)
    - Use special In and Out instructions to perform I/O transfers.
    - Advantage
      - I/O devices deal with few address lines.
    - Note
      - A separate I/O address space does not necessarily mean that the I/O address lines are physically separate from the memory address lines.

I/O Device Interface (6)



Summary

- 知识点：I/O Device Interface
  - Constituents of I/O Interface
  - Functions of interface
  - Addressing mode of I/O Interface
    - Memory-mapped I/O
    - Isolated I/O

What is Program-Controlled I/O

- Program-controlled I/O
  - CPU executes a program that communicates with I/O module
    - Read/write commands
    - Sensing status
    - Transferring data
  - CPU waits for I/O module to complete operation
- Disadvantage
  - It wastes CPU time

Example (1)

- Example: Consider a task that reads characters typed on a keyboard, stores these data in the memory, and displays the same characters on a display screen.

Example (2)

- The difference in speed between the processor and I/O devices creates the need for mechanisms to synchronize the transfer of data between them.
  - On input, the processor waits for a signal indicating that a character key has been struck and that its code is available in some buffer register associated with the keyboard. Then the processor proceeds to read that code.
  - On output, the processor sends the first character and then waits for a signal from the display that the character has been received. It then sends the second character, and so on.

Example (3)

- Signaling Protocol for I/O Devices
  - Assume that the I/O devices have a way to send a '*ready*' signal to the processor.
  - For keyboard, indicates character can be read so processor uses Load to access data register.
  - For display, indicates character can be sent so processor uses Store to access data register.
  - The '*ready*' signal in each case is a status flag in status register that is polled by processor.

Example (4)

- Example I/O Registers
  - For sample I/O programs that follow, assume specific addresses & bit positions for registers.
  - Registers are 8 bits in width and word-aligned.
  - For example, keyboard has KIN status flag in bit b_1 of KBD_STATUS reg. at address 0x4004.
  - Processor polls KBD_STATUS register, checking whether KIN flag is 0 or 1.
  - If KIN is 1, processor reads KBD_DATA register.

Example (5)

- Keyboard&Display Interface Register Organization

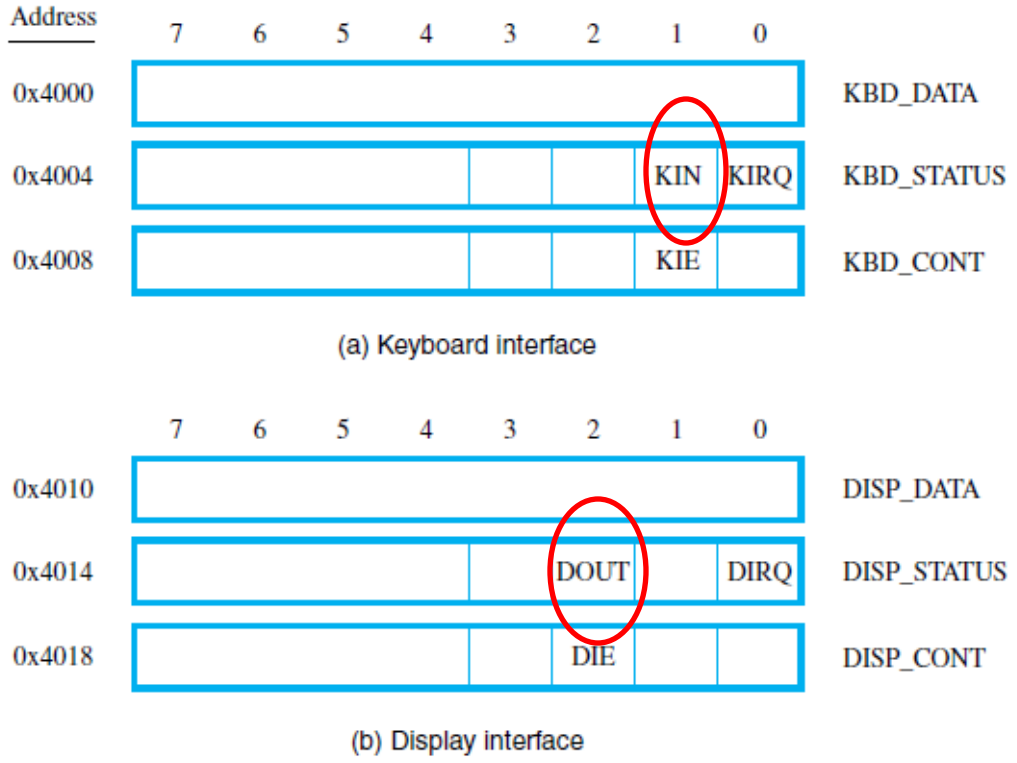


Figure 3.3 Registers in the keyboard and display interfaces.

Example (6)

- Wait Loop for Polling I/O Status
  - Program-controlled I/O implemented with a wait loop for polling keyboard status register:

```
READWAIT: LoadByte    R4, KBD_STATUS
          And         R4, R4, #2
          Branch_if_[R4]=0   READWAIT
          LoadByte    R5, KBD_DATA
```

  - Keyboard circuit places character in KBD_DATA and sets KIN flag in KBD_STATUS.
  - Circuit clears KIN flag when KBD_STATUS read.

Example (7)

- Wait Loop for Polling I/O Status
  - Similar wait loop for display device:

```
WRITEWAIT:LoadByte  R4,DISP_STATUS
          And       R4, R4, #4
          Branch_if_[R4]=0  WRITEWAIT
          StoreByte  R5, DISP_DATA
```

  - Display circuit sets DOUT flag in DISP_STATUS after previous character has been displayed.
  - Circuit automatically clears DOUT flag when DISP_STATUS register is read.

Example (8)

- RISC- and CISC-style I/O Programs
  - Consider complete programs that use polling to read, store, and display a line of characters.
  - Each keyboard character *echoed* to display.
  - Program finishes when carriage return (CR) character is entered on keyboard.
  - LOC is address of first character in stored line.
  - CISC has TestBit, CompareByte instructions as well as auto-increment addressing mode.

Example (9)

- RISC Program

```
        Move                R2, #LOC          Initialize pointer register R2 to point to the
                                                address of the first location in main memory
                                                where the characters are to be stored.
        MoveByte            R3, #CR           Load ASCII code for Carriage Return into R3.
READ:   LoadByte            R4, KBD_STATUS    Wait for a character to be entered.
        And                 R4, R4, #2        Check the KIN flag.
        Branch_if_[R4]=0    READ
        LoadByte            R5, KBD_DATA      Read the character from KBD_DATA
                                                (this clears KIN to 0).
        StoreByte           R5, (R2)          Write the character into the main memory and
        Add                 R2, R2, #1          increment the pointer to main memory.
ECHO:   LoadByte            R4, DISP_STATUS   Wait for the display to become ready.
        And                 R4, R4, #4        Check the DOUT flag.
        Branch_if_[R4]=0    ECHO
        StoreByte           R5, DISP_DATA     Move the character just read to the display
                                                buffer register (this clears DOUT to 0).
        Branch_if_[R5]≠[R3] READ              Check if the character just read is the
                                                Carriage Return. If it is not, then
                                                branch back and read another character.
```

Example (10)

- CISC Program

```
        Move        R2, #LOC          Initialize pointer register R2 to point to the
                                        address of the first location in main memory
                                        where the characters are to be stored.
READ:   TestBit     KBD_STATUS, #1    Wait for a character to be entered
        Branch=0    READ              in the keyboard buffer KBD_DATA.
        MoveByte    (R2), KBD_DATA    Transfer the character from KBD_DATA into
                                        the main memory (this clears KIN to 0).
ECHO:   TestBit     DISP_STATUS, #2   Wait for the display to become ready.
        Branch=0    ECHO
        MoveByte    DISP_DATA, (R2)   Move the character just read to the display
                                        buffer register (this clears DOUT to 0).
        CompareByte (R2)+, #CR        Check if the character just read is CR
                                        (carriage return). If it is not CR, then
        Branch≠0    READ              branch back and read another character.
                                        Also, increment the pointer to store the
                                        next character.
```

Summary

- 知识点：Program-controlled I/O
  - Principle
  - Give the advantages and disadvantages of programmable I/O.

Content of this lecture

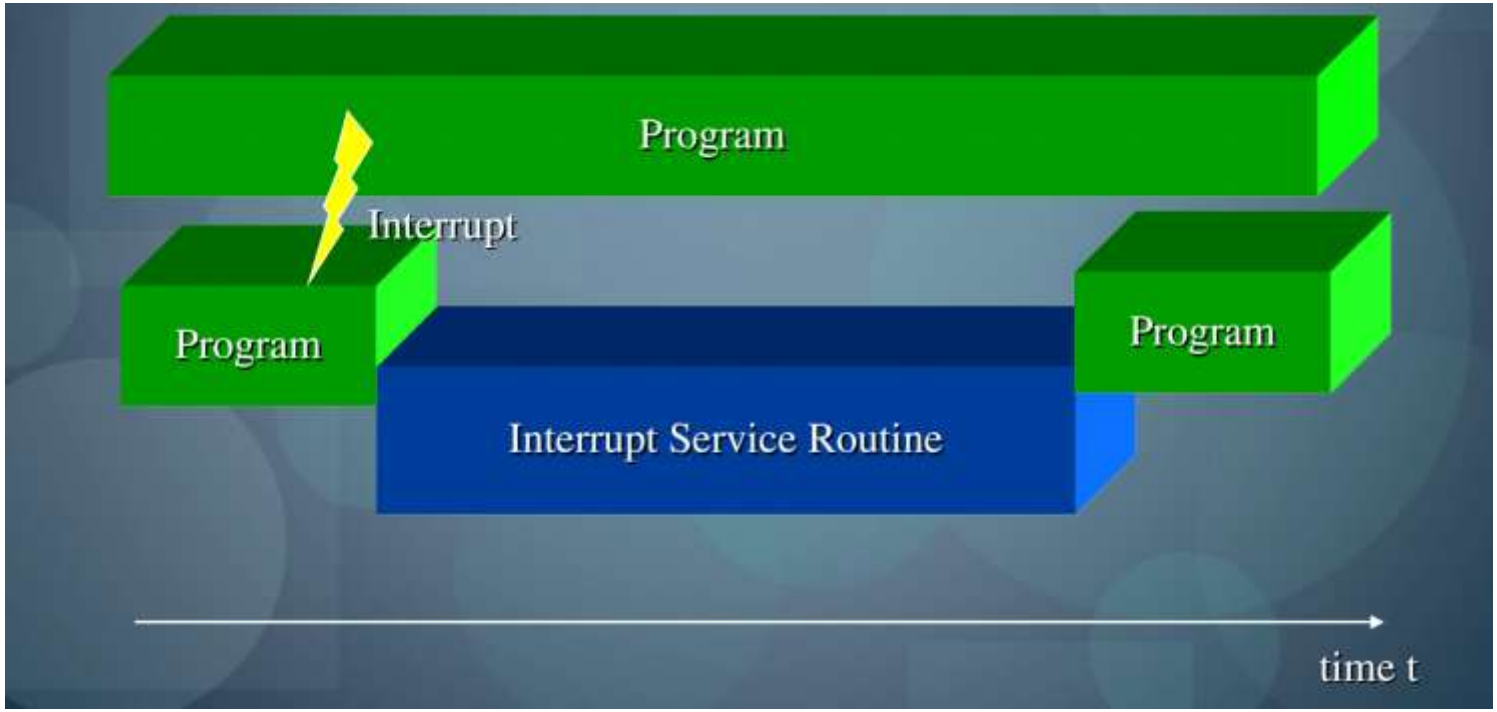
- 3.2 Interrupts
  - What is an interrupt?
  - Interrupt Example
  - Advantage of Interrupt-driven I/O
  - Concepts of Interrupt
  - Interrupt Processing
  - Types of Interrupt
  - Enabling and Disabling Interrupt
  - Summary

Content of this lecture

- 3.2 Interrupts (ctd.)
  - Identify Interrupt Source
  - Multi-level Interrupt
  - Simultaneous Interrupt
  - Summary

What is an Interrupt?

- An interrupt is an event that causes the processor to stop its current program execution and switch to performing an interrupt service routine.



Interrupt Example (1)

- Example 3.1
  - Consider a task that requires continuous extensive computations to be performed and the results to be displayed on a display device.
  - The displayed results must be updated every ten seconds. The 10- second intervals can be determined by a simple timer circuit, which generates an appropriate signal. The processor treats the timer circuit as an input device that produces a signal that can be interrogated. The timer circuit raise an interrupt request once every 10 seconds. In response, the processor displays the latest results.

Interrupt Example (2)

- Example 3.1(ctd.)
  - The task can be implemented with a program that consists of two routines, COMPUTE and DISPLAY.
  - The processor continuously executes the COMPUTE routine. When it receives an interrupt request from the timer, it suspends the execution of the COMPUTE routine and executes the DISPLAY routine which sends the latest results to the display device. Upon completion of the DISPLAY routine, the processor resumes the execution of the COMPUTE routine. Since the time needed to send the results to the display device is very small compared to the 10-second interval, the processor in effect spends almost all of its time executing the COMPUTE routine.

Interrupt Example (3)

- Example 3.1 (ctd.)

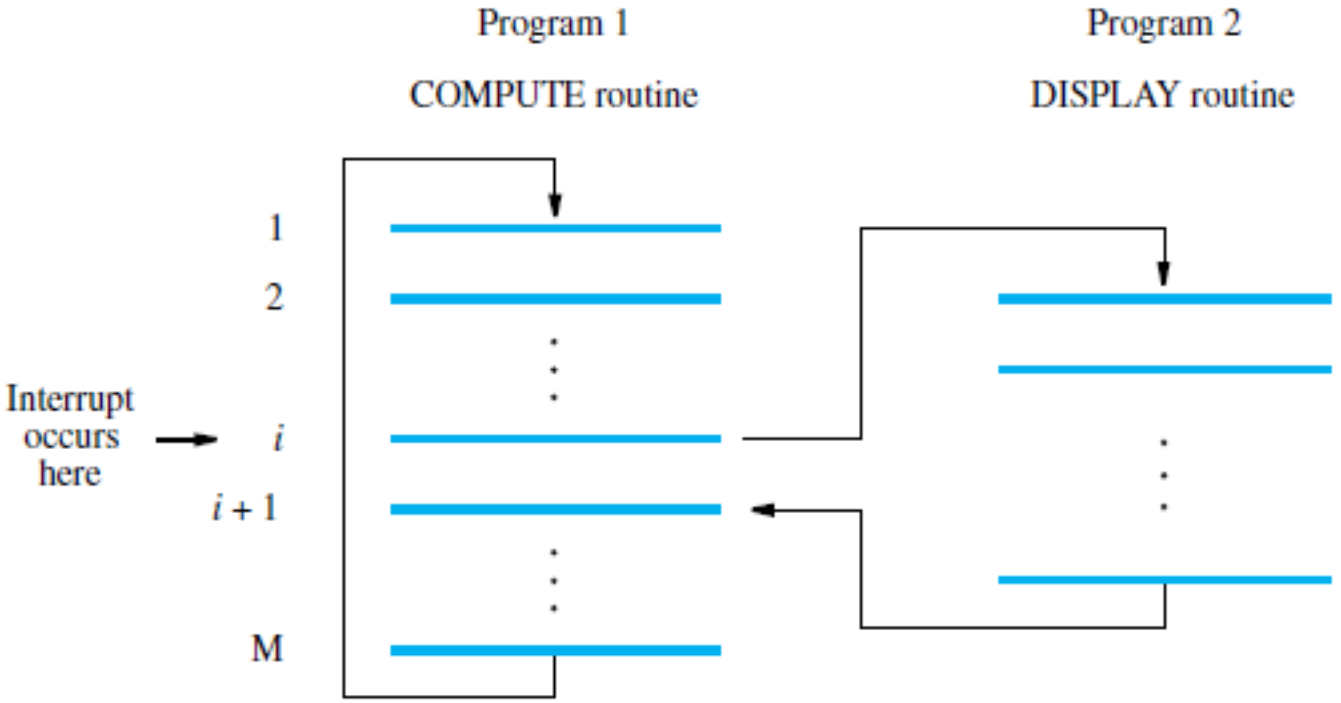


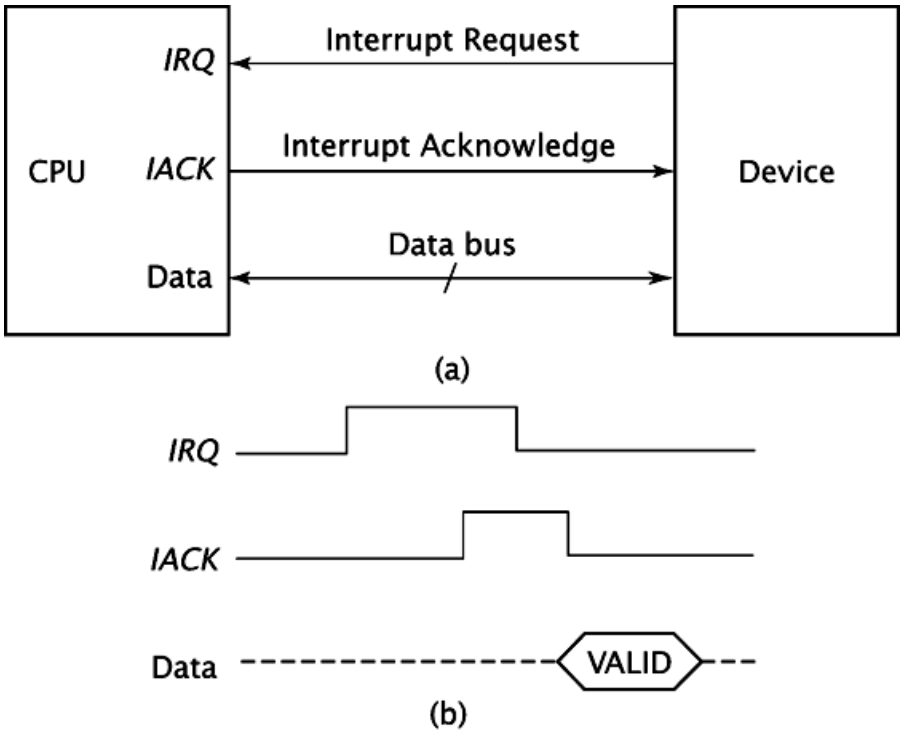
Figure 3.6 Transfer of control through the use of interrupts.

Advantages of Interrupt-driven I/O

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

Concepts of Interrupt (1)

- Interrupt Request
  - A signal that an I/O device sends to the processor through one of the bus control lines.
- Interrupt Acknowledge
  - The CPU issue the signal to acknowledges the interrupt.



Concepts of Interrupt (2)

- Interrupt-Service Routine (Interrupt Handler)
  - The routine executed in response to an interrupt request is called the interrupt-service routine.
- Interrupt Latency
  - The delay between the time an interrupt request is received and the start of execution of the interrupt-service routine.

Subroutine and ISR (1)

- A subroutine performs a function required by the program from which it is called. As such, potential changes to status information and contents of registers are anticipated.

Subroutine and ISR (2)

- An interrupt-service routine may not have any relation to the portion of the program being executed at the time the interrupt request is received. Therefore, before starting execution of the interrupt service routine, status information and contents of processor registers that may be altered in unanticipated ways during the execution of that routine must be saved. This saved information must be restored before execution of the interrupted program is resumed.

Interrupts vs Procedure Calls

Interrupts

- Initiated by both software and hardware.
- Can handle anticipated and unanticipated internal as well as external events.
- ISRs or interrupt handlers are memory resident.
- Use numbers to identify an interrupt service.
- (E)FLAGS register is saved automatically.

Procedure Calls

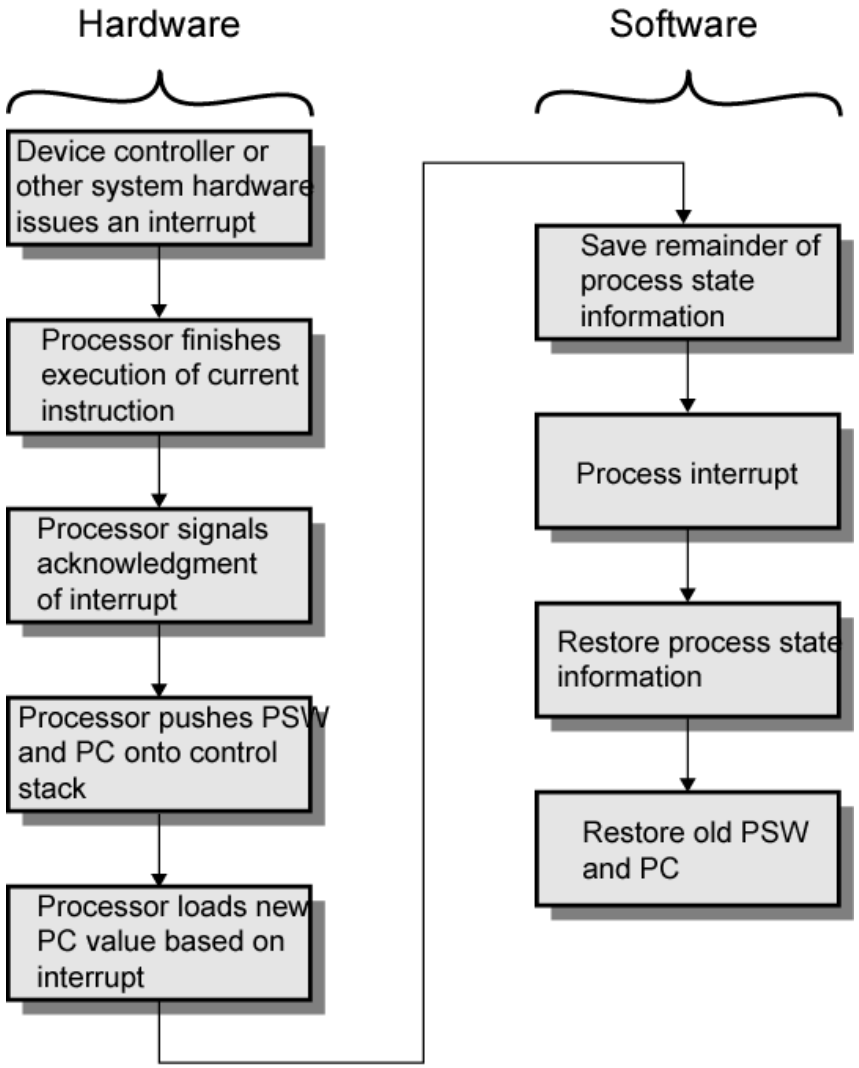
- Can only be initiated by Software.
- Can handle anticipated events that are coded into the program.
- Typically loaded along with the program.
- Use meaningful names to indicate their function.
- Do not save the (E)FLAGS register.

Interrupt Processing (1)

- Interrupt Processing
  - When an interrupt occurs (and is accepted), the execution of the current program is suspended
  - Must save PC, Registers in Process Control Block (PCB)
  - Interrupt service routine executes to service the interrupt

Interrupt Processing (2)

- Interrupt Processing Flowchart



Interrupt Processing (3)

- Details of Interrupt Processing
  - Processor Acknowledgement (part of handling interrupt)
    - Method 1: Processor issues interrupt-acknowledge signal.
    - Method 2: The execution of an instruction in the interrupt-service routine that access a status or data register in the device interface implicitly informs the device that its interrupt request has been recognized.
  - Save and Restore Information
    - The information that needs to be saved and restored typically includes the condition code flags and the contents of any registers used by both the interrupted program and the interrupt-service routine.
    - The task of saving and restoring information can be done automatically by the processor or by program instructions.

Interrupt Processing (4)

- Details of Interrupt Processing (ctd.)
  - Save and Restore Information (ctd.)
    - Most modern processors save only the minimum amount of information needed to maintain the integrity of program execution. (Typically the contents of PC and PSW)
    - Any additional information that needs to be saved must be saved by program instructions at the beginning of the interrupt-service routine and restored at the end of the routine.

Types of Interrupts (1)

- There are 2 types of interrupts, each with several different uses:
  - Hardware Interrupts
  - Software Interrupts
- Procedures for processing all types of interrupts are almost identical.
- Hardware Interrupts
  - The interrupt signal is raised by external device or hardware.
  - Used by CPU to interact with input/output devices; also used to initiate transfers.

Types of Interrupts (2)

- Hardware Interrupts (ctd.)
  - Hardware interrupts can be classified into two types
    - Maskable Interrupt: The hardware interrupts which can be delayed when a much highest priority interrupt has occurred to the processor.
    - Non Maskable Interrupt: The hardware which cannot be delayed and should process by the processor immediately.

Types of Interrupts (3)

- Software Interrupts
  - An interrupt that is caused by software, usually by a program in user mode.
  - Software interrupts can be classified into two types:
    - Normal Interrupts: the interrupts which are caused by the software instructions
    - Exception: unplanned interrupts while executing a program is called Exception.
      - E.g: Divide by zero exception, Arithmetic overflow, Page faults, Invalid instruction codes.

Enabling and Disabling Interrupts (1)

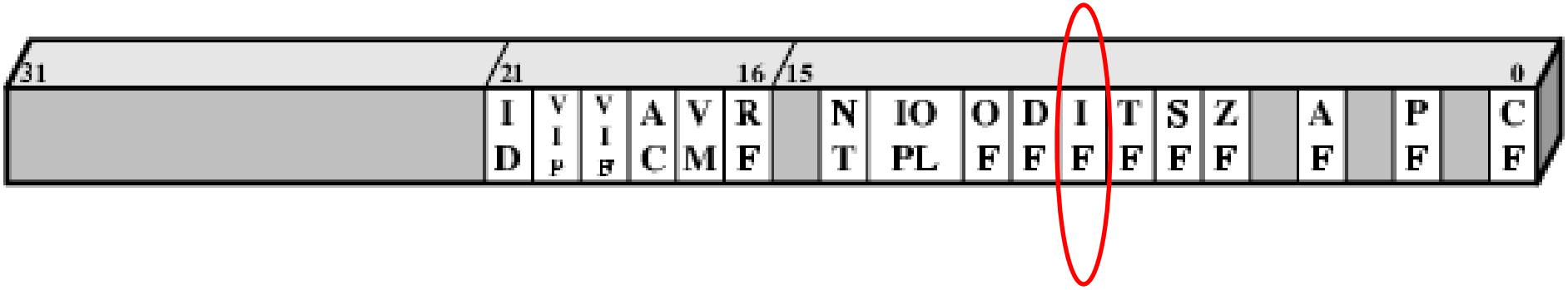
- Enabling and disabling interrupts are fundamental to all computers.
  - The interruption of program execution must be carefully controlled because interrupts can arrive at any time, they may alter the sequence of events from that envisaged by the programmer.
  - It may be necessary to guarantee that a particular sequence of instructions is executed to the end without interruption because the interrupt service routine may change some of the data used by the instructions in question.

Enabling and Disabling Interrupts (2)

- Interrupt Control at Processor End
  - Set Interrupt-enable bit in the Program Status register
  - Suitable for a simple processor with only one interrupt-request line.
  - Have the processor automatically disable interrupts (clear the Interrupt-enable bit) before starting the execution of the interrupt-service routine. (IE=0)
  - When a Return-from-interrupt instruction is executed, the contents of the PS are restored from the stack, setting the Interrupt-enable bit back to 1. (IE=1)

Enabling and Disabling Interrupts (3)

- Interrupt Control at Processor End (ctd.)
  - Example: Interrupt enable flag of x86 architecture



ID = Identification flag
VIP = Virtual interrupt pending
VIF = Virtual interrupt flag
AC = Alignment check
VM = Virtual 8086 mode
RF = Resume flag
NT = Nested task flag
IOPL = I/O privilege level
OF = Overflow flag

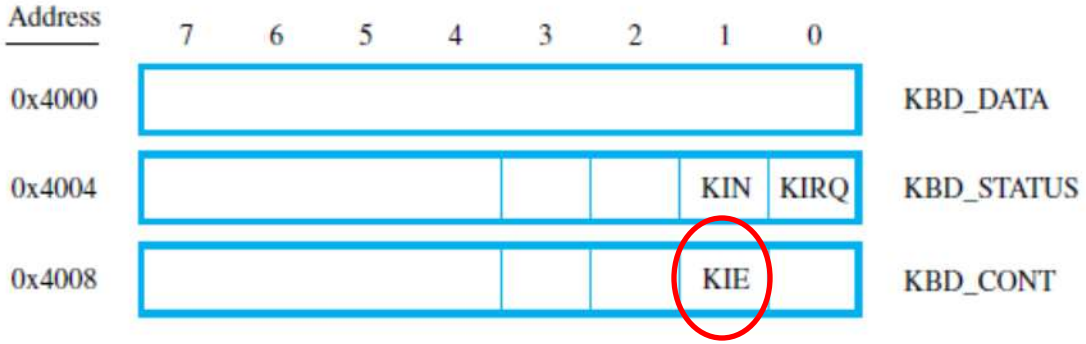
DF = Direction flag
IF = Interrupt enable flag
TF = Trap flag
SF = Sign flag
ZF = Zero flag
AF = Auxiliary carry flag
PF = Parity flag
CF = Carry flag

Enabling and Disabling Interrupts (4)

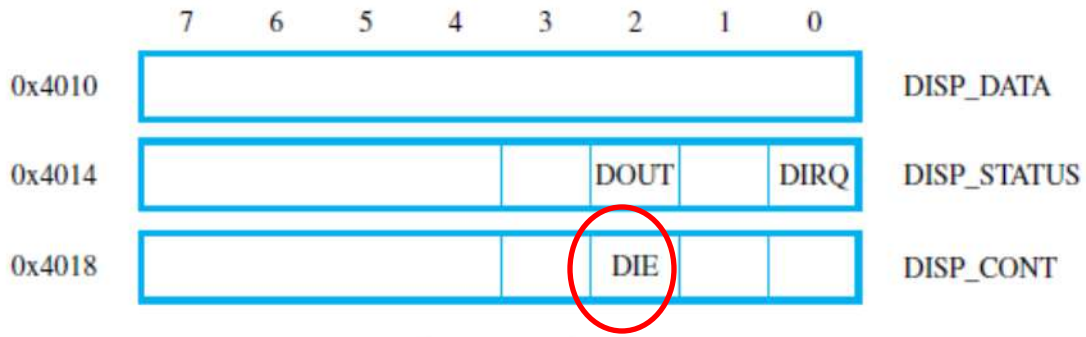
- Interrupt Control at I/O Device End
  - It is important to ensure that interrupt requests are generated only by those I/O devices that the processor is currently willing to recognize.
  - Need a mechanism in the interface circuits of individual devices to control whether a device is allowed to interrupt the processor.
  - An interrupt-enable bit in a control register determines whether the device is allowed to generate an interrupt request.

Enabling and Disabling Interrupts (5)

- Interrupt Control at I/O Device End (ctd.)
  - Example



(a) Keyboard interface



(b) Display interface

Figure 3.3 Registers in the keyboard and display interfaces.

Summary

- 知识点：Interrupt Concepts and Processing
  - What is interrupt?
  - Advantages of interrupt-driven I/O
  - Concepts of Interrupt
    - Interrupt request
    - Interrupt acknowledge
    - Interrupt handler
    - Interrupt latency
  - Subroutine and interrupt-service routine
  - Interrupt processing
  - Enabling and disabling interrupt
    - At processor end
    - At device end

单选题 2分

设置

1. In an interrupt process, the usage of saving PC is _____.

A to make CPU find the entry address of the interrupt service rountine

B to continue from the program breakpoint when returning from interrupt

C to make CPU and peripherals working in parallel

D to enable interrupt nesting

提交

单选题 2分

设置

2. During the processing of an interrupt, which of the following task is done by software?

A Signal acknowledgement of interrupt

B Push PSW and PC onto control stack

C Load new PC value based on interrupt

D Save remainder of process state information

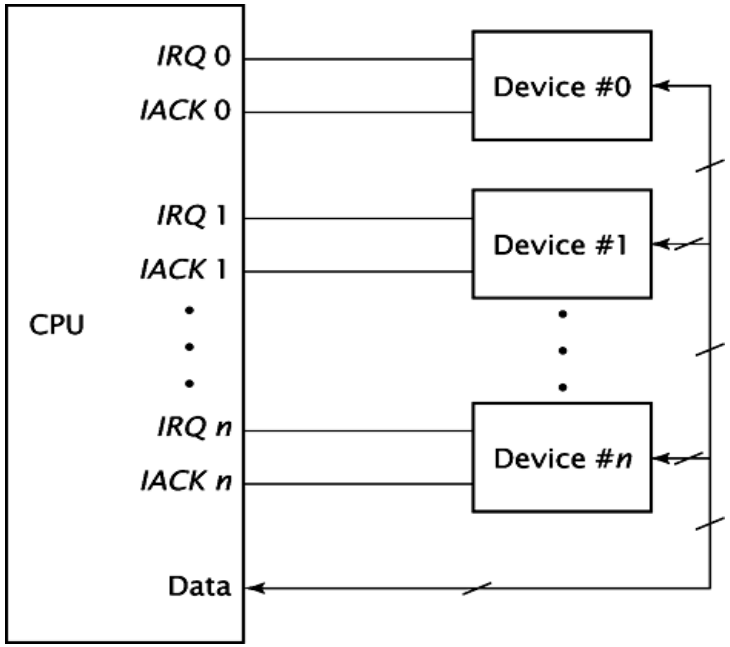
提交

Multiple Devices Interrupt System Design Issues

- How do you identify the module issuing the interrupt? How the processor obtain the starting address of the appropriate routine of different devices?—Identify Interrupt Source
- Should a device be allowed to interrupt the processor while another interrupt is being serviced? —Multi-level Interrupt
- How should two or more simultaneous interrupt requests be handled? —Simultaneous Interrupt

Identify Interrupt Source (1)

- Case 1: Multiple Interrupt-request Lines
  - Provide multiple interrupt lines between the processor and the I/O interface.
  - Even if multiple lines are used, it is likely that each line will have multiple I/O interface attached to it.



Identify Interrupt Source (2)

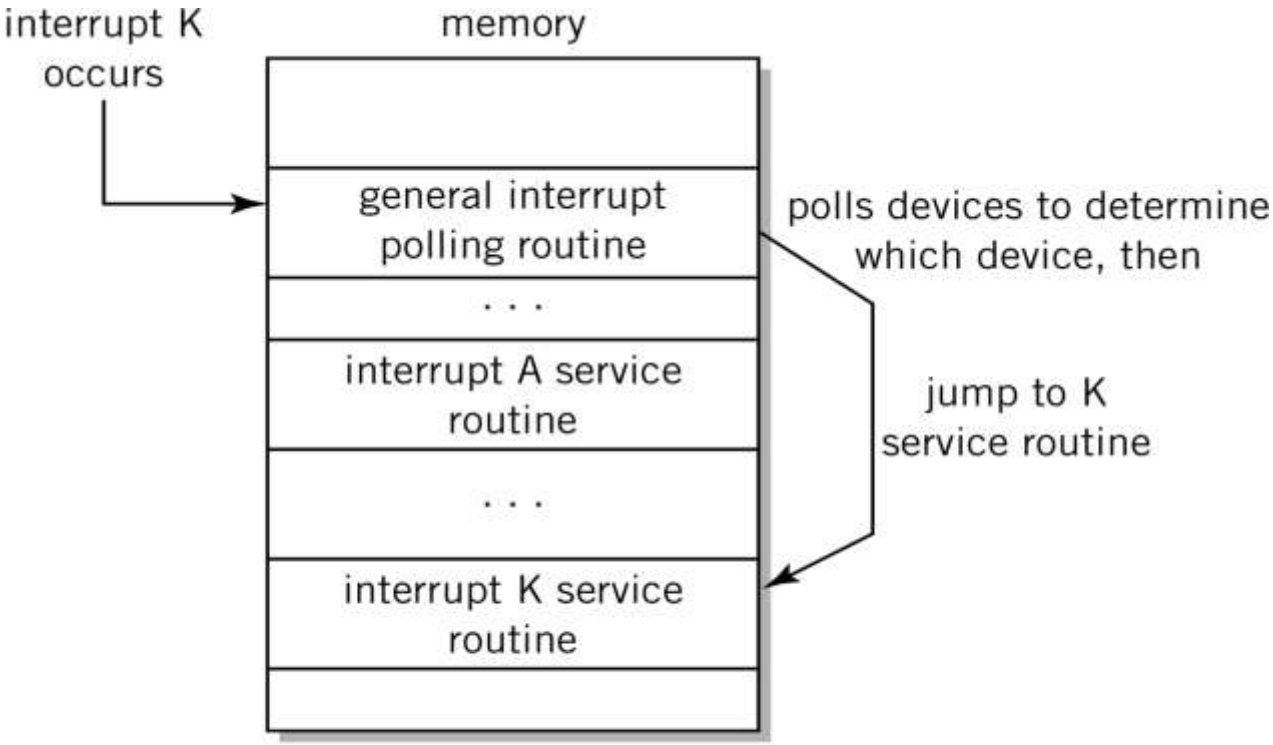
- Case2: Common Interrupt-request Line
  - Polling (Non-vectored Interrupt)
  - Vectored Interrupt
- Polling (Non-vectored Interrupt)
  - What is non-vectored Interrupt?
    - An interrupt is received by the CPU, and it jumps the program counter to a fixed address in hardware.
  - Useful for small systems where there are few interrupt sources and the software structure is straightforward.

Identify Interrupt Source (3)

- Polling (Non-vectored Interrupt) (ctd.)
  - Implementation
    - The poll could be in the form of a separate command line (e.g., TEST I/O). The processor raises TEST I/O and places the address of a particular I/O interface on the address lines.
    - The processor then reads the status register of each I/O interface. (When a device raises an interrupt request, it sets IRQ bit in its status register to 1.)

Identify Interrupt Source (4)

- Polling (Non-vectored Interrupt) (ctd.)
  - Example



Englander: The Architecture of Computer
Hardware and Systems Software, 2nd edition
Chapter 8, Figure 08-11

Identify Interrupt Source (5)

- Polling (Nonvectored Interrupt) (ctd.)
  - Registers in the Keyboard and Display

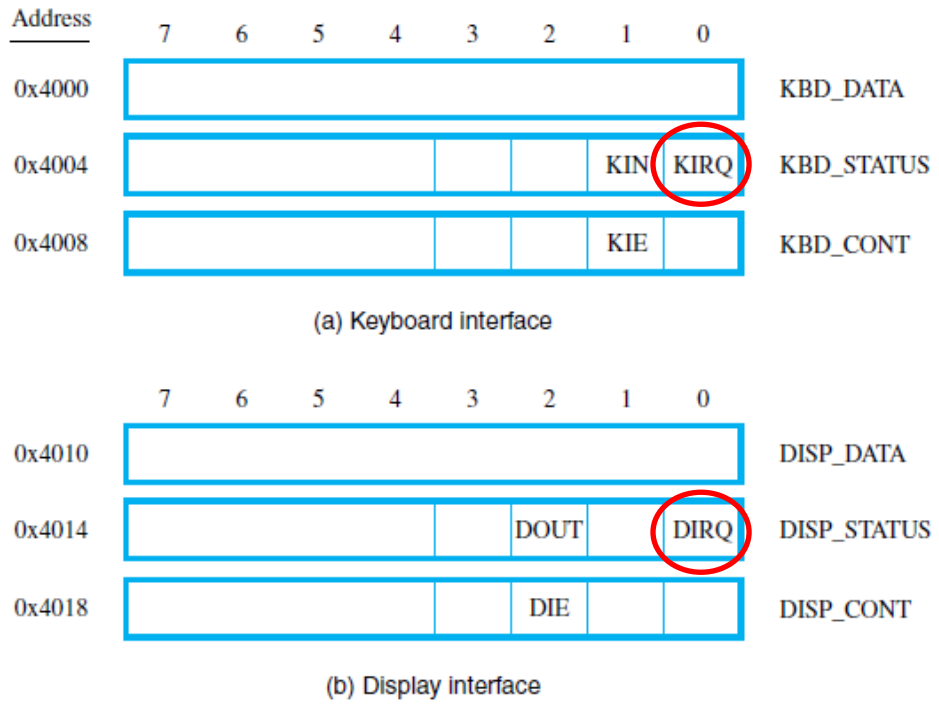


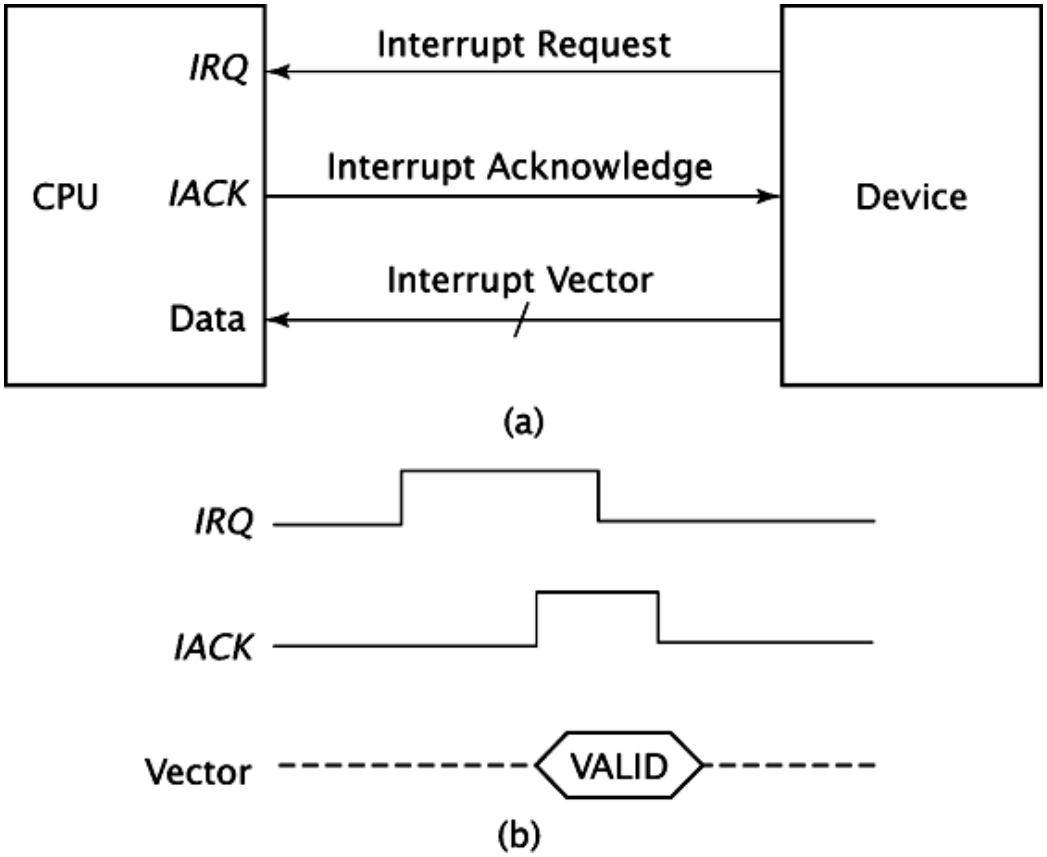
Figure 3.3 Registers in the keyboard and display interfaces.

Identify Interrupt Source (6)

- Vectored Interrupt
  - What is vectored Interrupt?
    - An interrupt scheme where the interrupting device identifies itself by giving interrupt vector when generating interrupts.
  - Interrupt Vector
    - An interrupt vector is the memory address of an interrupt handler, or an index into an array called an interrupt vector table or dispatch table.
    - Interrupt vector tables contain the memory addresses of interrupt handlers.

Identify Interrupt Source (7)

- Vectored Interrupt (ctd.)
  - Figure

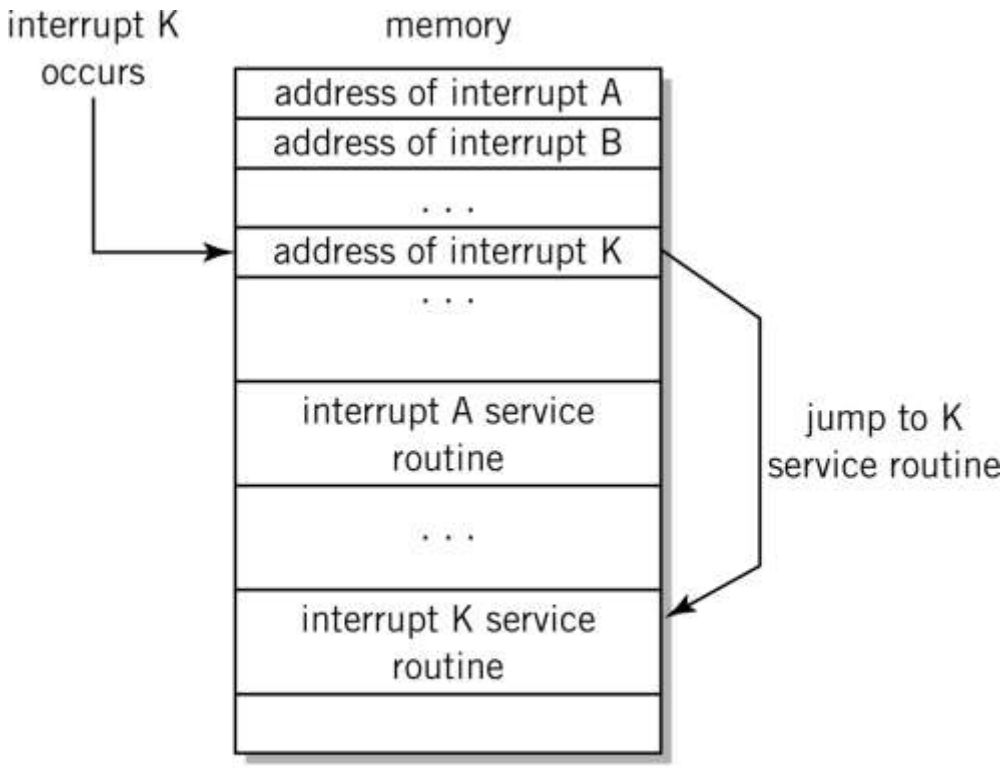


Identify Interrupt Source (8)

- Vectored Interrupt (ctd.)
  - Figure (ctd.)
    - When a device sends an interrupt request, the processor may not ready to receive the interrupt-vector code immediately.
    - When the processor is ready to receive the interrupt-vector code, it activates the interrupt-acknowledge line, INTA
    - The I/O device responds by sending its interrupt-vector code and turning off the INTR signal.

Identify Interrupt Source (9)

- Vectored Interrupt (ctd.)



Englander: The Architecture of Computer Hardware and Systems Software, 2nd edition
Chapter 8, Figure 08-10

Identify Interrupt Source (10)

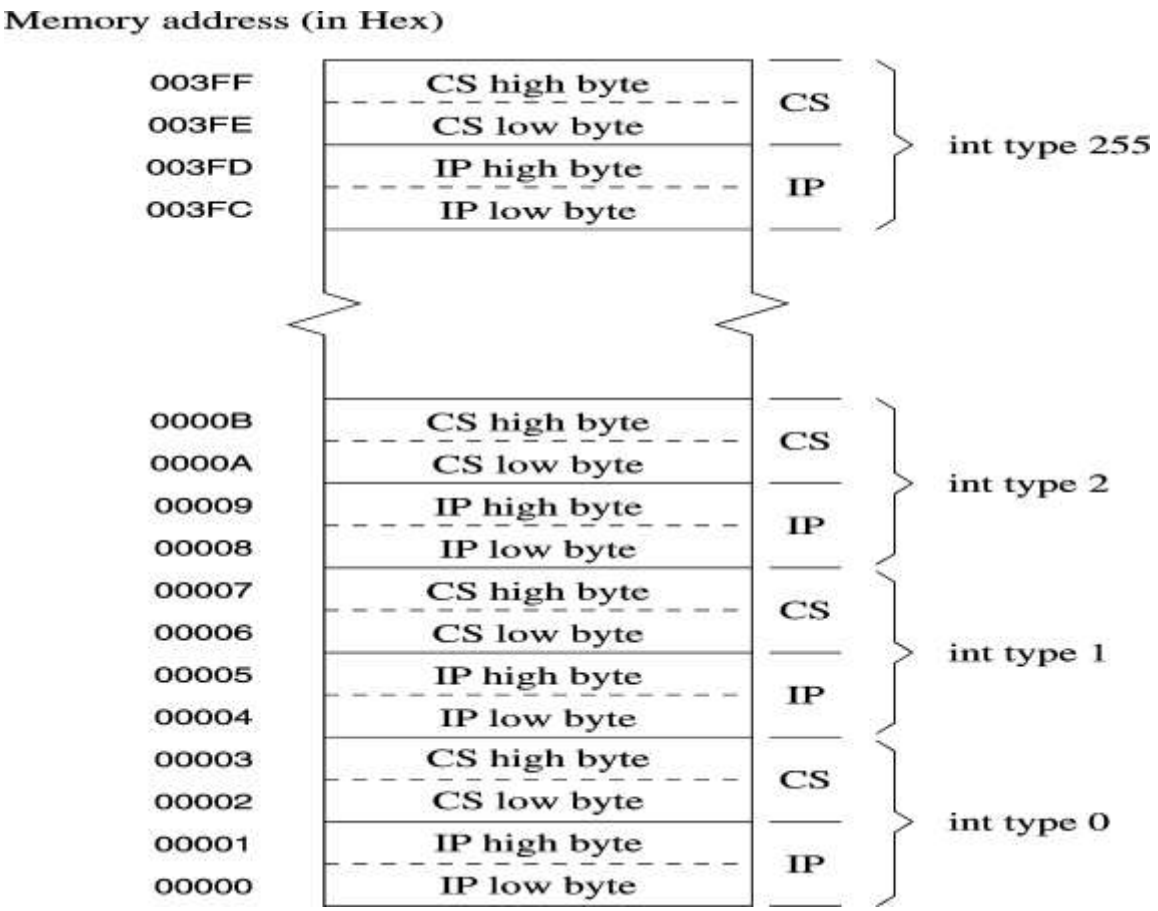
- Vectored Interrupt (ctd.)
  - Example: Pentium Real Mode (16-bit)
    - In the 8088/8086 processor as well as in the 80386/80486/Pentium processors operating in Real Mode (16-bit operation), the interrupt vector is a pointer to the Interrupt Vector Table.
    - The Interrupt Vector Table occupies the address range from 00000H to 003FFH (the first 1024 bytes in the memory map).
    - Each entry in the Interrupt Vector Table is 4 bytes long.

Identify Interrupt Source (11)

- Vectored Interrupt (ctd.)
  - Interrupt Vector Table Example: Pentium Real Mode (16 bit)
    - Up to 256 interrupts are supported (0 to 255).
    - The first 5 vectors are reserved by Intel to be used by the processor.
    - The vectors 5 to 255 are free to be used by the user.

Identify Interrupt Source (12)

Pentium Interrupt
Vector Table



1.Each entry in this table consists of a CS:IP pointer to the associated ISRs.

2.Each entry or vector requires 4 bytes:2 bytes for specifying CS; 2 bytes for the offset.

Identify Interrupt Source (13)

Interrupt Number to Vector Translation

- Interrupt numbers range from 0 to 255.
- Interrupt number acts as an index into the interrupt vector table.
- Since each vector takes 4 bytes, interrupt number is multiplied by 4 to get the corresponding ISR pointer.

Example

- For interrupt 66(42H), the memory address is

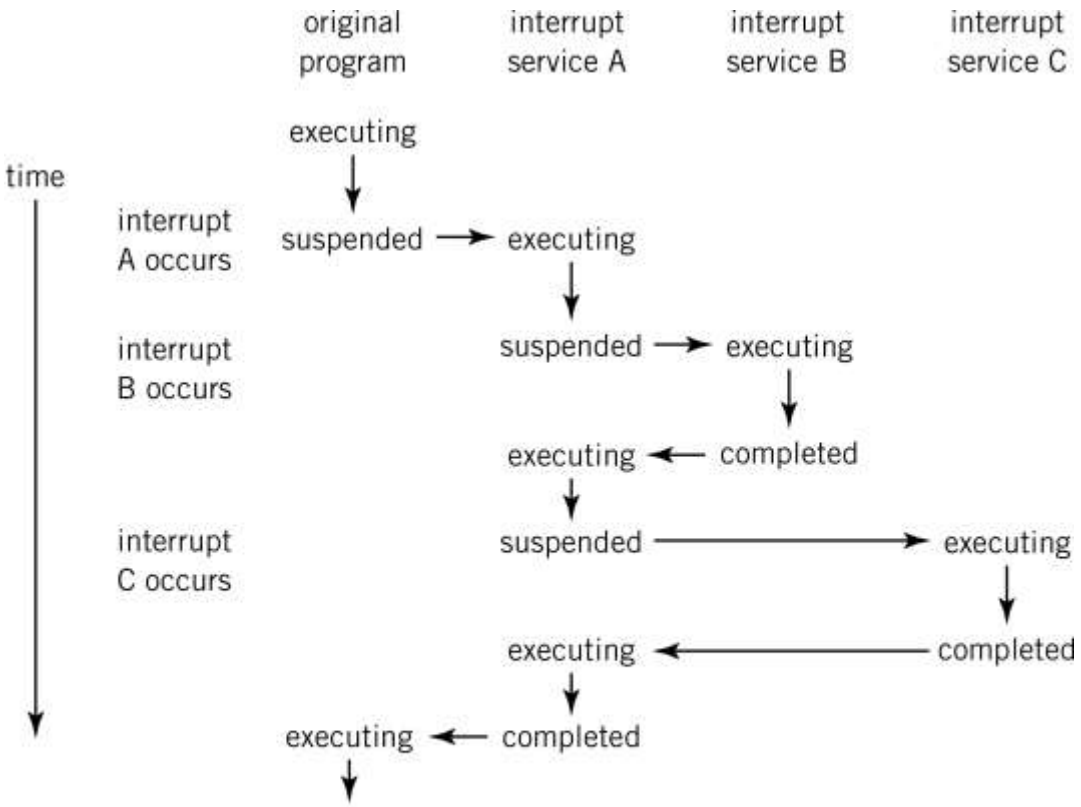
  4*42H = 108H
- The first two bytes at 108H are taken as the offset value.
- The next two bytes (i.e., at address 10AH) are used as the CS value

Multi-level Interrupt (1)

- Single-level Interrupt
  - Whether one device or several I/O devices, interrupts should be disabled during the execution of an interrupt-service routine.
- Multi-level Interrupt (Interrupt Nesting)
  - It is necessary to accept another device interrupt during the execution of an interrupt-service routine for a device.
  - I/O devices are organized in a priority structure. An interrupt request from a high-priority device should be accepted while the processor is serving another request from a lower-priority device.

Multi-level Interrupt (2)

- Multi-level Interrupt (ctd.)
  - Example



Englander: The Architecture of Computer Hardware and Systems Software, 2nd edition
Chapter 8, Figure 08-12

Multi-level Interrupt (3)

- Multi-Level Priority
  - Assign a priority level to the processor that can be changed under program (privileged instructions) control.
  - The priority level of the processor is the priority of the program that is currently being executed.
  - The processor accepts interrupts only from devices that have priorities higher than its own.
  - At the time the execution of an interrupt-service routine for some device is started, the priority of the processor is raised to that of the device.
  - The processor's priority can be encoded in a few bits of the processor status register.

Multi-level Interrupt (4)

- Multi-level Priority (ctd.)
  - Example

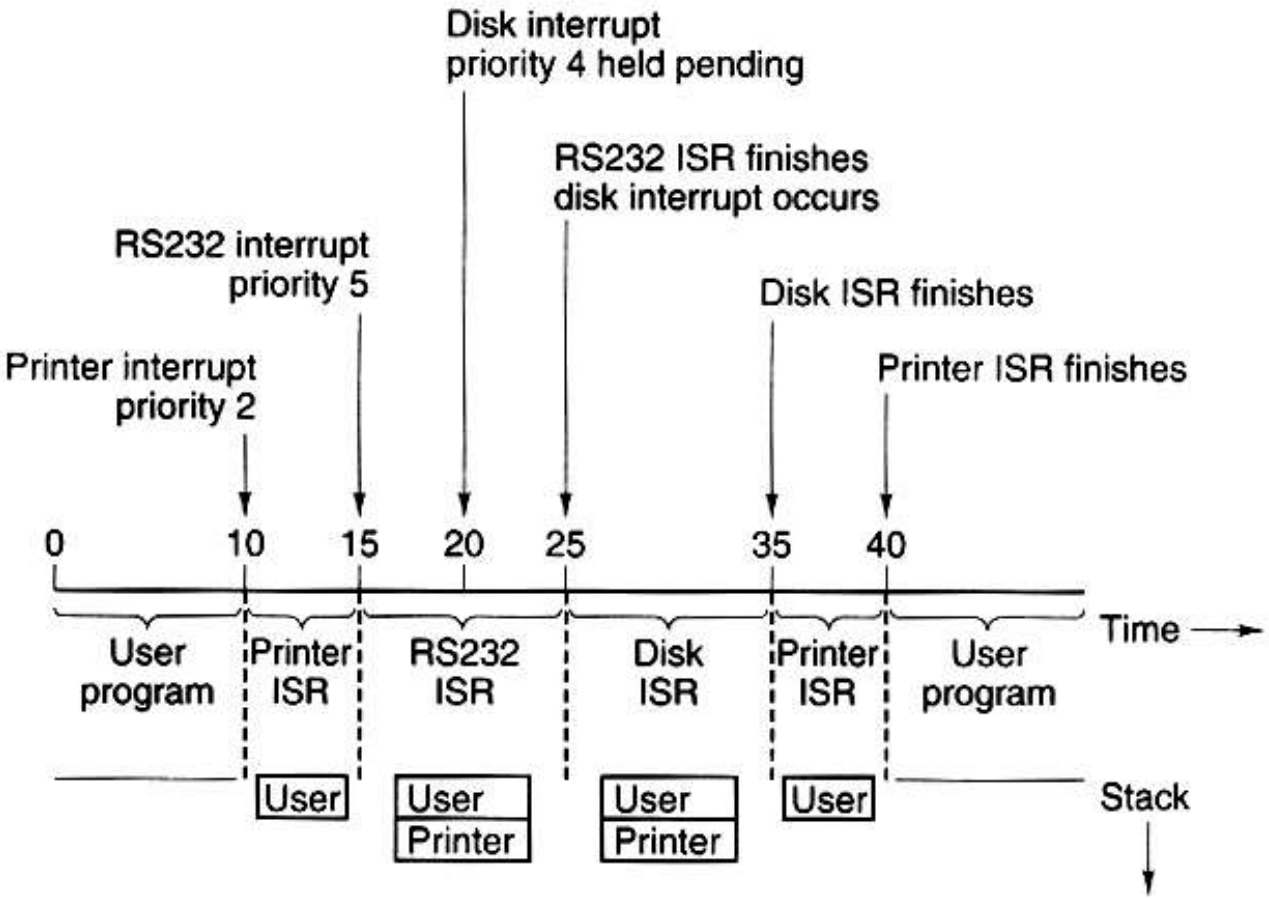
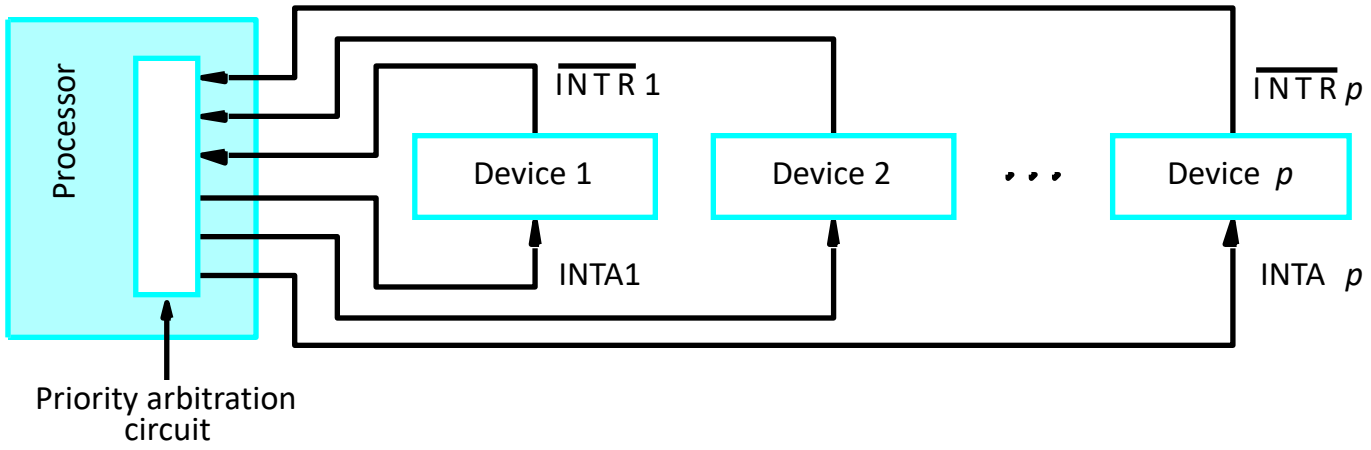


Figure 5-44. Time sequence of multiple interrupt example.

Multi-level Interrupt (5)

- Multi-Level Priority (ctd.)
  - One Example of Hardware Implementation Figure



Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

Simultaneous Interrupt (1)

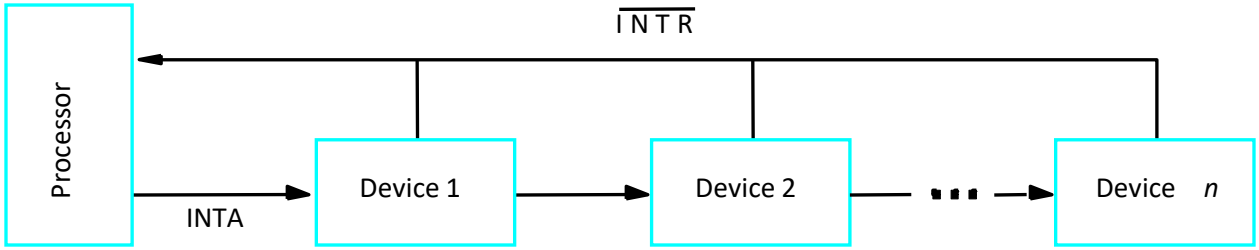
- Case 1: Individual device has individual interrupt-request and acknowledge lines
  - The processor simply accepts the request having the highest priority.
  - It allows the processor to accept interrupt requests from some devices but not from others, depending upon their priorities.
- Case 2: Several devices share one interrupt-request line
  - Software Poll
  - Daisy Chain (Hardware Poll)
  - Priority Group

Simultaneous Interrupt (2)

- Software Poll
  - The processor poll the status register of the I/O device.
  - Priority is determined by the order in which the devices are polled.

Simultaneous Interrupt (3)

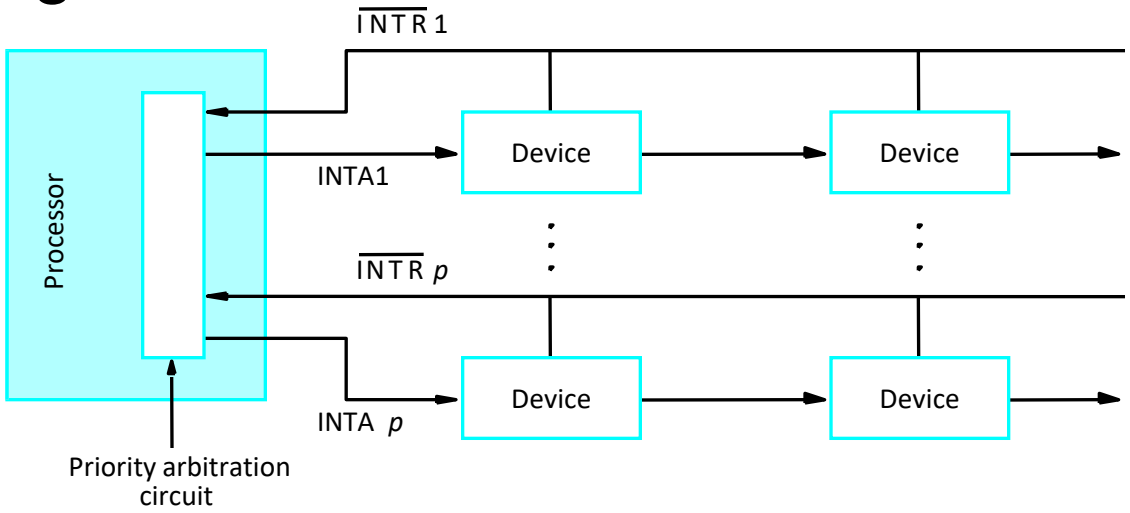
- Daisy Chain
  - Figure



  - In the daisy chain arrangement, the device that is electrically closest to the processor has the highest priority.
  - The second device along the chain has second highest priority, and so on.

Simultaneous Interrupt (4)

- Priority Group
  - Figure



Processor Control Registers (1)

- Processor Status Register (PS)
- IPS register is where PS is automatically saved when an interrupt request is recognized
- IENABLE register allows the processor to selectively respond to individual I/O devices.
  - Has one bit per device
- IPENDING has one bit per device to indicate if interrupt request has not yet been serviced

Processor Control Registers (2)

- Control Registers Figure

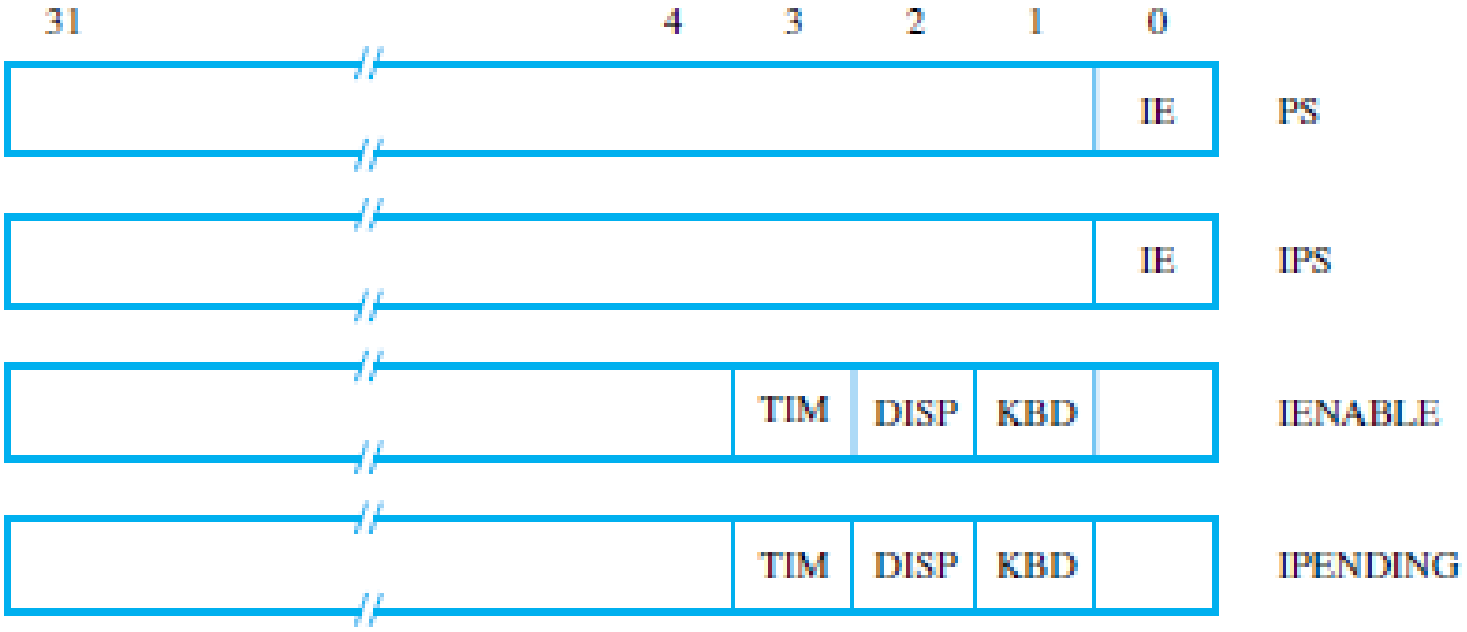


Figure 3.7 Control registers in the processor.

Processor Control Registers (3)

- Accessing Control Registers
  - Use special Move instructions that transfer values to and from general-purpose registers
  - Transfer pending interrupt requests to R4:

    MoveControl R4, IPENDING
  - Transfer current processor IE setting to R2:

    MoveControl R2, PS
  - Transfer desired bit pattern in R3 to IENABLE:

    MoveControl IENABLE, R3

Examples of Interrupt Programs (1)

- **Example 3.2** Let us consider again the task of reading a line of characters typed on a keyboard, storing the characters in the main memory, and displaying them on a display device. Now, we will use interrupts with the keyboard, but polling with the display.
  - Assume that a specific memory location, ILOC, is dedicated for dealing with interrupts, and that it contains the first instruction of the interrupt-service routine.

Examples of Interrupt Programs (2)

- **Example 3.2** (ctd.)
  - Whenever an interrupt request arrives at the processor, and processor interrupts are enabled, the processor will automatically:
    - Save the contents of the program counter, either in a processor register that holds the return address or on the processor stack.
    - Save the contents of the status register PS by transferring them into the IPS register, and clear the IE bit in the PS.
    - Load the address ILOC into the program counter

Examples of Interrupt Programs (3)

- Interrupt Service Routine of Example 3.2

Interrupt-service routine

```
ILOC:   Subtract          SP, SP, #8          Save registers.
        Store             R2, 4(SP)
        Store             R3, (SP)
        Load              R2, PNTR            Load address pointer.
        LoadByte          R3, KBD_DATA        Read character from keyboard.
        StoreByte         R3, (R2)            Write the character into memory
        Add               R2, R2, #1            and increment the pointer.
        Store             R2, PNTR            Update the pointer in memory.
ECHO:   LoadByte          R2, DISP_STATUS     Wait for display to become ready.
        And               R2, R2, #4
        Branch_if_[R2]=0  ECHO
        StoreByte         R3, DISP_DATA       Display the character just read.
        Move              R2, #CR             ASCII code for Carriage Return.
        Branch_if_[R3]≠[R2] RTRN              Return if not CR.
        Move              R2, #1
        Store             R2, EOL             Indicate end of line.
        Clear             R2                  Disable interrupts in
        StoreByte         R2, KBD_CONT          the keyboard interface.
RTRN:   Load              R3, (SP)            Restore registers.
        Load              R2, 4(SP)
        Add               SP, SP, #8
        Return-from-interrupt
```

Examples of Interrupt Programs (4)

- Main Program of Example 3.2

Main program

```
START:  Move          R2, #LINE
        Store         R2, PNTR          Initialize buffer pointer.
        Clear         R2
        Store         R2, EOL           Clear end-of-line indicator.
        Move          R2, #2            Enable interrupts in
        StoreByte     R2, KBD_CONT       the keyboard interface.
        MoveControl   R2, IENABLE
        Or            R2, R2, #2        Enable keyboard interrupts in
        MoveControl   IENABLE, R2        the processor control register.
        MoveControl   R2, PS
        Or            R2, R2, #1
        MoveControl   PS, R2            Set interrupt-enable bit in PS.
        next instruction
```

Examples of Interrupt Programs (5)

- **Example 3.3** Suppose a program needs to display a page of text stored in the memory. This can be done by having the processor send a character whenever the display interface is ready, which may be indicated by an interrupt request. Assume that both the display and the keyboard are used by this program, and that both are enabled to raise interrupt requests.

Examples of Interrupt Programs (6)

- **Example 3.3** (ctd.)
  - To use interrupts for both keyboard & display, call subroutines from ILOC service routine
  - Service routine reads IPENDING register
  - Checks which device bit(s) is (are) set to determine which subroutine(s) to call
  - Service routine must save/restore Link register
  - Also need separate pointer variable to indicate output character for next display interrupt

Examples of Interrupt Programs (7)

- Interrupt Handler of Example 3.3

Interrupt handler

```
ILOC:       Subtract    SP, SP, #12        Save registers.
            Store       LINK_reg, 8(SP)
            Store       R2, 4(SP)
            Store       R3, (SP)
            MoveControl R2, IPENDING       Check contents of IPENDING.
            And         R3, R2, #4         Check if display raised the request.
            Branch_if_[R3]=0 TESTKBD       If not, check if keyboard.
            Call        DISR               Call the display ISR.
TESTKBD:    And         R3, R2, #2         Check if keyboard raised the request.
            Branch_if_[R3]=0 NEXT          If not, then check next device.
            Call        KISR               Call the keyboard ISR.
NEXT:       ...                            Check for other interrupts.

            Load        R3, (SP)           Restore registers.
            Load        R2, 4(SP)
            Load        LINK_reg, 8(SP)
            Add         SP, SP, #12
            Return-from-interrupt
```

Figure 3.7 Control registers in the processor

Summary

- 知识点：Multiple device interrupt system Design Issues
  - Identify Interrupt Source
    - Polling (Non-vectored Interrupt)
    - Vectored Interrupt
  - Multiple-level Interrupt (Interrupt Nesting)
  - Simultaneous Interrupt

单选题 2分

设置

3. With interrupt-driven I/O, if two or more devices request service at the same time, _____.

A the device closest to the CPU gets priority

B the device that is fastest gets priority

C the device assigned the highest priority is serviced first

D the system is likely to crash

提交

Content of this lecture

- 8.4 Direct Memory Access
- Summary

Direct Memory Access (1)

- Interrupt driven and programmed I/O require active CPU intervention.
  - Transfer rate is limited.
  - CPU is tied up.
- Solution: DMA
  - Used for high-speed block transfers between a device and memory.
  - During the transfer, the CPU is not involved.
  - Typical DMA devices:
    - Disk drives, Tape drives
  - Remember
    - Keyboard data rate → 0.01 KB/s (1 byte every 100 ms)
    - Disk drive data rate → 2,000 KB/s (1 byte every 0.5 μs)

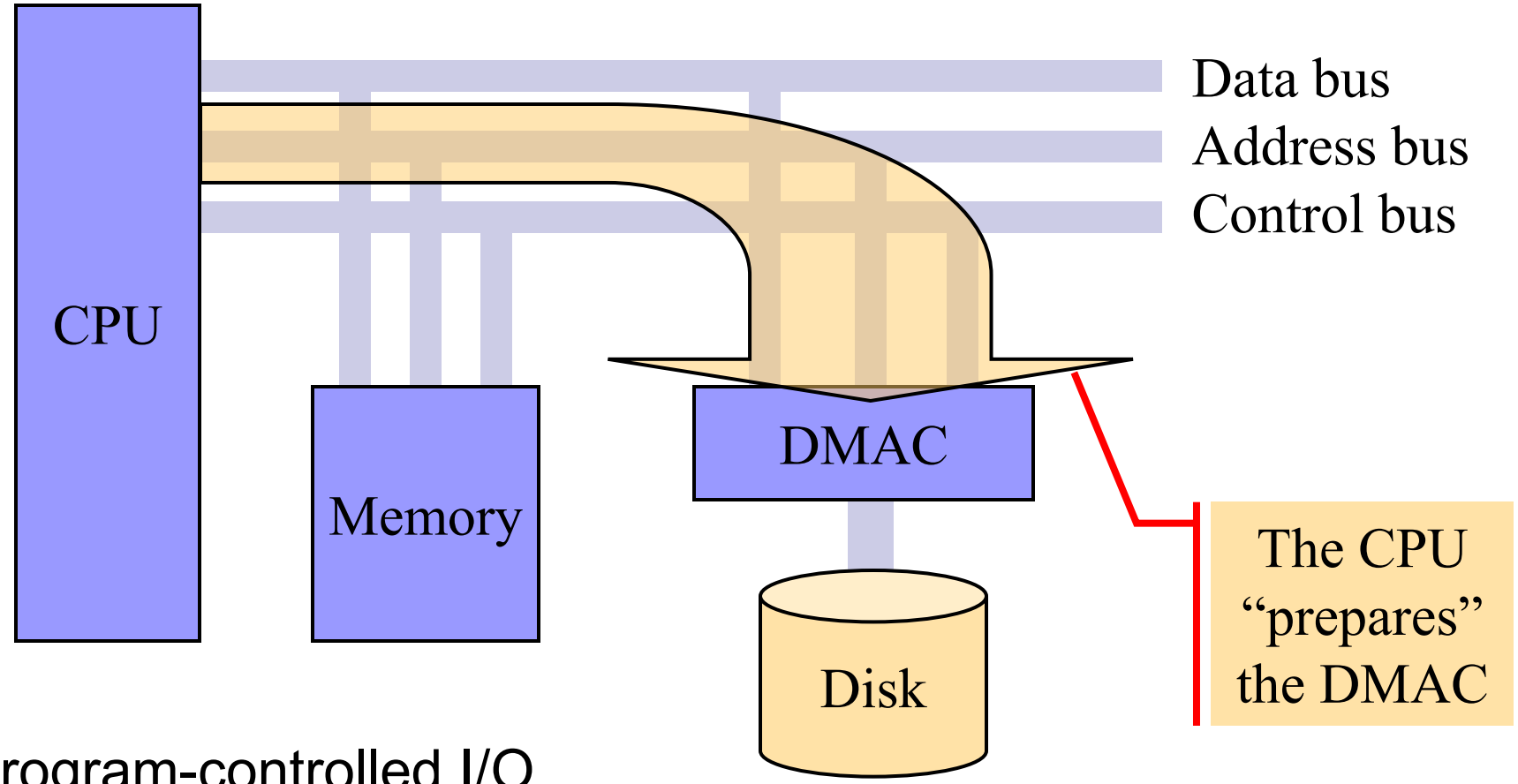
Transfer rate is too high to be controlled by software executing on the CPU

Direct Memory Access (2)

- DMA Operation
  - 1.The CPU “prepares” the DMA operation by transferring information to a DMA controller (DMAC):
    - Location of the data on the device
    - Location of the data in memory
    - Size of the block to transfer
    - Direction of the transfer
    - Mode of data transfer (burst/cycle stealing/transparent)
  - 2. When the device is ready to transfer data, the DMAC takes control of the system buses (DMA controller deals with transfer)
  - 3. DMAC sends interrupt when finished

Direct Memory Access (3)

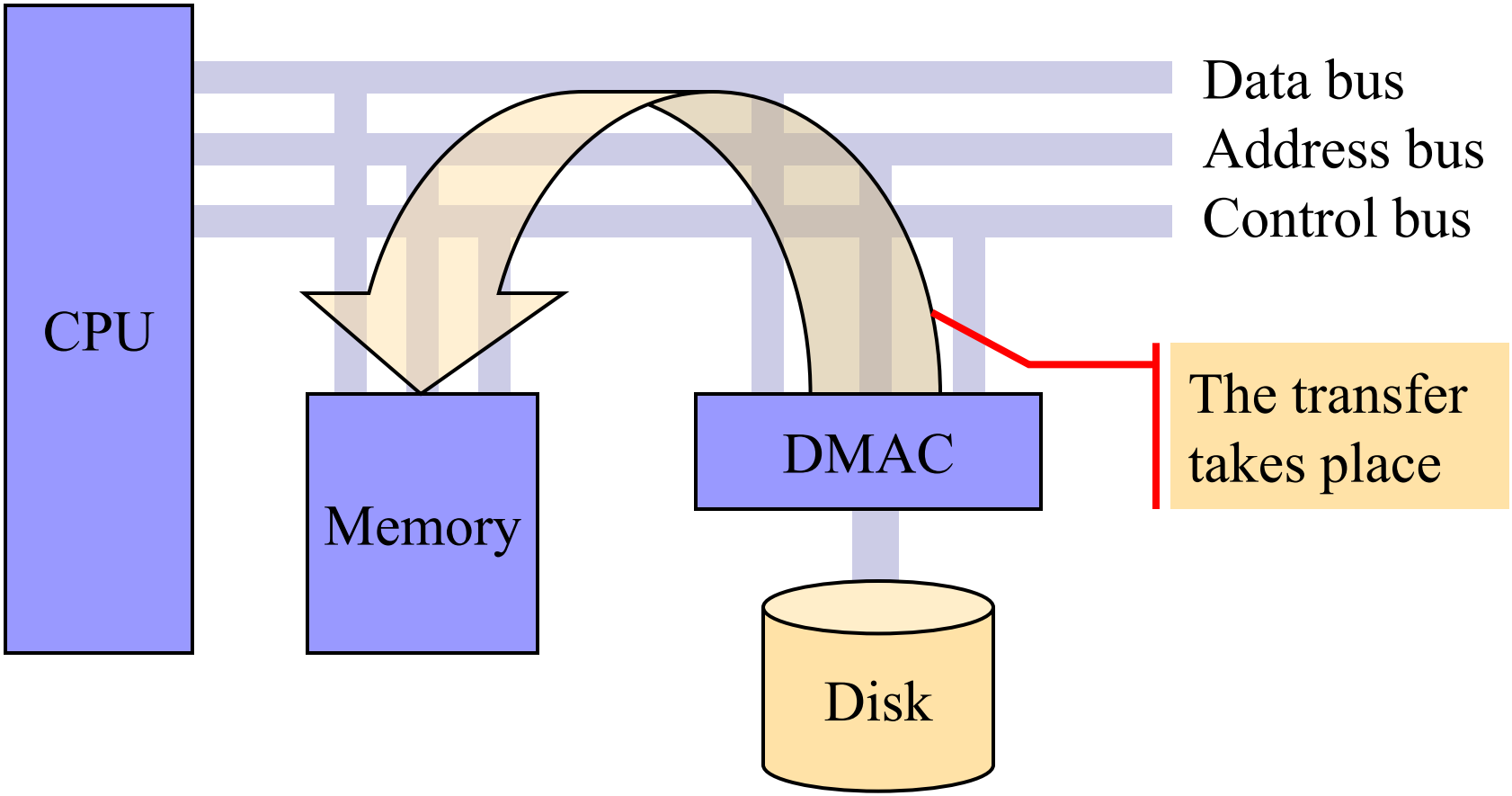
- DMA Operation (ctd.)



Program-controlled I/O

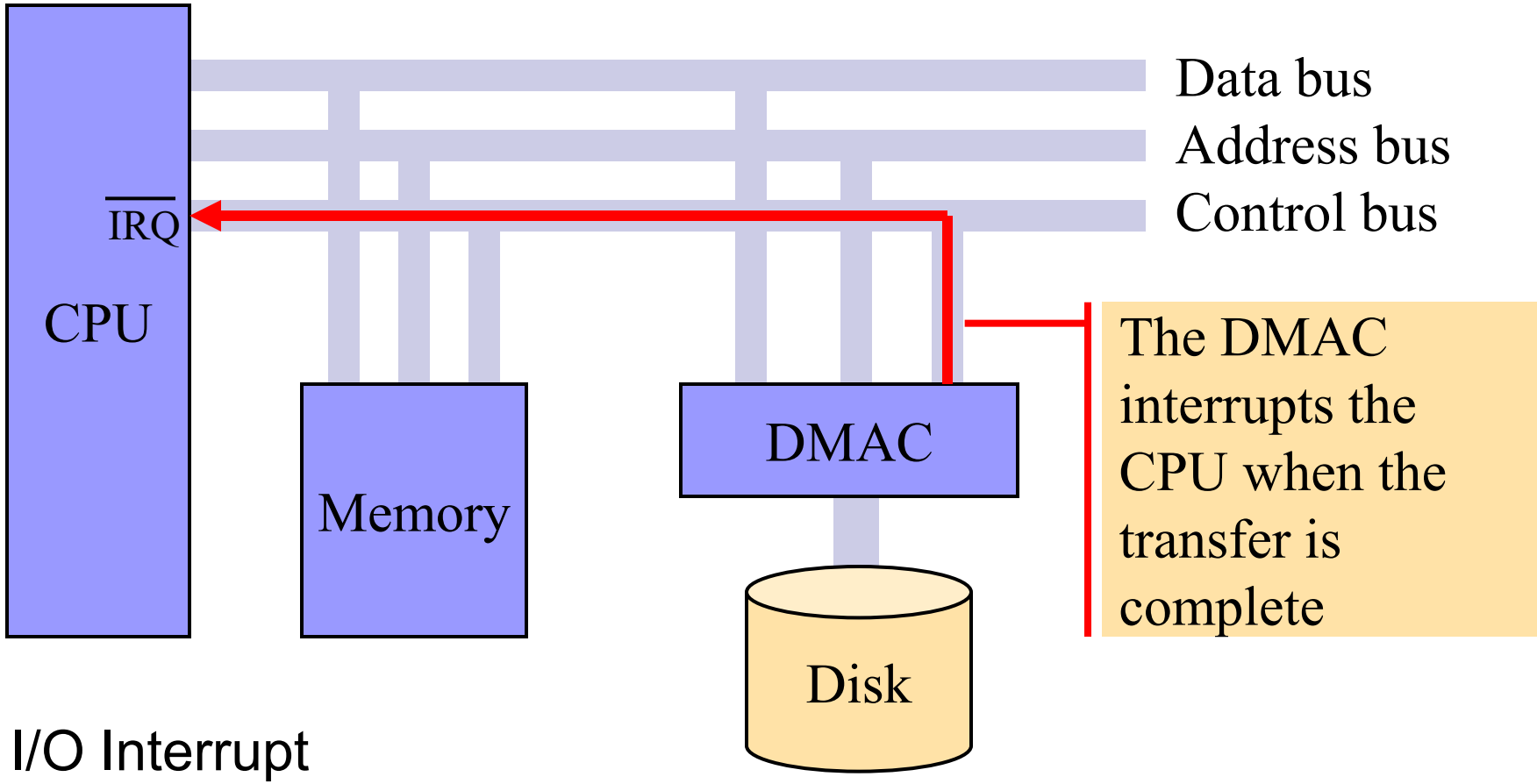
Direct Memory Access (4)

- DMA Operation (ctd.)



Direct Memory Access (5)

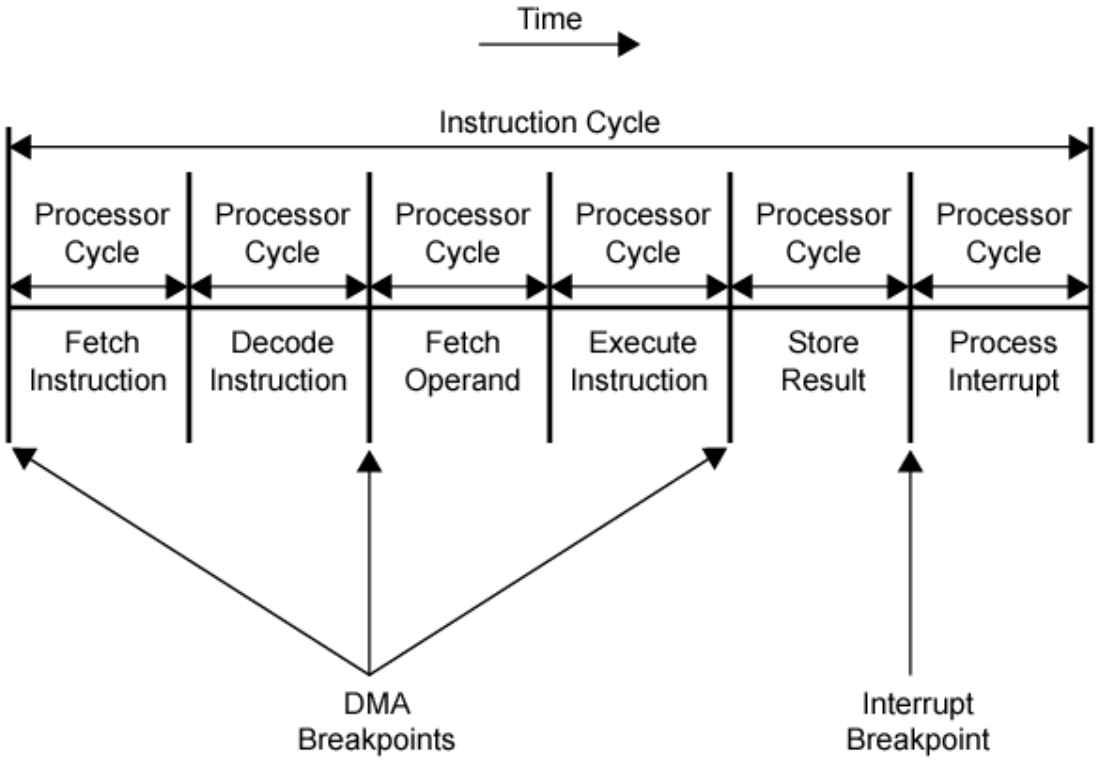
- DMA Operation (ctd.)



I/O Interrupt

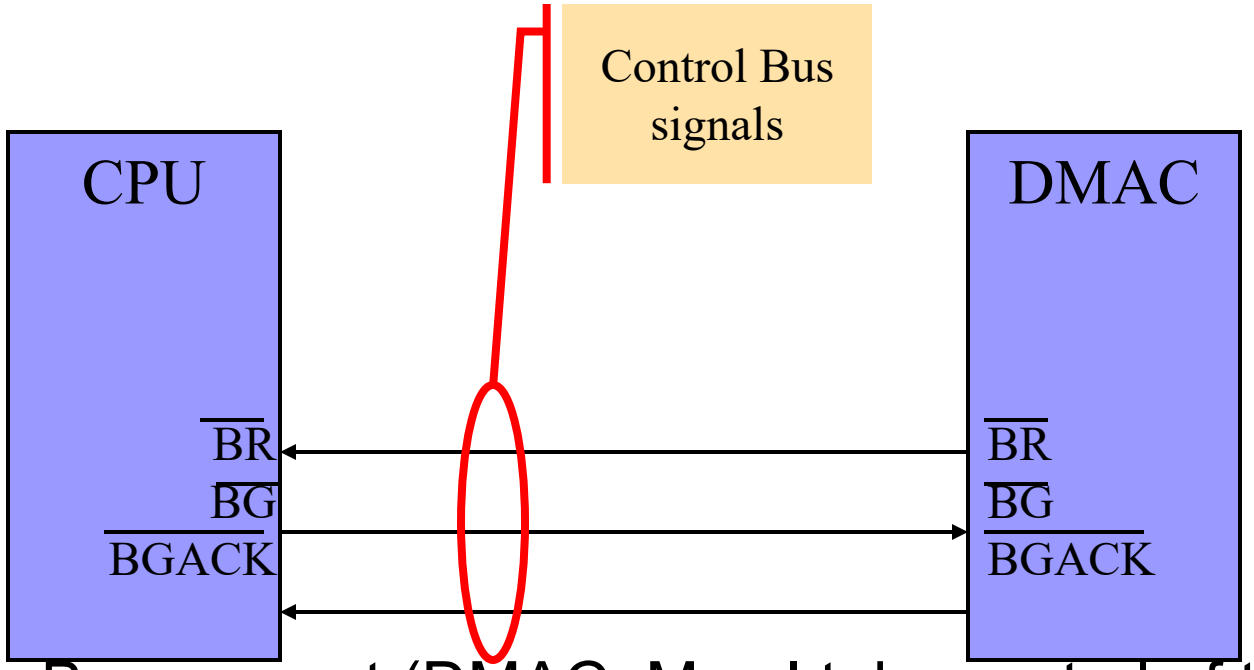
Direct Memory Access (6)

- DMA Operation (ctd.)
  - DMA and Interrupt Breakpoints During an Instruction Cycle



Direct Memory Access (7)

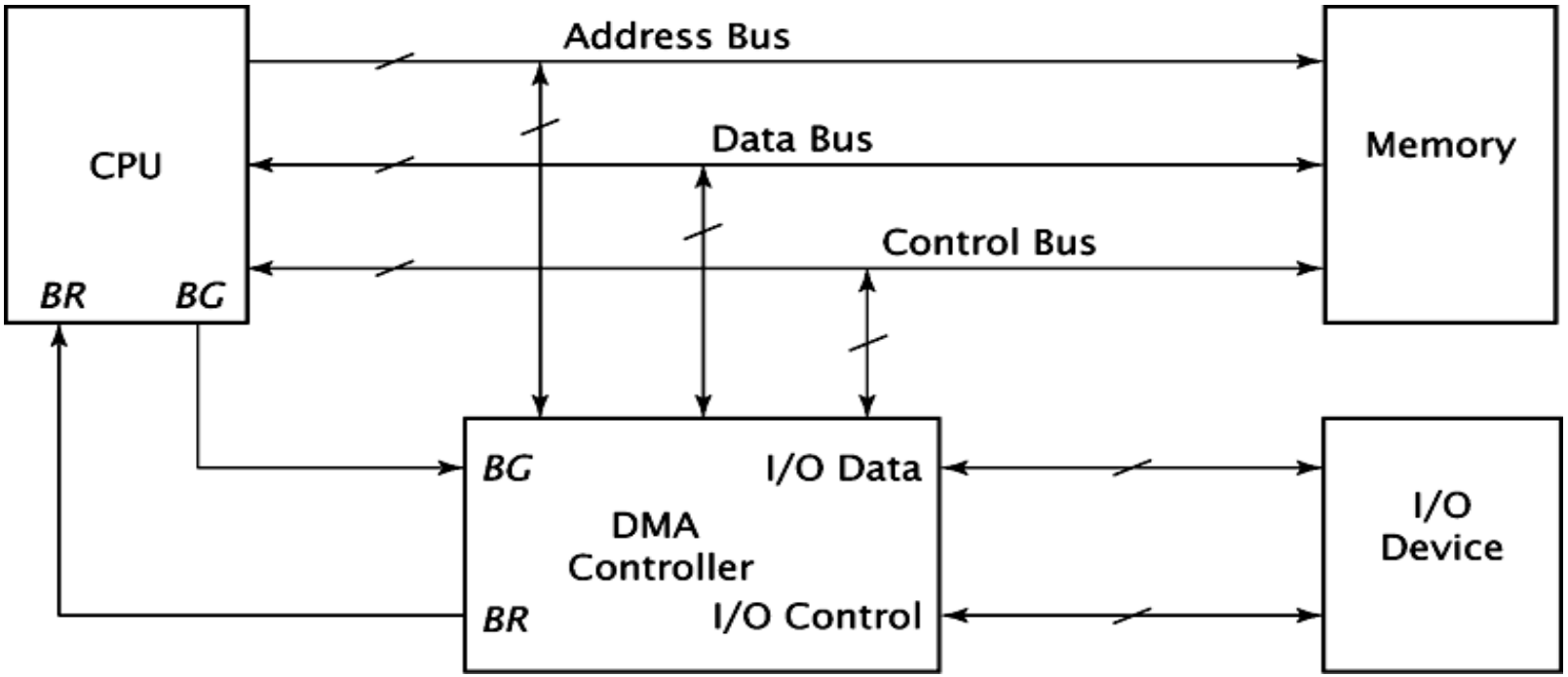
- Bus Control



  - BR = Bus request (DMAC: May I take control of the system buses?)
  - BG = Bus grant (CPU: Yes, here you go.)
  - BGACK = BG acknowledge (DMAC: Thanks, I've got control.)

Direct Memory Access (8)

- Implementing DMA in a Computer System
  - A computer system with DMA controller



Direct Memory Access (9)

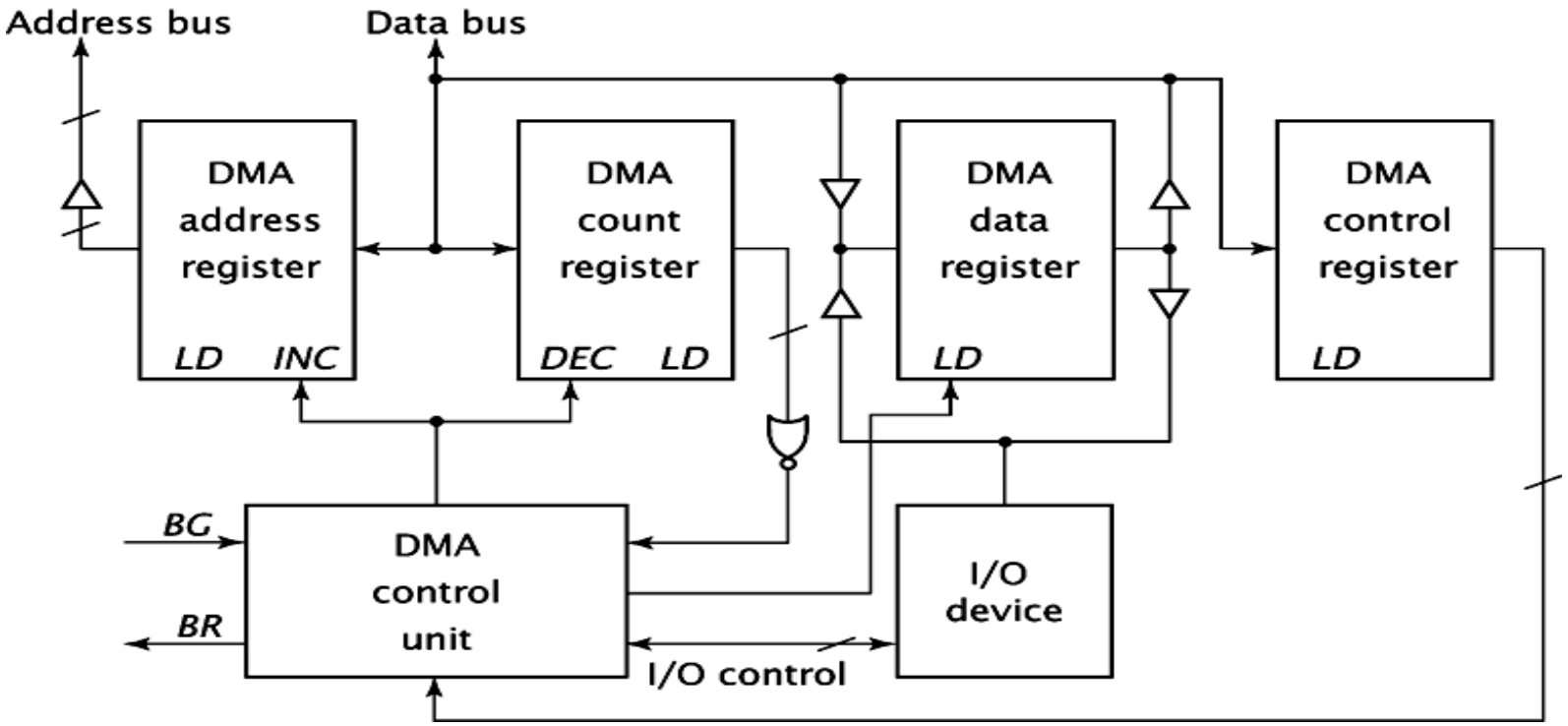
- Implementing DMA in a Computer System (ctd.)
  - Function of DMA Controller
    - For words transferred, the DMA controller
      - Provides the memory address
      - Provides all the bus signal that control data transfer
      - Increment the memory address for successive words
      - Keep track of the number of transfers

Direct Memory Access (10)

- Implementing DMA in a Computer System (ctd.)
  - Registers in a DMA Controller
    - DMA Address Register contains the memory address to be used in the data transfer.
    - DMA Count Register (Word Count Register) contains the number of bytes of data to be transferred.
    - DMA Control Register accepts commands from the CPU.
    - Although not shown in this fig., most DMA controllers also have a Status Register.

Direct Memory Access (11)

- Implementing DMA in a Computer System (ctd.)
  - Registers in a DMA Controller (ctd.)



Direct Memory Access (12)

- Implementing DMA in a Computer System (ctd.)
  - Registers in a DMA Controller (ctd.)

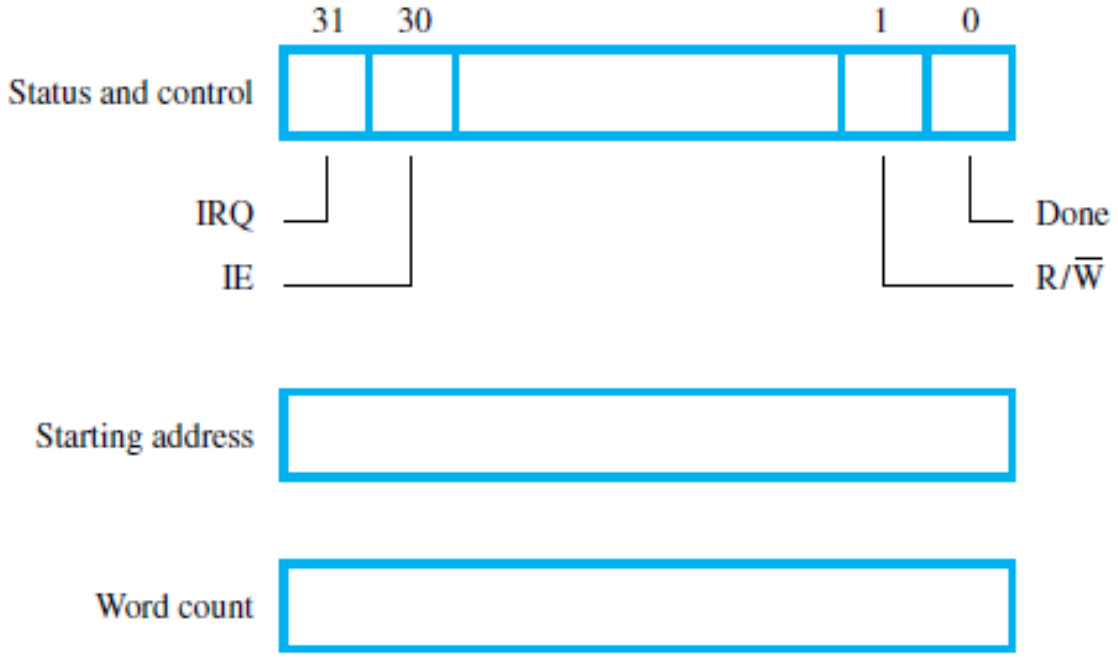


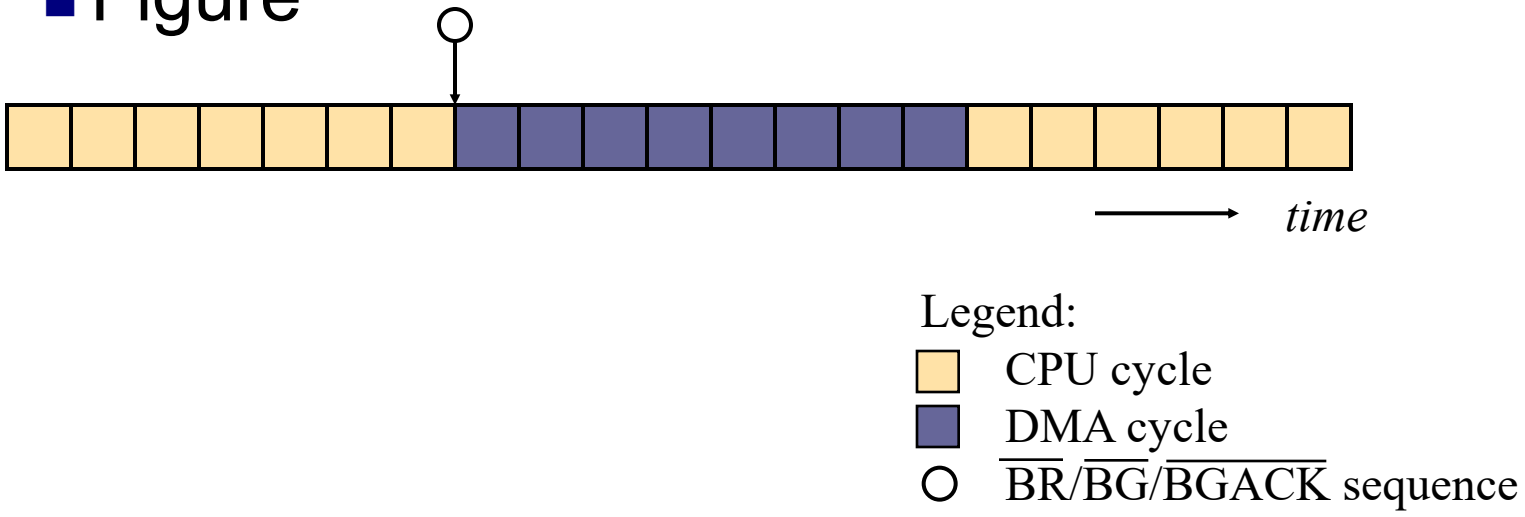
Figure 8.12 Typical registers in a DMA controller.

Direct Memory Access (13)

- DMA Data Transfer Modes
  - Modes vary by how the DMA controller determines when to transfer data, but the actual data transfer process is the same for all the modes.
  - Burst Mode (Block Mode)
    - An entire block of data is transferred in one contiguous sequence.
    - Once the DMAC is granted access to the system buses by the CPU, it transfers all bytes of data in the data block before releasing control of the system buses back to the CPU.

Direct Memory Access (14)

- DMA Data Transfer Mode (ctd.)
  - Burst Mode (ctd.)
    - DMAC relinquishes control of the system buses by releasing BGACK signal.
    - Figure



Direct Memory Access (15)

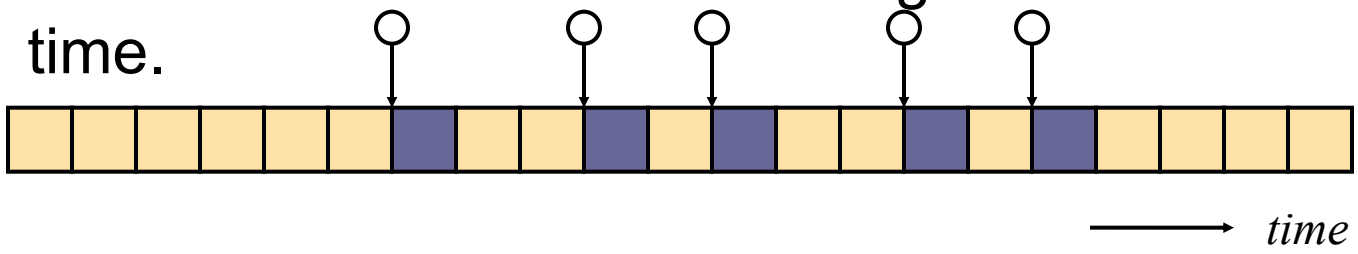
- DMA Data Transfer Mode (ctd.)
  - Burst Mode (ctd.)
    - It is the fastest DMA modeUsage
    - It is useful for loading programs or data files into memory.
    - Disadvantages
      - Render the processor inactive for relatively long periods of time.
      - Low memory efficiency.

Direct Memory Access (16)

- DMA Data Transfer Mode(ctd.)
  - Cycle Stealing Mode (Single Byte Mode)
    - DMAC obtains access to the system buses as in burst mode, using BR & BG signals.
    - However, it transfers one byte of data and then de-asserts BR, returning control of the system buses to the CPU. It continually issues requests via BR, transferring one byte of data per request, until it has transferred it's entire block of data.
    - A BR-BG-BGACK sequence occurs for every transfer, until the block is completely transferred.

Direct Memory Access (17)

- DMA Data Transfer Mode(ctd.)
  - Cycle Stealing Mode (ctd.)
    - The data block is not transferred as quickly as in burst mode, but the CPU is not idled for as long as in that mode.
    - Usage
      - Useful for controllers monitoring data in real time.



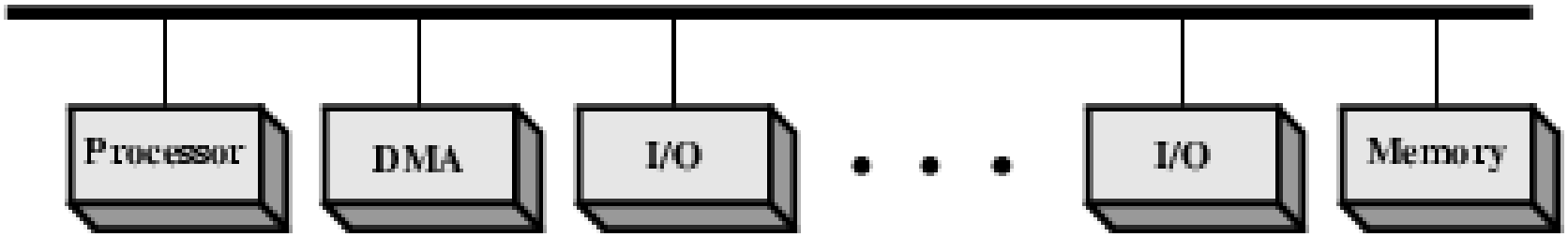
Legend:
- CPU cycle
- DMA cycle
- $\overline{BR}/\overline{BG}/\overline{BGACK}$ sequence

Direct Memory Access (18)

- DMA Data Transfer Mode (ctd.)
  - Transparent Mode
    - The DMAC only transfers data when the CPU is performing operations that do not use the system bus. It is transparent to CPU.
    - Note
      - The DMAC does not need to request the control of the system bus.
    - Advantage
      - Processor never stops executing its program.
    - Disadvantage
      - The hardware needed to determine when the processor is not using the system bus can be quite complex and relatively expensive.

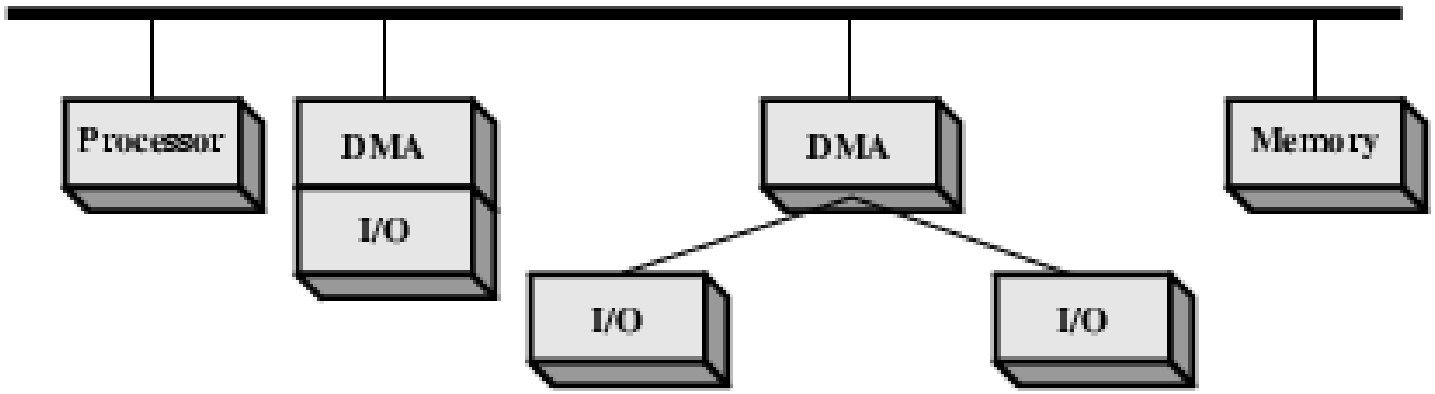
Direct Memory Access (19)

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



Direct Memory Access(20)

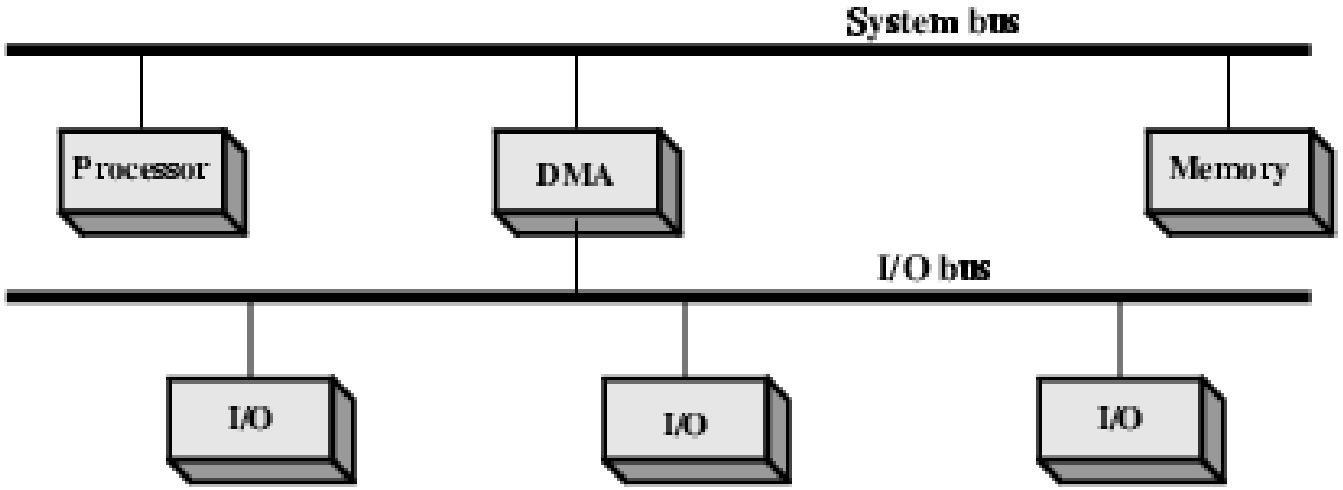
- DMA Configurations (ctd.)
  - Single Bus, Integrated DMA controller
    - Controller may support >1 device
    - Each transfer uses bus once
      - DMA to memory
    - CPU is suspended once



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

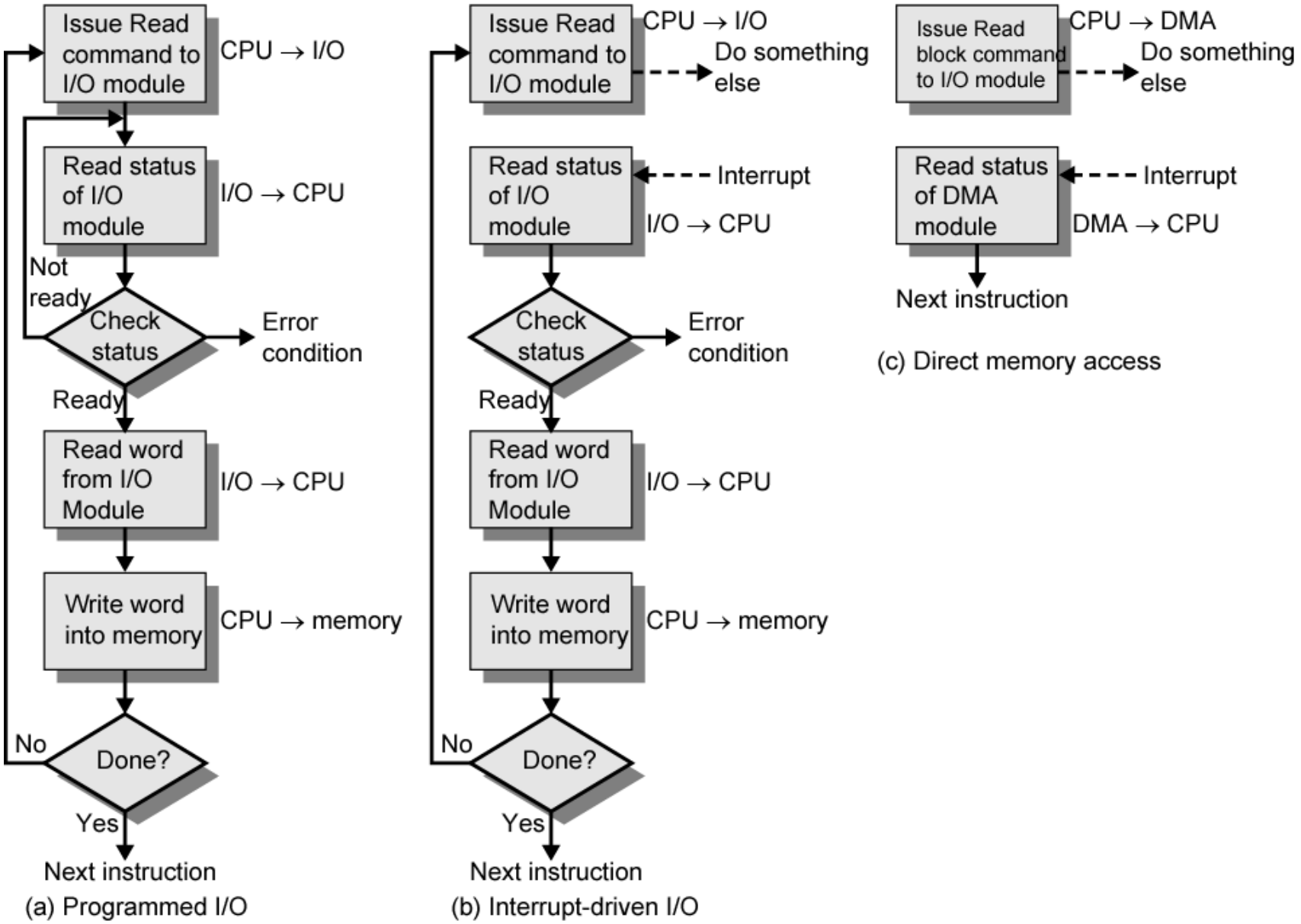
DMA Configurations (21)

- DMA Configurations (ctd.)
  - Separate I/O Bus
    - Bus supports all DMA enabled devices
    - Each transfer uses bus once
      - DMA to memory
    - CPU is suspended once



(c) I/O bus

Three Techniques for Input of a Block of Data



单选题 2分

设置

Which method bypasses the CPU for certain types of data transfer?

A Software interrupts

B Polled I/O

C Interrupt-driven I/O

D Direct Memory Access (DMA)

提交

Summary

- 知识点：DMA
  - DMA Operation
  - Registers in a DMA Controller
  - DMA Data Transfer Mode

Homework

- P126 3.1, 3.3, 3.6
- 补充：
  - Three devices, A, B, and C, are connected to the bus of a computer. I/O transfers for all three devices use interrupt control. Interrupt nesting for devices A and B is not allowed, but interrupt requests from C may be accepted while either A or B is being serviced. Suggest different ways in which this can be accomplished in each of the following cases;
    - (a) the computer has one interrupt-request line
    - (b) two interrupt-request lines, INTR1 and INTR2, are available, with INTR1 having higher priority.