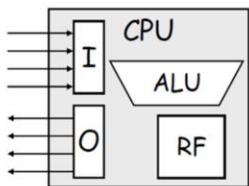


سافتار و زبان کامپیوٹر

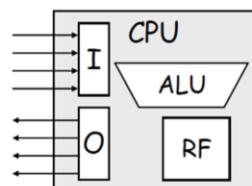
فصل هشت

وادی/فوجی-وقفه



Computer Structure & Machine Language

Chapter Eight Input/Output & Interrupt



Copyright Notice

Parts (text & figures of this lecture are adopted from:

- ④ M. M. Mano, C. R. Kime & T. Martin, “Logic & Computer Design Fundamentals”, 5th Ed., Pearson, 2015
- ④ D. Patterson, J. Hennessy, “Computer Organization & Design, The Hardware/Software Interface, MIPS Edition”, 4th Ed., MK Publishing, 2012
- ④ W. Stallings, “Computer Organization and Architecture, Designing Performance”, 10th Ed., Pearson, 2016



Spring 2025

3

William Stallings



Spring 2025

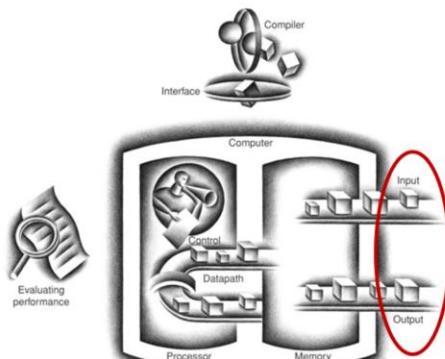
4

Contents

- I/O Subsystem
- I/O Devices (Instances & Characteristics)
- I/O Interfaces
 - Memory-mapped vs. isolated I/O
- I/O Transactions
- Modes of Transfer
 - Interrupt vs. Programmed I/O
 - Direct Memory Access (DMA)
- Operating system's Role



Input/Output Subsystem



Spring 2025

6

The input and output subsystem of a computer provides an efficient mode of communication between the CPU and the outside environment.

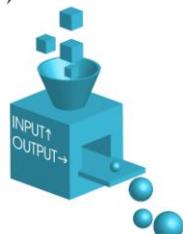
Programs and data must be entered into the memory for processing, and results obtained from computations must be recorded or displayed.

Devices that the CPU controls directly are said to be connected *online*. These devices communicate directly with the CPU or transfer binary information into or out of the memory upon command from the CPU.

Input or output devices attached to the computer online are called *peripherals*.

Contents

- I/O Subsystem
- I/O Devices (**Instances & Characteristics**)
- I/O Interfaces
 - Memory-mapped vs. isolated I/O
- I/O Transactions
- Modes of Transfer
 - Interrupt vs. Programmed I/O
 - Direct Memory Access (DMA)
- Operating system's Role

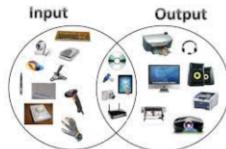


Spring 2025

7

I/O Devices

- Same components for all kinds of computers
 - Desktop, server, embedded
- Input/output includes
 - User-interface devices
 - Display, keyboard, mouse
 - Storage devices
 - Hard disk, CD/DVD, flash
 - Network adapters
 - For communicating with other computers



Spring 2025

8

keyboards, displays, printers, magnetic drives, compact disc read-only memory (CD-ROM) drives, digital video disc read-only memory (DVD-ROM) drives, network devices or other communication interfaces, scanners, and sound cards with speakers and microphones, analog-to-digital converters, digital-to-analog converters, and other data-acquisition and control components

I/O Devices Characteristics - 1

- Behavior:

- Input (read only)
- output (write only, cannot be read)
- storage (can be reread and usually rewritten)



I/O Devices Characteristics - 2

- Partner:

- A **human** or a **machine** is at the other end of the I/O device
 - feeding data on input or
 - reading data on output



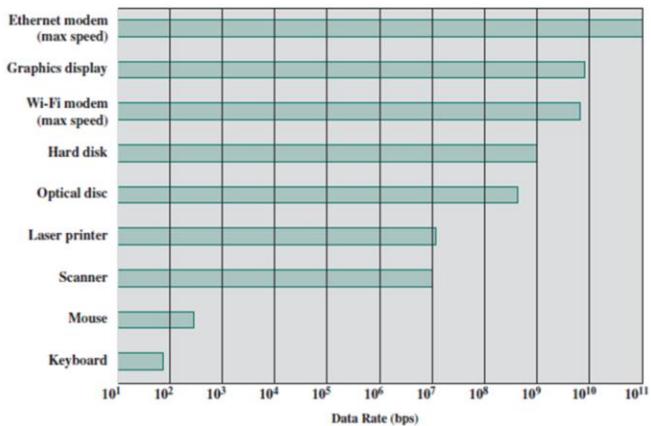
I/O Devices Characteristics - 3

- Data rate:

- The peak rate at which data can be transferred between the I/O device & memory or processor



Typical I/O Device Data Rates



I/O Devices Characteristics

- Behavior (input, output, storage)
- Partner (a human or a machine)
- Data rate
- e.g. keyboard:
 - an input device
 - used by a human
 - with a peak data rate of about 10 bytes per second



Diversity of I/O Devices

Device	Behavior	Partner	Data rate (Mbit/sec)
Keyboard	Input	Human	0.0001
Mouse	Input	Human	0.0038
Voice input	Input	Human	0.2640
Sound input	Input	Machine	3.0000
Scanner	Input	Human	3.2000
Voice output	Output	Human	0.2640
Sound output	Output	Human	8.0000
Laser printer	Output	Human	3.2000
Graphics display	Output	Human	800.0000–8000.0000
Cable modem	Input or output	Machine	0.1280–6.0000
Network/LAN	Input or output	Machine	100.0000–10000.0000
Network/wireless LAN	Input or output	Machine	11.0000–54.0000
Optical disk	Storage	Machine	80.0000–220.0000
Magnetic tape	Storage	Machine	5.0000–120.0000
Flash memory	Storage	Machine	32.0000–200.0000
Magnetic disk	Storage	Machine	800.0000–3000.0000



I/O devices can be distinguished by whether they serve as input, output, or storage devices; their communication partner (people or other computers); and their peak communication rates. The data rates span eight orders of magnitude.

Note that a network can be an input or an output device, but cannot be used for storage.

Transfer rates for devices are always quoted in base 10, so that 10 Mbit/sec = 10,000,000 bits/sec

Contents

- I/O Subsystem
- I/O Devices (Instances & Characteristics)
- I/O Interfaces
 - Memory-mapped vs. isolated I/O
- I/O Transactions
- Modes of Transfer
 - Interrupt vs. Programmed I/O
 - Direct Memory Access (DMA)
- Operating system's Role



Spring 2025

15

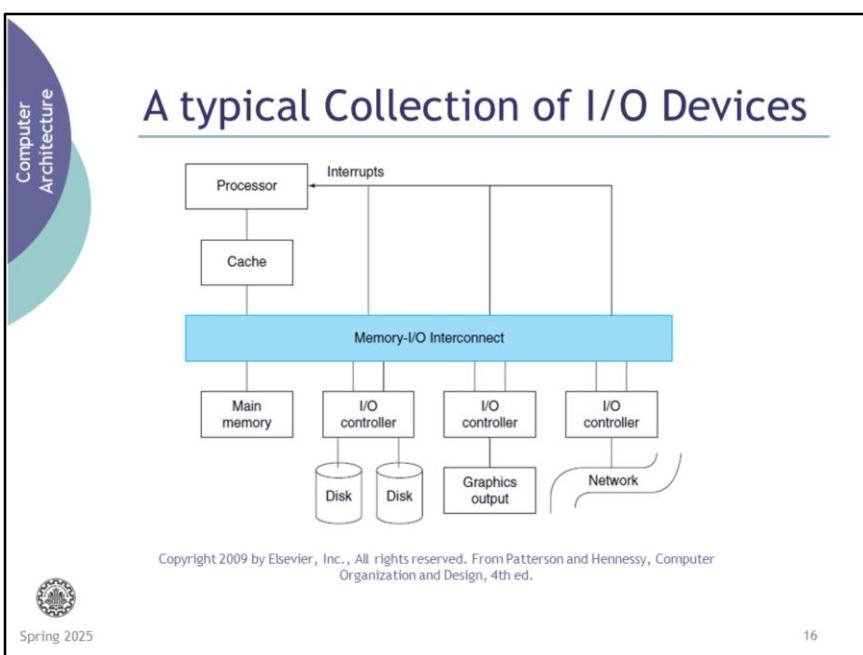


FIGURE 6.1 A typical collection of I/O devices.

The connections between the I/O devices, processor, and memory are historically called *buses*, although the term means shared parallel wires and most I/O connections today are closer to dedicated serial lines. Communication among the devices and the processor uses both interrupts and protocols on the interconnect, as we will see in this chapter.

I/O Interfaces

- Special hw components between CPU & peripheral
 - Peripherals are often electromechanical
 - A conversion of signals may be required
 - Data transfer rate of peripherals is usually different from CPU clock rate
 - A synchronization mechanism may be needed
 - Data codes and formats in peripherals differ from CPU and memory word format
 - Operating modes of peripherals differ from each other and each must be controlled independently



Peripherals connected to a computer need special communication links to interface them with the CPU. The purpose of these links is to resolve the differences in the properties of the CPU and memory and the properties of each peripheral.

To resolve these differences, computer systems include special hardware components between the CPU and the peripherals to supervise and synchronize all input and output transfers. These components are called *interface units*, because they interface between the bus from the CPU and the peripheral device.

In addition, each device has its own controller to supervise the operations of the particular mechanism of that peripheral.

Figure 11-5: Connection of I/O Devices to CPU

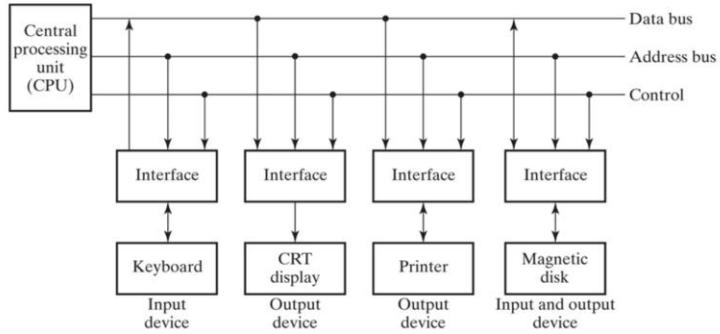
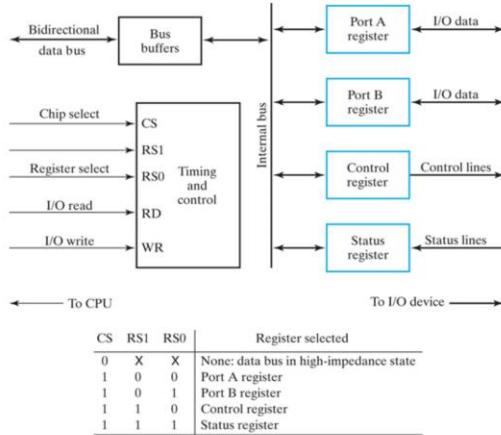


Figure 11-6: Example of I/O Interface Unit



It consists of two data registers called *ports*, a control register, a status register, a bidirectional data bus, and timing and control circuits.

The function of the interface is to translate the signals between the CPU buses and the I/O device and to provide the needed hardware to satisfy the two sets of timing constraints.

The I/O data from the device can be transferred into either port A or port B.

The interface may operate with an output device, with an input device, or with a device that requires both input and output.

The *control register* receives control information from the CPU. By loading appropriate bits into this register, the interface and the device can be placed in a variety of operating modes.

The bits in the status register are used for status conditions and for recording errors that may occur during data transfer. (e.g. port A has received a new data item from the device)

The interface registers communicate with the CPU through the bidirectional data bus.

The address bus selects the interface unit through the chip select input and the two register select inputs.

A circuit (usually a decoder or a gate) detects the address assigned to the interface registers. This circuit sets the chip select (CS) input when the interface is selected by the address bus.

The two *register select inputs* RS_1 and RS_0 are usually connected to the two least significant lines of the address bus.

The contents of the selected register are transferred into the CPU via the data bus when the I/O read signal is set. The CPU transfers binary information into the selected register via the data bus when the I/O write input is set.

I/O Bus Configuration

- Memory mapped I/O
 - uses common data, address, and control buses for both memory and I/O
 - **common** address space
- Isolated I/O
 - shares a common address bus and data bus, but uses different control lines for memory and I/O
 - **isolated** address space



Memory-mapped I/O:

The common address space is shared between the interface units and memory words, each having distinct addresses.

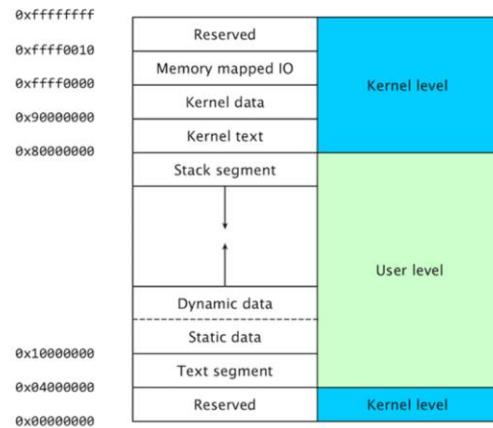
Computers that adopt the memory-mapped scheme read and write from interface units as if they were assigned memory addresses by using the same instructions that read from and write to memory.

Isolated I/O:

Have separate read and write lines for memory and I/O. To read or write from memory, the CPU activates the memory read or memory write control. To perform input to or output from an interface, the CPU activates the read I/O or write I/O control, using special instructions.

In this way, the addresses assigned to memory and I/O interface units are independent from each other and are distinguished by separate control lines.

Memory-Mapped I/O in MIPS



Memory-Mapped I/O

- Common address space is shared between the interface units and memory words, each having distinct addresses
- Performed by simple **load/store** instructions
- Memory & devices on bus programmed to respond only to their own address ranges



Spring 2025

22

Memory-Mapped I/O (cont.)

☺ Pros

- easy to handle memory locations
(software perspective)

⊗ Cons

- caching may cause memory incoherency
- slow (consumes CPU cycles)



If the CPU cache retains a cached copy of a memory-mapped control register, and the external device updates that register, the cached copy becomes outdated. When the CPU attempts to read or write to the memory-mapped region, it may use the outdated cached value instead of the current value in the control register.

Isolated I/O

- Dedicated I/O instructions
 - part of ISA
- Output
 - values are written to an output register
 - on the output pins of an external port
- Input
 - values are read from an input register
 - from the input pins of an external port
- Example: Intel x86 IN/OUT instructions



Isolated I/O (cont.)

☺ Pros

- low latency
 - simpler decoding

⊗ Cons

- not implemented by all CPUs
- limited addressing
 - predetermined # of I/O signals



Contents

- I/O Subsystem
- I/O Devices (Instances & Characteristics)
- I/O Interfaces
 - Memory-mapped vs. isolated I/O
- I/O Transactions
- Modes of Transfer
 - Interrupt vs. Programmed I/O
 - Direct Memory Access (DMA)
- Operating system's Role



Spring 2025

26

I/O Transactions

- An I/O transaction consists of:
 - sending address and command
 - sending or receiving data
- Types of I/O transactions:
 - Input
 - from input devices to memory or processor
 - Output
 - from memory or processor to output devices



Spring 2025

27

I/O transaction is a sequence of operations over the interconnect that includes a request and may include a response, either of which may carry data.

A transaction is initiated by a single request and may take many individual bus operations.

Sync/Async Interconnection

- **Synchronous** interconnection

- includes a **clock** in the control lines
- uses a fixed protocol for communicating that is relative to the clock

- **Asynchronous** interconnection

- not clocked
- uses a **handshaking** protocol for coordinating usage rather than a clock



Spring 2025

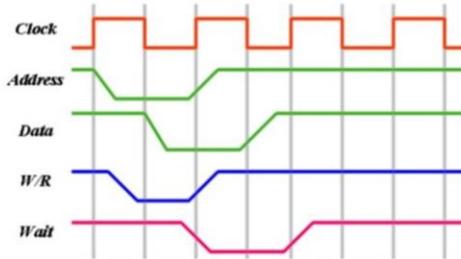
28

A handshaking protocol consists of a series of steps in which the sender and receiver proceed to the next step only when both parties agree that the current step has been completed.

The protocol is implemented with an additional set of control lines.

Synchronous Interconnection

- a **clock** in the control lines
- a **fixed** communicating protocol relative to the clock



e.g. for performing a read from memory, we might have a protocol that transmits the address and read command on the first clock cycle, using the control lines to indicate the type of request. The memory might then be required to respond with the data word on the 5th clock

Synch Interconnection (pros/cons)

- ⌚ can be implemented **easily** in a small finite-state machine
- ⌚ the bus can run **fast**, and the interface logic will be **small**
- ⌚ Every device on the bus must run at the **same clock rate**
- ⌚ **cannot be long** if fast (due to clock skew problems)



Asynchronous Interconnection

- not clocked
- uses a **handshaking** protocol for coordinating usage rather than a clock
 - consists of a series of steps in which the sender and receiver proceed to the next step only when both parties agree.
 - the **protocol** is implemented with an additional set of control lines



Spring 2025

31

The CPU, interface, and I/O device are likely to have different clocks that are not synchronized with each other.

Asynch Interconnection (pros/cons)

- ☺ can accommodate a **wide variety of devices** of different speeds
- ☺ the bus can be **lengthened** without worrying about clock skew or synchronization problems
- ⊗ ?



Spring 2025

32

Cons: Need to extra control lines for handshaking

Typical Asynchronous Protocols

- Strobing

- source-initiated
- destination-initiated



- Handshaking

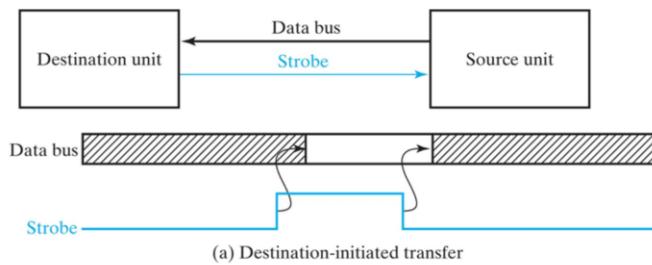
- source-initiated
- destination-initiated



Spring 2025

33

Figure 11-7: Asynchronous Transfer Using Strobing



In the shaded area of the data signal, the data is invalid.

A change in Strobe at the tail of each arrow causes a change on the data bus at the head of the arrow.

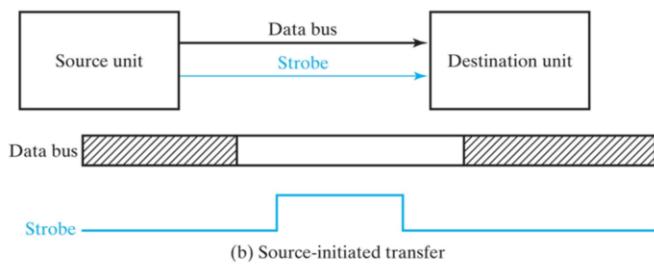
The destination unit changes Strobe from 0 to 1. When the value 1 on Strobe reaches the source unit, the unit responds by placing the data on the data bus.

The destination unit expects the data to be available, at worst, a fixed amount of time after Strobe goes to 1.

At that time, the destination unit captures the data in a register and changes Strobe from 1 to 0.

In response to the 0 value on Strobe, the source unit removes the data from the bus.

Figure 11-7: Asynchronous Transfer Using Strobing



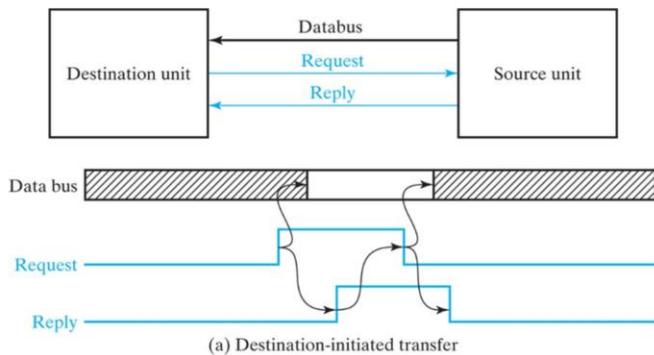
The source unit places the data on the data bus. After a short time required for the data to settle on the bus, the source unit changes Strobe from 0 to 1.

In response to Strobe equal to 1, the destination unit sets up the transfer to one of its registers.

The source then changes Strobe from 1 to 0, which triggers the transfer into the register at the destination.

Finally, after a short time required to ensure that the register transfer is done, the source removes the data from the data bus, completing the transfer.

Figure 11-8: Asynchronous Transfer Using Handshaking



In the initial state both Request and Reply are reset and in the 00 state.

The destination unit initiates the transfer by enabling Request.

The source unit responds by placing the data on the bus.

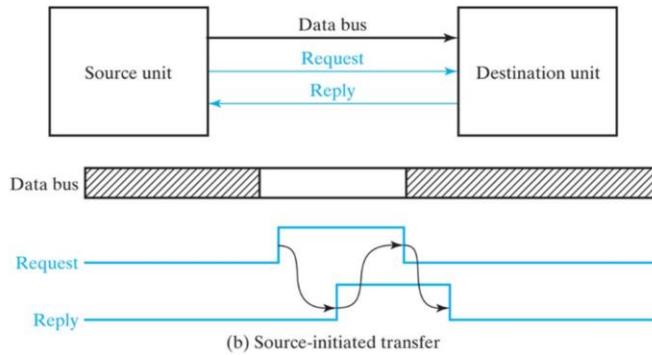
After a short time for settling of the data on the bus, the source unit activates Reply, to signal the presence of the data.

In response to Reply, the destination unit captures the data in a register and resets Request.

The source unit then resets Reply, and the system goes to the initial state.

The destination unit may not make another request until the source unit has shown its readiness to provide new data by disabling Reply.

Figure 11-8: Asynchronous Transfer Using Handshaking



The source controls the interval between when the data is applied and when Request changes to 1 and between when Request changes to 0 and when the data is removed.

The handshaking scheme provides a high degree of flexibility and reliability, because the successful completion of a data transfer relies on active participation by both units.

If one unit is faulty, the data transfer will not be completed.

Such an error can be detected by means of a time-out mechanism, which produces an alarm if the data transfer is not completed within a predetermined time interval.

Contents

- I/O Subsystem
- I/O Devices (Instances & Characteristics)
- I/O Interfaces
 - Memory-mapped vs. isolated I/O
- I/O Transactions
- Modes of Transfer
 - Interrupt vs. Programmed I/O
 - Direct Memory Access (DMA)
- Operating system's Role

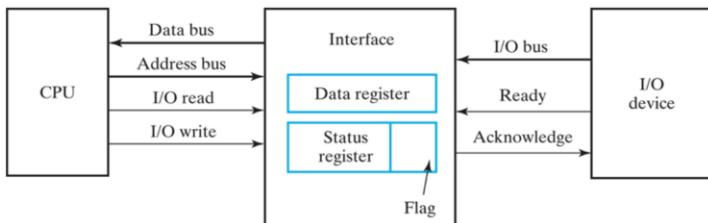


Modes of Transfer

- Data transfer to and from peripherals may be handled in three modes:
 - Data transfer under **program control**
 - **Interrupt-initiated** data transfer
 - Direct Memory Access (**DMA**) transfer



Figure 11-13: Data Transfer from I/O Device to CPU



The device transfers bytes of data one at a time as they are available.

When a byte is available, the device places it on the I/O bus and sets Ready.

The interface accepts the byte into its data register and sets Acknowledge.

The interface sets a bit in the status register, which we will refer to as a *flag*.

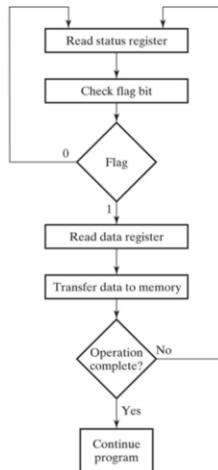
The device can now reset Ready, but it will not transfer another byte until Acknowledge is reset by the interface, according to the handshaking procedure established in Section 11-3.

Under program control, the CPU must check the flag to determine whether there is a new byte in the interface data register.

This is done by reading the contents of the status register into a CPU register and checking the value of the flag.

If the flag is equal to 1, the CPU reads the data from the data register. The flag is then cleared to 0 either by the CPU or the interface. Once the flag is cleared, the interface resets Acknowledge, and the device can transfer the next data byte.

Figure 11-14: Flowchart for CPU Program to Input Data



The flowchart assumes that the device is sending a sequence of bytes that must be stored in memory.

The program continually examines the status of the interface until the flag is set to 1.

Each byte is brought into the CPU and transferred to memory until all of the data have been transferred.

Program Controlled I/O Transfer

- Checks the status of I/O device periodically to determine the need to service
 - ⊕ Simple
 - ⊕ More predictable I/O overhead
 - ⊕ can waste a lot of processor time
- Used in real-time applications because of predictability



The program-controlled data transfer is used only in systems that are dedicated to monitor a device continuously.

The difference in information transfer rate between the CPU and the I/O device makes this type of transfer inefficient:

- read the status register and check the flag in 100 ns,
- input data rate 100 bytes/s (one byte every 10,000 μ s),
- meaning that the CPU will check the flag 100,000 times between each transfer

Predictable means we can estimate how much cycle that used for operations duo to this fact

Interrupt-Initiated I/O Transfer

- A scheme that **interrupts** the processor to indicate that an I/O device needs **attention**
 - Overcomes CPU waiting
 - No repeated CPU checking of device
 - I/O device interrupts when ready
- I/O device
 - **Raises** the interrupt
 - **Identifies** itself



The interface informs the computer when it is ready to transfer data. While the CPU is running a program, it does not check the flag.

However, when the flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed that the flag has been set. The CPU drops what it is doing to take care of the input or output transfer.

Program Interrupt

- Used to handle situations that require a departure from the normal program sequence
- Transfers control from a program that is currently running on another service program as a result of an **externally** or **internally** generated request
- Control returns to the original program after the service program is executed



Spring 2025

44

ISR: Interrupt Service Routine (Interrupt Handler)

Interrupt vs. Procedure Call

- The interrupt is usually initiated at an **unpredictable** point in the program by an external or internal signal, rather than the execution of an instruction
- The address of the service program that processes the interrupt request is determined by a **hardware** procedure, rather than the address field of an instruction
- In response to an interrupt, it is necessary to store information that defines all or part of the contents of the **register set**, rather than storing only the program counter



After the computer has been interrupted and the appropriate service program executed, the computer must return to exactly the same state that it was in before the interrupt occurred.

Types of Interrupts

- **External** Interrupts
 - come from external sources
- **Internal** Interrupts
 - arise from the invalid or erroneous use of an instruction or data
- **Software** Interrupts
 - initiated by executing an instruction



Processing Interrupts

- Very similar to the execution of a procedure call instruction:
 - The contents of the register set of the processor are temporarily **stored** in memory (typically pushed onto a memory stack)
 - The address of the interrupt service program is **loaded into the PC**
- The address of the service program is chosen by the hardware



Service Program Address

- Some computers assign one memory location for the beginning address of the service program
 - the service program must then determine the source of the interrupt and proceed to service it
- Other computers assign a separate memory location for each possible interrupt source
 - sometimes, the interrupt source hardware itself supplies the address of the service routine
- In any case, the computer must possess some form of **hardware procedure** for selecting a branch address for servicing the interrupt



Spring 2025

48

Interrupt Cycle

- Just **before going to fetch** the next instruction, the control checks for any interrupt signals
- If an interrupt has occurred, control goes to a hw interrupt cycle
 - Program Counter (PC) is pushed onto the stack
 - Branch address for the particular interrupt is then transferred to PC
 - the control goes to fetch the next instruction (beginning of the ISR)
- At the beginning of the service routine, part or all of the register set are pushed onto the stack
- The last instruction in the service routine is a return from the interrupt instruction
 - The stack is popped to retrieve the return address and registers



Spring 2025

49

Most computers will not respond to an interrupt until the instruction that is in the process of being executed is completed

Software Interrupts

- a special **call** instruction that behaves like an **interrupt** rather than a procedure call
- can be used by the programmer to initiate an interrupt procedure at any desired point in the program
- Typical use: **system call**
 - provides a means for switching from user mode to system mode



A system call instruction provides a means for switching from user mode to system mode.

Certain operations in the computer may be performed by the operating system only in system mode.

For example, a complex input or output procedure is done in system mode. In contrast, a program written by a user must run in user mode.

When an input or output transfer is required, the user program causes a software interrupt, which stores the contents of the *PSR* (with the mode bit set to “user”), loads new *PSR* contents (with the mode bit set to “system”), and initiates the execution of a system program. The calling program must pass information to the operating system in order to specify the particular task that is being requested.

Internal Interrupts

- also known as **exceptions** or traps
- Sources:
 - arithmetic overflow
 - attempt to divide by zero
 - invalid opcode
 - memory stack overflow
 - protection violation
- Corresponding service programs determine the corrective measure to be taken in each case



A protection violation is an attempt to address an area of memory that is not supposed to be accessed by the currently executing program.

External Interrupts

- Sources:
 - I/O device
 - Requesting transfer of data
 - Completing transfer of data
 - Time-out of an event
 - Impending power failure
- may have single or multiple interrupt input lines

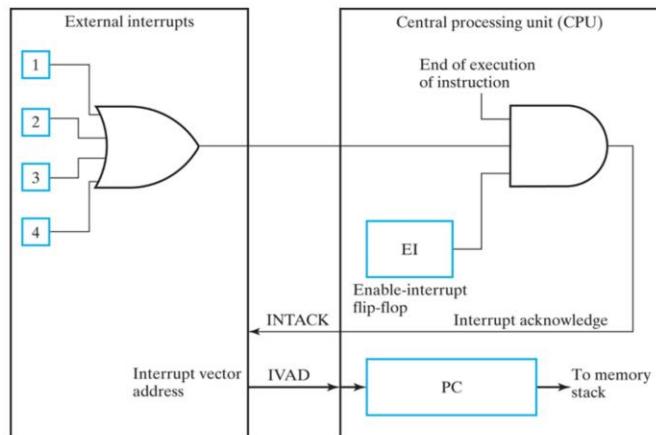


A time-out interrupt may result from a program that is in an endless loop and thus exceeds its time allocation.

A power-failure interrupt may have as its service program a few instructions that transfer the complete contents of the register set of the processor into a nondestructive memory such as a disk in the few milliseconds before power ceases.

If there are more interrupt sources than there are interrupt inputs in the computer, two or more sources are Ored to form a common line.

Figure 9-9: Example of External Interrupt Configuration



Design Issues

- How do you identify the module issuing the interrupt?
- How do you deal with multiple interrupts?
 - i.e. an interrupt handler being interrupted



Spring 2025

54

Identifying Interrupting Module

- Multiple interrupt lines
- Software poll
- Daisy Chain or Hardware poll
- Parallel Priority Hardware
- Bus arbitration

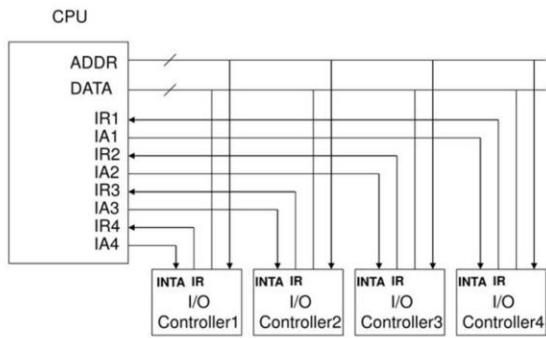


Spring 2025

55

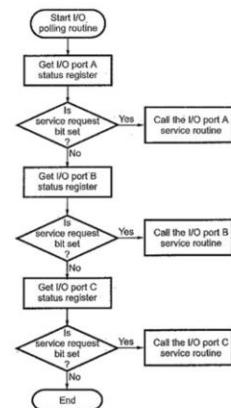
Multiple Interrupt Lines

- Different line for each module
- ⌚ Limits number of devices



Software Poll

- CPU asks each module in turn
- ⌚ Time consuming



Hardware Poll (Daisy Chain)

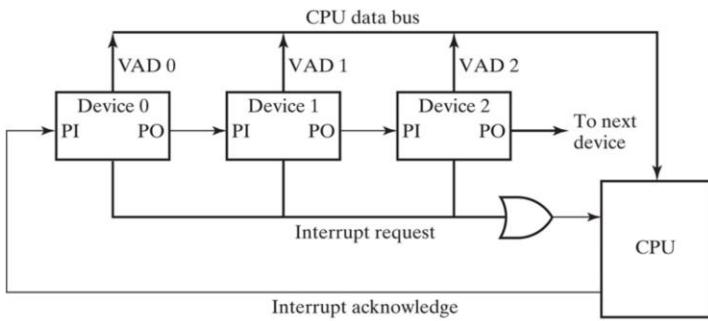
- Interrupt Acknowledge sent down a chain
- Module responsible places **vector** on bus
- CPU uses vector to identify interrupt



Spring 2025

58

Figure 11-15: Daisy Chain Priority Interrupt



PI: Priority In

PO: Priority Out

VAD: Vector Address

Figure 11-16: One Stage of the Daisy Chain Priority Arrangement

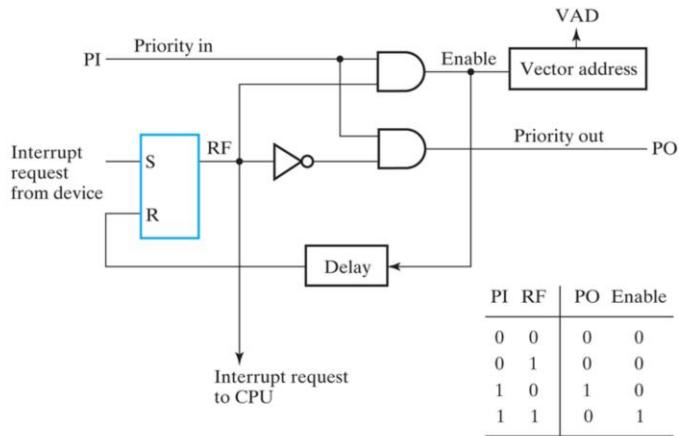
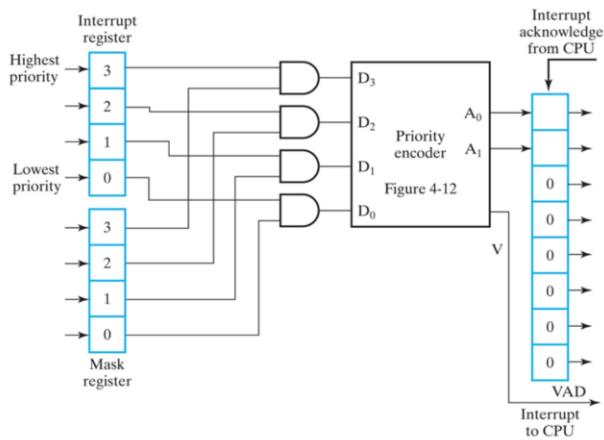
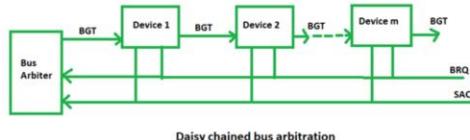
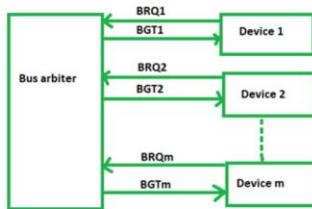


Figure 11-17: Parallel Priority Interrupt Hardware



Bus Arbitration

- Module must claim the bus before it can raise interrupt

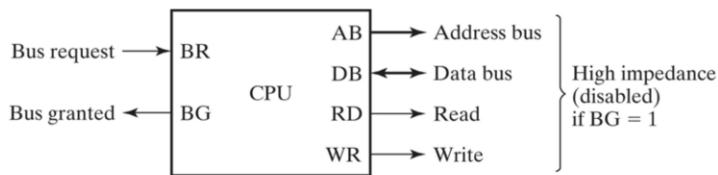


Spring 2025

62

In a computer system, multiple devices, such as the CPU, memory, and I/O controllers, are connected to a common communication pathway, known as a bus. In order to transfer data between these devices, they need to have access to the bus. Bus arbitration is the process of resolving conflicts that arise when multiple devices attempt to access the bus at the same time.

Figure 11-18: CPU Bus Control Signals



The bus request (*BR*) input is used to request the CPU to relinquish control of the buses.

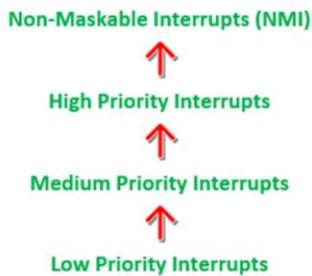
When *BR* input is active, the CPU places the address bus, the data bus, and the read and write lines into a high-impedance state.

After this is done, the CPU activates the bus granted (*BG*) output to inform the external DMA that it can take control of the buses.

As long as the *BG* line is active, the CPU is unable to proceed with any operations requiring access to the buses.

Multiple Interrupts

- Each interrupt has a priority
- Higher priority lines can interrupt lower priority lines

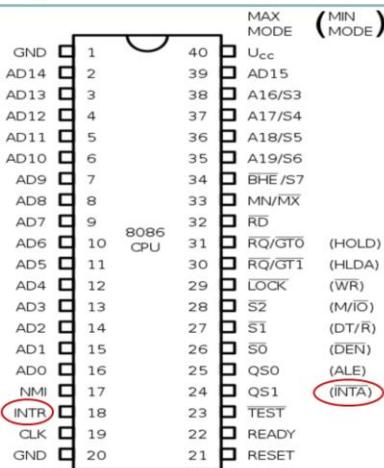


Handling Priority

- Multiple interrupt lines
 - processor picks the interrupt line with highest priority
- Software poll
 - order in which modules are polled determines priority
- Hardware poll (Daisy Chain)
 - order of modules on a daisy chain determines priority
- Parallel Priority Hardware
 - Priority determined by the priority encoder
- Bus Arbitration
 - Priority determined by the bus arbiter



8086 Chip



Spring 2025

66

Contents

- I/O Subsystem
- I/O Devices (Instances & Characteristics)
- I/O Interfaces
 - Memory-mapped vs. isolated I/O
- I/O Transactions
- Modes of Transfer
 - Interrupt vs. Programmed I/O
 - Direct Memory Access (DMA)
- Operating system's Role



Spring 2025

70

High-bandwidth I/O

- Programmed I/O & interrupt are suitable for **low** bandwidth devices
 - Data transfer is done in small no of bytes
- For **higher** bandwidth (e.g. hard disks)
 - Programmed I/O is used in real-time applications
 - Interrupt driven approach
 - Helps OS to work on other tasks while actual I/O transfer is done
 - But the overhead of each transfer is still intolerable
- Solution: **Direct Memory Access (DMA)**



Direct Memory Access (DMA)

- A mechanism that provides a device controller with the ability to **transfer data directly** to or from the **memory** without involving the processor
- Implemented with a specialized controller that transfers data between an I/O device and memory **independent of the processor**



Modes of DMA Transfer

- Burst Transfer

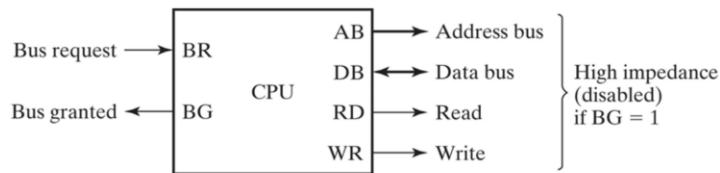
- Transfer is made for an entire block of memory words, suspending operation of the CPU until the **entire block** is transferred

- Single-cycle Transfer (cycle-stealing)

- Transfer is made one word at a time between executions of CPU instructions
 - CPU merely delays its bus operations for one memory cycle to allow the direct memory-I/O transfer to **steal** one memory cycle



Figure 11-18: CPU Bus Control Signals



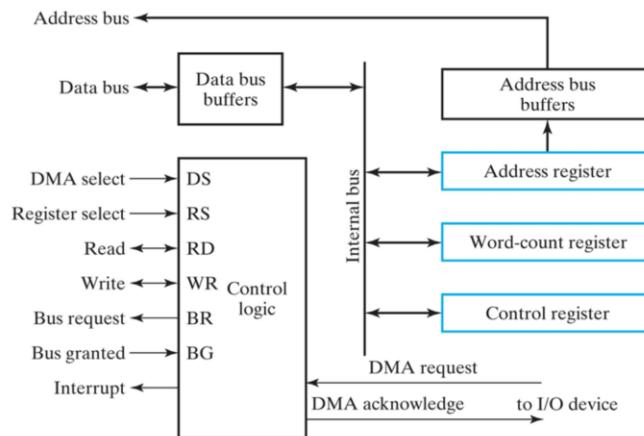
The bus request (*BR*) input is used by the DMA controller to request the CPU to relinquish control of the buses.

When *BR* input is active, the CPU places the address bus, the data bus, and the read and write lines into a high-impedance state.

After this is done, the CPU activates the bus granted (*BG*) output to inform the external DMA that it can take control of the buses.

As long as the *BG* line is active, the CPU is unable to proceed with any operations requiring access to the buses.

Figure 11-19: Block Diagram of a DMA Controller



The address register and address lines are used for direct communication with memory.

The word-count register specifies the number of words that must be transferred.

The control register specifies the mode of transfer.

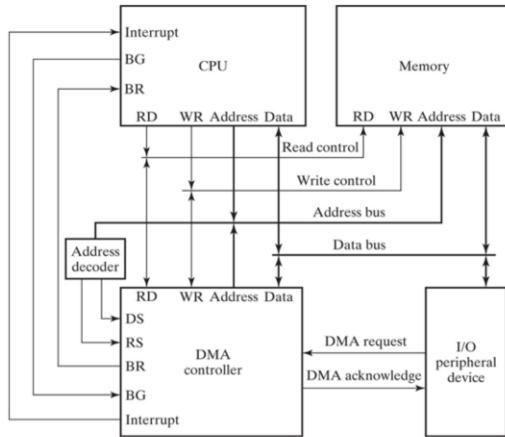
All registers in the DMA appear to the CPU as I/O interface registers. Thus, the CPU can read from or write to the DMA registers under program control via the data bus.

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



Figure 11-20: DMA Transfer in a Computer System



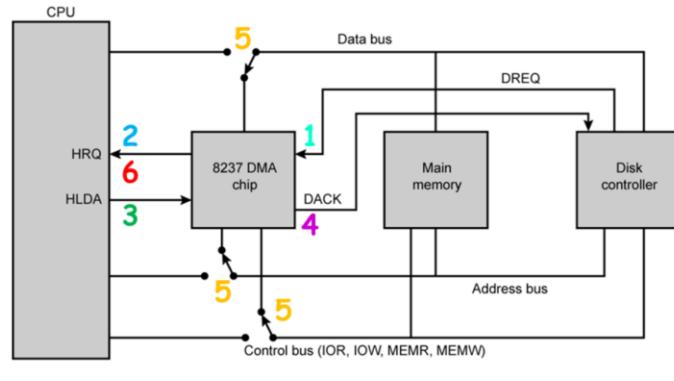
The bus request (BR) input is used by the DMA controller to request the CPU to relinquish control of the buses.

When BR input is active, the CPU places the address bus, the data bus, and the read and write lines into a high-impedance state.

After this is done, the CPU activates the bus granted (BG) output to inform the external DMA that it can take control of the buses.

As long as the BG line is active, the CPU is unable to proceed with any operations requiring access to the buses.

DMA Usage of Systems Bus



DACK = DMA acknowledge
DREQ = DMA request
HLDA = HOLD acknowledge (bus acknowledge)
HRQ = HOLD request (bus request)

Spring 2025

78

HRQ (Hold Request) is equivalent to BR (Bus Request)

HLDA (Hold Acknowledge) is equivalent to BG (Bus Granted)

- 1- The peripheral device will request the service of DMA by pulling DREQ high
- 2- The DMA controller will put a high on its HRQ, signaling the CPU through its HOLD pin that it needs to use the buses
- 3- The CPU will finish the present bus cycle and responds to the HRQ by putting high on its HLDA
- 4- DMA controller will activate DACK which tells the peripheral device that it can start to transfer the data
- 5- DMA controller starts to transfer the data from memory to peripheral ...
- 6- After transfer is complete DMA controller deactivates HRQ, signaling the CPU that it can regain control over its buses

Contents

- I/O Subsystem
- I/O Devices (Instances & Characteristics)
- I/O Interfaces
 - Memory-mapped vs. isolated I/O
- I/O Transactions
- Modes of Transfer
 - Interrupt vs. Programmed I/O
 - Direct Memory Access (DMA)
- Operating system's Role



Operating System

- Interface between a user's program and the hw
- Provides variety of services and supervisory functions
 - Handling basic input and output operations
 - Allocating storage and memory
 - Providing for protected sharing of the computer among multiple applications using it simultaneously
- e.g., Linux, MacOS, and Windows



Spring 2025

80

Why Operating System?

- OS responsibilities arise from I/O characteristics:
 - Multiple programs using the processor **share** the I/O system
 - I/O systems often use **interrupts** to communicate information about I/O operations
 - Interrupts cause a transfer to kernel or supervisor mode, so they must be handled by the OS
 - The low-level control of an I/O device is **complex**
 - requires managing a set of concurrent events
 - requirements for correct device control are detailed



Operating System's Role

- Guarantees that a user's program accesses **only** the portions of an I/O device to which the user has **rights**
 - e.g., must not allow a program to read or write a file on disk if the owner of the file has not granted access to this program
 - In a system with shared I/O devices, protection could not be provided if user programs could perform I/O directly
- ...



Spring 2025

82

Operating System's Role (cont.)

- Guarantees that a user's program accesses **only** the portions of an I/O device to which the user has **rights**
- Provides **abstractions** for accessing devices by supplying routines that handle low-level device operations
- Handles the **interrupts** generated by I/O devices, just as it handles the **exceptions** generated by a program
- Tries to provide **equitable** access to the shared I/O resources
- Tries to **schedule** accesses to enhance system throughput



Spring 2025

83

OS Communication with I/O

- Give commands to the I/O devices:
 - read, write, disk seek, ...
- Be notified when the I/O device has completed an operation or has encountered an error
 - e.g., when a disk completes a seek, it will notify OS
- Transfer data between memory and I/O device
 - e.g., the block being read from a disk must be moved from disk to memory



Outlines

- I/O Devices
- I/O Interfaces
- Memory-mapped vs. isolated I/O
- I/O Transactions
- Program Controlled Transfer
- Interrupt Initiated Transfer
- Direct Memory Access (DMA)
- Operating system's Role

