I/O FUNDAMENTALS

ANISHA M. LAL

Examples of I/O Devices

- A personal computer
  - Keyboard (input)
  - Mouse (input)
  - Monitor (output)
  - Printer (output)

Introduction

- The I/O subsystem provides an efficient mode of communication between CPU and outside environment
- Devices that are under the direct control of computer are said to be connected on-line
- Input or output devices attached to the computer are also referred as peripherals.
- I/O interface or I/O Module provides a method for transferring information between internal storage and external i/o devices.

I/O interface

Resolves the *differences* between the computer and peripheral devices

Peripherals - Electromechanical Devices
CPU or Memory - Electronic Device

- **Data Transfer Rate**
  - Peripherals - Usually slower
  - CPU or Memory - Usually faster than peripherals
  - Some kinds of Synchronization mechanism may be needed

- **Unit of Information**
  - Peripherals - Byte
  - CPU or Memory - Word

- **Operating Modes**
  - Peripherals - Asynchronous
  - CPU or Memory - Synchronous

I/O Bus and Memory Bus

- *MEMORY BUS* **is for information transfers between CPU and the Main Memory.**

- *I/O BUS* **is for information transfers between CPU and I/O devices through their I/O interface.**

- **Many computers use a common single bus system for both memory and I/O interface units**

  - Use one common bus but separate control lines for each function

  - Use one common bus with common control lines for both functions

Isolated I/O

Isolated I/O

- Separate I/O read/write control lines in addition to memory read/write control lines

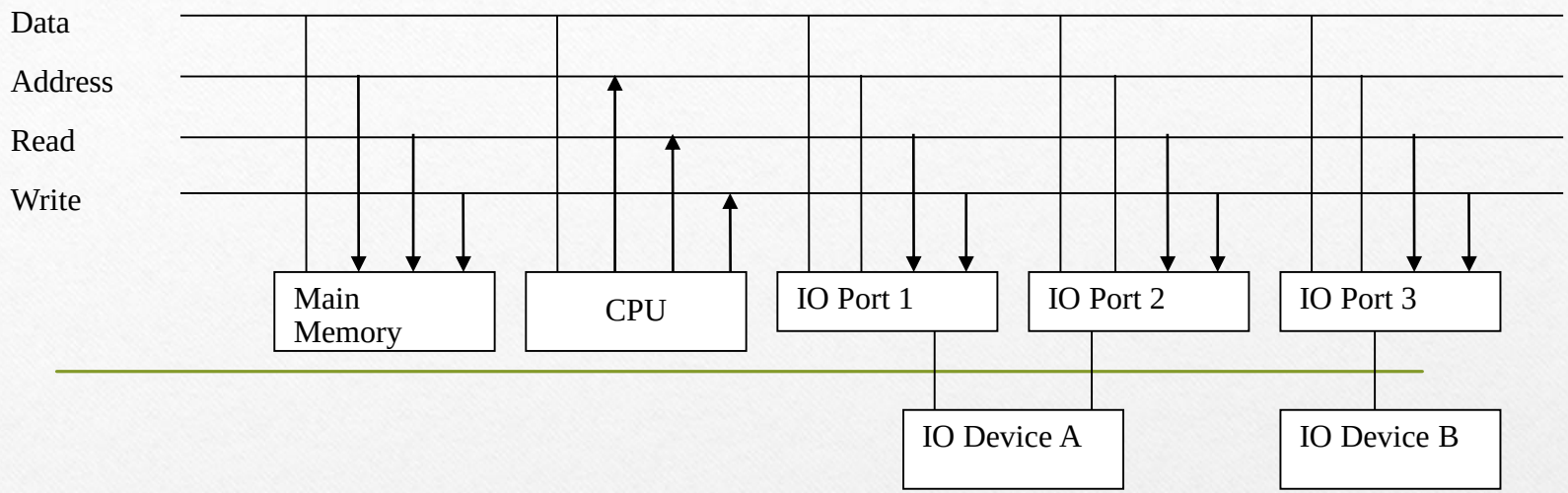
- Separate (isolated) memory and I/O address spaces

- Distinct input and output instructions

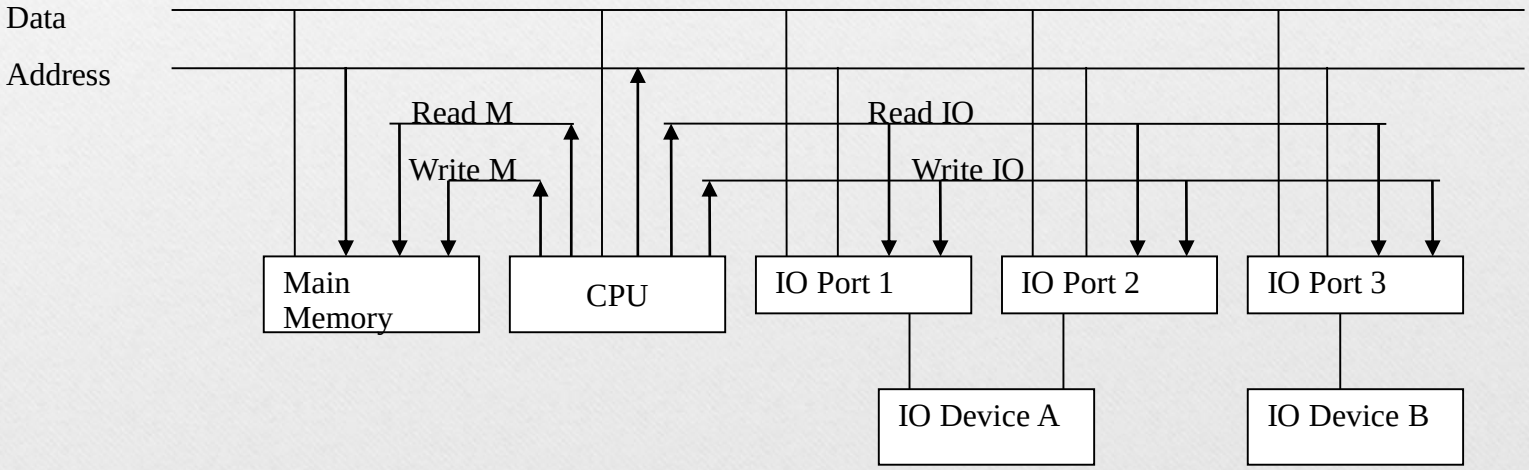
- When CPU fetches and decodes the opcode of an I/O instruction, it places the address into the common address lines. Also enables read/write control lines => the address in the address lines is for interface register and not for a memory word.

Memory Mapped I/O

- A single set of read/write control lines
- Memory and I/O addresses share the common address space .
  - Reduces memory address range available.
- No specific input or output instruction
- The same memory reference instructions can be used for I/O transfers
- When the bus sees certain addresses, it knows they are not memory addresses, but are addresses for accessing I/O devices.



Memory-Mapped I/O



I/O Mapped I/O

Asynchronous Data Transfer

Synchronous - All devices derive the timing information from common clock line.

Asynchronous - No common clock

Asynchronous Data Transfer:-

Asynchronous data transfer between two independent units requires that *control signals* be transmitted between the communicating units *to indicate the time at which data is being transmitted*

Asynchronous Data Transfer

Two Asynchronous Data Transfer Methods:

Strobe pulse :-

- A strobe pulse is supplied by one unit to indicate the other unit when the transfer has to occur.

Handshaking:-

- A control signal is accompanied with each data being transmitted to indicate the presence of data.

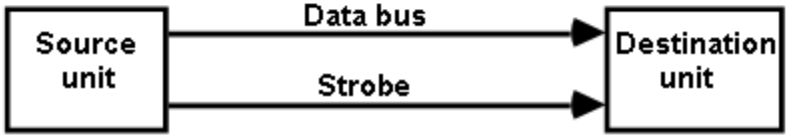
- The receiving unit responds with another control signal to acknowledge receipt of the data.

Strobe Control

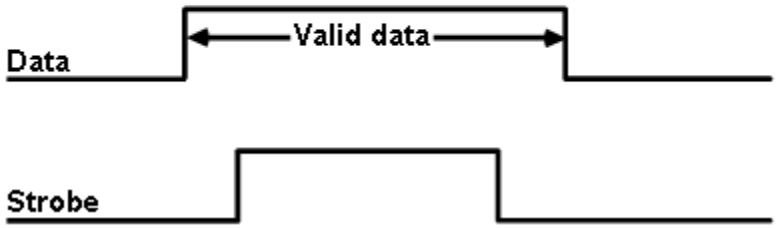
- The strobe may be activated by either the source or the destination unit
- Data bus carries the binary information from source to destination

Source-Initiated Strobe for Data Transfer

Block Diagram

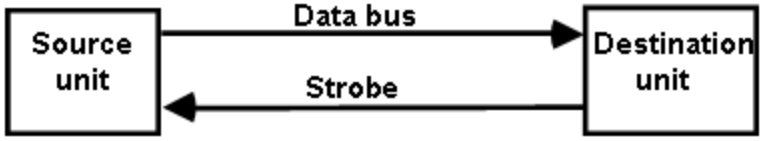


Timing Diagram

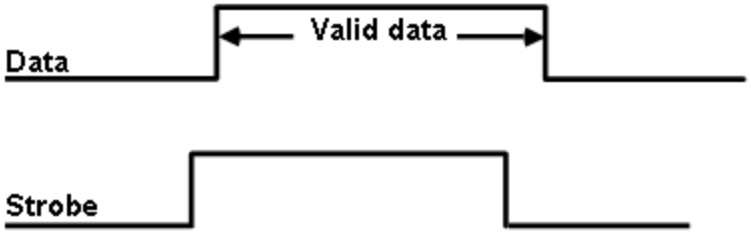


Destination-Initiated Strobe for Data Transfer

Block Diagram



Timing Diagram



Source initiated strobe

- It could be a memory write control signal from the CPU to a memory unit.
- Source is the CPU, places a word on the data bus and informs the memory unit which is destination, that this is a write operation.
- Disadvantage – no way of knowing whether the destination unit has actually received data.

Destination initiated strobe

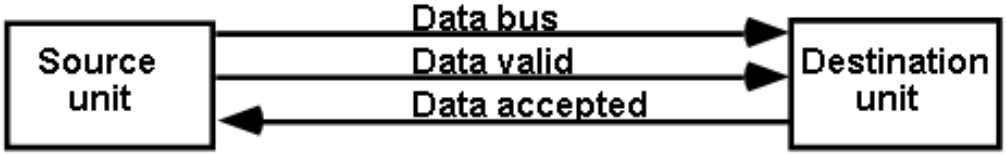
- It could be a memory read control signal.
- CPU, the destination initiates the read operation to inform the memory which is the source to place a selected word into the data bus.
- Disadvantage - no way of knowing whether the source has actually placed the data on the bus.

Hand Shaking

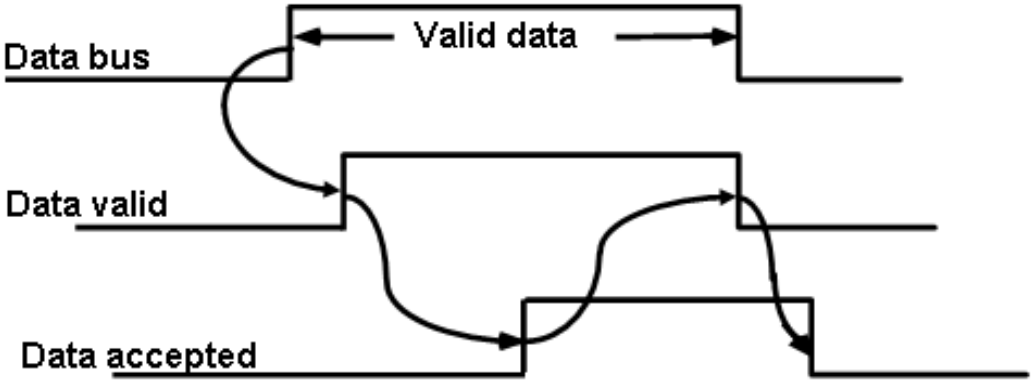
- The handshake method introduces a second control signal to provide a *reply* to the unit that initiates the transfer.
- 2 types
  - Source initiated transfer
  - Destination initiated transfer
- Provides high degree of flexibility and reliability
- Incompletion of data transfer can be detected by means of a timeout mechanism
- The timeout signal can be used to interrupt the processor and hence execute a service routine that takes appropriate error recovery action.

SOURCE-INITIATED TRANSFER USING HANDSHAKE

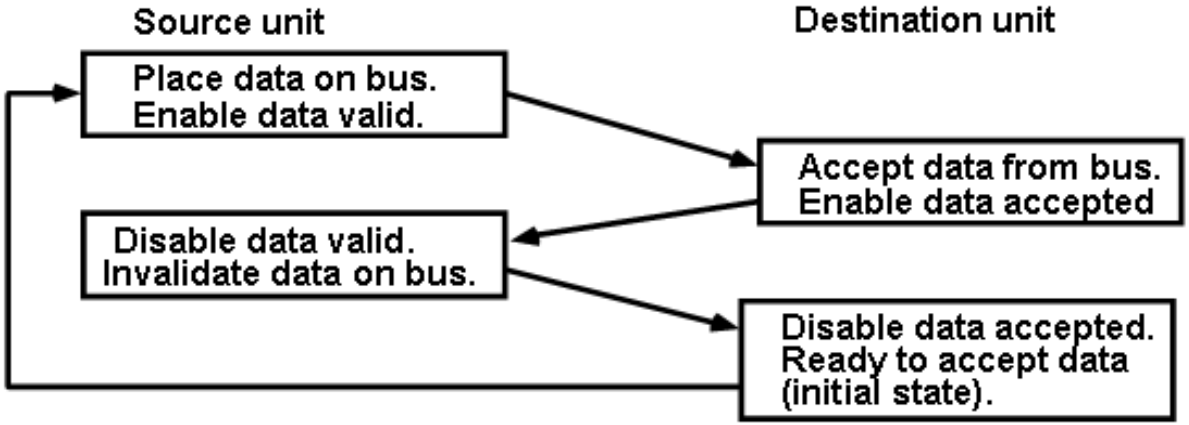
Block Diagram



Timing Diagram

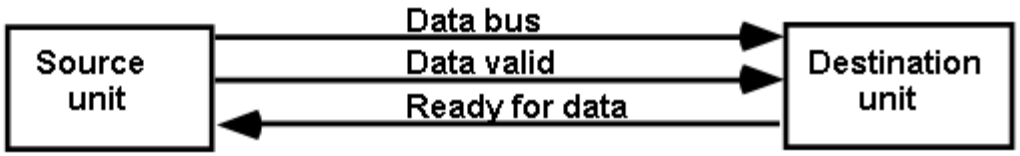


Sequence of Events

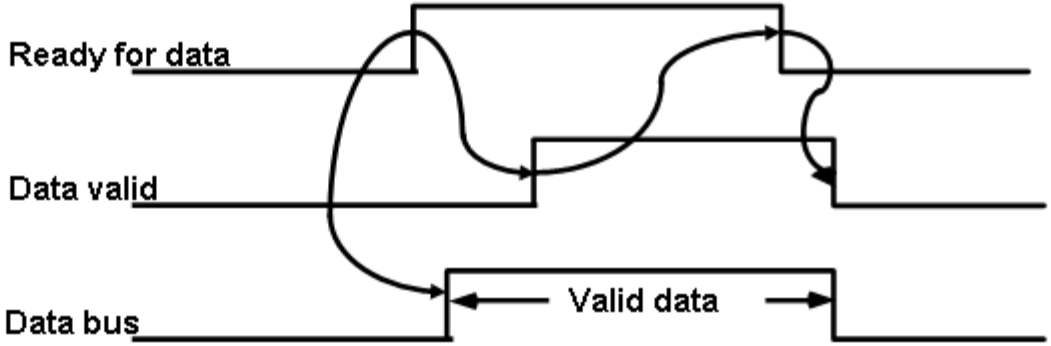


DESTINATION-INITIATED TRANSFER USING HANDSHAKE

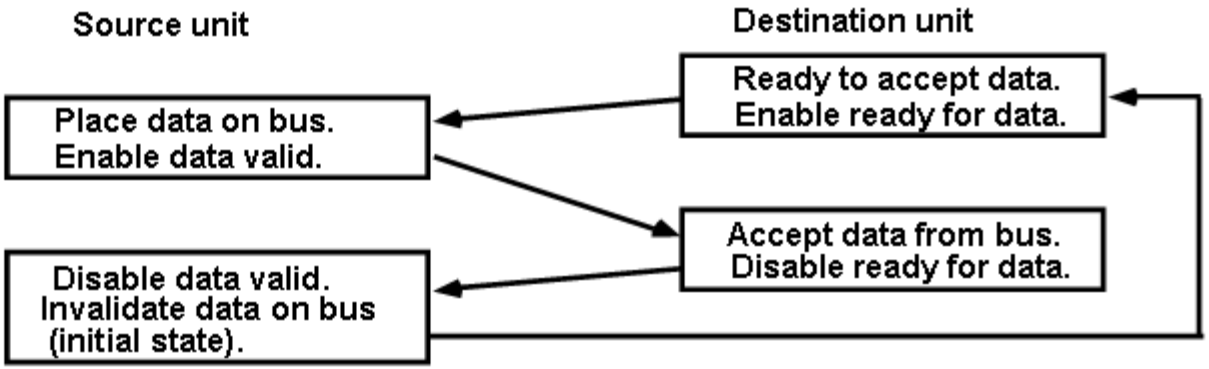
Block Diagram



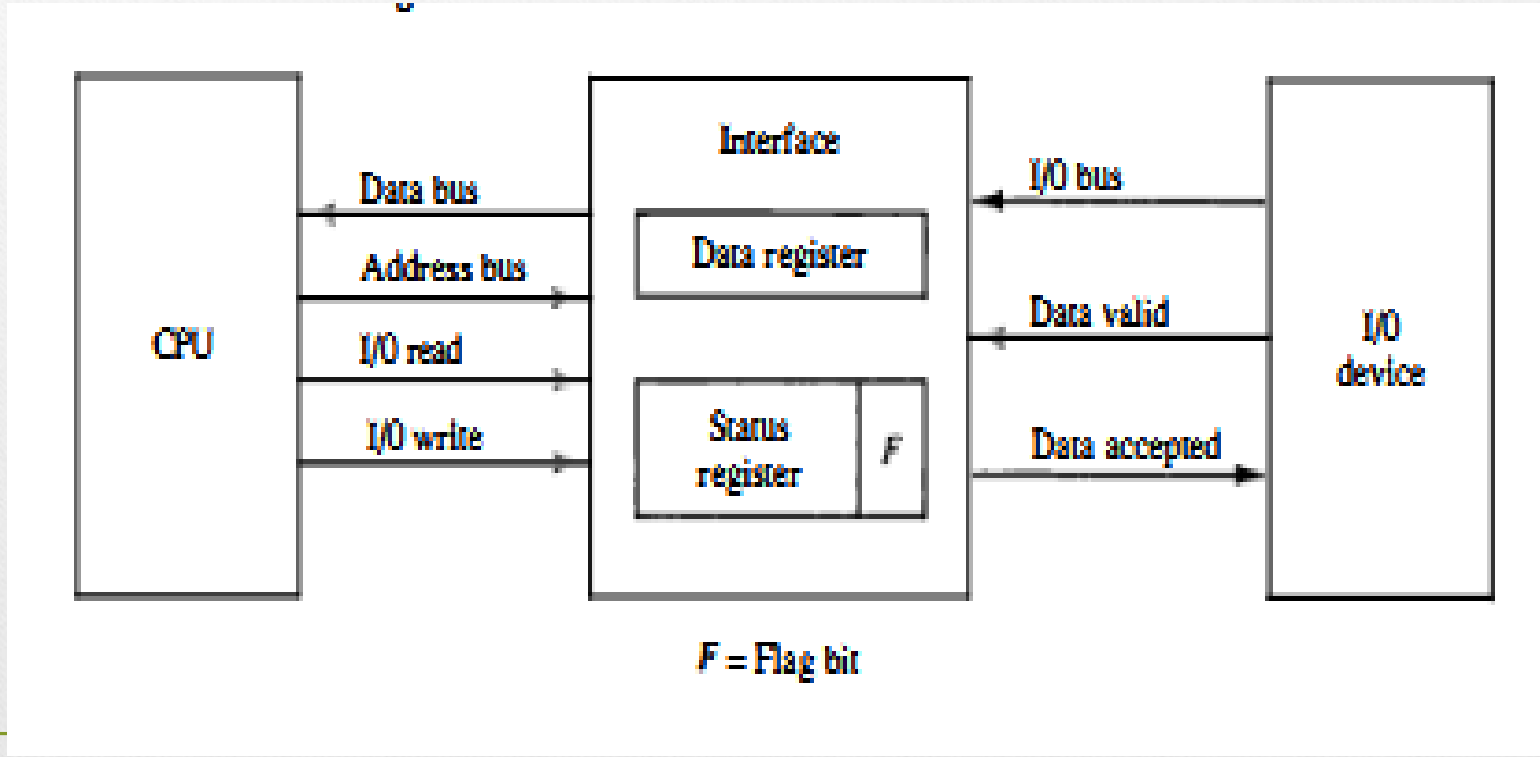
Timing Diagram



Sequence of Events



- Strobe pulse: Data transfer between CPU and Interface
- Handshaking signals:- Data transfer between IO interface and peripheral device.



Modes of Transfer

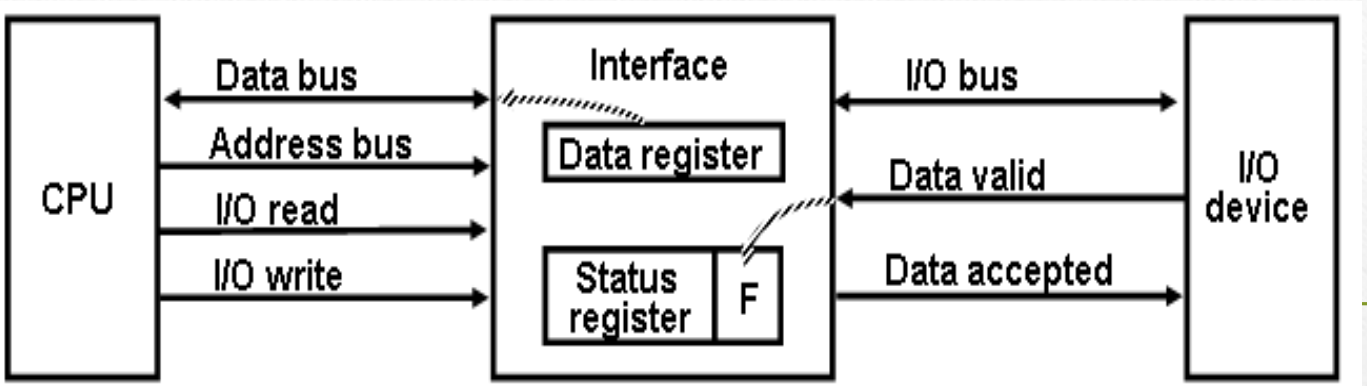
- 3 different Data Transfer Modes between the central computer (CPU or Memory) and peripherals;
  - Program-Controlled I/O or programmed I/O
  - Interrupt-Initiated I/O or Interrupt driven I/O
  - Direct Memory Access (DMA)
- Programmed I/O
  - Transferring data under program control requires constant monitoring of the peripheral by CPU.
  - I/O device does not have direct access to memory
  - Programmed I/O is time consuming process since it keeps the processor busy needlessly.

Programmed I/O

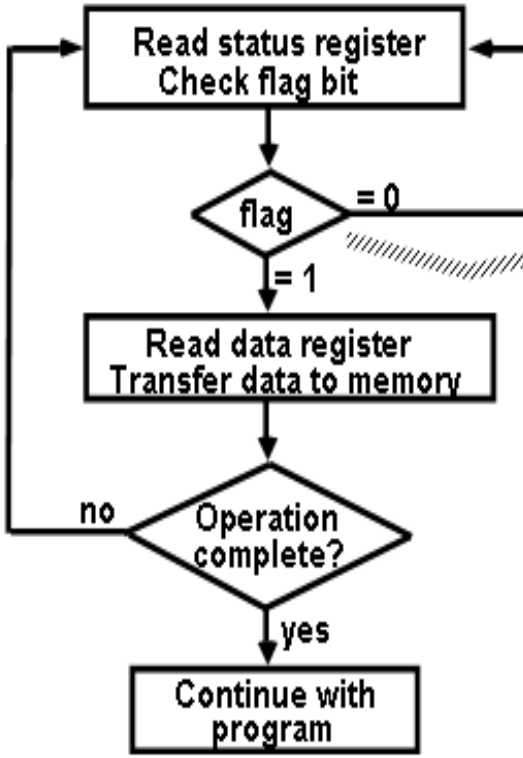
- More instructions may be required during the transfer of data from I/O device to memory
  - Input instruction: from I/P device to CPU
  - Store instruction: CPU->memory
  - To verify the data availability
  - To count the number of words transferred
- A common programming task is to transfer a block of words from an I/O device and store them in a memory buffer
- Useful in small low speed systems or in dedicated systems to monitor a device continuously.

Programmed I/O

Data Transfer from I/O device to CPU



Flow chart for CPU program to input data



Polling or Status Checking

- Continuous CPU involvement
- CPU slowed down to I/O speed
- Simple
- Least hardware

Interrupt-Initiated I/O

- Polling takes valuable CPU time
- Open communication only when some data has to be passed -> *Interrupt*.
- I/O interface, instead of the CPU, monitors the I/O device
- When the interface determines that the I/O device is ready for data transfer, it generates an *Interrupt Request* to the CPU
- Upon detecting an interrupt, CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing
- 2 methods in which the CPU chooses the branch address of the service routine
  - Non-vectored interrupt – branch address is assigned to a fixed location
  - Vectored interrupt – interrupt supplies the branch info (direct or indirect address of ISR) to the CPU

References

- M. M. Mano, Computer System Architecture, Prentice-Hall

Direct Memory Access (DMA)

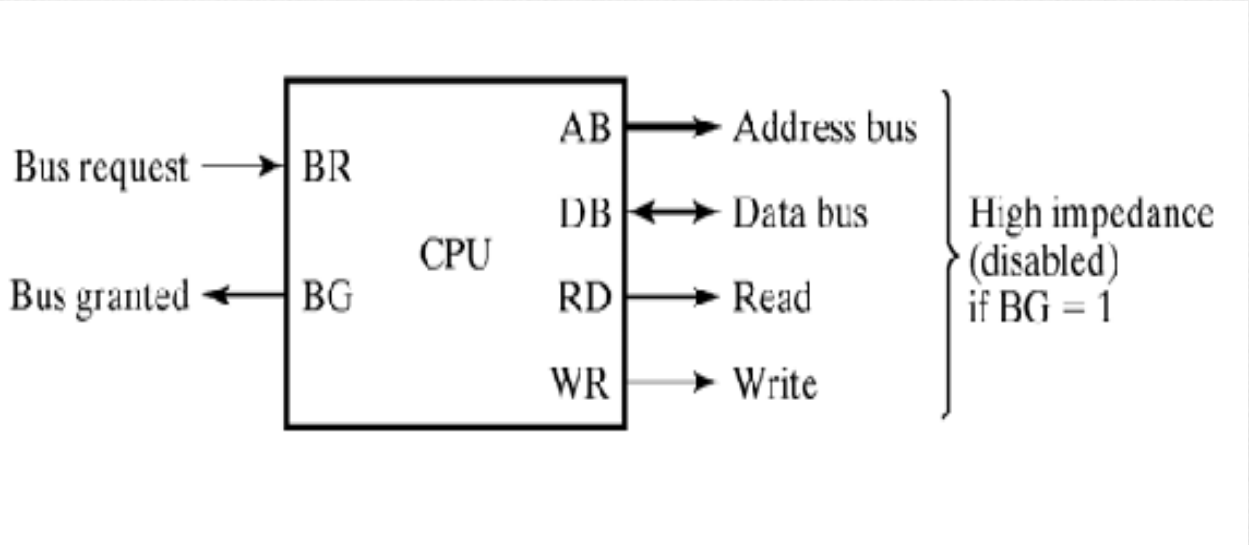
ANISHA M. LAL

DMA (DIRECT MEMORY ACCESS)

NEED

- Peripheral devices manage the memory bus directly and improve the speed of transfer between peripheral device and memory without the intervention of CPU.

CPU Bus signals for DMA



Two control signals used to facilitate the DMA transfer:

- BUS REQUEST
- BUS GRANT

BUS REQUEST:

-Signal from DMA controller to the CPU requesting to relinquish control of the buses.

-When this input is high CPU stops executing the instruction and places the address, data, read and write lines in high impedance.

BUS GRANT:

-Signal from CPU to DMA controller to the buses are in high impedance state.

-DMA takes the control of buses and memory transfer occur without the

intervention of CPU.

At the end of the transfer, bus request line is disabled followed by bus grant signal disable and CPU returns to its normal operation

DMA TRANSFER MODES

- BURST TRANSFER
- CYCLE STEALING

BURST TRANSFER:

- block consisting of number of words is transferred in a continuous burst while

DMA controller is master of memory buses.

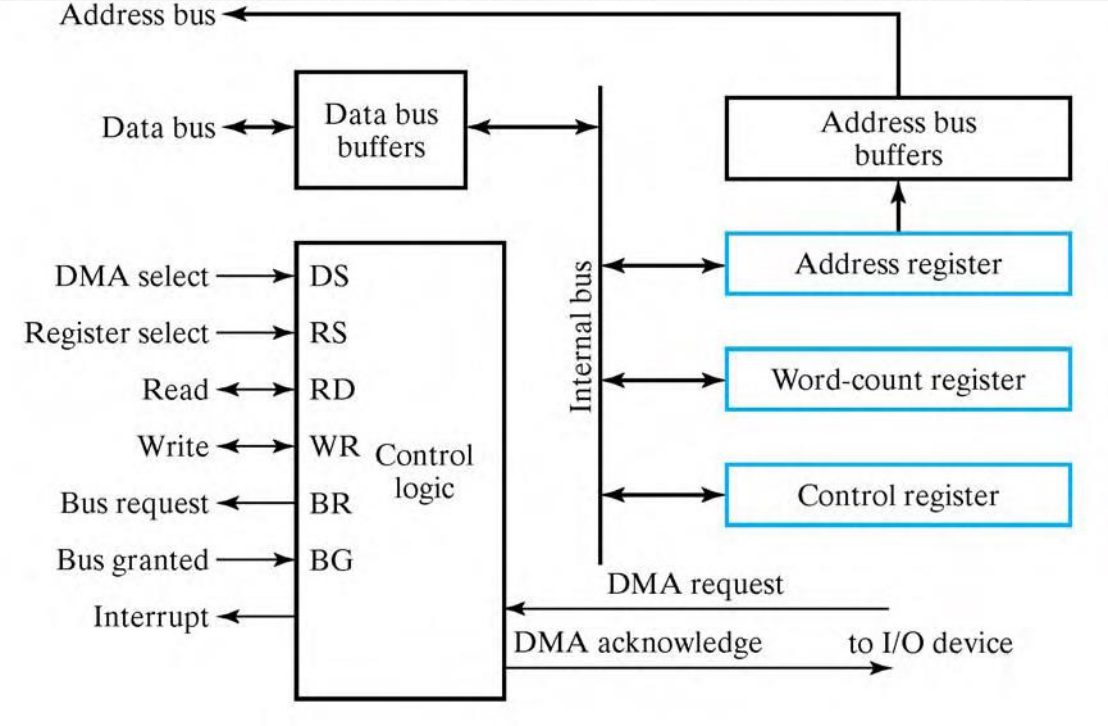
- used in high speed peripheral devices such as magnetic disk.

CYCLE STEALING:

- transfers one data word at a time after which it must return the control of the buses to CPU.

- CPU delays its operation for one memory cycle to allow the direct memory transfer to steal one memory cycle

DMA CONTROLLER



-DMA controller is similar to an interface and is used to communicate with the CPU

and IO device.

-Communication to the CPU is established by the data and control lines.

-Communication with IO device is established by DMA request and Acknowledge.

-when BG = 0 , CPU communicate with DMA register through data bus to read or write to DMA registers.

-When BG =1, CPU relinquished the buses and the DMA can communicate directly with memory by specifying address in the address bus and enabling read/write control signals.

-DMA controller consists of address, word count and control register.

Address Register: contains address to specify the desired location in memory and is incremented after each word is transferred to memory.

Word Count Register: holds number of words to be transferred and decremented after each transfer and tested for zero.

Control Register: specifies the mode of transfer. f

-DMA is initialized by the CPU by executing a program consisting of IO instructions that include the address for selecting particular DMA registers.

-CPU initializes the DMA by sending the following information.

1. Starting address of memory block where data are available for read and where data to be stored in write operation.

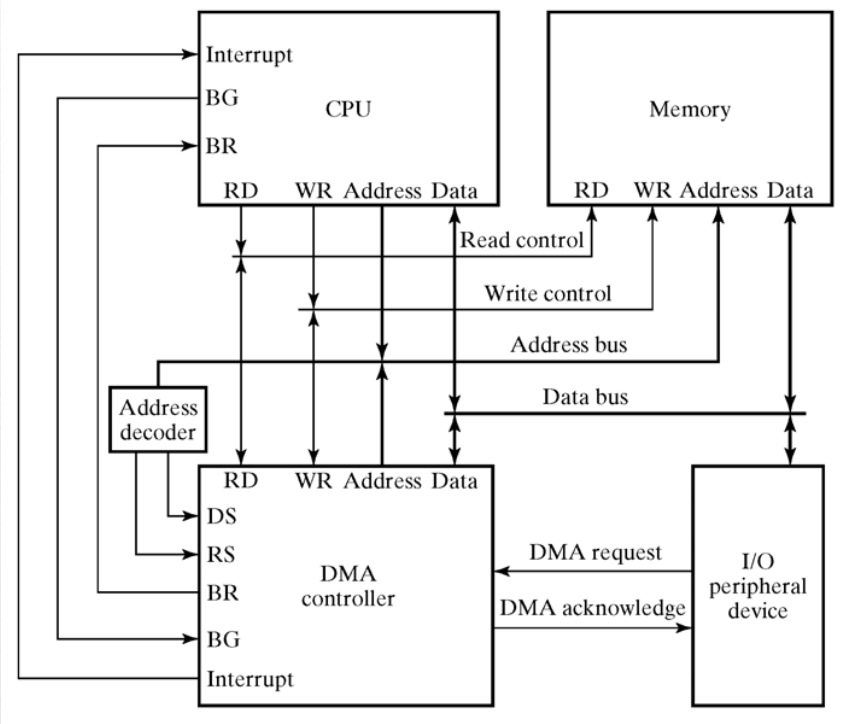
2. The word count – specifies the number of words in memory block

3. Control to specify the mode of transfer such read or write.

4. Control to start the DMA transfer.

- Once the DMA is initialized, CPU stops communicating with the DMA, and DMA now starts and continues to transfer data between memory and peripheral unit until an entire block is transferred.

DMA TRANSFER



-DMA has its own address, which activates the DS and RS lines.

-CPU initializes the DMA through the data bus.

-When peripheral device sends a DMA request, DMA controller activates BR, informing CPU to relinquish the buses. CPU responds by enabling BG.

-DMA transfers the value of address register in to address bus and enables DMA Acknowledge.

-Peripheral devices transfers the word in to data bus (write) or receives a word (read).

- After each word is transferred, DMA increments the address register and decrements its word count register.

-If the word count doesn't reach zero, DMA checks the request line from peripheral devices.

-If the peripheral speed is slow, the DMA request line may come somewhat later. In this case DMA disables the BR line to CPU, so that CPU continues to execute its program.

-Word count reaches zero, DMA stops transfer and removes BR to CPU.

- DMA controller may have more than one channel. Each channel has a request and acknowledge pair of control signals which are connected to separate peripheral devices and has its own address register and word count register.

-Priority is established among the channels ,such that high priority channels are serviced first.

Application:

- Fast transfer of information between magnetic disks and memory

- Updating the display in an interactive terminal.

Interrupt

ANISHA M. LAL

Interrupt

What is an interrupt?

An interrupt is a signal from a device attached to a computer or from a program within the computer that causes the CPU to stop its normal program execution and perform service related to the event.

Examples of interrupts :I/O completion, divide-by-0, etc.

Maskable Interrupt: It is a hardware interrupt that may be ignored by setting a bit in an interrupt mask register's (IMR) bit-mask.

Nonmaskable Interrupt:is a hardware interrupt that does not have a bit-mask associated with it - meaning that it can never be ignored. NMIs are often used for timers, especially watchdog timers

Interrupt Service Routine

A complete interrupt service cycle includes

1. Saving the program counter value in the stack
2. Saving the CPU status (including the CPU status register and some other registers) in the stack
3. Identifying the cause of interrupt
4. Resolving the starting address of the corresponding interrupt service routine
5. Executing the interrupt service routine
6. Restoring the CPU status and the program counter from the stack
7. Restarting the interrupted program (continue)

Interrupt Service Routine

An interrupt handler, also known as an interrupt service routine (ISR), is a callback subroutine in an operating system whose execution is triggered by the reception of an interrupt. Interrupt handlers have a multitude of functions, which vary based on the reason the interrupt was generated.

Interrupt Overhead

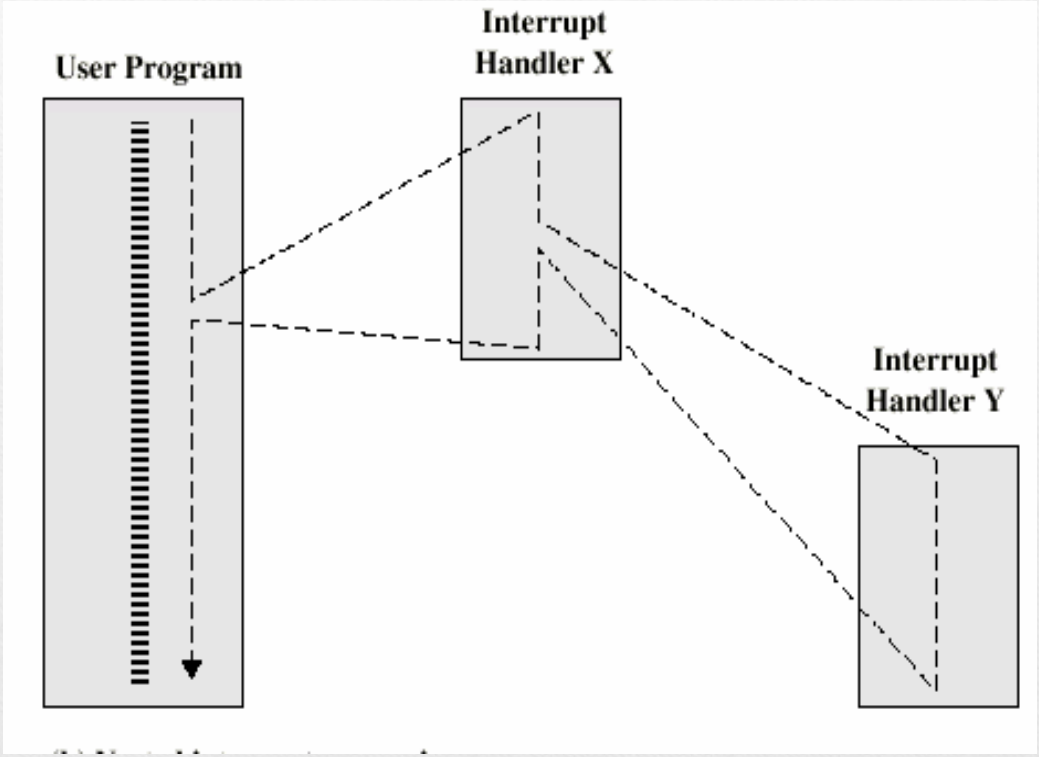
-The interrupt overhead is caused by context switching (storing and restoring the state of CPU)

-On interrupt handler entry, the context of the current process and its thread must be saved. On exit, it must be restored.

-On handler entry, memory locations different from the memory locations in the cache are used, and therefore cache updates are required.

Interrupt nesting:

-An interrupt can happen while executing an ISR. This is called in*terrupt* nesting.



PRIORITY INTERRUPT

ANISHA M. LAL

PRIORITY INTERRUPT

-In Interrupt Initiated I/O transfer, the first task of the interrupt system is to identify the source of the interrupt. In the case several sources requesting for service, the system should decide which device to serve first.

-Priority Interrupt is a system which establishes a priority over the devices to determine which condition is to be serviced.

-Higher priority is assigned to the devices which if delayed, could have serious consequences. (High speed devices i.e. Magnetic disk)

-Establishing the priority of simultaneous interrupts can be done by software or hardware.

-Software – polling procedure is used to assign the priority.

-When the interrupt comes from several sources, the control is changed to a common branch address.

-Program that takes care of interrupts begins at the branch address and polls the interrupting sources in sequence.

-The order in which they are tested determines the priority of each interrupt.

-The device which is tested first has the highest priority and if it has set a interrupt, then control branches to a service routine for this source.

-Disadvantage: If there are many interrupts the time required to poll them can exceed the time available to service the I/O device.

Hardware Priority Interrupt:

- A hardware unit which function as an overall manager, accepts interrupt requests from many sources, determines the highest priority source and issues an interrupt request to CPU.

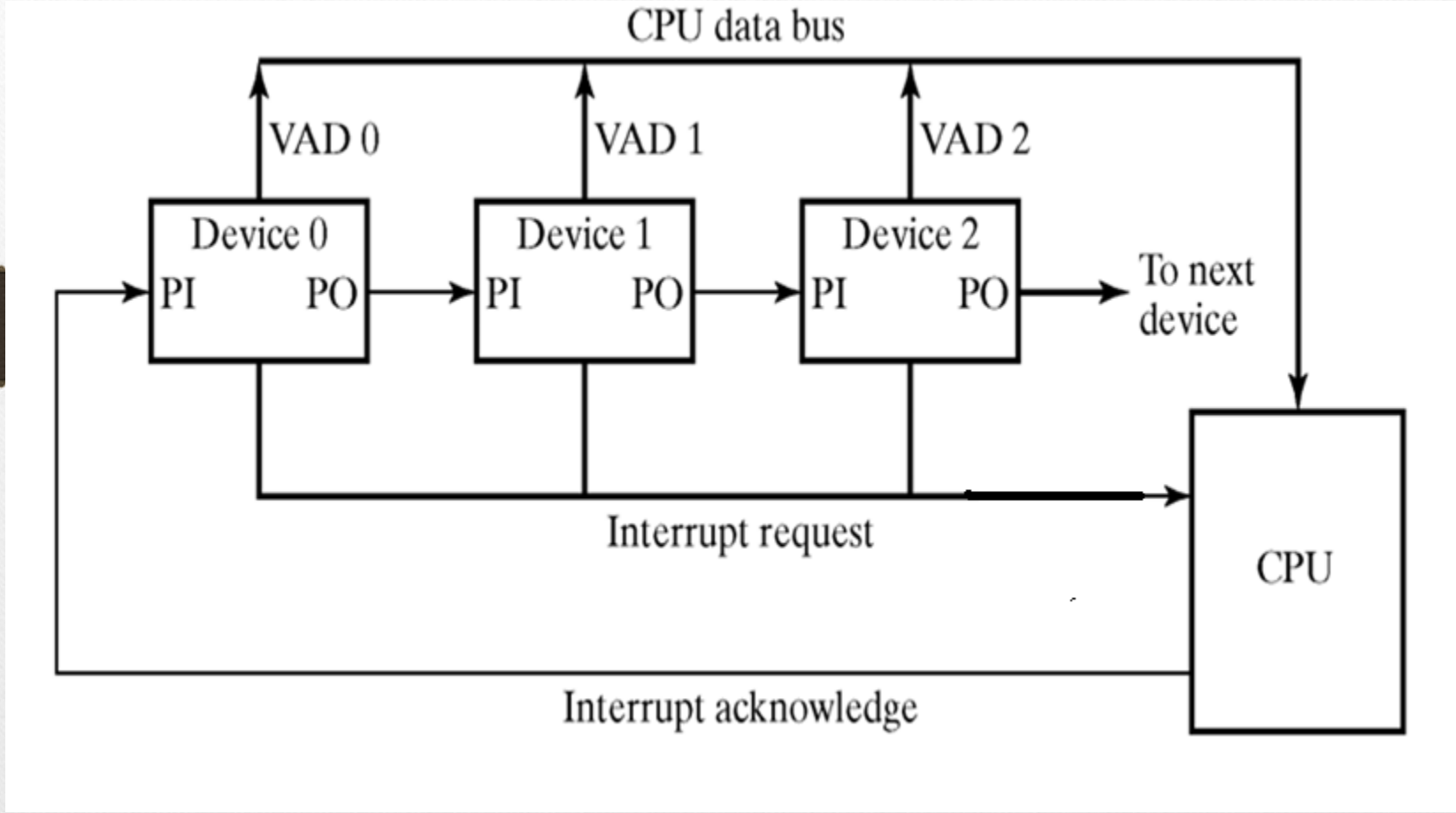
- Each interrupt source has its own interrupt vector to access its own service routine directly.

- The hardware priority interrupt function can be established in two ways:

1. Serial connection of interrupt Lines (Daisy- chaining method)

2. Parallel connection of interrupt Lines.

DAISY CHAIN PRIORITY INTERRUPT



- Serial connection of all devices that request an interrupt.

-Highest priority device is placed in the first position followed by the lower priority devices.

-Interrupt request line is common to all devices.

-If any device places interrupt signal, the interrupt request line is driven to low level state and enables an interrupt input to CPU.

-CPU responds by giving the interrupt acknowledge signal and the signal is received by device 1 at its priority in (PI).

-If device 1 has given the interrupt, it blocks the acknowledge signal by placing a 0 in PO output and inserts its own vector address (VAD) in to the data bus.

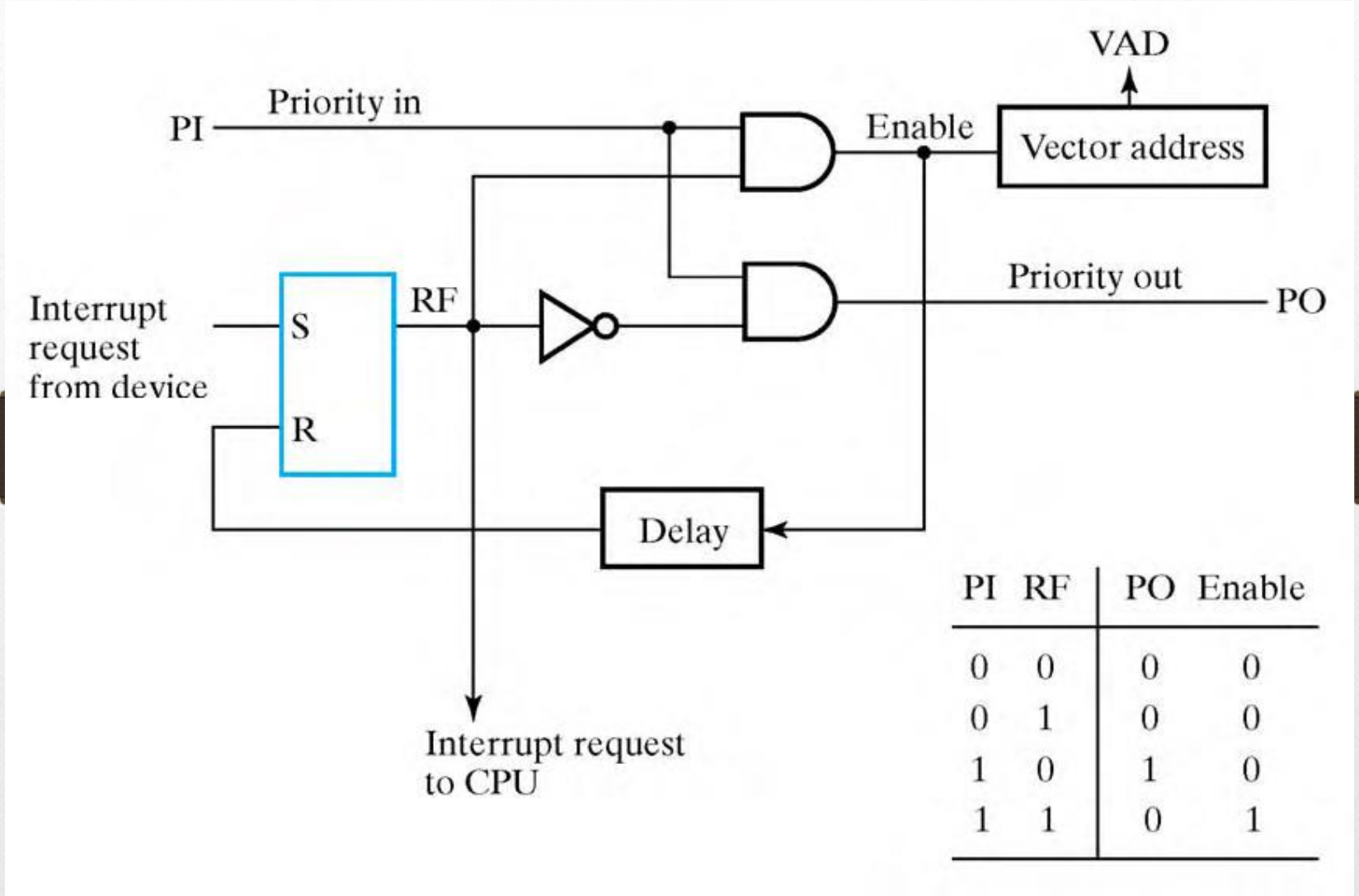
-If device 1 has not given the interrupt, it passes the acknowledge to the next device by placing 1 in PO.

-Three condition: 1. PI =0 (No acknowledge signal) PO= 0

2. PI =1 (device1 gives interrupt) PO =0

3. PI = 1 (device1 doesn't gives interrupt) PO =1

One stage of Daisy- chain priority arrangement



PI	RF	PO	Enable
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

PARALLEL PRIORITY INTERRUPT

-Interrupt Register: whose bits are set separately by the interrupt signal from each device and cleared by program instruction.

-Mask register: purpose is to control the status of each interrupt request. Its bit are set or reset by program instructions.

-The mask register can be programmed to disable lower priority interrupts while a higher priority device is being serviced and also provide a facility that allows a high priority device to interrupt the CPU while lower priority device is being serviced.

-Priority Logic:

-High speed device i.e. magnetic disk is given highest priority followed by printer, character reader and keyboard.

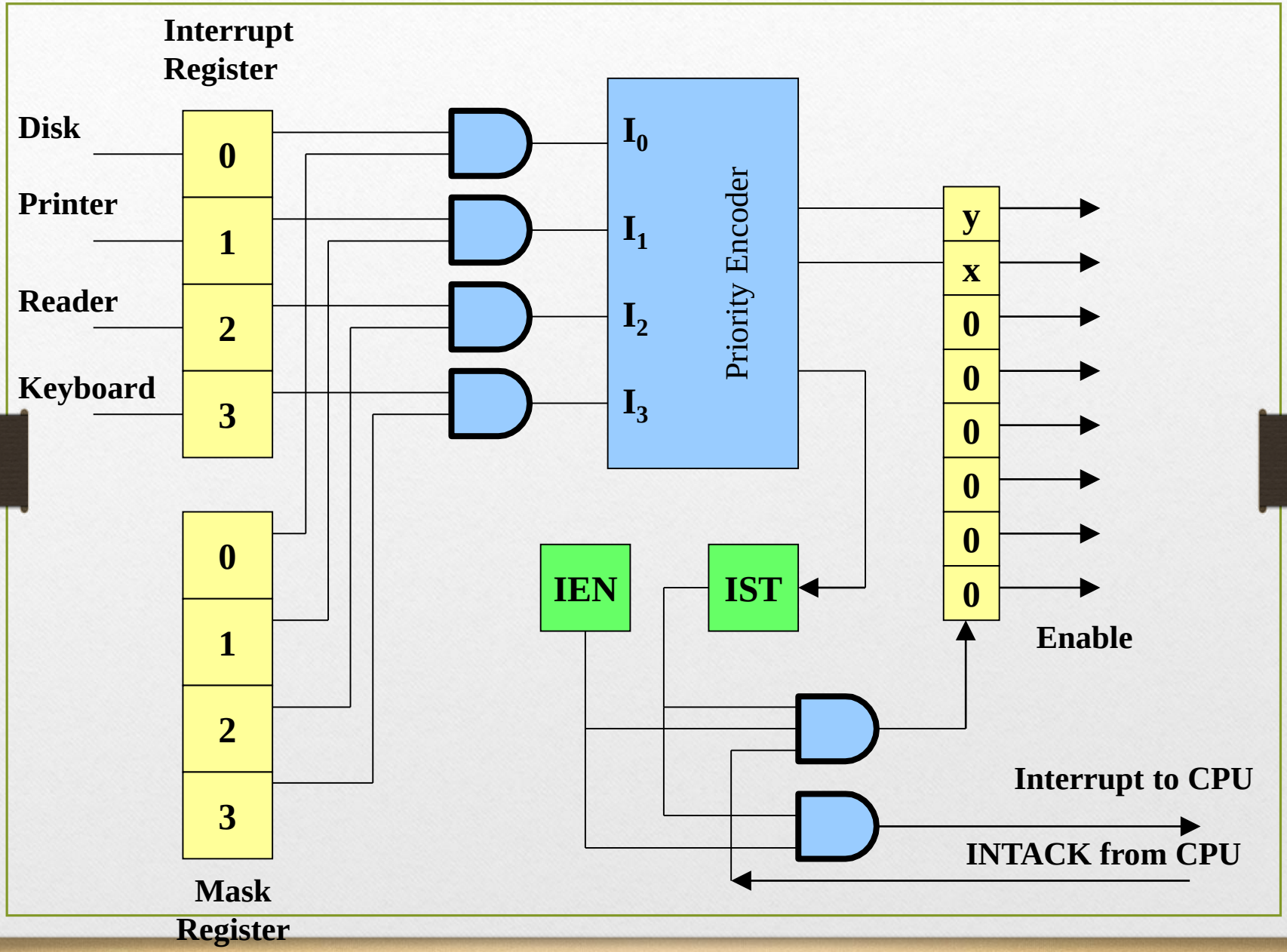
-The interrupt bit and its corresponding mask bit are applied to an AND gate to produce inputs to the priority encoder.

-Interrupt is recognized only if its corresponding mask bit is set to 1 by the program.

-Priority encoder generates two bits of the vector address which is transferred to the CPU.

-IST – Encoder sets this when one or more inputs are equal to 1.

-IEN – to provide overall control over the interrupt system.


Interrupt Register
Disk
Printer
Reader
Keyboard
0
1
2
3
0
1
2
3
Mask Register
I_0
I_1
I_2
I_3
Priority Encoder
y
x
0
0
0
0
0
0
Enable
IEN
IST
Interrupt to CPU
INTACK from CPU

INTACK: Signal from CPU which enables the bus buffers in the output register and a vector address VAD is placed into the data bus.

Priority Encoder:

-It is a circuit that implements the priority function.

-Logic of Priority encoder is such that if two or more inputs arrive at the same time, the input having higher priority will take precedence.

-Truth table:

$X = I0'I1'$ $Y = I0'I1+I0'I2'$ $IST = I0 + I1 + I2 + I3$

Inputs	outputs
(I0 I1 I2 I3)	x y IST
1 x x x	0 0 1
0 1 x x	0 1 1
0 0 1 x	1 0 1
0 0 0 1	1 1 1
0 0 0 0	x x 0

- The Interrupt enable flip-flop (IEN) can be set or cleared by program instructions.
- A programmer can therefore allow interrupts (clear IEN) or disallow interrupts (set IEN)
- At the end of each instruction cycle the CPU checks IEN and IST. If either is equal to zero, control continues with the next instruction. If both = 1, the interrupt is handled.
- Interrupt micro-operations:
  - SP←SP – 1 (Decrement stack pointer)
  - M[SP] ← PC Push PC onto stack
  - INTACK ← 1 Enable interrupt acknowledge
  - PC ←VAD Transfer vector address to PC
  - IEN ←0 Disable further interrupts
  - Go to fetch next instruction