Peripheral Devices

Input Devices

- Keyboard
- Optical input devices
  - Card Reader
  - Paper Tape Reader
  - Bar code reader
  - Digitizer
  - Optical Mark Reader
- Magnetic Input Devices
  - Magnetic Stripe Reader
- Screen Input Devices
  - Touch Screen
  - Light Pen
  - Mouse

Output Devices

- Card Puncher, Paper Tape Puncher
- CRT
- Printer (Impact, Ink Jet, Laser, Dot Matrix)
- Plotter

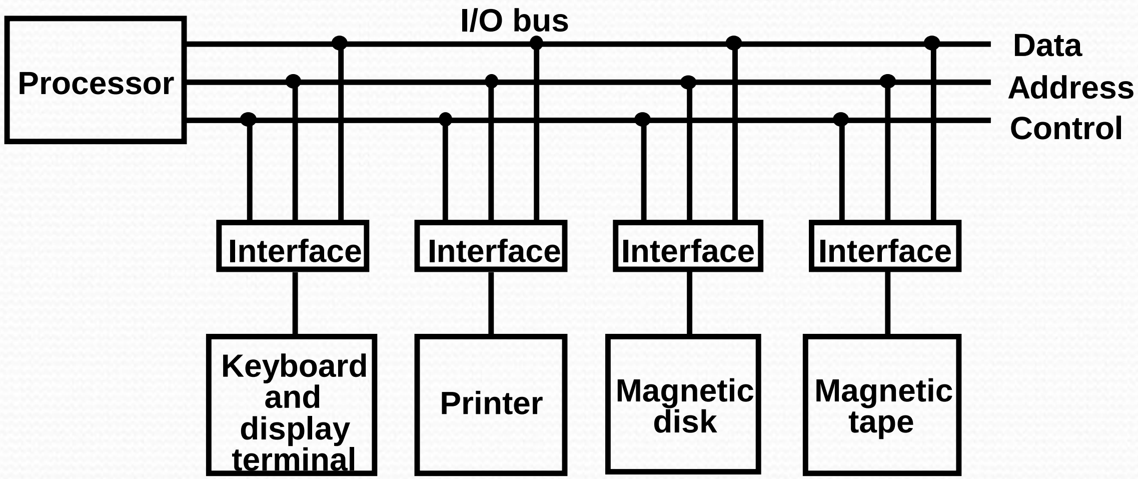
I/O Interface

- **Provides a method for transferring information between internal storage (such as memory and CPU registers) and external I/O devices**
- **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, Block, …**
    - **CPU or Memory – Word**

Computer system include special H/W components i.e. ***interface units,*** between CPU and peripherals to supervise and synchronize all I/O transfers, to resolve these differences.

  - **Data representations may differ:** Operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to CPU.

I/O Bus and Interface



Each peripheral has an interface module associated with it

Interface

- **Decodes the device address (device code)**
- **Decodes the commands (operation)**
- **Provides signals for the peripheral controller**
- **Synchronizes the data flow and supervises the transfer rate between peripheral and CPU or Memory**

Typical I/O instruction

Op. code	Device address	Function code

(Command)

I/O Bus and Memory Bus

Functions of Buses

- ***MEMORY BUS*** **is for information transfers between CPU and the MM**
- ***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
- **Some computer systems use two separate buses, one to communicate with memory and the other with I/O interfaces**

  - Communication between CPU and all interface units is via a common I/O Bus

  - An interface connected to a peripheral device may have a number of *data registers* , a *control register*, and a *status register*

  - A command is passed to the peripheral by sending to the appropriate interface register

  - Function code and sense lines are not needed (Transfer of data, control, and status information is always via the common I/O Bus)

The interpretation of the command depends on the peripheral that the processor is addressing. There are four types of commands that an interface may receive. They are classified as control, status, data output, and data input.

A *control command* is issued to activate the peripheral and to inform it what to do. For example, a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the tape, or to start the tape moving in the forward direction.

A *status command* is used to test various status conditions in the interface and the peripheral. For example, the computer may wish to check the status of the peripheral before a transfer is initiated.

A *data output command* causes the interface to respond by transferring data from the bus into one of its registers. Consider an example with a tape unit. The computer starts the tape moving by issuing a control command. The processor then monitors the status of the tape by means of a status command. When the tape is in the correct position, the processor issues a data output command. The interface responds to the address and command and transfers the information from the data lines in the bus to its buffer register. The interface then communicates with the tape controller and sends the data to be stored on tape.

The *data input command* is the opposite of the data output. In this case the interface receives an item of data from the peripheral and places it in its buffer register. The processor checks if data are available by means of a status command and then issues a data input command. The interface places the data on the data lines, where they are accepted by the processor.

Isolated vs. Memory Mapped I/O

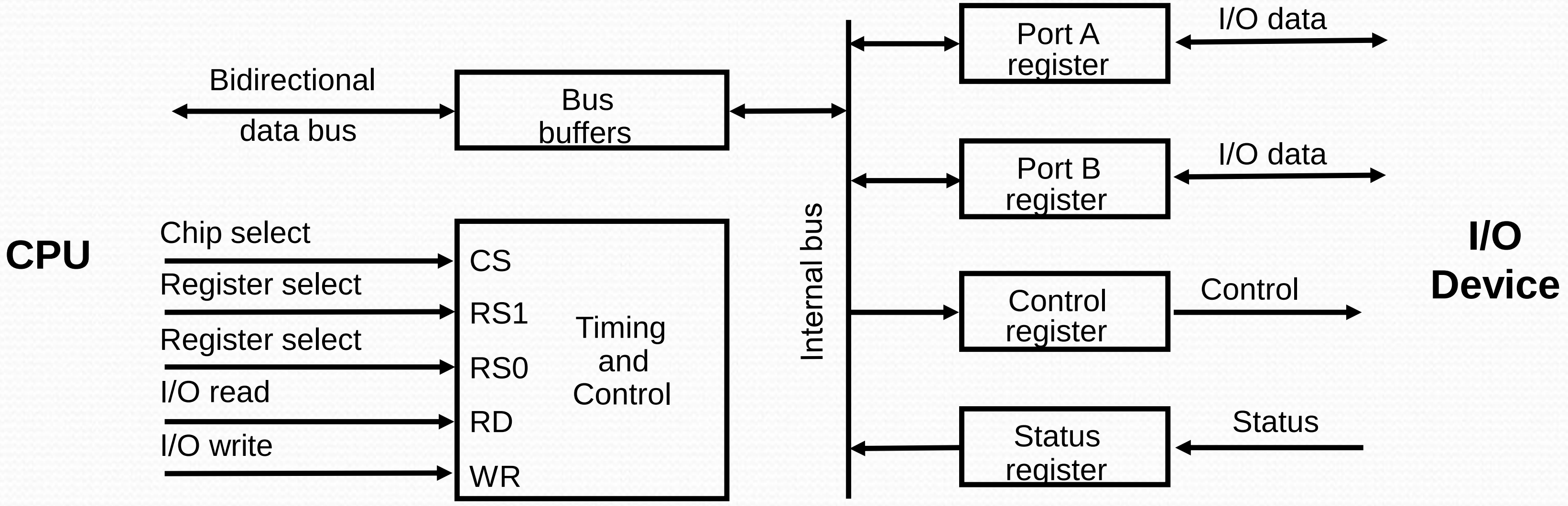
Isolated I/O

- 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

Memory-mapped I/O

- A single set of read/write control lines
  (no distinction between memory and I/O transfer)
- 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
- Considerable flexibility in handling I/O operations

I/O Interface



CS	RS1	RS0	Register selected
0	x	x	None - data bus in high-impedence
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

Programmable Interface

- Information in each port can be assigned a meaning depending on the mode of operation of the I/O device
→ Port A = Data; Port B = Command; Port C = Status

- CPU initializes(loads) each port by transferring a byte to the Control Register
→ Allows CPU can define the mode of operation of each port
→ *Programmable Port*: By changing the bits in the control register, it is possible to change the interface characteristics

Asynchronous Data Transfer

Synchronous and Asynchronous Operations

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*

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

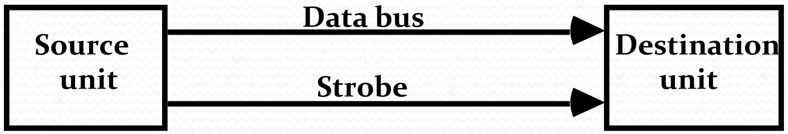
- The receiving unit responds with another control signal to acknowledge receipt of the data

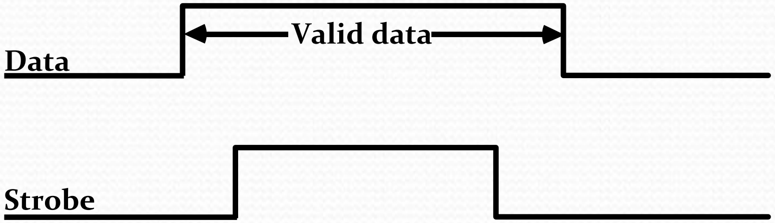
Strobe Control

* Employs a single control line to time each transfer
* The strobe may be activated by either the source or the destination unit

Source-Initiated Strobe for Data Transfer

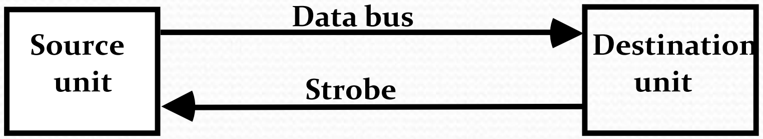
Block Diagram



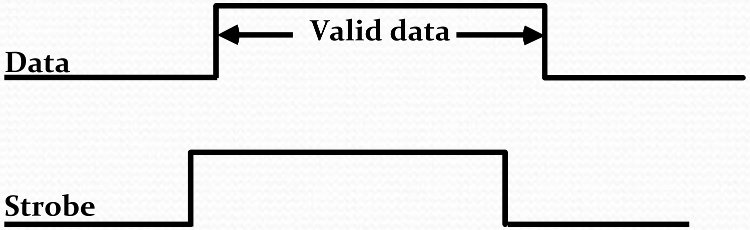


Destination-Initiated Strobe for Data Transfer

Block Diagram



Timing Diagram



Handshaking

Strobe Methods

Source-Initiated

The source unit that initiates the transfer has no way of knowing whether the destination unit has actually received data

Destination-Initiated

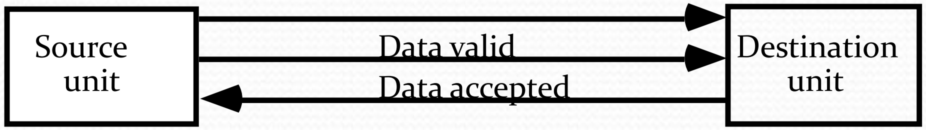
The destination unit that initiates the transfer no way of knowing whether the source has actually placed the data on the bus

To solve this problem, the *HANDSHAKE* method introduces a second control signal to provide a *Reply* to the unit that initiates the transfer

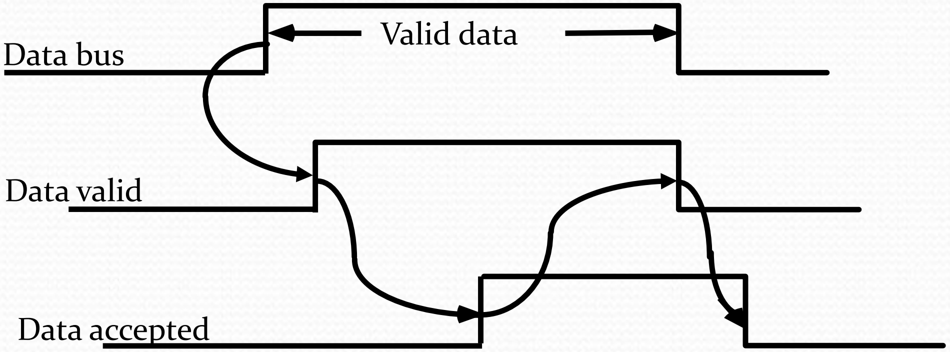
Source Initiated Transfer using Handshaking

* **Allows arbitrary delays from one state to the next**
* **Permits each unit to respond at its own data transfer rate**
* **The rate of transfer is determined by the slower unit**

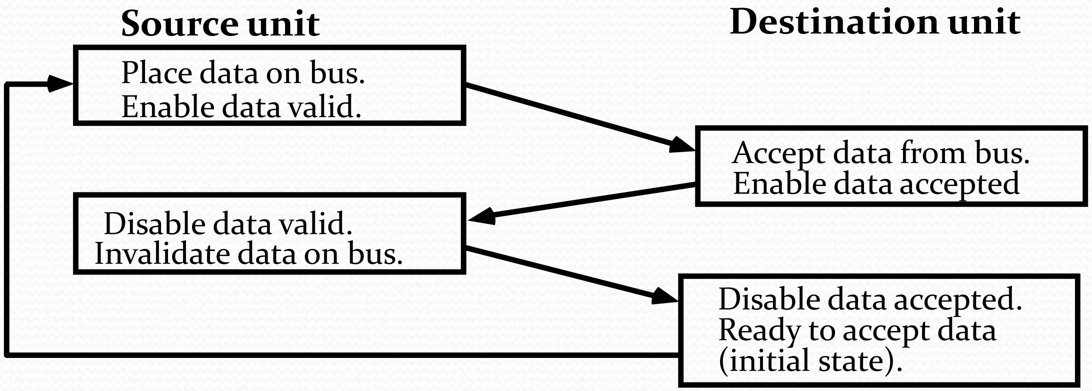
Block Diagram



Timing Diagram



Sequence of Events


Source unit

Destination unit

Place data on bus.
Enable data valid.

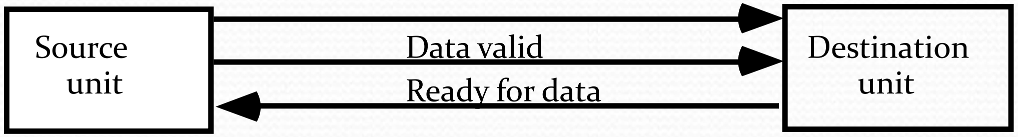
Accept data from bus.
Enable data accepted

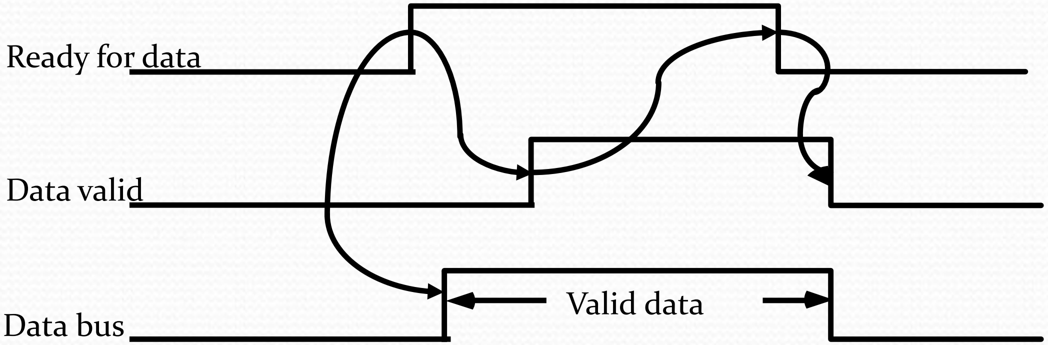
Disable data valid.
Invalidate data on bus.

Disable data accepted.
Ready to accept data
(initial state).

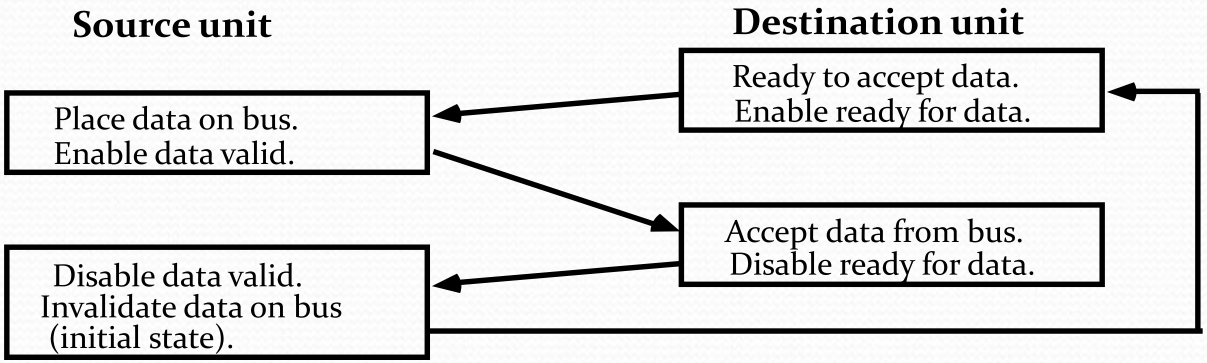
Destination Initiated Transfer using Handshaking

Block Diagram





Sequence of Events



Source unit

Destination unit

Ready to accept data.
Enable ready for data.

Place data on bus.
Enable data valid.

Accept data from bus.
Disable ready for data.

Disable data valid.
Invalidate data on bus
(initial state).

*** Handshaking 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, data transfer will not be completed -> Can be detected by means of a *timeout* mechanism**

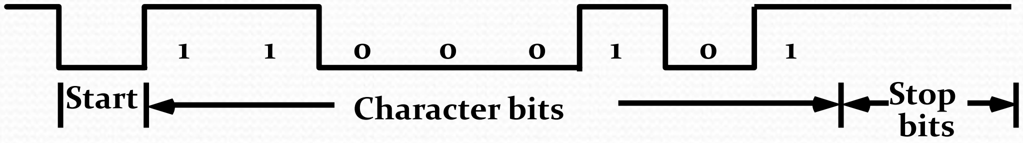
Asynchronous Serial Transfer

Four Different Types of Transfer

Asynchronous serial transfer
Synchronous serial transfer
Asynchronous parallel transfer
Synchronous parallel transfer

Asynchronous Serial Transfer

- Employs special bits which are inserted at both ends of the character code
- Each character consists of three parts; Start bit; Data bits; Stop bits.



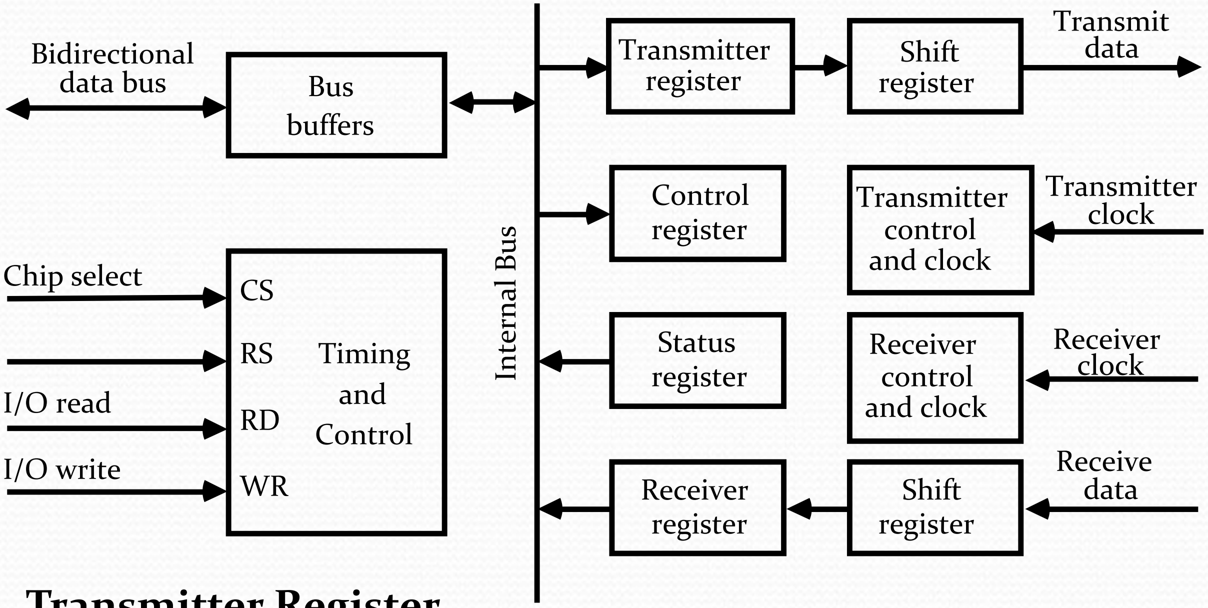
A character can be detected by the receiver from the knowledge of 4 rules;

- When data are not being sent, the line is kept in the 1-state (idle state)
- The initiation of a character transmission is detected by a *Start Bit* , which is always a 0
- The character bits always follow the *Start Bit*
- After the last character , a *Stop Bit* is detected when the line returns to the 1-state for at least 1 bit time

The receiver knows in advance the transfer rate of the bits and the number of information bits to expect

Universal Asynchronous Receiver Transmitter

A typical asynchronous communication interface available as an IC



CS	RS	Oper.	Register selected
0	x	x	None
1	0	WR	Transmitter register
1	1	WR	Control register
1	0	RD	Receiver register
1	1	RD	Status register

Transmitter Register

- Accepts a data byte(from CPU) through the data bus
- Transferred to a shift register for serial transmission

Receiver

- Receives serial information into another shift register
- Complete data byte is sent to the receiver register

Status Register Bits

- Used for I/O flags and for recording errors

Control Register Bits

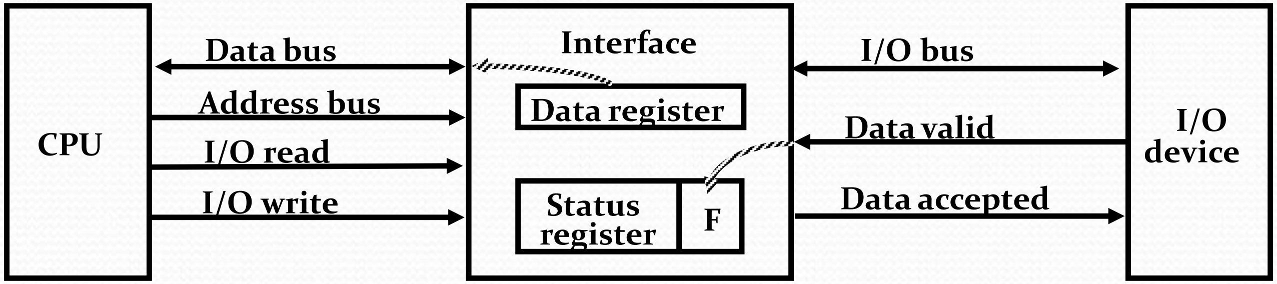
- Define baud rate, no. of bits in each character, whether to generate and check parity, and no. of stop bits

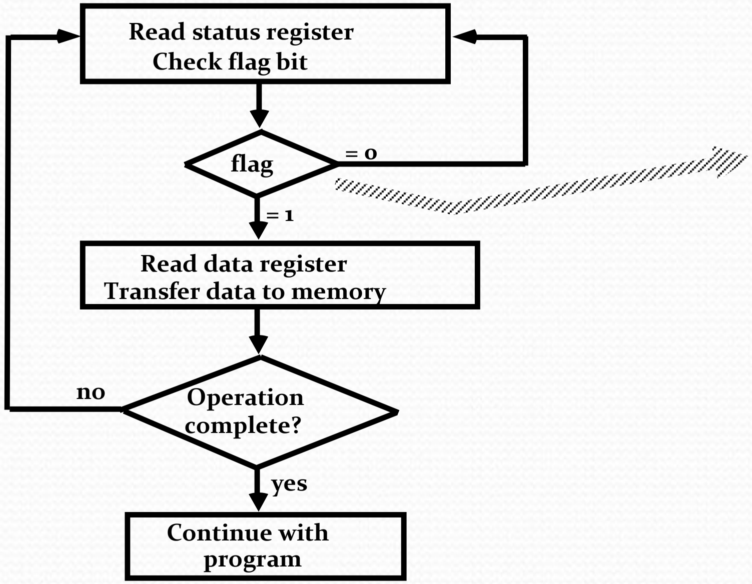
Modes of Transfer – Programmed I/O

3 different Data Transfer Modes between the central computer(CPU or Memory) and peripherals;

- **Program-Controlled I/O**
- **Interrupt-Initiated I/O**
- **Direct Memory Access (DMA)**

Program-Controlled I/O(Input Dev to CPU)





Polling or Status Checking

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

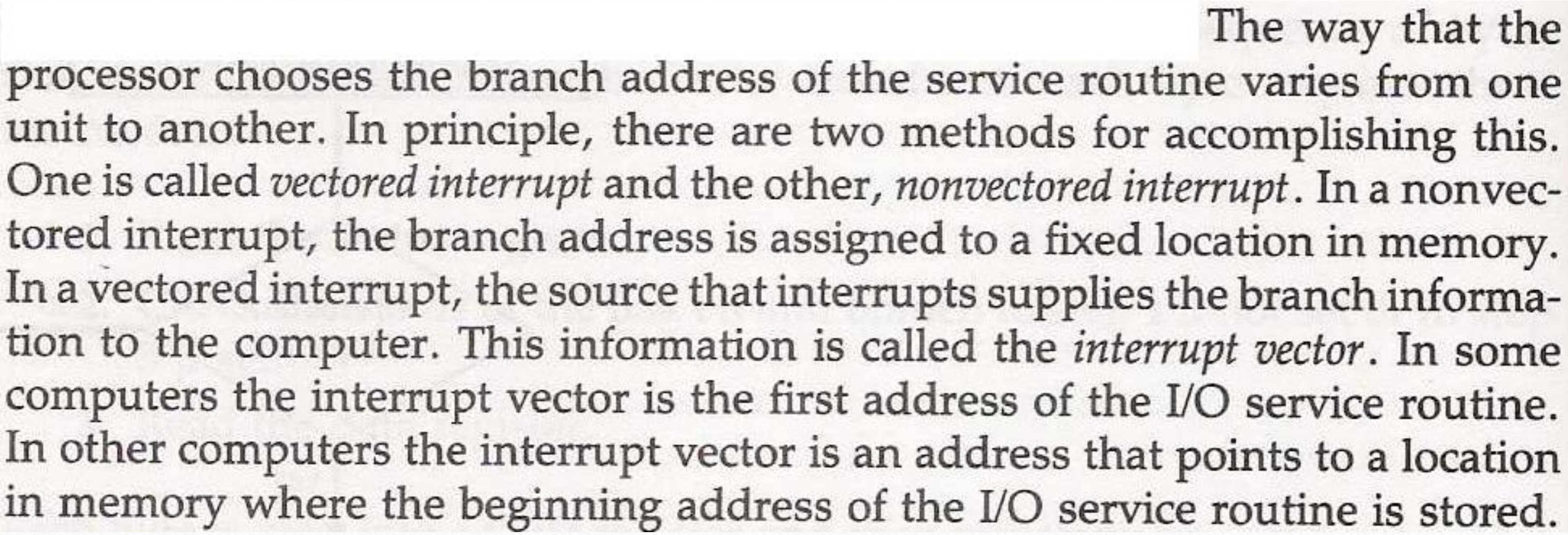
Modes of Transfer – Interrupted I/O & DMA

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

DMA (Direct Memory Access)

- Large blocks of data transferred at a high speed to or from high speed devices, magnetic drums, disks, tapes, etc.
- DMA controller Interface that provides I/O transfer of data directly to and from the memory and the I/O device
- CPU initializes the DMA controller by sending a memory address and the number of words to be transferred
- Actual transfer of data is done directly between the device and memory through DMA controller
-> Freeing CPU for other tasks



The way that the processor chooses the branch address of the service routine varies from one unit to another. In principle, there are two methods for accomplishing this. One is called *vectored interrupt* and the other, *nonvectored interrupt*. In a nonvectored interrupt, the branch address is assigned to a fixed location in memory. In a vectored interrupt, the source that interrupts supplies the branch information to the computer. This information is called the *interrupt vector*. In some computers the interrupt vector is the first address of the I/O service routine. In other computers the interrupt vector is an address that points to a location in memory where the beginning address of the I/O service routine is stored.

Priority Interrupts

Priority

- Determines which interrupt is to be served first when two or more requests are made simultaneously
- Also determines which interrupts are permitted to interrupt the computer while another is being serviced
- Higher priority interrupts can make requests while servicing a lower priority interrupt

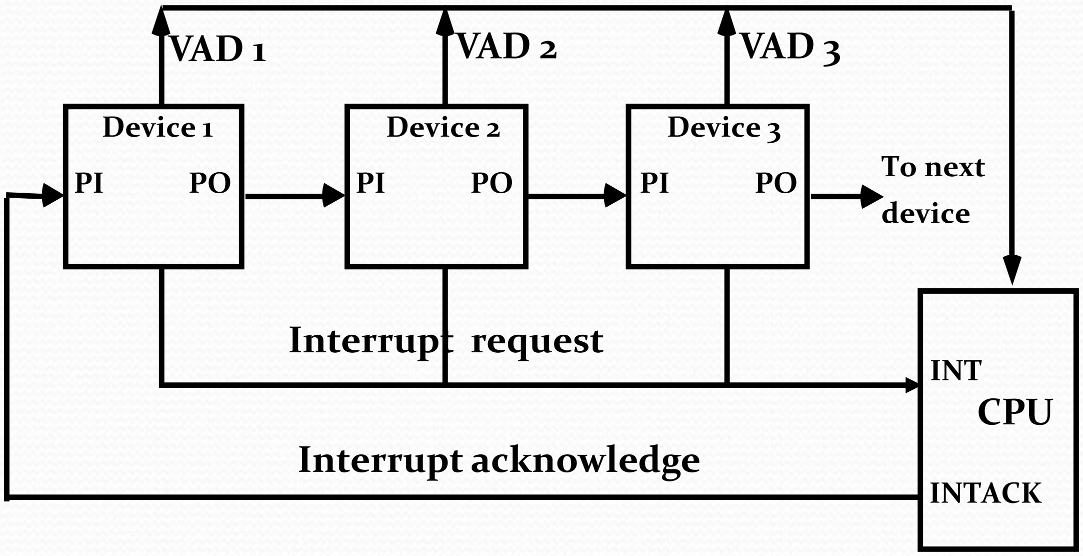
Priority Interrupt by Software(Polling)

- Priority is established by the order of polling the devices(interrupt sources)
- Flexible since it is established by software
- Low cost since it needs a very little hardware
- Very slow

Priority Interrupt by Hardware

- Require a priority interrupt manager which accepts all the interrupt requests to determine the highest priority request
- Fast since identification of the highest priority interrupt request is identified by the hardware
- Fast since each interrupt source has its own interrupt vector to access directly to its own service routine

Hardware Priority Interrupts – Daisy Chain



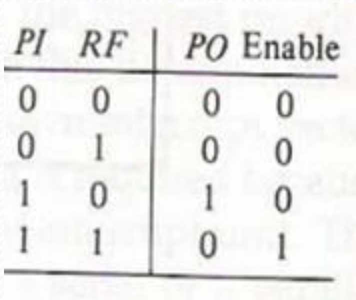
* Serial hardware priority function

* Interrupt Request Line

- Single common line

* Interrupt Acknowledge Line

- Daisy-Chain



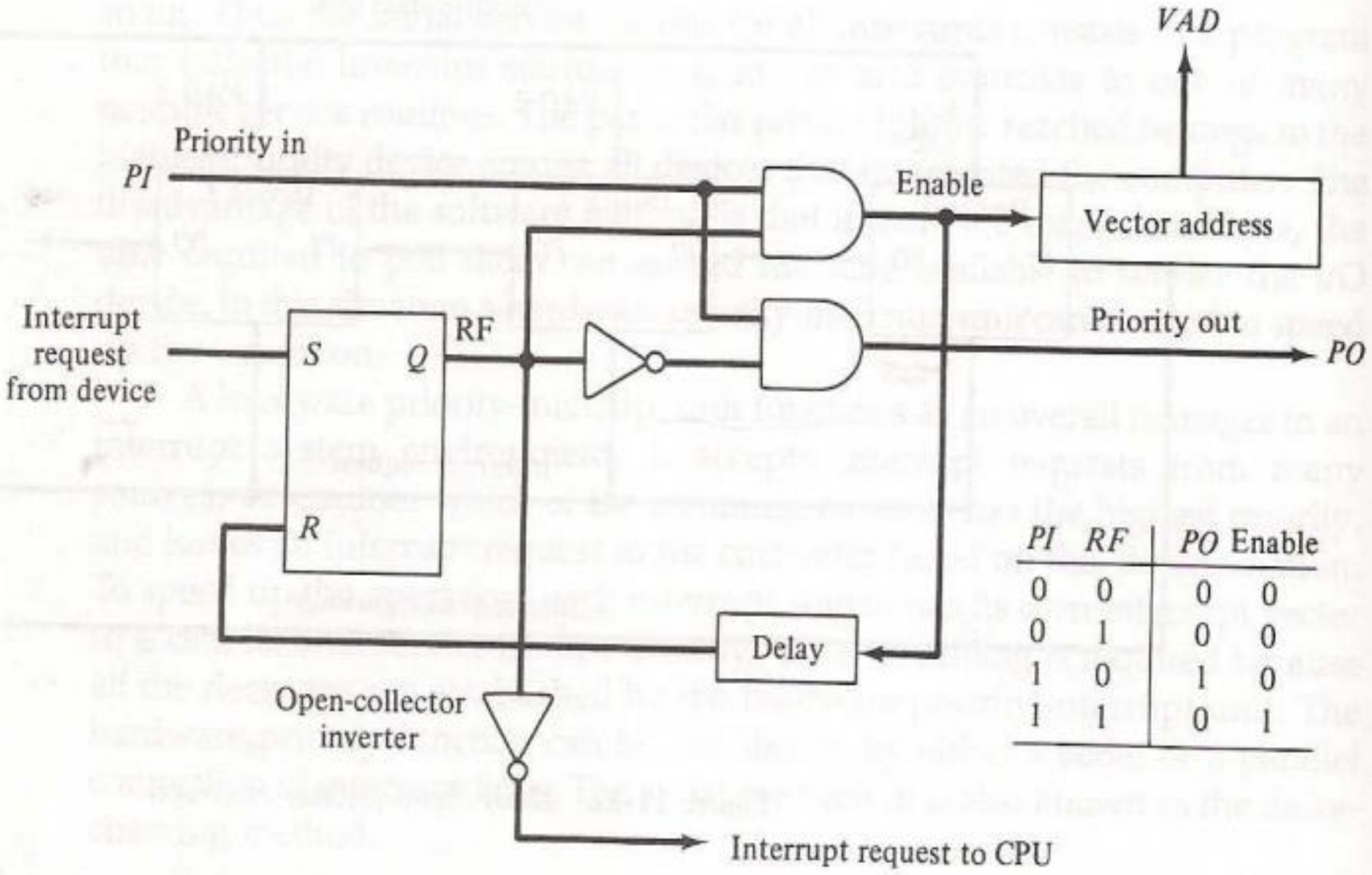
PI	*RF*	*PO*	Enable
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

Interrupt Request from any device(>=1)

-> CPU responds by INTACK <- 1

-> Any device receives signal(INTACK) 1 at PI puts the VAD on the bus

Among interrupt requesting devices the only device which is physically closest to CPU gets INTACK=1, and it blocks INTACK to propagate to the next device

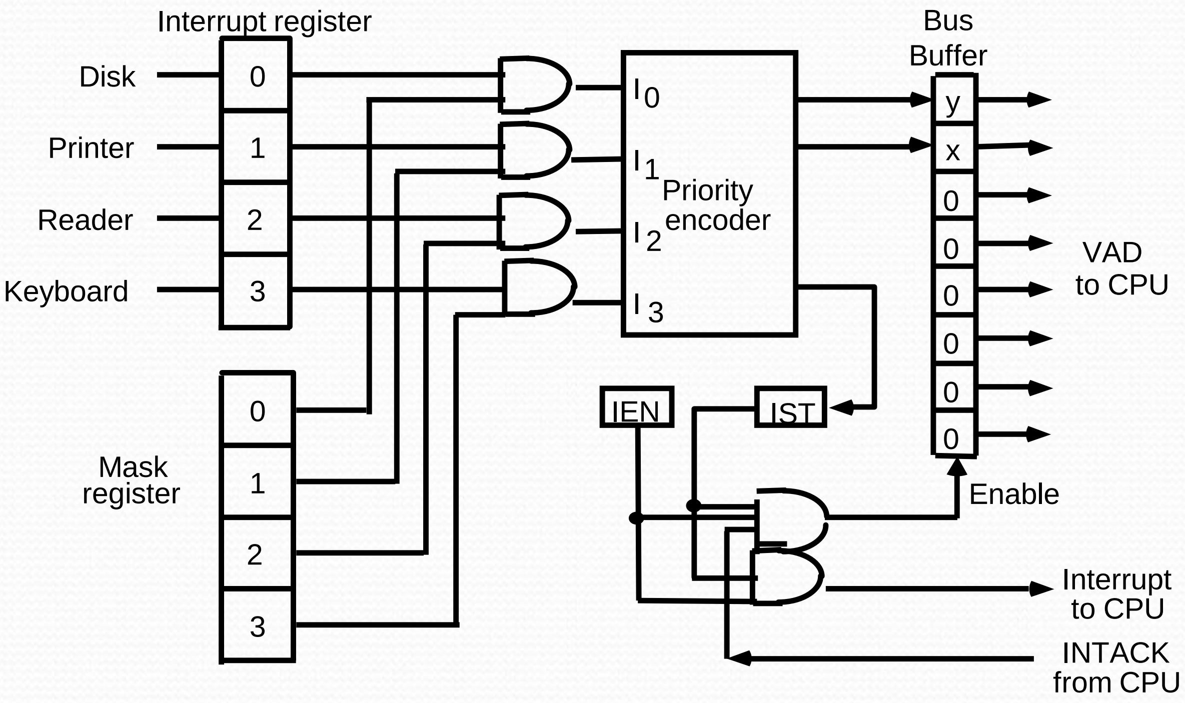


PI	RF	PO	Enable
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

Figure 11-13 One stage of the daisy-chain priority arrangement.

It shows the internal logic that must be included within each device when connected in daisy-chaining scheme.

Parallel Priority Interrupts



IEN: Set or Clear by instructions ION or IOF
IST: Represents an unmasked interrupt has occurred. INTACK enables tristate Bus Buffer to load VAD generated by the Priority Logic

Interrupt Register:
- Each bit is associated with an Interrupt Request from different Interrupt Source - different priority level
- Each bit can be cleared by a program instruction

Mask Register:
- Mask Register is associated with Interrupt Register. It is used to control status of each interrupt request.
- Each bit can be set or cleared by an Instruction

Priority Encoder

Determines the highest priority interrupt when more than one interrupts take place

Priority Encoder Truth table

Inputs				Outputs			
I_0	I_1	I_2	I_3	x	y	IST	Boolean functions
1	d	d	d	0	0	1	
0	1	d	d	0	1	1	
0	0	1	d	1	0	1	$x = I_0' \ I_1'$
0	0	0	1	1	1	1	$y = I_0' I_1 + I_0' I_2'$
0	0	0	0	d	d	0	$(IST) = I_0 + I_1 + I_2 + I_3$

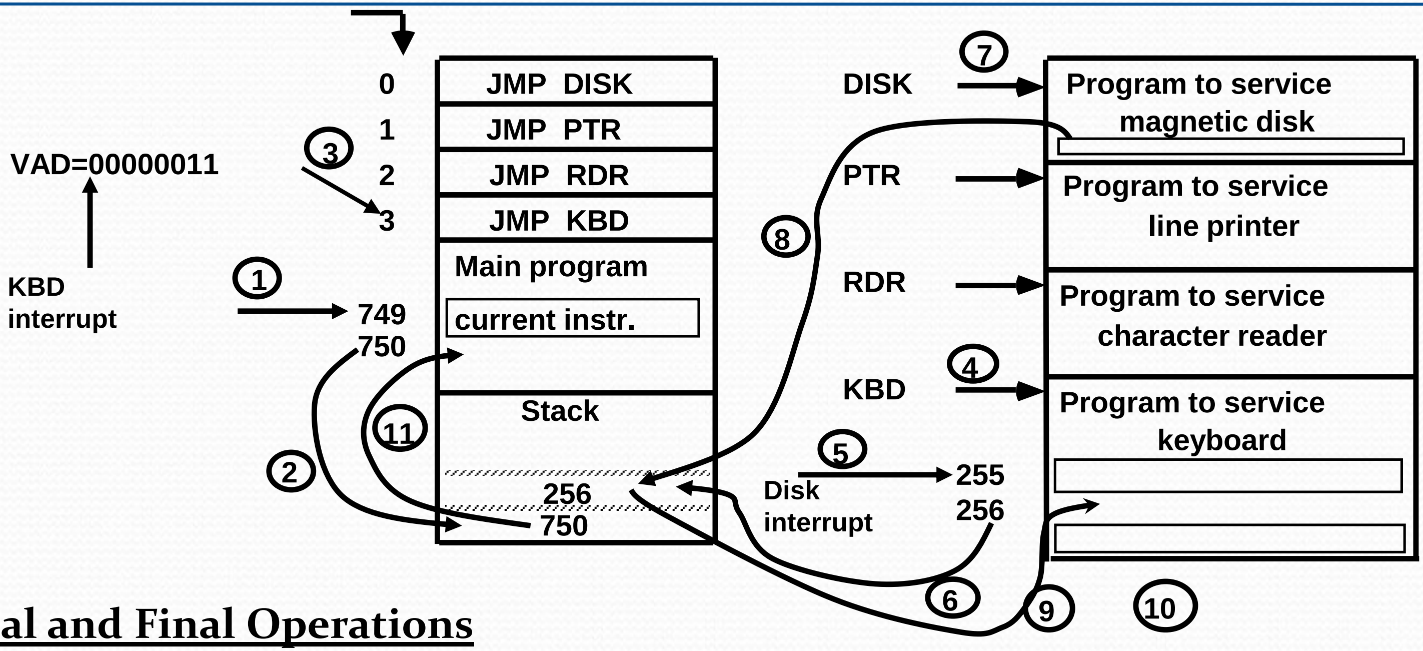
Interrupt Cycle

At the end of each Instruction cycle

- CPU checks IEN and IST
- If IEN • IST = 1, CPU -> Interrupt Cycle

SP ←SP - 1	Decrement stack pointer
M[SP] ← PC	Push PC into stack
INTACK ← 1	Enable interrupt acknowledge
PC ← VAD	Transfer vector address to PC
IEN ← 0	Disable further interrupts
Go To Fetch	to execute the first instruction in the interrupt service routine

Initial and Final Operations



Initial and Final Operations

Each interrupt service routine must have an initial and final set of operations for controlling the registers in the hardware interrupt system

Initial Sequence
[1] Clear lower level Mask reg. bits
[2] IST <- 0
[3] Save contents of CPU registers
[4] IEN <- 1
[5] Go to Interrupt Service Routine

Final Sequence
[1] IEN <- 0
[2] Restore CPU registers
[3] Clear the bit in the Interrupt Reg
[4] Set lower level Mask reg. bits
[5] Restore return address, IEN <- 1

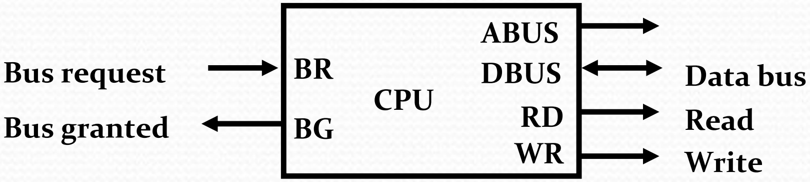
Direct Memory Access

*** Block of data transfer from high speed devices, Drum, Disk, Tape**

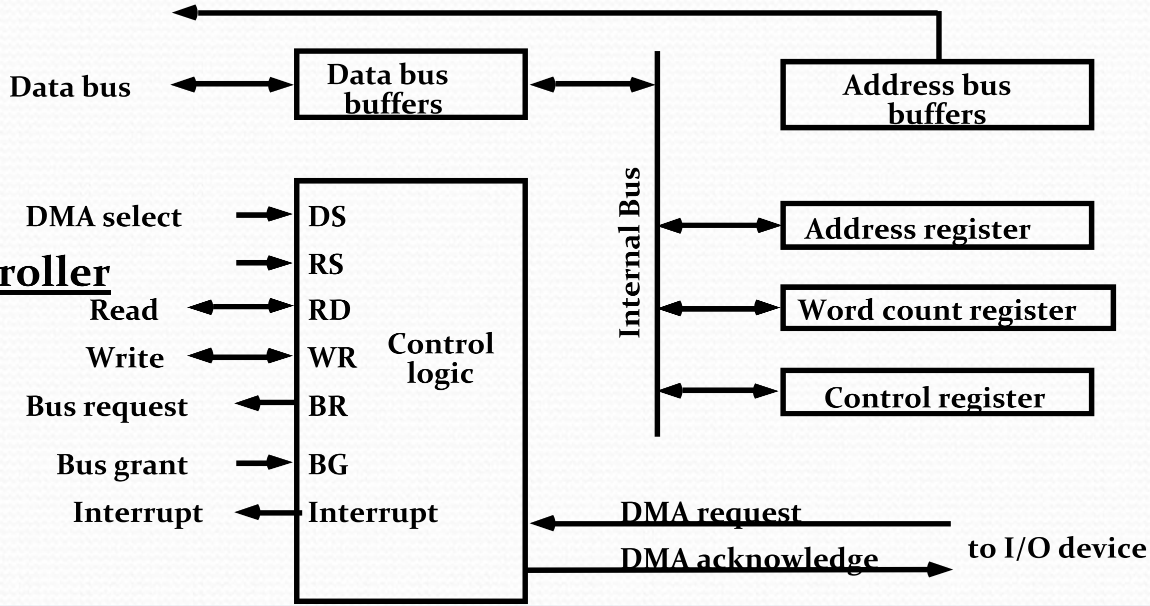
*** DMA controller - Interface which allows I/O transfer directly between Memory and Device, freeing CPU for other tasks**

*** CPU initializes DMA Controller by sending memory address and the block size (number of words)**

CPU bus signals for DMA transfer



Block diagram of DMA controller



DMA I/O Operation

Starting an I/O
- **CPU executes instruction to**
  - **Load Memory Address Register**
  - **Load Word Counter**
  - **Load Function(Read or Write) to be performed**
  - **Issue a GO command**

Upon receiving a GO Command DMA performs I/O operation as follows independently from CPU

Input

[1] Input Device <- R (Read control signal)
[2] Buffer(DMA Controller) <- Input Byte; and
assembles the byte into a word until word is full
[4] M <- memory address, W(Write control signal)
[5] Address Reg <- Address Reg +1; WC(Word Counter) <- WC - 1
[6] If WC = 0, then Interrupt to acknowledge done, else go to [1]

Output

[1] M <- M Address, R
M Address R <- M Address R + 1, WC <- WC - 1
[2] Disassemble the word
[3] Buffer <- One byte; Output Device <- W, for all disassembled bytes
[4] If WC = 0, then Interrupt to acknowledge done, else go to [1]

Cycle Stealing

While DMA I/O takes place, CPU is also executing instructions

DMA Controller and CPU both access Memory -> Memory Access Conflict

Memory Bus Controller

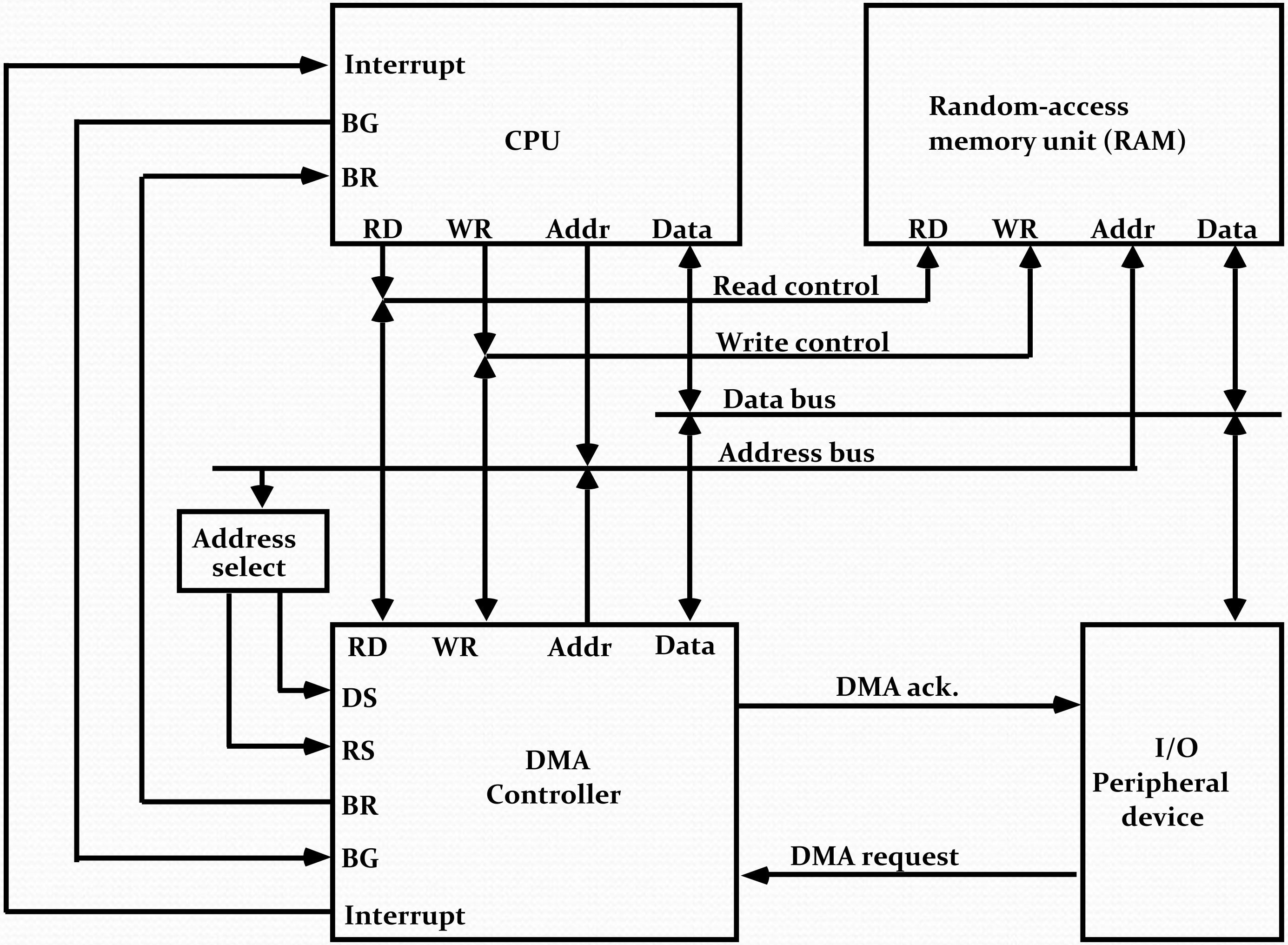
- Coordinating the activities of all devices requesting memory access
- Priority System

Memory accesses by CPU and DMA Controller are interwoven,
with the top priority given to DMA Controller
-> Cycle Stealing

Cycle Steal

- CPU is usually much faster than I/O(DMA), thus
CPU uses the most of the memory cycles
- DMA Controller steals the memory cycles from CPU
- For those stolen cycles, CPU remains idle
- For those slow CPU, DMA Controller may steal most of the memory
cycles which may cause CPU remain idle long time

DMA Transfer



I/O Processor - Channel

Channel

- Processor with direct memory access capability that communicates with I/O devices

- Channel accesses memory by cycle stealing

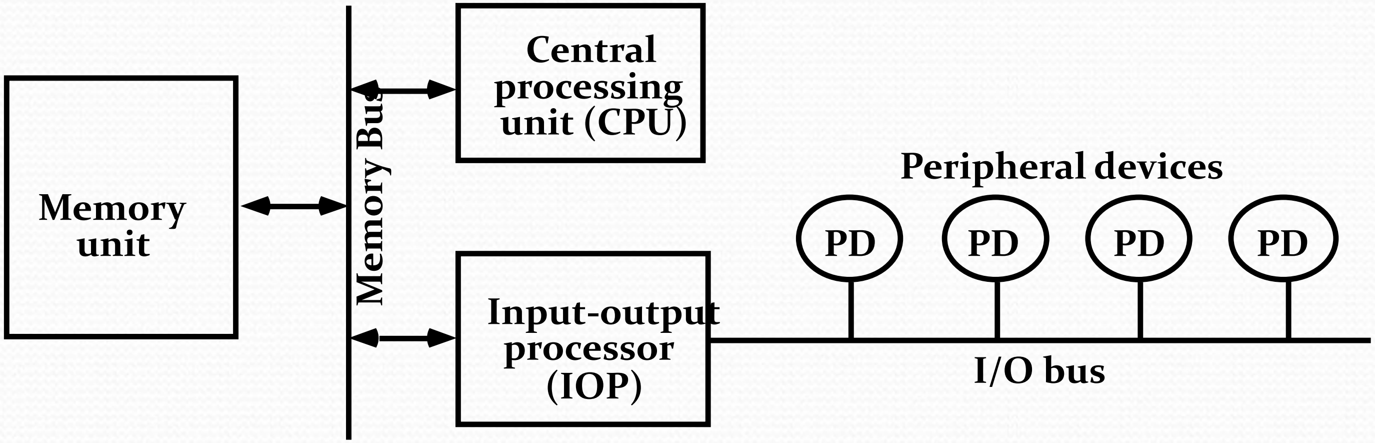
- Channel can execute a Channel Program

- Stored in the main memory

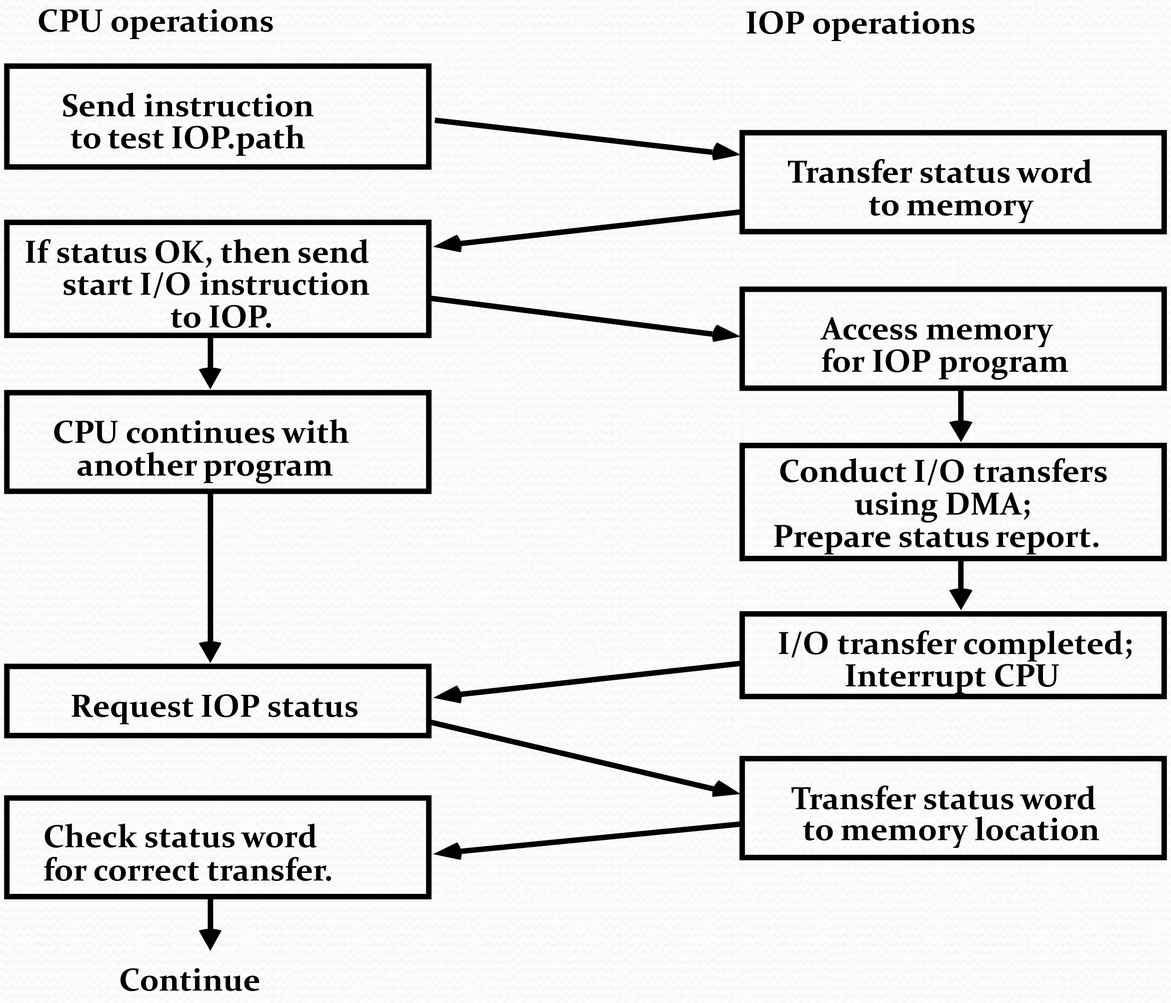
- Consists of Channel Command Word(CCW)

- Each CCW specifies the parameters needed by the channel to control the I/O devices and perform data transfer operations

- CPU initiates the channel by executing an channel I/O class instruction and once initiated, channel operates independently of the CPU



Channel CPU Communication


CPU operations
IOP operations
Send instruction to test IOP.path
Transfer status word to memory
If status OK, then send start I/O instruction to IOP.
Access memory for IOP program
CPU continues with another program
Conduct I/O transfers using DMA; Prepare status report.
I/O transfer completed; Interrupt CPU
Request IOP status
Transfer status word to memory location
Check status word for correct transfer.
Continue