UNIT-V

INPUT-OUTPUT ORGANIZATION

Peripheral Devices:

The Input / output organization of computer depends upon the size of computer and the peripherals connected to it. The I/O Subsystem of the computer, provides an efficient mode of communication between the central system and the outside environment

The most common input output devices are:

i) Monitor

ii) Keyboard

iii) Mouse

iv) Printer

v) Magnetic tapes

The devices that are under the direct control of the computer are said to be connected online.

Input - Output Interface

Input Output Interface provides a method for transferring information between internal storage and external I/O devices.

Peripherals connected to a computer need special communication links for interfacing them with the central processing unit.

The purpose of communication link is to resolve the differences that exist between the central computer and each peripheral.

The Major Differences are:-

1. Peripherals are electromechnical and electromagnetic devices and CPU and memory are electronic devices. Therefore, a conversion of signal values may be needed.
2. The data transfer rate of peripherals is usually slower than the transfer rate of CPU and consequently, a synchronization mechanism may be needed.
3. Data codes and formats in the peripherals differ from the word format in the CPU and memory.

4. The operating modes of peripherals are different from each other and must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To Resolve these differences, computer systems include special hardware components between the CPU and Peripherals to supervises and synchronizes all input and out transfers

- These components are called Interface Units because they interface between the processor bus and the peripheral devices.

I/O BUS and Interface Module

It defines the typical link between the processor and several peripherals.

The I/O Bus consists of data lines, address lines and control lines.

The I/O bus from the processor is attached to all peripherals interface.

To communicate with a particular device, the processor places a device address on address lines.

Each Interface decodes the address and control received from the I/O bus, interprets them for peripherals and provides signals for the peripheral controller.

It is also synchronizes the data flow and supervises the transfer between peripheral and processor.

Each peripheral has its own controller.

For example, the printer controller controls the paper motion, the print timing

The control lines are referred as I/O command. The commands are as following:

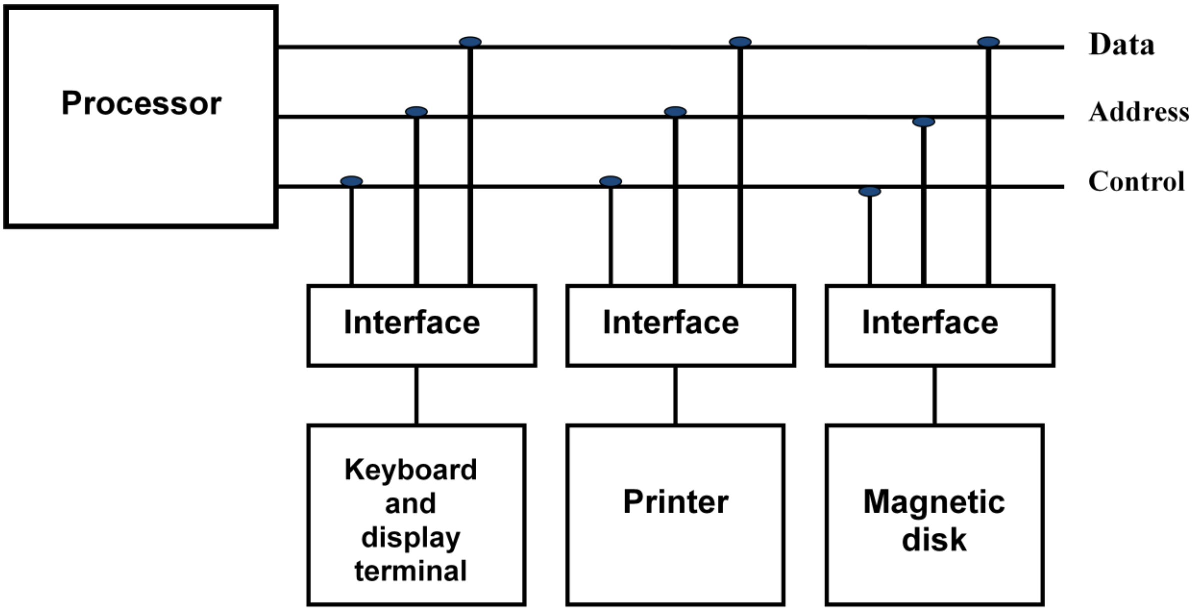
Control command- A control command is issued to activate the peripheral and to inform it what to do.

Status command- A status command is used to test various status conditions in the interface and the peripheral.

Data Output command- A data output command causes the interface to respond by transferring data from the bus into one of its registers.

Data Input command- The data input command is the opposite of the data output.

In this case the interface receives on item of data from the peripheral and places it in its buffer register. I/O Versus Memory Bus



Connection of I/O bus to input-output devices

To communicate with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address and read/write control lines. There are 3 ways that computer buses can be used to communicate with memory and I/O:

i. Use two Separate buses , one for memory and other for I/O.

ii. Use one common bus for both memory and I/O but separate control lines for each.

iii. Use one common bus for memory and I/O with common control lines.

I/O Processor

In the first method, the computer has independent sets of data, address and control buses one for accessing memory and other for I/O. This is done in computers that provides a separate I/O processor (IOP). The purpose of IOP is to provide an independent pathway for the transfer of information between external device and internal memory.

Asynchronous Data Transfer :

This Scheme is used when speed of I/O devices do not match with microprocessor, and timing characteristics of I/O devices is not predictable. In this method, process initiates the device and check its status. As a result, CPU has to wait till I/O device is ready to transfer data. When device is ready CPU issues instruction for I/O transfer. In this method two types of techniques are used based on signals before data transfer.

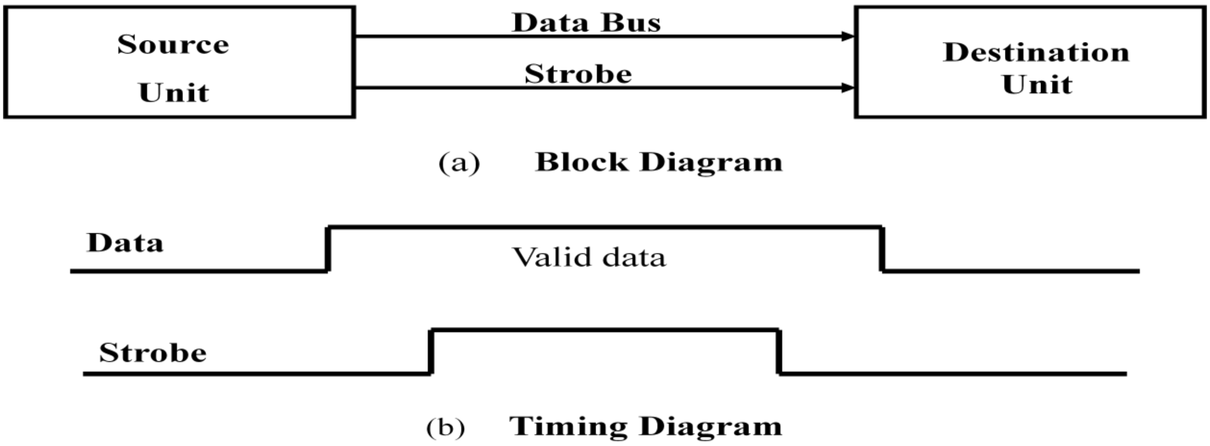
i. Strobe Control

ii. Handshaking

Strobe Signal :

The strobe control method of Asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either the source or the destination unit.

Data Transfer Initiated by Source Unit:



Source-Initiated strobe for Data Transfer

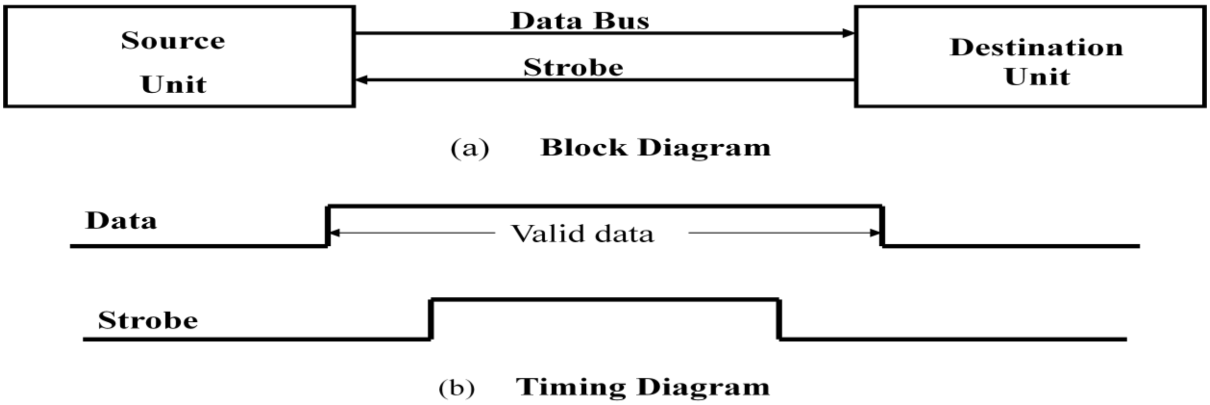
In the block diagram fig. (a), the data bus carries the binary information from source to destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available.

The timing diagram fig. (b) the source unit first places the data on the data bus. The information on the data bus and strobe signal remain in the active state to allow the destination unit to receive the data.

Data Transfer Initiated by Destination Unit:

In this method, the destination unit activates the strobe pulse, to informing the source to provide the data. The source will respond by placing the requested binary information on the data bus.

The data must be valid and remain in the bus long enough for the destination unit to accept it. When accepted the destination unit then disables the strobe and the source unit removes the data from the bus.



(a) Block Diagram

(b) Timing Diagram

Destination-Initiated strobe for Data Transfer

Disadvantage of Strobe Signal :

The disadvantage of the strobe method is that, the source unit initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was places in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on bus. The Handshaking method solves this problem.

Handshaking:

The handshaking method solves the problem of strobe method by introducing a second control signal that provides a reply to the unit that initiates the transfer.

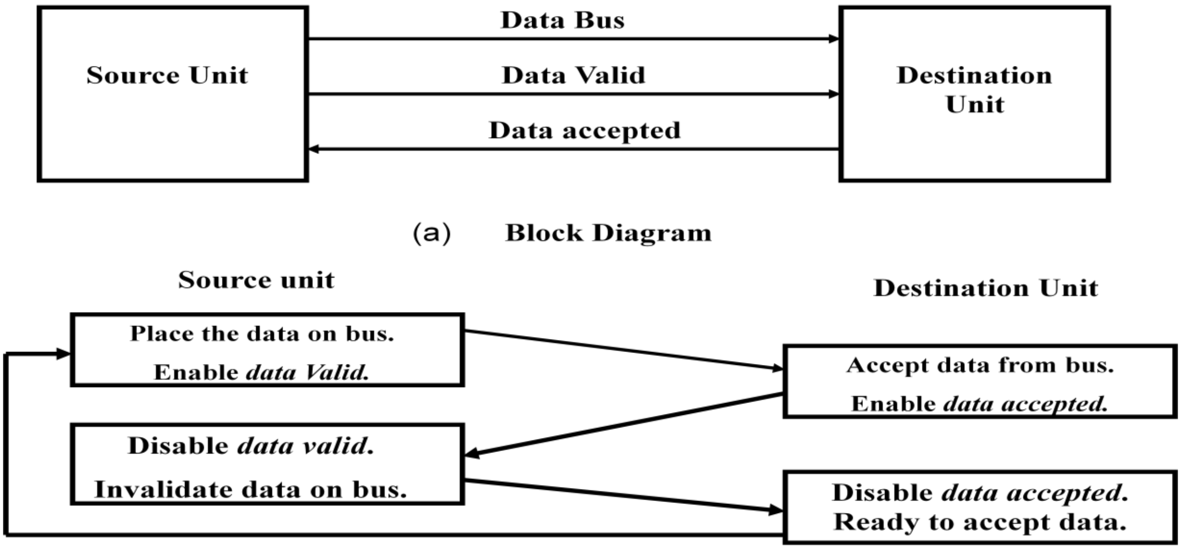
Principle of Handshaking:

The basic principle of the two-wire handshaking method of data transfer is as follow:

One control line is in the same direction as the data flows in the bus from the source to destination. It is used by source unit to inform the destination unit whether there a valid data in the bus. The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept the data. The sequence of control during the transfer depends on the unit that initiates the transfer.

Source Initiated Transfer using Handshaking:

The sequence of events shows four possible states that the system can be at any given time. The source unit initiates the transfer by placing the data on the bus and enabling its *data valid* signal. The *data accepted* signal is activated by the destination unit after it accepts the data from the bus. The source unit then disables its *data accepted* signal and the system goes into its initial state.



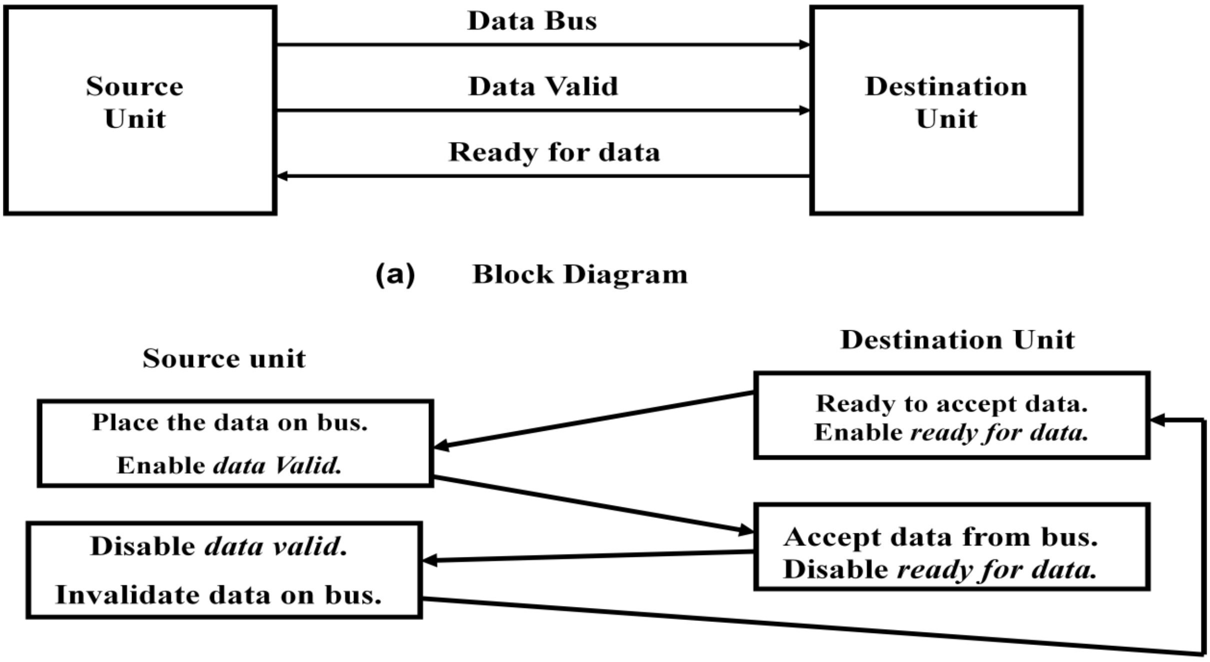
(a) Block Diagram

(b) Sequence of events

Destination Initiated Transfer Using Handshaking:

The name of the signal generated by the destination unit has been changed to *ready for data* to reflects its new meaning. The source unit in this case does not place data on the bus until after it receives the *ready for data* signal from the destination unit. From there on, the handshaking procedure follows the same pattern as in the source initiated case.

The only difference between the Source Initiated and the Destination Initiated transfer is in their choice of Initial sate.



(a) Block Diagram

(b) Sequence of events

Destination-Initiated transfer using Handshaking

Advantage of the Handshaking method:

- The Handshaking scheme provides degree of flexibility and reliability because the successful completion of data transfer relies on active participation by both units.
- If any of one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a *Timeout mechanism* which provides an alarm if the data is not completed within time.

Asynchronous Serial Transmission:

The transfer of data between two units is serial or parallel. In parallel data transmission, n bit in the message must be transmitted through n separate conductor path. In serial transmission, each bit in the message is sent in sequence one at a time.

Parallel transmission is faster but it requires many wires. It is used for short distances and where speed is important. Serial transmission is slower but is less expensive.

In Asynchronous serial transfer, each bit of message is sent a sequence at a time, and binary information is transferred only when it is available. When there is no information to be transferred, line remains idle.

In this technique each character consists of three points :

i. Start bit

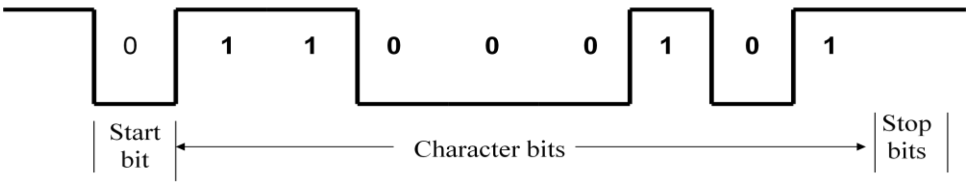
ii. Character bit

iii. Stop bit

i. Start Bit- First bit, called start bit is always zero and used to indicate the beginning character.

ii. Stop Bit- Last bit, called stop bit is always one and used to indicate end of characters. Stop bit is always in the 1- state and frame the end of the characters to signify the idle or wait state.

iii. Character Bit- Bits in between the start bit and the stop bit are known as character bits. The character bits always follow the start bit.



Asynchronous Serial Transmission

Serial Transmission of Asynchronous is done by two ways:

a) Asynchronous Communication Interface

b) First In First out Buffer

Asynchronous Communication Interface:

It works as both a receiver and a transmitter. Its operation is initialized by CPU by sending a byte to the control register.

The transmitter register accepts a data byte from CPU through the data bus and transferred to a shift register for serial transmission.

The receive portion receives information into another shift register, and when a complete data byte is received it is transferred to receiver register.

CPU can select the receiver register to read the byte through the data bus. Data in the status register is used for input and output flags.

First In First Out Buffer (FIFO):

A First In First Out (FIFO) Buffer is a memory unit that stores information in such a manner that the first item is in the item first out. A FIFO buffer comes with separate input and output terminals. The important feature of this buffer is that it can input data and output data at two different rates.

When placed between two units, the FIFO can accept data from the source unit at one rate, rate of transfer and deliver the data to the destination unit at another rate.

If the source is faster than the destination, the FIFO is useful for source data arrive in bursts that fills out the buffer. FIFO is useful in some applications when data are transferred asynchronously.

Modes of Data Transfer :

Transfer of data is required between CPU and peripherals or memory or sometimes between any two devices or units of your computer system. To transfer a data from one unit to another one should be sure that both units have proper connection and at the time of data transfer the receiving unit is not busy. This data transfer with the computer is Internal Operation.

All the internal operations in a digital system are synchronized by means of clock pulses supplied by a common clock pulse Generator. The data transfer can be

i. Synchronous or

ii. Asynchronous

When both the transmitting and receiving units use same clock pulse then such a data transfer is called Synchronous process. On the other hand, if the there is not concept of clock pulses and the sender operates at different moment than the receiver then such a data transfer is called Asynchronous data transfer.

The data transfer can be handled by various modes. some of the modes use CPU as an intermediate path, others transfer the data directly to and from the memory unit and this can be handled by 3 following ways:

i. Programmed I/O

ii. Interrupt-Initiated I/O

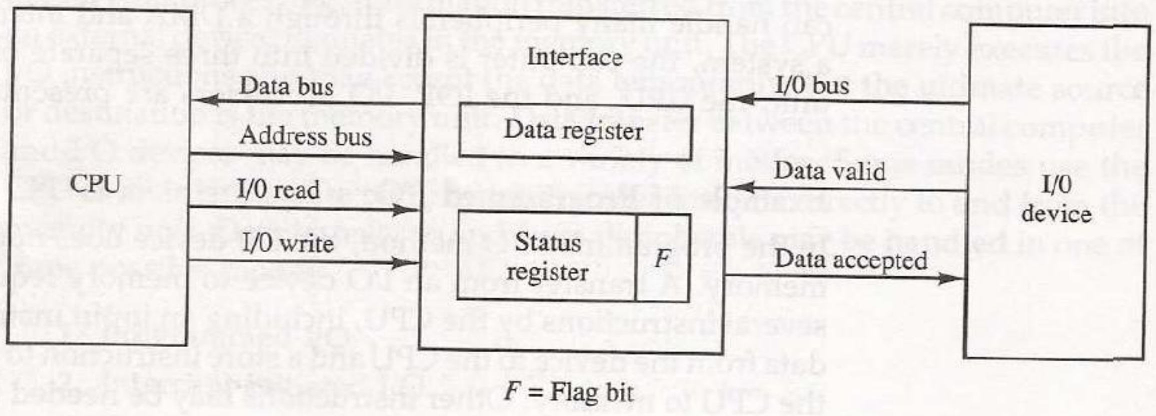
iii. Direct Memory Access (DMA)

Programmed I/O Mode:

In this mode of data transfer the operations are the results in I/O instructions which is a part of computer program. Each data transfer is initiated by a instruction in the program. Normally the transfer is from a CPU register to peripheral device or vice-versa.

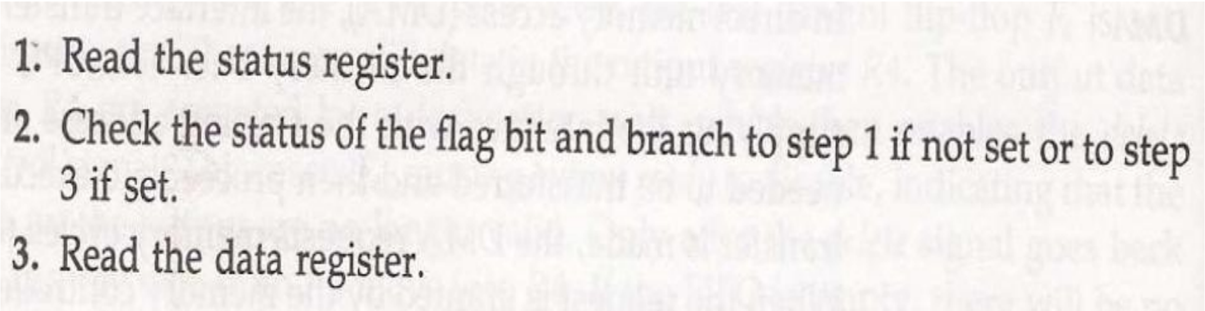
Once the data is initiated the CPU starts monitoring the interface to see when next transfer can made. The instructions of the program keep close tabs on everything that takes place in the interface unit and the I/O devices.

Figure Data transfer from I/O device to CPU.



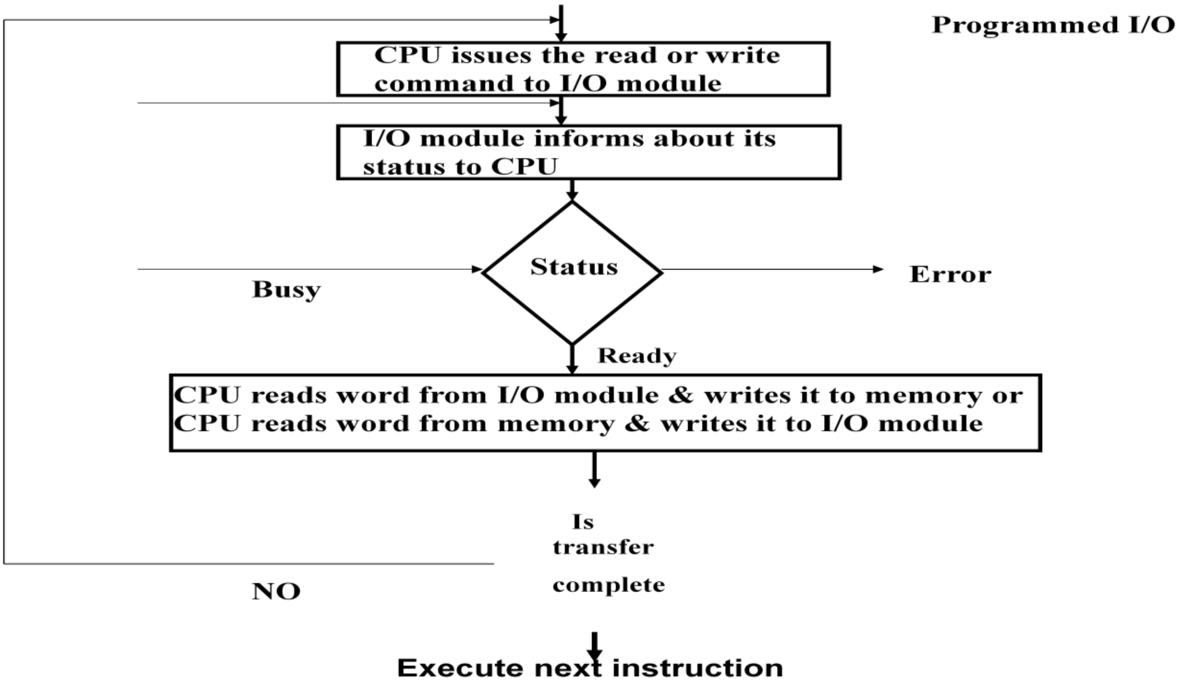
F = Flag bit

- The transfer of data requires three instructions:



1. Read the status register.
2. Check the status of the flag bit and branch to step 1 if not set or to step 3 if set.
3. Read the data register.

In this technique CPU is responsible for executing data from the memory for output and storing data in memory for executing of Programmed I/O as shown in Flowchart-:


Programmed I/O
CPU issues the read or write command to I/O module
I/O module informs about its status to CPU
Status
Busy
Error
Ready
CPU reads word from I/O module & writes it to memory or CPU reads word from memory & writes it to I/O module
Is transfer complete
NO
Execute next instruction

Drawback of the Programmed I/O :

The main drawback of the Program Initiated I/O was that the CPU has to monitor the units all the times when the program is executing. Thus the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process and the CPU time is wasted a lot in keeping an eye to the executing of program.

To remove this problem an Interrupt facility and special commands are used.

Interrupt-Initiated I/O :

In this method an interrupt facility an interrupt command is used to inform the device about the start and end of transfer. In the meantime the CPU executes other program. When the interface determines that the device is ready for data transfer it generates an Interrupt Request and sends it to the computer.

When the CPU receives such an signal, it temporarily stops the execution of the program and branches to a service program to process the I/O transfer and after completing it returns back to task, what it was originally performing.

- In this type of IO, computer does not check the flag. It continue to perform its task.

- Whenever any device wants the attention, it sends the interrupt signal to the CPU.
- CPU then deviates from what it was doing, store the return address from PC and branch to the address of the subroutine.
- There are two ways of choosing the branch address:
    - Vectored Interrupt
    - Non-vectored Interrupt
- In vectored interrupt the source that interrupt the CPU provides the branch information. This information is called interrupt vectored.
- In non-vectored interrupt, the branch address is assigned to the fixed address in the memory.

Priority Interrupt:

- There are number of IO devices attached to the computer.
- They are all capable of generating the interrupt.
- When the interrupt is generated from more than one device, priority interrupt system is used to determine which device is to be serviced first.
- Devices with high speed transfer are given higher priority and slow devices are given lower priority.
- Establishing the priority can be done in two ways:
    - Using Software
    - Using Hardware
- A pooling procedure is used to identify highest priority in software means.

Polling Procedure :

- There is one common branch address for all interrupts.
- Branch address contain the code that polls the interrupt sources in sequence. The highest priority is tested first.
- The particular service routine of the highest priority device is served.
- The disadvantage is that time required to poll them can exceed the time to serve them in large number of IO devices.

Using Hardware:

- Hardware priority system function as an overall manager.

- It accepts interrupt request and determine the priorities.
- To speed up the operation each interrupting devices has its own interrupt vector.
- No polling is required, all decision are established by hardware priority interrupt unit.
- It can be established by serial or parallel connection of interrupt lines.

Serial or Daisy Chaining Priority:

- Device with highest priority is placed first.
- Device that wants the attention send the interrupt request to the CPU.
- CPU then sends the INTACK signal which is applied to PI(priority in) of the first device.
- If it had requested the attention, it place its VAD(vector address) on the bus. And it block the signal by placing 0 in PO(priority out)
- If not it pass the signal to next device through PO(priority out) by placing 1.
- This process is continued until appropriate device is found.
- The device whose PI is 1 and PO is 0 is the device that send the interrupt request.

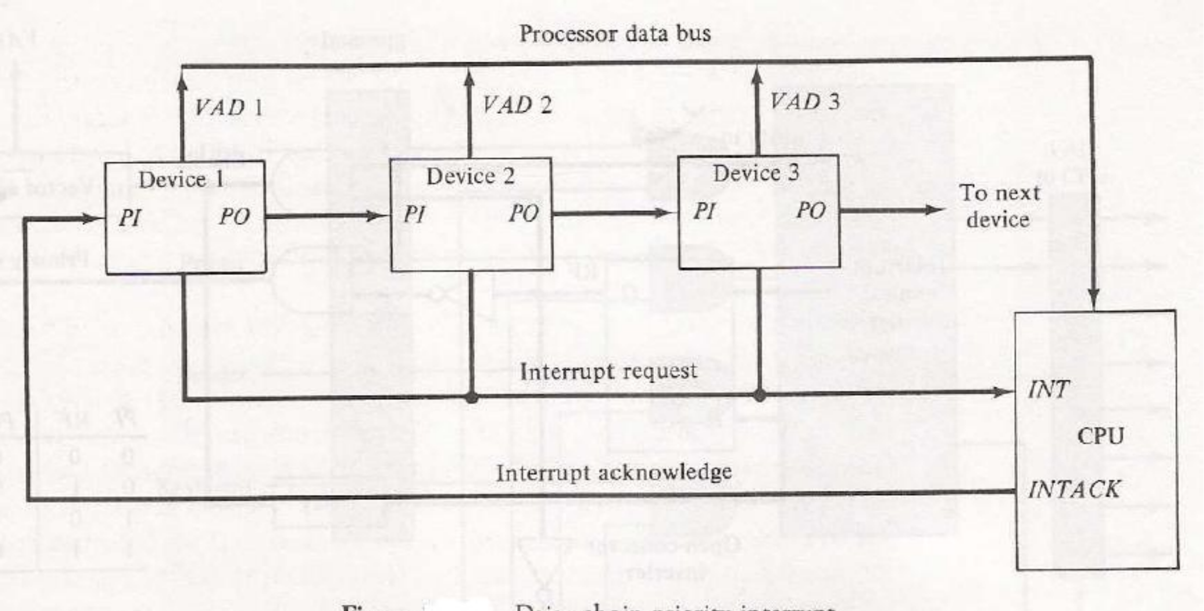
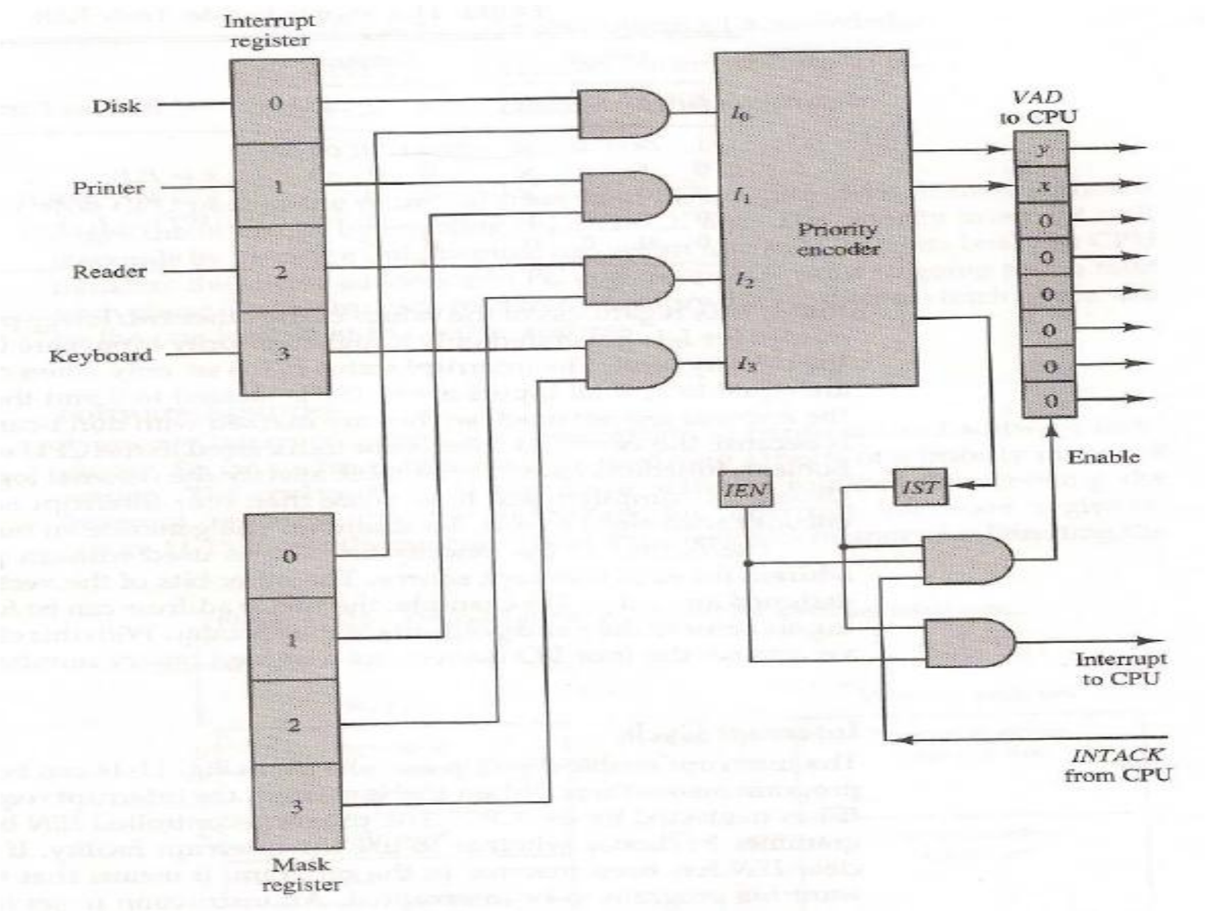


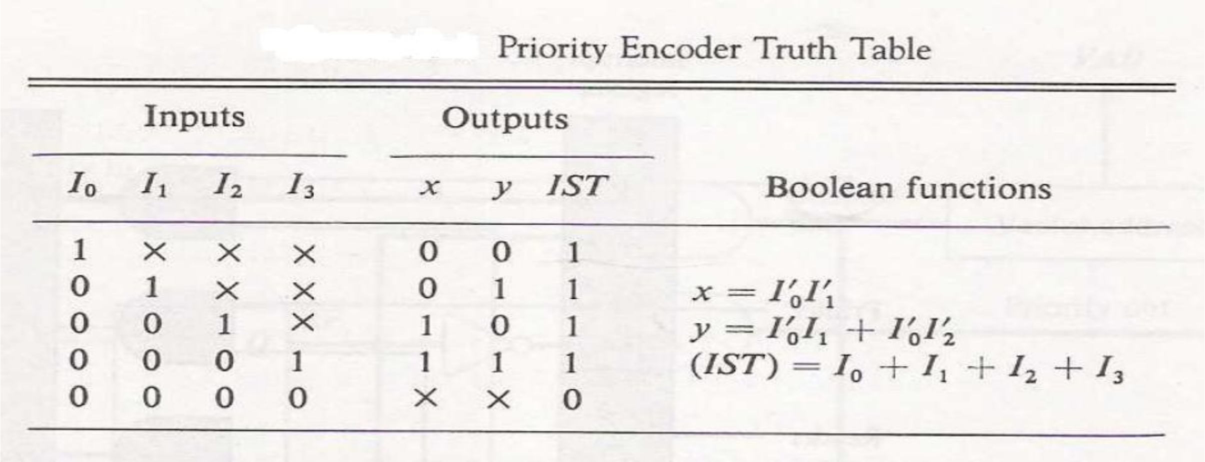
Figure Daisy-chain priority interrupt.

Parallel Priority Interrupt :

- It consist of interrupt register whose bits are set separately by the interrupting devices.
- Priority is established according to the position of the bits in the register.

- Mask register is used to provide facility for the higher priority devices to interrupt when lower priority device is being serviced or disable all lower priority devices when higher is being serviced.
- Corresponding interrupt bit and mask bit are ANDed and applied to priority encoder.
- Priority encoder generates two bits of vector address.
- Another output from it sets IST(interrupt status flip flop).

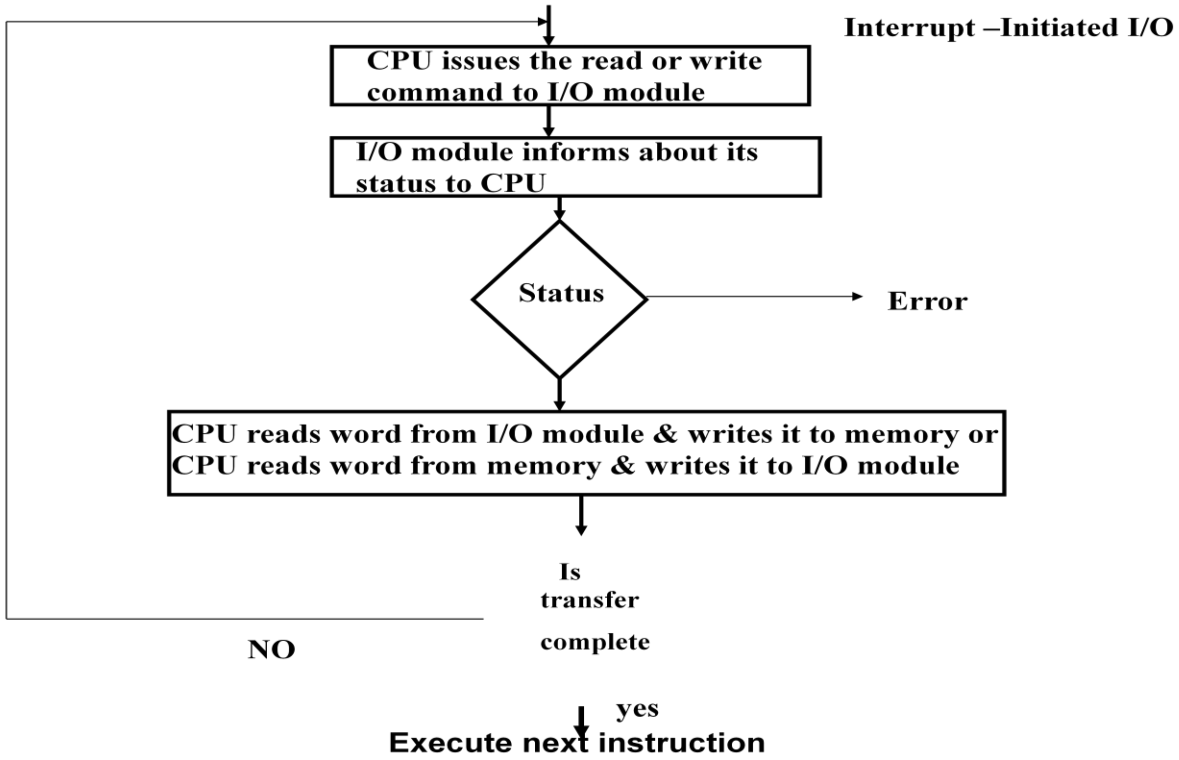




Priority Encoder Truth Table

Inputs				Outputs			
I_0	I_1	I_2	I_3	x	y	IST	Boolean functions
1	×	×	×	0	0	1	
0	1	×	×	0	1	1	$x = I'_0 I'_1$
0	0	1	×	1	0	1	$y = I'_0 I_1 + I'_0 I'_2$
0	0	0	1	1	1	1	$(IST) = I_0 + I_1 + I_2 + I_3$
0	0	0	0	×	×	0	

The Execution process of Interrupt–Initiated I/O is represented in the flowchart:


Interrupt –Initiated I/O
CPU issues the read or write command to I/O module
I/O module informs about its status to CPU
Status
Error
CPU reads word from I/O module & writes it to memory or CPU reads word from memory & writes it to I/O module
Is transfer complete
NO
yes
Execute next instruction

Direct Memory Access (DMA):

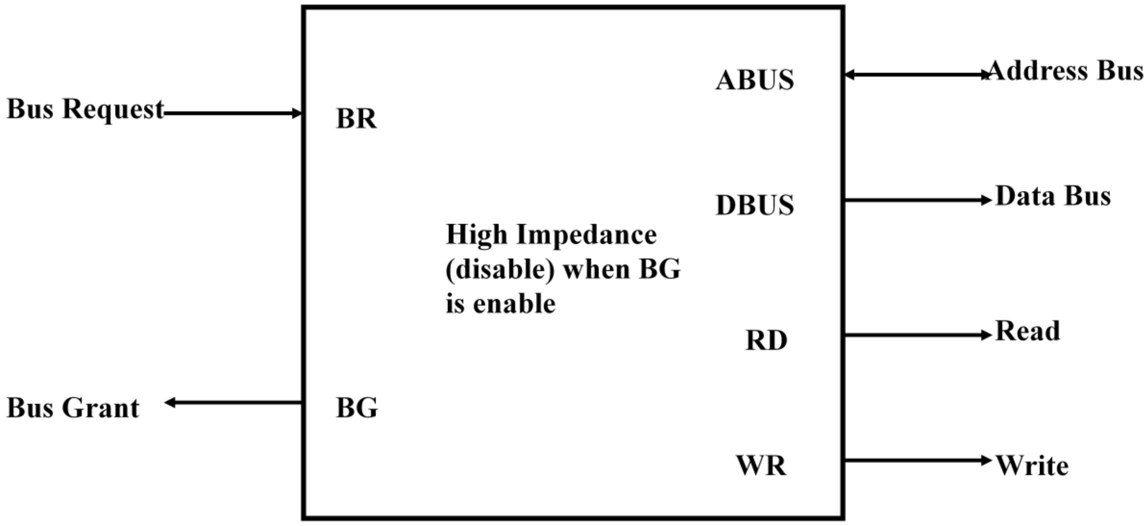
In the Direct Memory Access (DMA) the interface transfer the data into and out of the memory unit through the memory bus. The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access (DMA).

During the DMA transfer, the CPU is idle and has no control of the memory buses. A DMA Controller takes over the buses to manage the transfer directly between the I/O device and memory.

The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessor is to disable the buses through special control signals such as:

- Bus Request (BR)
- Bus Grant (BG)

These two control signals in the CPU that facilitates the DMA transfer. The *Bus Request (BR)* input is used by the *DMA controller* to request the CPU. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, data bus and read write lines into a *high Impedance state*. High Impedance state means that the output is disconnected.



CPU bus Signals for DMA Transfer

The CPU activates the *Bus Grant (BG)* output to inform the external DMA that the Bus Request (BR) can now take control of the buses to conduct memory transfer without processor.

When the DMA terminates the transfer, it disables the *Bus Request (BR)* line. The CPU disables the *Bus Grant (BG)*, takes control of the buses and return to its normal operation.

The transfer can be made in several ways that are:

i. DMA Burst

ii. Cycle Stealing

i) DMA Burst :- In DMA Burst transfer, a block sequence consisting of a number of memory words is transferred in continuous burst while the DMA controller is master of the memory buses.

ii) Cycle Stealing :- Cycle stealing allows the DMA controller to transfer one data word at a time, after which it must returns control of the buses to the CPU.

DMA Controller:

The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. The DMA controller has three registers:

i. Address Register

ii. Word Count Register

iii. Control Register

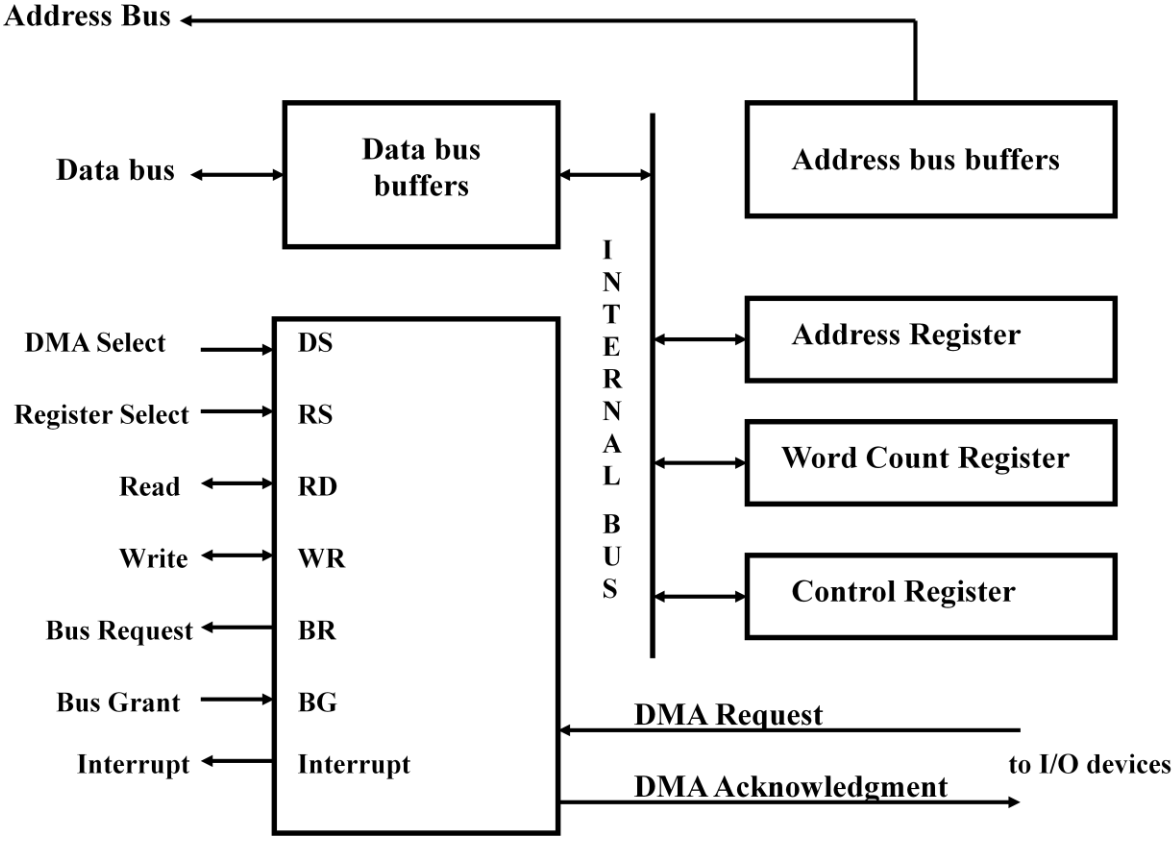
i. Address Register :- Address Register contains an address to specify the desired location in memory.

ii. Word Count Register :- WC holds the number of words to be transferred. The register is incre/decre by one after each word transfer and internally tested for zero.

i. Control Register :- Control Register specifies the mode of transfer

The unit communicates with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (Register select) inputs. The RD (read) and WR (write) inputs are bidirectional.

When the BG (Bus Grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When BG =1, the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.



Block Diagram of DMA Controller

DMA Transfer:

The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can transfer between the peripheral and the memory.

When BG = 0 the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers. When BG=1, the RD and WR are output lines from the DMA controller to the random access memory to specify the read or write operation of data.

Summary :

- Interface is the point where a connection is made between two different parts of a system.
- The strobe control method of Asynchronous data transfer employs a single control line to time each transfer.
- The handshaking method solves the problem of strobe method by introducing a second control signal that provides a reply to the unit that initiates the transfer.
- Programmed I/O mode of data transfer the operations are the results in I/O instructions which is a part of computer program.
- In the Interrupt Initiated I/O method an interrupt facility an interrupt command is used to inform the device about the start and end of transfer.
- In the Direct Memory Access (DMA) the interface transfer the data into and out of the memory unit through the memory bus.

Input-Output Processor:

- It is a processor with direct memory access capability that communicates with IO devices.
- IOP is similar to CPU except that it is designed to handle the details of IO operation.
- Unlike DMA which is initialized by CPU, IOP can fetch and execute its own instructions.
- IOP instruction are specially designed to handle IO operation.

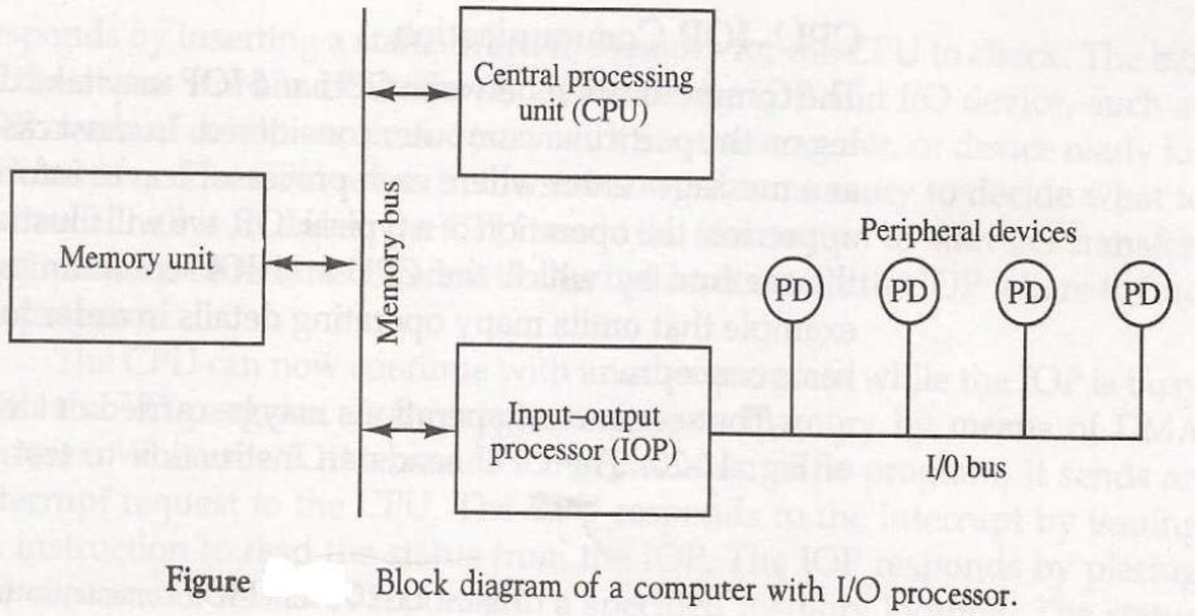
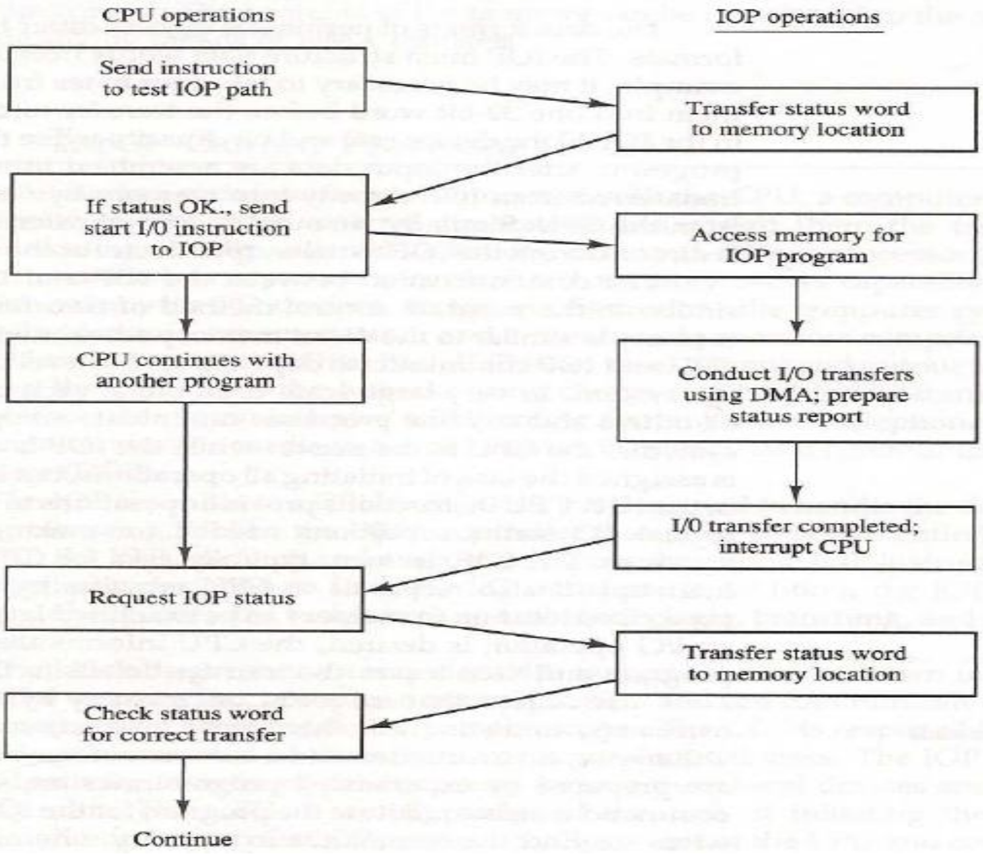


Figure Block diagram of a computer with I/O processor.

- Memory occupies the central position and can communicate with each processor by DMA.
- CPU is responsible for processing data.
- IOP provides the path for transfer of data between various peripheral devices and memory.
- Data formats of peripherals differ from CPU and memory. IOP maintain such problems.
- Data are transfer from IOP to memory by stealing one memory cycle.
- Instructions that are read from memory by IOP are called commands to distinguish them from instructions that are read by the CPU.

Figure 11-20 CPU-IOP communication.


CPU operations
IOP operations
Send instruction to test IOP path
Transfer status word to memory location
If status OK., send start I/0 instruction to IOP
Access memory for IOP program
CPU continues with another program
Conduct I/O transfers using DMA; prepare status report
I/0 transfer completed; interrupt CPU
Request IOP status
Transfer status word to memory location
Check status word for correct transfer
Continue

Instruction *that are read from memory by an IOP*

» Distinguish from instructions that are read by the CPU

» Commands are prepared by experienced programmers and are stored in memory

» Command word = IOP program

PIPELINE AND VECTOR PROCESSING

Parallel processing:

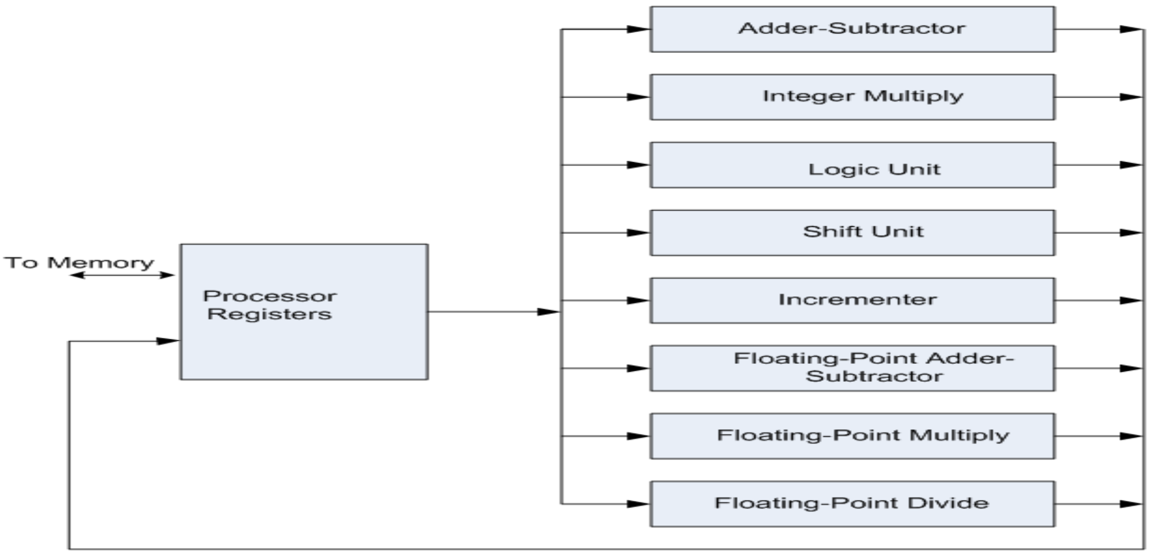
- Parallel processing is a term used for a large class of techniques that

are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.

- It refers to techniques that are used to provide simultaneous data processing.
- The system may have two or more ALUs to be able to execute two or more instruction at the same time.
- The system may have two or more processors operating concurrently.
- It can be achieved by having multiple functional units that perform same or different operation simultaneously.
- Example of parallel Processing:
  - Multiple Functional Unit:

Separate the execution unit into eight functional units operating in parallel.

- There are variety of ways in which the parallel processing can be classified
  - Internal Organization of Processor
  - Interconnection structure between processors
  - Flow of information through system



Architectural Classification:

- – Flynn's classification
  - » Based on the multiplicity of *Instruction Streams* and *Data Streams*
  - » Instruction Stream
    - Sequence of Instructions read from memory
  - » Data Stream
    - Operations performed on the data in the processor

		Number of *Data Streams*	
		Single	Multiple
Number of *Instruction Streams*	Single	SISD	SIMD
	Multiple	MISD	MIMD

- SISD represents the organization containing single control unit, a processor unit and a memory unit. Instruction are executed sequentially and system may or may not have internal parallel processing capabilities.
- SIMD represents an organization that includes many processing units under the supervision of a common control unit.
- MISD structure is of only theoretical interest since no practical system has been constructed using this organization.
- MIMD organization refers to a computer system capable of processing several programs at the same time.

The main difference between multicomputer system and multiprocessor system is that the multiprocessor system is controlled by one operating system that provides interaction between processors and all the component of the system cooperate in the solution of a problem.

- Parallel Processing can be discussed under following topics:
  - ○ Pipeline Processing
  - ○ Vector Processing
  - ○ Array Processors

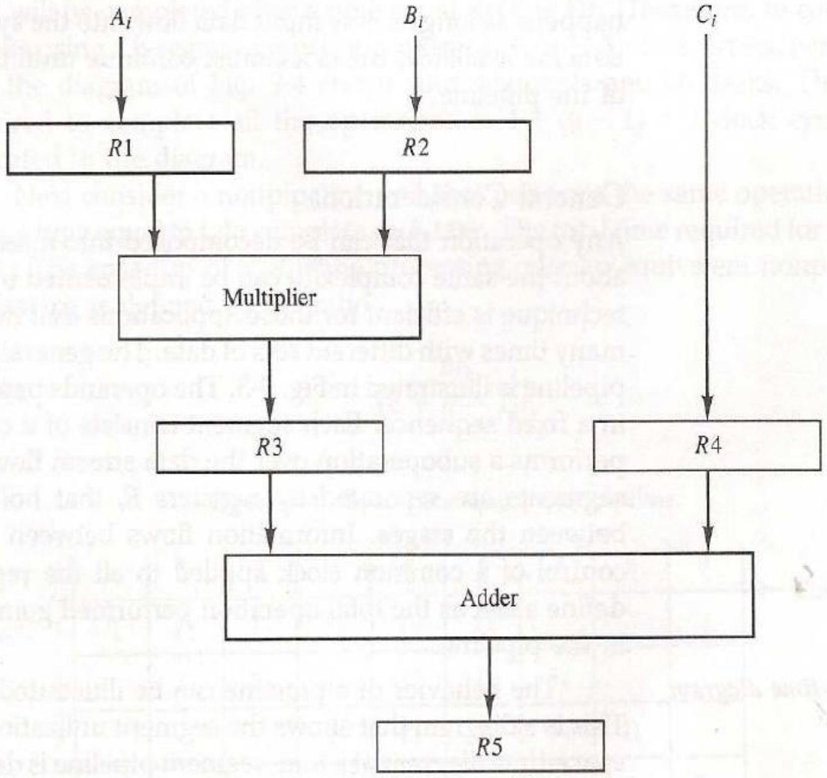
PIPELINING:

- A technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments.
- It is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segments that operates concurrently with all other segments.
- Each segment performs partial processing dictated by the way task is partitioned.
- The result obtained from each segment is transferred to next segment.
- The final result is obtained when data have passed through all segments.

- Suppose we have to perform the following task:
- Each sub operation is to be performed in a segment within a pipeline. Each segment has one or two registers and a combinational circuit.

$$A_i * B_i + C_i \qquad \text{for } i = 1, 2, 3, \ldots, 7$$

Figure Example of pipeline processing.

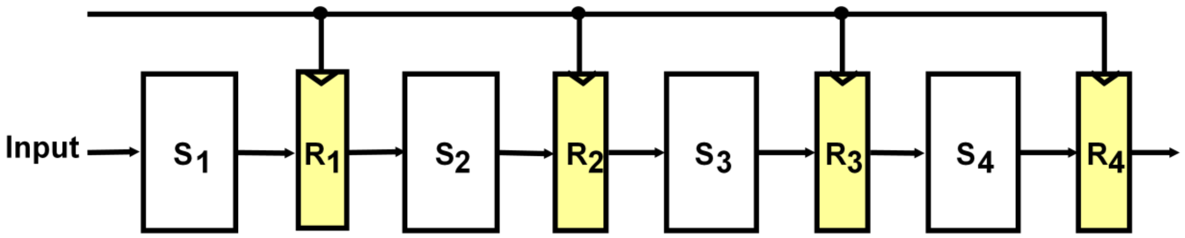


Suboperations in each segment: $R1 \leftarrow A_i$, $R2 \leftarrow B_i$ Load A_i and B_i
$R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$ Multiply and load C_i
$R5 \leftarrow R3 + R4$ Add

OPERATIONS IN EACH PIPELINE STAGE:

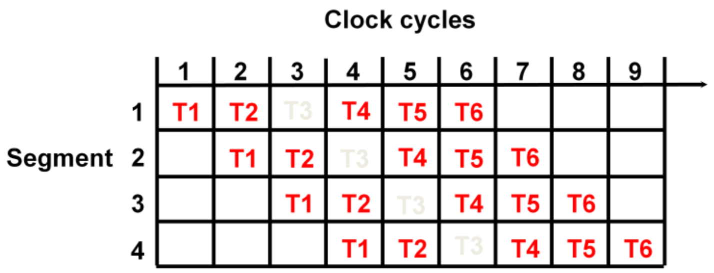
Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A1	B1	---	---	-------
2	A2	B2	A1 * B1	C1	-------
3	A3	B3	A2 * B2	C2	A1 * B1 + C1
4	A4	B4	A3 * B3	C3	A2 * B2 + C2
5	A5	B5	A4 * B4	C4	A3 * B3 + C3
6	A6	B6	A5 * B5	C5	A4 * B4 + C4
7	A7	B7	A6 * B6	C6	A5 * B5 + C5
8			A7 * B7	C7	A6 * B6 + C6
9					A7 * B7 + C7

- General Structure of a 4-Segment Pipeline



- Space-Time Diagram

The following diagram shows 6 tasks T1 through T6 executed in 4segments.



Clock cycles

Segment	1	2	3	4	5	6	7	8	9
1	T1	T2	T3	T4	T5	T6			
2		T1	T2	T3	T4	T5	T6		
3			T1	T2	T3	T4	T5	T6	
4				T1	T2	T3	T4	T5	T6

No matter how many segments, once the pipeline is full, it takes only one clock period to obtain an output.

PIPELINE SPEEDUP:

Consider the case where a k-segment pipeline used to execute n tasks.

- n = 6 in previous example

- k = 4 in previous example

- Pipelined Machine (k stages, n tasks)
  - The first task t1 requires k clock cycles to complete its operation since there are k segments
  - The remaining n-1 tasks require n-1 clock cycles
  - The n tasks clock cycles = k+(n-1) (9 in previous example)

- Conventional Machine (Non-Pipelined)
  - Cycles to complete each task in nonpipeline = k
  - For n tasks, n cycles required is

- Speedup (S)
  - S = Nonpipeline time /Pipeline time

- For n tasks: S = nk/(k+n-1)
- As n becomes much larger than k-1; Therefore, S = nk/n = k

PIPELINE AND MULTIPLE FUNCTION UNITS:

Example:

- 4-stage pipeline
- 100 tasks to be executed
- 1 task in non-pipelined system; 4 clock cycles

Pipelined System : k + n - 1 = 4 + 99 = 103 clock cycles

Non-Pipelined System : n*k = 100 * 4 = 400 clock cycles

Speedup : S_k = 400 / 103 = 3.88

Types of Pipelining:

- Arithmetic Pipeline
- Instruction Pipeline

ARITHMETIC PIPELINE:

- Pipeline arithmetic units are usually found in very high speed computers.
- They are used to implement floating point operations.

- We will now discuss the pipeline unit for the floating point addition and subtraction.
- The inputs to floating point adder pipeline are two normalized floating point numbers.
- A and B are mantissas and a and b are the exponents.
- The floating point addition and subtraction can be performed in four segments.

Floating-point adder:

[1] Compare the exponents

[2] Align the mantissa

[3] Add/sub the mantissa

[4] Normalize the result

$X = A \times 10^a = 0.9504 \times 10^3$

$Y = B \times 10^b = 0.8200 \times 10^2$

1) Compare exponents :

$3 - 2 = 1$

2) Align mantissas

$X = 0.9504 \times 10^3$

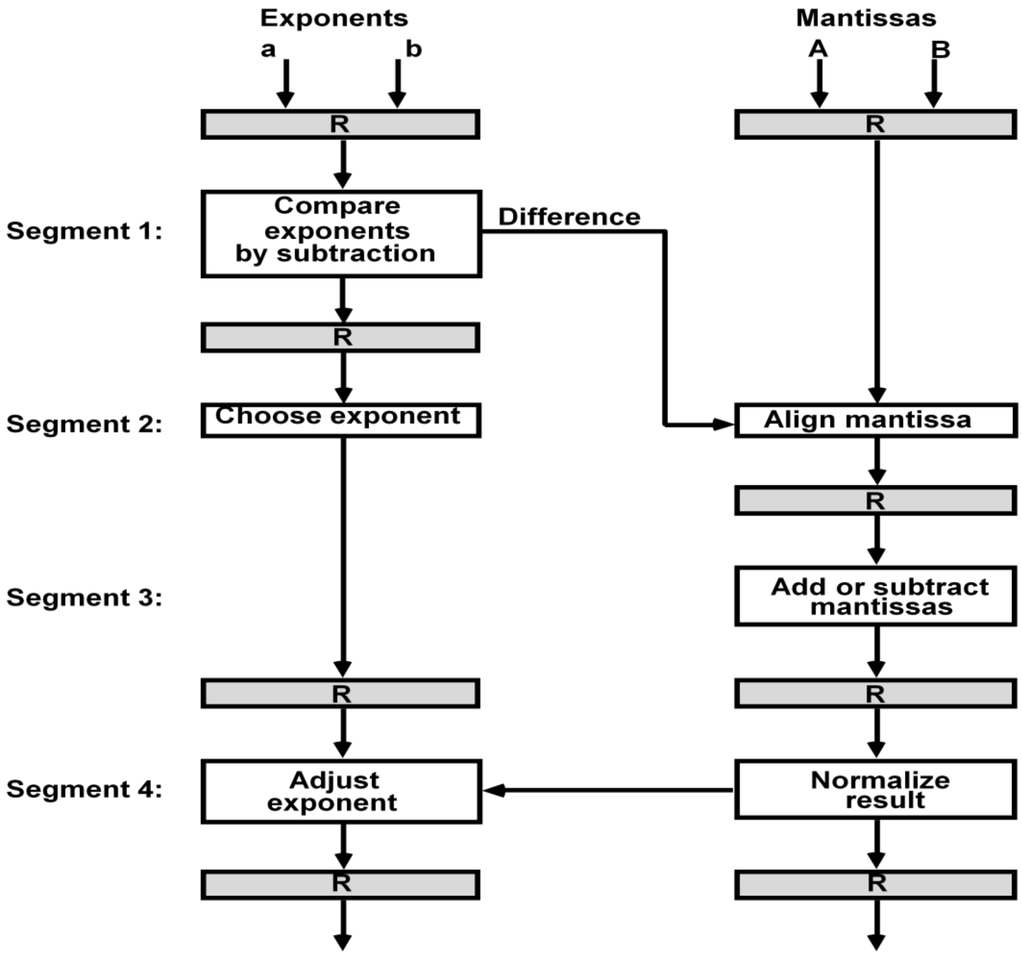
$Y = 0.08200 \times 10^3$

3) Add mantissas

$Z = 1.0324 \times 10^3$

4) Normalize result

$Z = 0.10324 \times 10^4$



Instruction Pipeline:

- Pipeline processing can occur not only in the data stream but in the instruction stream as well.
- An instruction pipeline reads consecutive instruction from memory while previous instruction are being executed in other segments.
- This caused the instruction fetch and execute segments to overlap and perform simultaneous operation.

Four Segment CPU Pipeline:

- FI segment fetches the instruction.
- DA segment decodes the instruction and calculate the effective address.
- FO segment fetches the operand.
- EX segment executes the instruction.

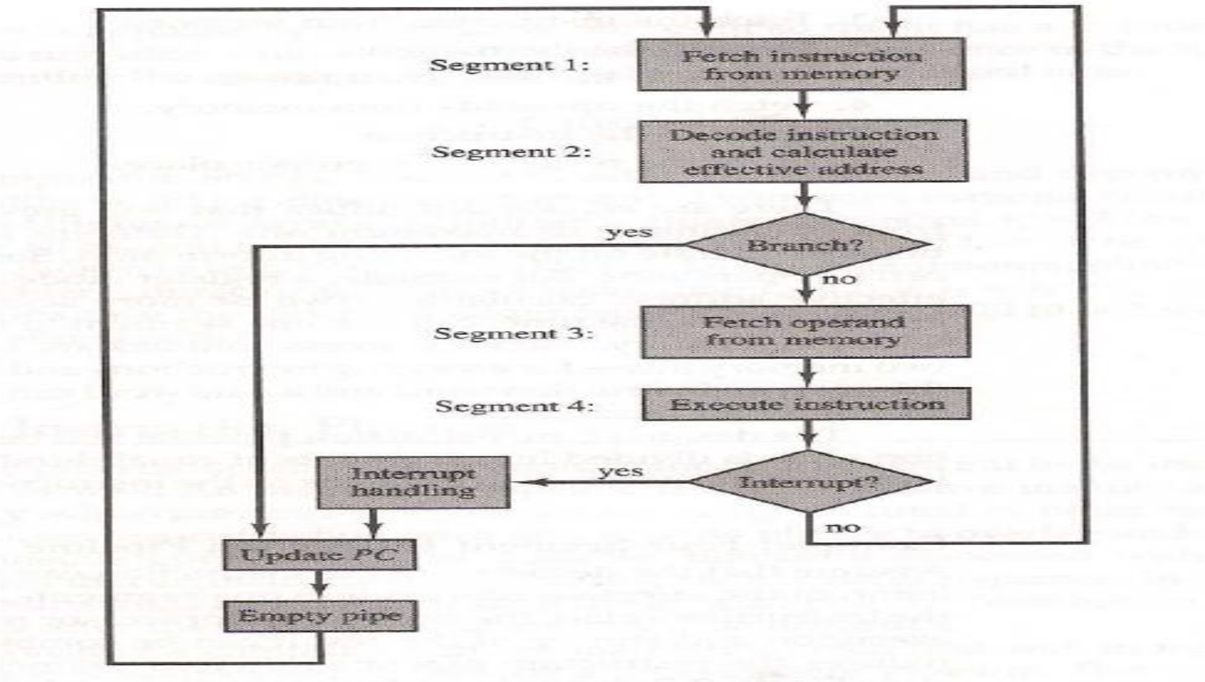
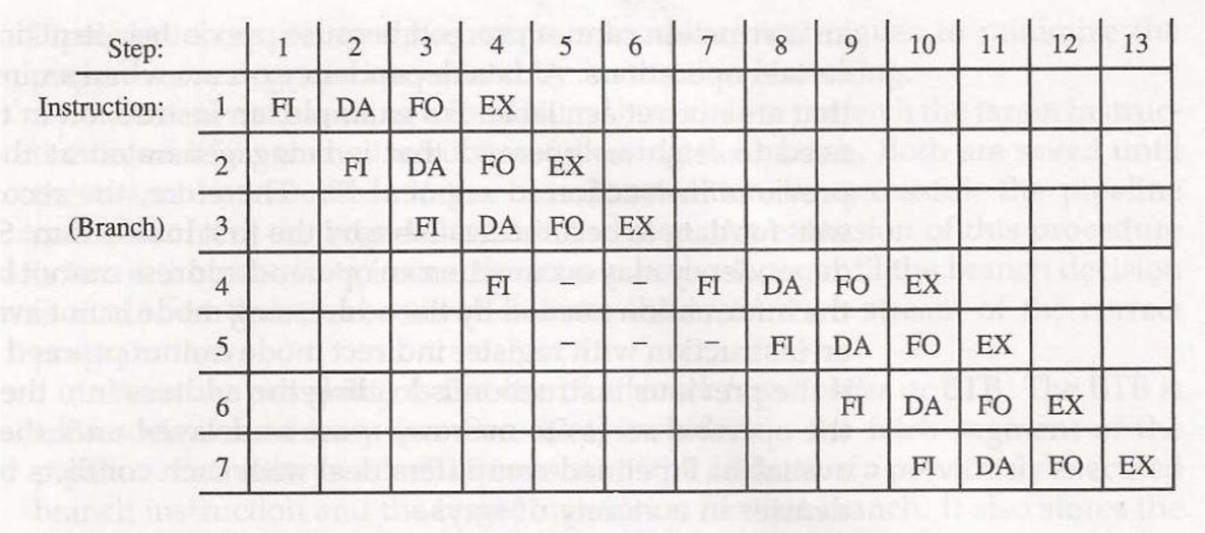


Figure Four-segment CPU pipeline.



Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Figure Timing of instruction pipeline.

INSTRUCTION CYCLE:

Pipeline processing can occur also in the instruction stream. An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.

Six Phases* in an Instruction Cycle

[1] Fetch an instruction from memory

[2] Decode the instruction

[3] Calculate the effective address of the operand

[4] Fetch the operands from memory

[5] Execute the operation

[6] Store the result in the proper place

* Some instructions skip some phases

* Effective address calculation can be done in the part of the decoding phase

* Storage of the operation result into a register is done automatically in the execution phase

==> 4-Stage Pipeline

[1] FI: Fetch an instruction from memory

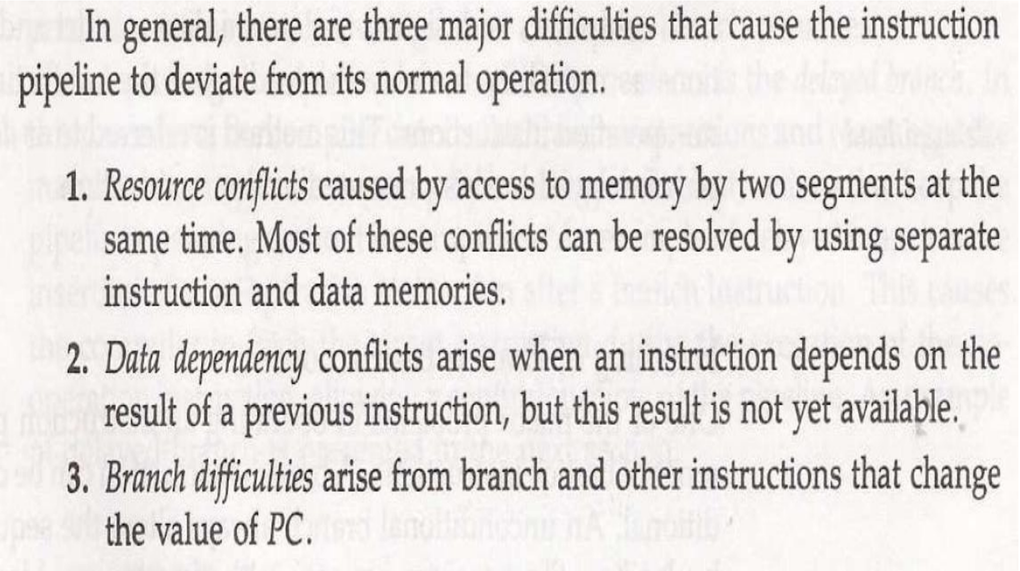
[2] DA: Decode the instruction and calculate the effective address of the operand

[3] FO: Fetch the operand

[4] EX: Execute the operation

Pipeline Conflicts :

- Pipeline Conflicts : 3 major difficulties



In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

1. *Resource conflicts* caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.
2. *Data dependency* conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
3. *Branch difficulties* arise from branch and other instructions that change the value of *PC*.

–

1) Resource conflicts: memory access by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.

2) Data dependency: when an instruction depend on the result of a previous instruction, but this result is not yet available.

Example: an instruction with register indirect mode cannot proceed to fetch the operand if the previous instruction is loading the address into the register.

3) Branch difficulties: branch and other instruction (interrupt, ret, ..) that change the value of PC.

Handling Data Dependency:

- This problem can be solved in the following ways:
  - Hardware interlocks: It is the circuit that detects the conflict situation and delayed the instruction by sufficient cycles to resolve the conflict.
  - Operand Forwarding: It uses the special hardware to detect the conflict and avoid it by routing the data through the special path between pipeline segments.
  - Delayed Loads: The compiler detects the data conflict and reorder the instruction as necessary to delay the loading of the conflicting data by inserting no operation instruction.

Handling of Branch Instruction:

- Pre fetch the target instruction.
- Branch target buffer(BTB) included in the fetch segment of the pipeline
- Branch Prediction
- Delayed Branch

RISC Pipeline:

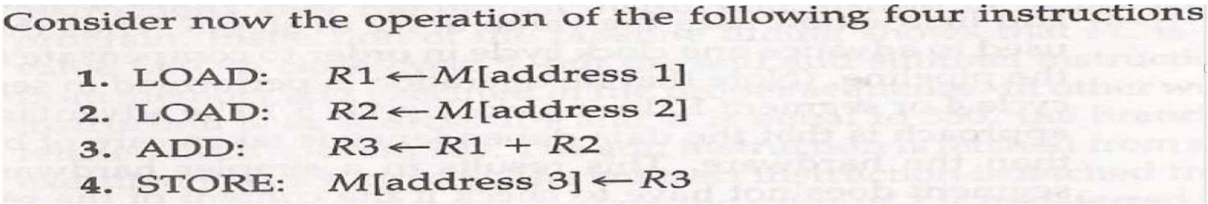
- Simplicity of instruction set is utilized to implement an instruction pipeline using small number of sub-operation, with each being executed in single clock cycle.

Since all operation are performed in the register, there is no need of effective address calculation.

Three Segment Instruction Pipeline:

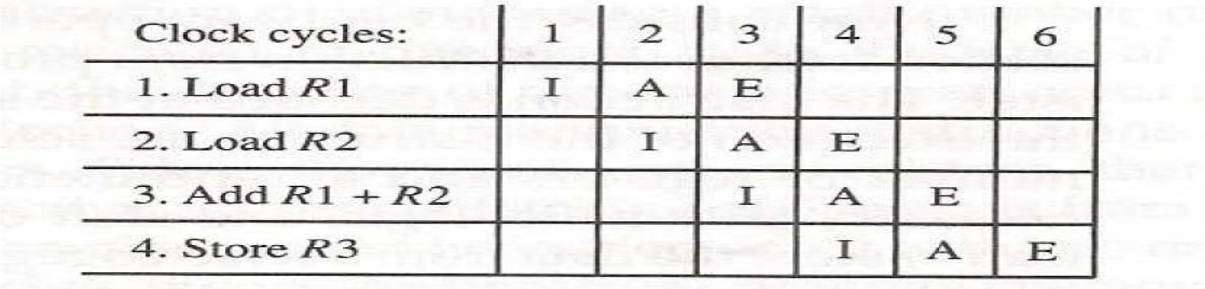
- I: Instruction Fetch
- A: ALU Operation
- E: Execute Instruction

Delayed Load:



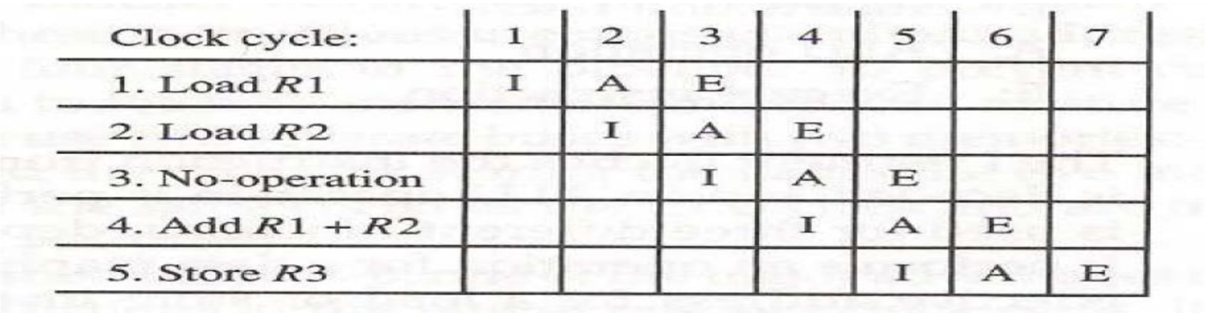
Consider now the operation of the following four instructions

1. LOAD: $R1 \leftarrow M[\text{address } 1]$
2. LOAD: $R2 \leftarrow M[\text{address } 2]$
3. ADD: $R3 \leftarrow R1 + R2$
4. STORE: $M[\text{address } 3] \leftarrow R3$



Clock cycles:	1	2	3	4	5	6
1. Load $R1$	I	A	E			
2. Load $R2$		I	A	E		
3. Add $R1 + R2$			I	A	E	
4. Store $R3$				I	A	E

Pipeline timing with data conflict

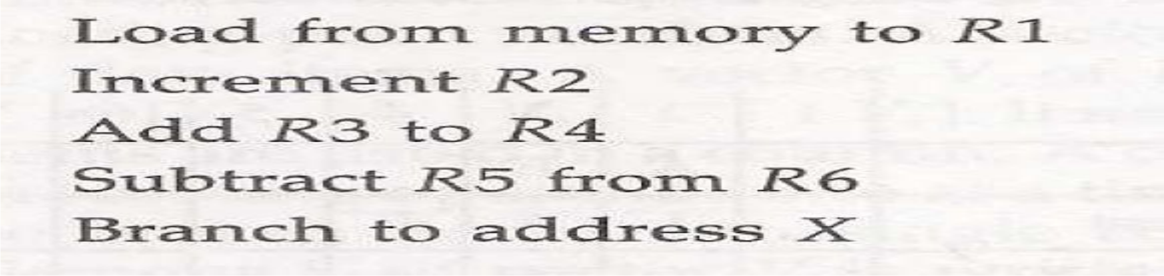


Clock cycle:	1	2	3	4	5	6	7
1. Load $R1$	I	A	E				
2. Load $R2$		I	A	E			
3. No-operation			I	A	E		
4. Add $R1 + R2$				I	A	E	
5. Store $R3$					I	A	E

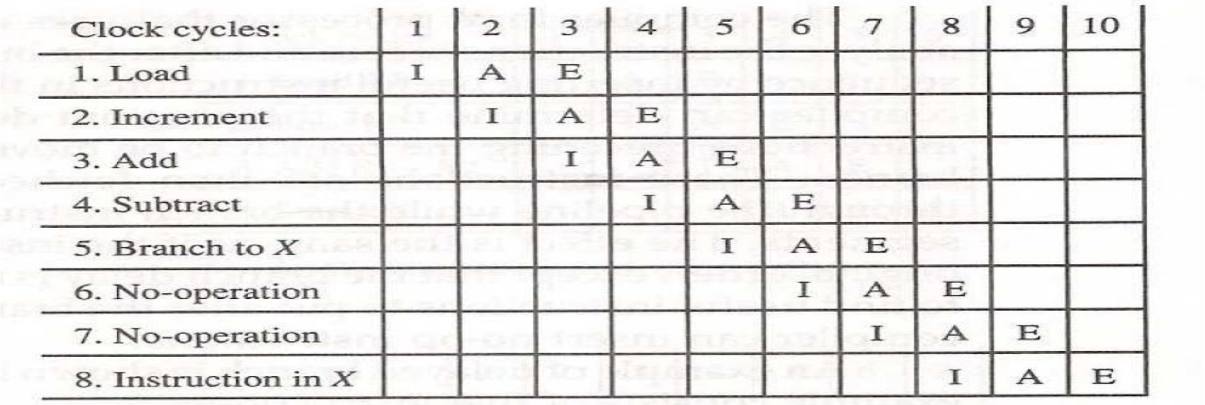
Pipeline timing with delayed load

Delayed Branch:

Let us consider the program having the following 5 instructions



Load from memory to $R1$
Increment $R2$
Add $R3$ to $R4$
Subtract $R5$ from $R6$
Branch to address X



Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to *X*					I	A	E			
6. No-operation						I	A	E		
7. No-operation							I	A	E	
8. Instruction in *X*								I	A	E

Using no-operation instructions

Organization of Intel 8085 Micro-Processor:

The microprocessors that are available today came with a wide variety of capabilities and architectural features. All of them, regardless of their diversity, are provided with at least the following functional components, which form the central processing unit (CPU) of a classical computer.

1. Register Section : A set of registers for temporary storage of instructions, data and address of data .
2. Arithmetic and Logic Unit : Hardware for performing primitive arithmetic and logical operations .
3. Interface Section : Input and output lines through which the microprocessor communicates with the outside world .
4. Timing and Control Section : Hardware for coordinating and controlling the activities of the various sections within the microprocessor and other devices connected to the interface section .

The block diagram of the microprocessor along with the memory and Input/Output (I/O) devices is shown in the Figure 11.1.

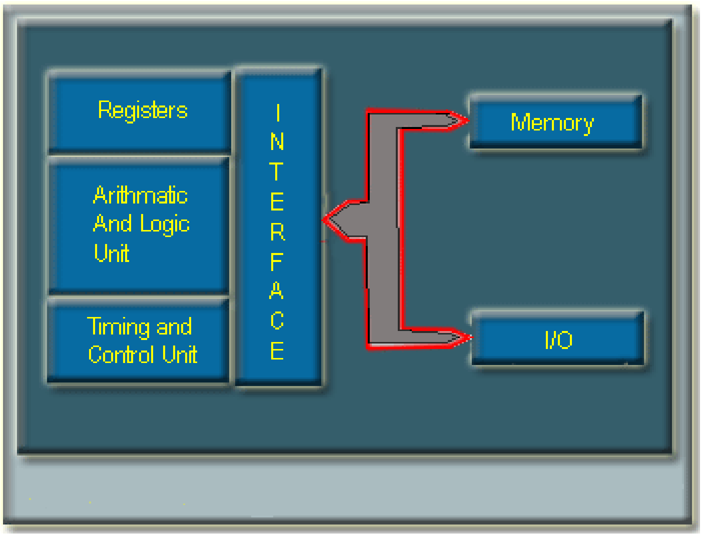


Figure 11.1: Block diagram of Micorprocessor with memory and I/O.

Intel Microprocessors:

Intel 4004 is the first 4-bit microprocessor introduced by Intel in 1971. After that Intel introduced its first 8-bit microprocessor 8088 in 1972.

These microprocessors could not last long as general-purpose microprocessors due to their design and performance limitations.

In 1974, Intel introduced the first general purpose 8-bit microprocessor 8080 and this is the first step of Intel towards the development of advanced microprocessor.

After 8080, Intel launched microprocessor 8085 with a few more features added to its architecture, and it is considered to be the first functionally complete microprocessor.

The main limitations of the 8-bit microprocessors were their low speed, low memory capacity, limited number of general purpose registers and a less powerful instruction set .

To overcome these limitations Intel moves from 8-bit microprocessor to 16-bit microprocessor.

In the family of 16-bit microprocessors, Intel's 8086 was the first one introduced in 1978 .

8086 microprocessor has a much powerful instruction set along with the architectural developments, which imparted substantial programming flexibility and improvement over the 8-bit microprocessor.

Microprocessor Intel 8085 :

Intel 8085 is the first popular microprocessor used by many vendors. Due to its simple architecture and organization, it is easy to understand the working principle of a microprocessor.

Register in the Intel 8085:

The programmable registers of 8085 are as follows -

- One 8-bit accumulator A.
- Six 8-bit general purpose register (GPR’s)
  B, C, D , E , H and L.
- The GPR’s are also accessible as three 16-bit register pairs BC, DE and HL.
  - There is a 16-bit program counter(PC), one 16-bit stack pointer(SP) and 8-bit flag register . Out of 8 bits of the flag register , only 5 bits are in use.

The programmable registers of the 8085 are shown in the Figure 11.2-

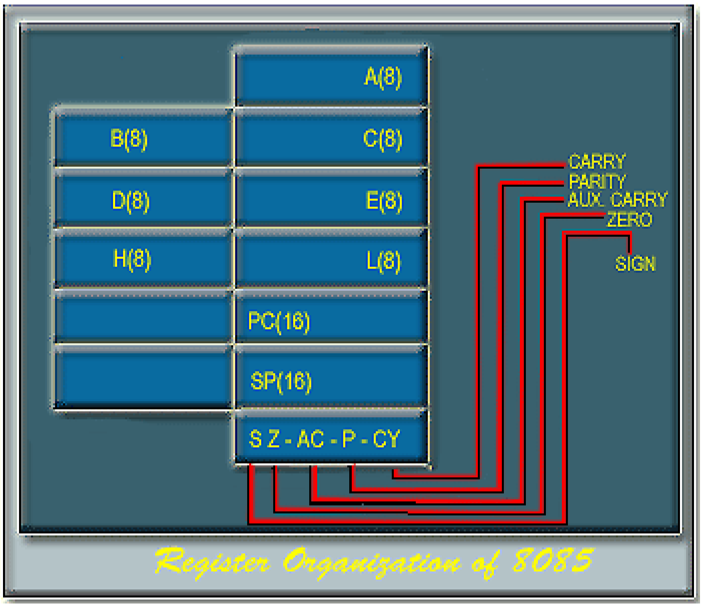


Figure 11.2: Register Organisation of 8085

Apart from these programmable registers , some other registers are also available which are not accessible to the programmer . These registers include -

- Instruction Register(IR).
- Memory address and data buffers(MAR & MDR).
    - MAR: Memory Address Register.
    - MDR: Memory Data Register.
- Temporary register for ALU use.

ALU of 8085 :

The 8-bit parallel ALU of 8085 is capable of performing the following operations –

Arithmetic : Addition, Subtraction, Increment, Decrement, Compare.

Logical : AND, OR, EXOR, NOT, SHIFT / ROTATE, CLEAR.

Because of limited chip area , complex operations like multiplication, division, etc are not available, in earlier processors like 8085.

The operations performed on binary 2's complement data.

The five flag bits give the status of the microprocessor after an ALU operation.

The carry (C) flag bit indicates whether there is any overflow from the MSB.

The parity (P) flag bit is set if the parity of the accumulater is even.

The Auxiliary Carry (AC) flag bit indicates overflow out of bit –3 (lower nibble) in the same manner, as the C-flag indicates the overflow out of the bit-7.

The Zero (Z) flag bit is set if the content of the accumulator after any ALU operations is zero.

The Sign(S) flag bit is set to the condition of bit-7 of the accumulator as per the sign of the contents of the accumulator(positive or negative).

The Interface Section:

Microprocessor chips are equipped with a number of pins for communication with the outside world. This is known as the system bus.
The interface lines of the Intel 8085 microprocessor are shown in the Figure 11.3 –

Address and Data Bus

The AD0 - AD7 lines are used as lower order 8-bit address bus and data bus , in time division multiplexed manner .

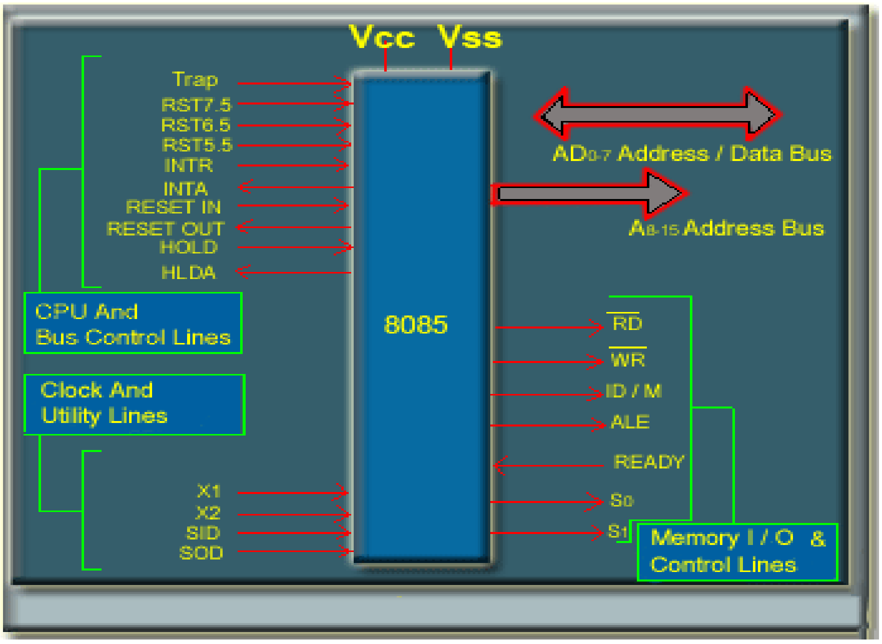
The A8 - A15 lines are used for higher order 8 bit of address bus.

There are seven memory and I/O control lines -

RD : indicates a READ operation when the signal is LOW .

WR : indicates a WRITE operation when the signal is LOW .

IO/M : indicates memory access for LOW and I/O access for HIGH .



ALE : ALE is an address latch enable signal , this signal is HIGH when address information is present in AD0-AD7 . The falling edge of ALU can be used to latch the address into an external buffer to de-multiples the address bus .

READY : READY line is used for communication with slow memory and I/O devices .

S0 and S1 : The status of the system bus is difined by the S0 and S1 lines as follows -

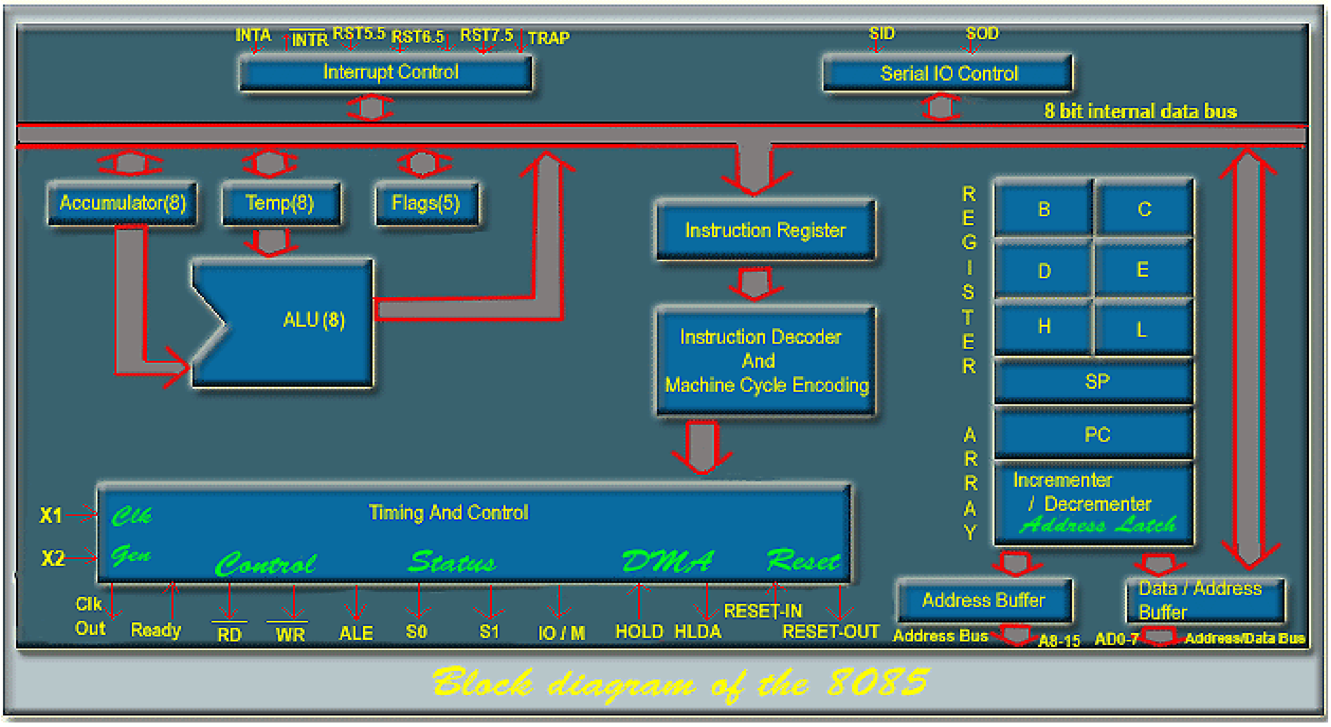
S1	S0	Operation Specified
0	0	Halt
0	1	Memory or I/O WRITE
1	0	Memory or I/O READ
1	1	Instruction Fetch

There are ten lines associated with CPU and bus control-

  - TRAP , RST7.5 , RST6.5 , RST5.5 and INTR are the Interrupt lines.
  - INTA: Interrupt acknowledge line.
  - RESET IN : This is the reset input signal to the 8085.
  - RESET OUT : The 8085 generates the RESET-OUT signal in response to RESET-IN signal , which can be used as a system reset signal .
  - HOLD : HOLD signal is used for DMA request.
  - HLDA : HLDA signal is used for DMA grant .
- Clock and Utility Lines :

X1 and X2: X1 and X2 are provided to connect a crystal or a RC network for generating theclockinternaltothe chip.
Sid: input line for serial data communication.
Sod: output line for serial data communication.
V_{cc} and v_{ss}: power supply.

- The block diagram of the Intel 8085 is shown in the Figure 11.4 -



Addressing Modes :

The 8085 has four different modes for addressing data stored in memory or in registers -

Direct: Bytes 2 and 3 of the instruction contains the exact memory address of the data item(the low-order bits of the address are in byte 2 , the high-order bits in byte 3).

Register: The instruction specifies the register or register pair in which the data are located.

Register Indirect: The instruction specifies a register pair which contains the memory address where the data are located .(the high-order bits of the address are in the first register of the pair and the low order bits in the second).

Immediate: The instruction contains the data itself . This is either and 8-bit quantity or a 16-bit quantity (least significant byte first , most significant byte second).

Unless directed by an interrupt or branch instruction the execution of instructions proceeds through consecutively increasing memory locations.

A branch instruction can specify the address of the next instruction to be executed in one of two ways -

Direct: The branch instruction contains the address of the next instruction to be executed .

REFERENCE :

1. COMPUTER SYSTEM ARCHITECTURE, MORRIS M. MANO, 3RD EDITION, PRENTICE HALL INDIA.
2. HTTP://NPTEL.AC.IN/COURSES