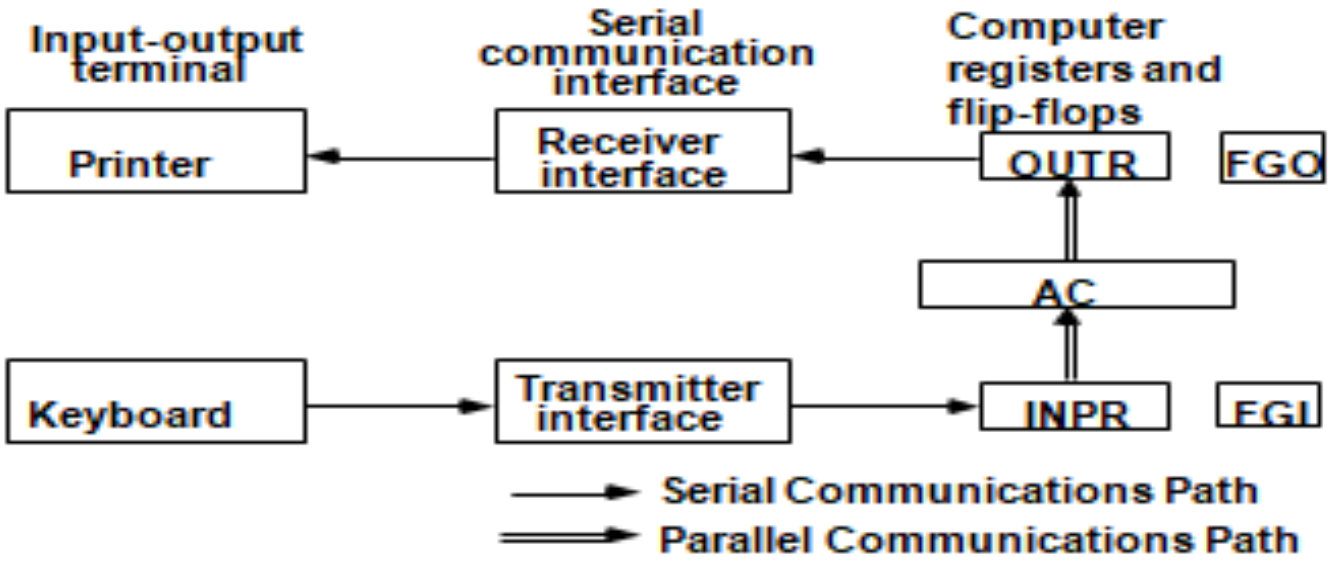
Overview

- **Peripheral Devices**
- **Input-Output Interface**
- Asynchronous Data Transfer
- Modes of Transfer
- Priority Interrupt
- Direct Memory Access
- Input-Output Processor
- Serial Communication


Input-output terminal
Serial communication interface
Computer registers and flip-flops
Printer
Receiver interface
OUTR
FGO
AC
Keyboard
Transmitter interface
INPR
FGI
Serial Communications Path
Parallel Communications Path

Input Output Organization

- **I/O Subsystem**
  - Provides an efficient mode of communication between the central system and the outside environment
- Programs and data must be entered into computer memory for processing and results obtained from computer must be recorded and displayed to user.
- When input transferred via slow keyboard processor will be idle most of the time waiting for information to arrive
  - Magnetic tapes, disks

Peripheral Devices

- **Devices that are under direct control of computer are said to be connected on-line.**
- **Input or output devices attached to the computer are also called peripherals.**
- **There are three types of peripherals :**
  - **Input peripherals**
  - **Output peripherals**
  - **Input-output peripherals**

Peripheral (or I/O Device)

Monitor (*Visual Output Device*) : CRT, LCD,LED
KeyBoard (*Input Device*) : light pen, mouse, touch screen, joy stick, digitizer
Printer (Hard Copy Device) : **Daisy wheel**, **dot matrix** and **laser printer**
Storage Device : Magnetic tape, magnetic disk, optical disk

Peripheral Devices

Input Devices

- Keyboard
- Optical input devices
  - Card Reader
  - Paper Tape Reader
  - Bar code reader
  - 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,LCD,LED
- Printer (Daisy Wheel, Dot Matrix, Laser)
- Plotter

Storage devices

- magnetic tapes(audio tape ,video tape)
- magnetic disc (HDD,FDD,BLU-RAY DISC)
- optical disc(CD,DVD)

Input Output Organization

➢**ASCII (*American Standard Code for Information Interchange*)**

- I/O communications usually involves transfer of alphanumeric information from the device and the computer.
- Standard binary code for alphanumeric character is **ASCII**
- ASCII Code :
  - It uses 7 bits to code 128 characters (94 printable and 34 non printing)
  - 7 bit - 00 - 7F (*0 - 127*)
- ASCII is 7 bits but most computers manipulate 8 bit quantity as a single unit called byte.

  80 - FF (128 - 255) : Greek, Italic type font
- Three types of control characters: Format effectors, Information separators and communication control

TABLE 11-1 American Standard Code for Information Interchange (ASCII)

$b_4 b_3 b_2 b_1$	$b_7 b_6 b_5$ = 000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	−	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	—	o	DEL

Control characters

NUL	Null
SOH	Start of heading
STX	Start of text
ETX	End of text
EOT	End of transmission
ENQ	Enquiry
ACK	Acknowledge
BEL	Bell
BS	Backspace
HT	Horizontal tab
LF	Line feed
VT	Vertical tab
FF	Form feed
CR	Carriage return
SO	Shift out
SI	Shift in
SP	Space
DLE	Data link escape
DC1	Device control 1
DC2	Device control 2
DC3	Device control 3
DC4	Device control 4
NAK	Negative acknowledge
SYN	Synchronous idle
ETB	End of transmission block
CAN	Cancel
EM	End of medium
SUB	Substitute
ESC	Escape
FS	File separator
GS	Group separator
RS	Record separator
US	Unit separator
DEL	Delete

- **Format Effectors**: control the layout of printing. They include familiar typewriter controls, such as backspace (BS), horizontal tabulation(HT), carriage return(CR).
- **Information separators**: used to separate data into divisions like paragraphs and pages. They include characters such as record separator (RS) and file separator(FS).
- **Communication Control characters**: these are useful during the transmission of text between remote terminals. These include STX(Start of text) and ETX(end of text)

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**

(1). Peripherals – Electromechanical or Electromagnetic Devices

CPU or Memory - Electronic Device

- **Conversion of signal values required**

(2). Data Transfer Rate

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

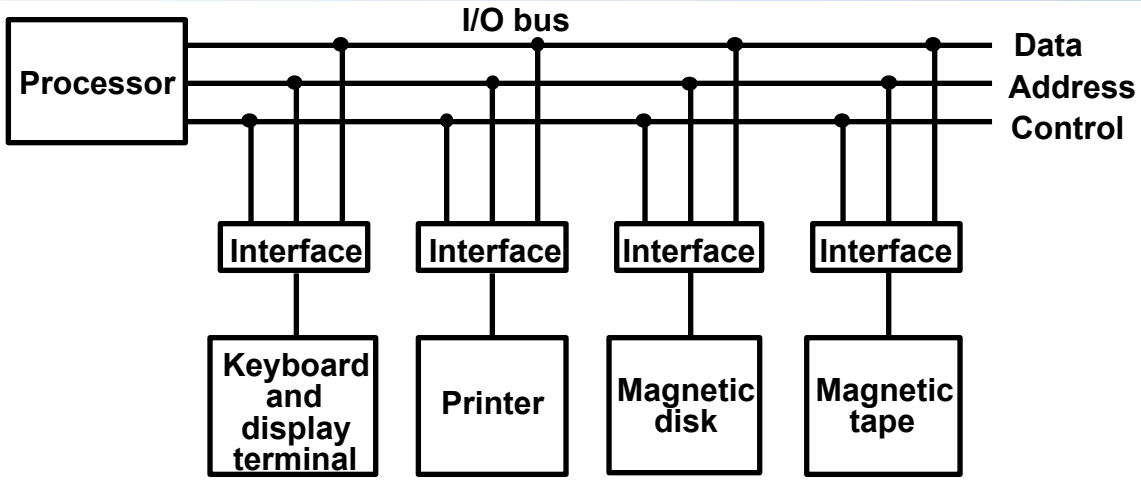
(3). Data formats or Unit of Information

- **Peripherals – Byte, Block, …**
- **CPU or Memory – Word**

(4). Operating modes of peripherals may differ

- **must be controlled so that not to disturbed other peripherals connected to CPU**

I/O Bus and Interface



Interface :

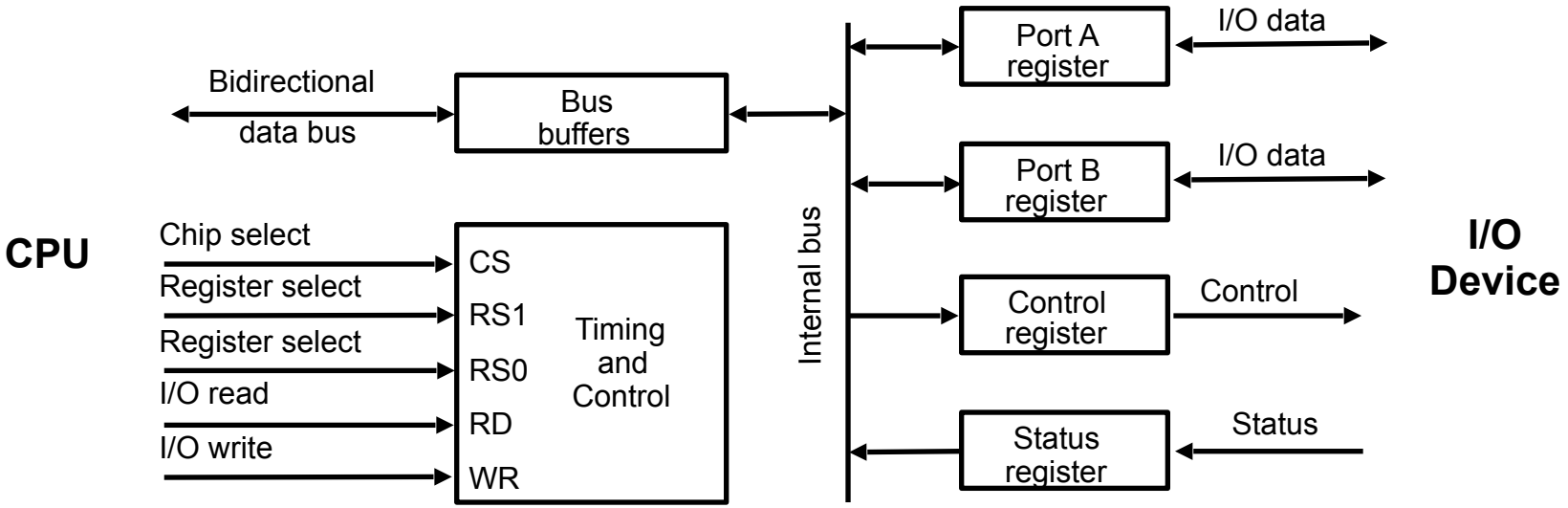
- **Decodes the device address (device code)**
- **Decodes the commands (operation)**
- **Provides signals for the peripheral controller**
- **Synchronizes the data flow and supervises the transfer rate between peripheral and CPU or Memory**

4 types of command interface can receive : control, status, data o/p and data i/p

I/O Bus and Interface

- Control command : is issued to activate peripheral and to inform what to do
- Status command : used to test various status condition in the interface and the peripherals
- data o/p command : causes the interface to respond by transferring data from the bus into one of its registers
- data i/p command : interface receives an item of data from the peripheral and places it in its buffer register.

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;

- 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

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**

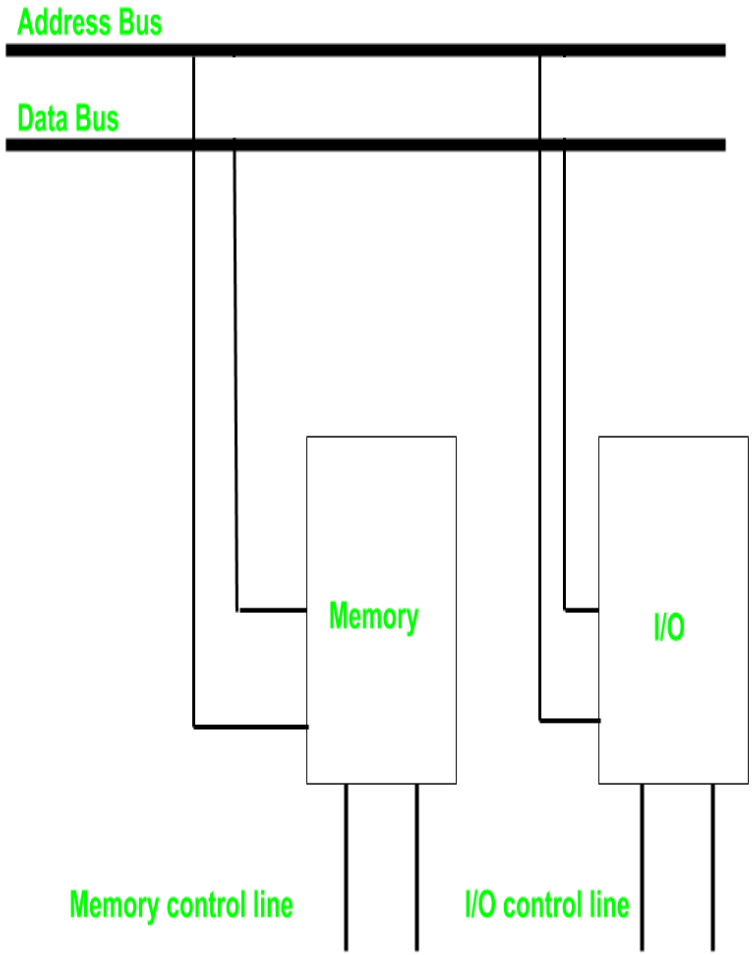
•3 ways to bus can communicate with memory and I/O :

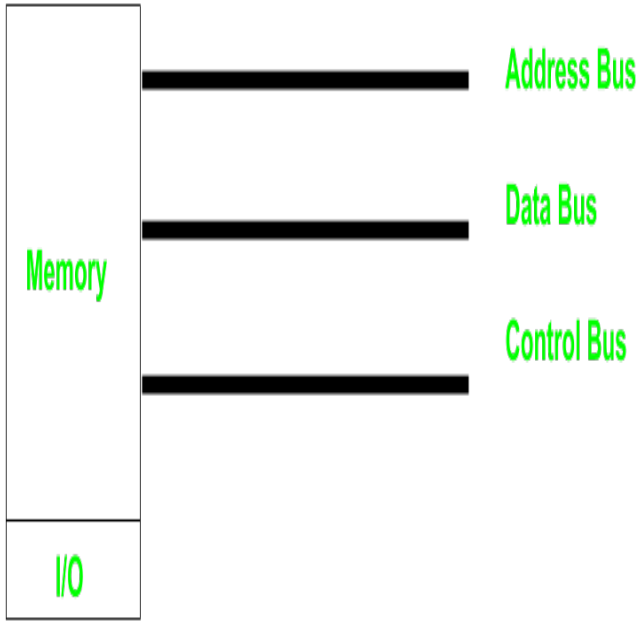
(1). use two separate buses, one to communicate with memory and the other with I/O interfaces

- **Computer has independent set of data, address and control bus one for accessing memory and another I/O.**
- **done in computers that have separate IOP other than CPU.**

(2). Use one common bus for memory and I/O but separate control lines for each

(3). Use one common bus for memory and I/O with common control lines for both


Address Bus
Data Bus
Memory
I/O
Memory control line
I/O control line


Memory
I/O
Address Bus
Data Bus
Control Bus

Isolated vs. Memory Mapped I/O

Isolated I/O

- Many computers use common bus to transfer information between memory or I/O.

- The distinction between memory transfer and I/O transfer is made through separate read and write line.

-In the isolated I/O configuration , the CPU has distinct input and output instructions and each of these instruction is associated with the address of an interface register.

- Distinct input and output instructions -each associated with address of interface register

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

- Peripheral Devices
- Input-Output Interface
- **Asynchronous Data Transfer**
- Modes of Transfer
- Priority Interrupt
- Direct Memory Access
- Input-Output Processor

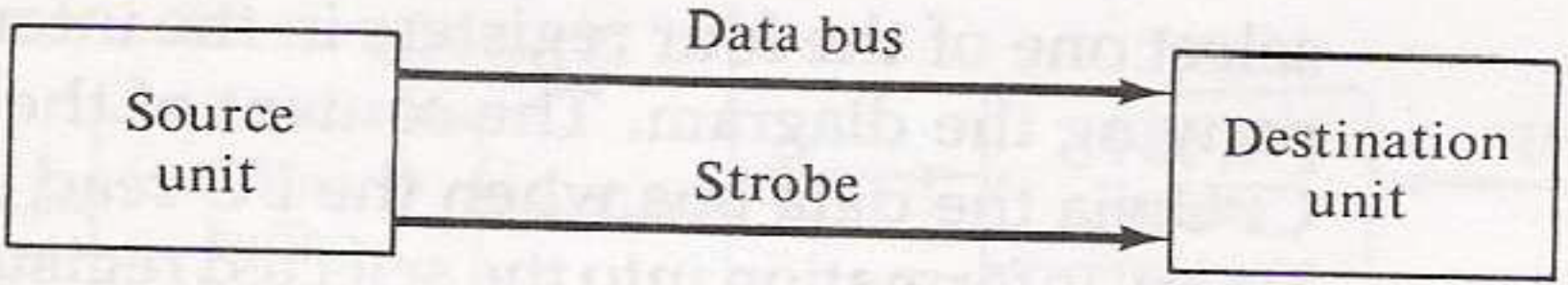
11-3. Asynchronous Data Transfer

→**Synchronous Data Transfer**:. Two units such as CPU and I/O Interface are designed independently of each other. If the registers in the **interface** share a common clock with **CPU** registers, the transfer between the two is said to be **synchronous.**

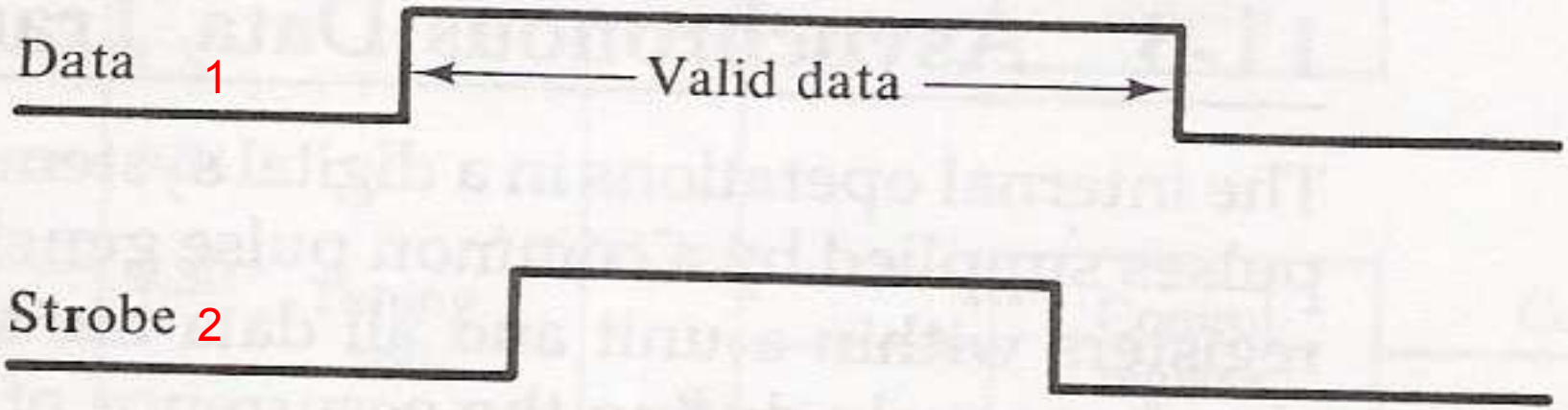
→**Asynchronous Data Transfer:** Internal timing in each unit (*CPU and Interface*) is independent. Each unit uses its own private clock for internal registers.

One way of achieving this is by means of

- **STROBE(**Control signal to indicate the time at which data is being transmitted) pulse and other method is
- **HANDSHAKING(**Agreement between two independent units).



(a) Block diagram



(b) Timing diagram

Figure 11-3 Source-initiated strobe for data transfer.

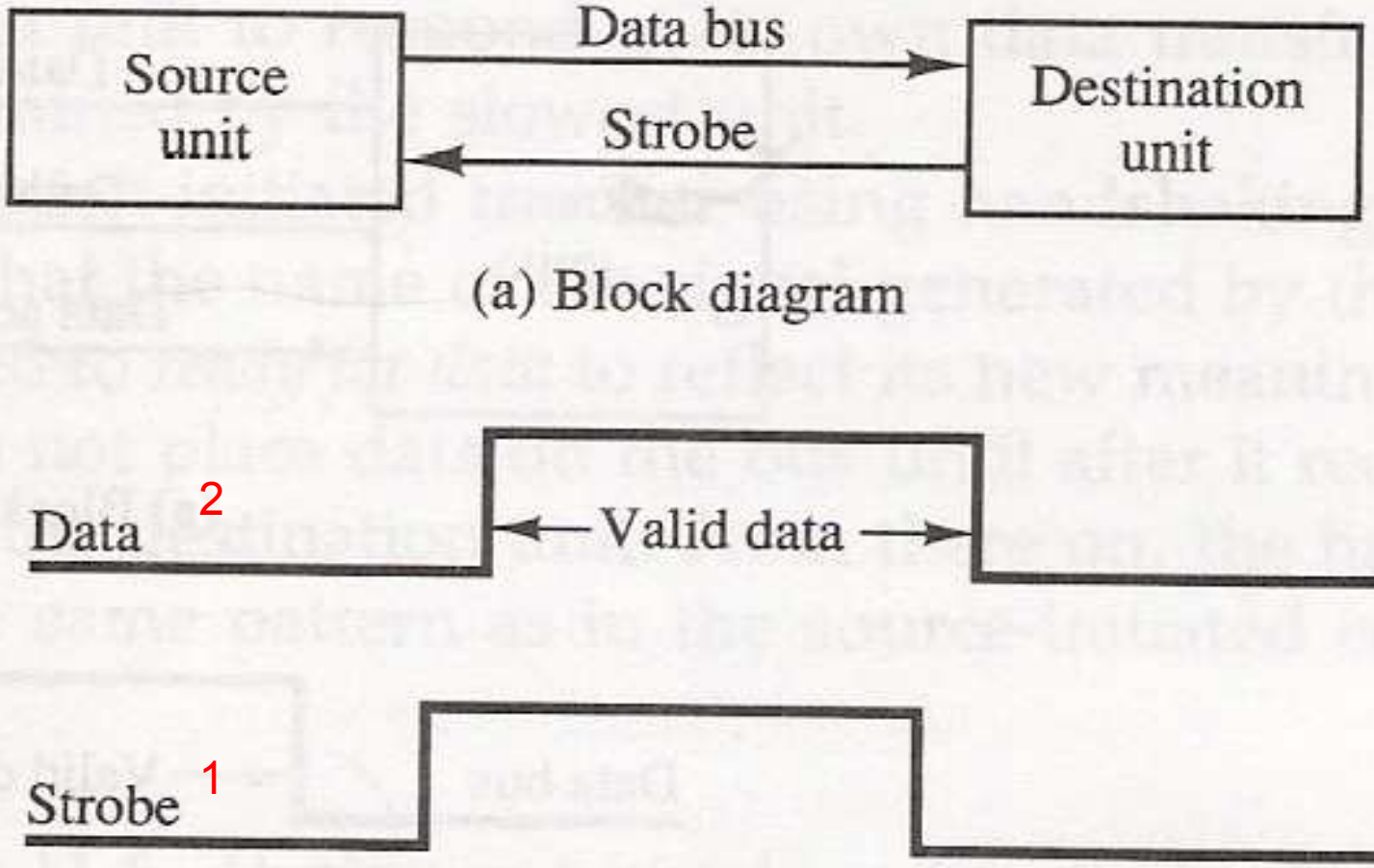
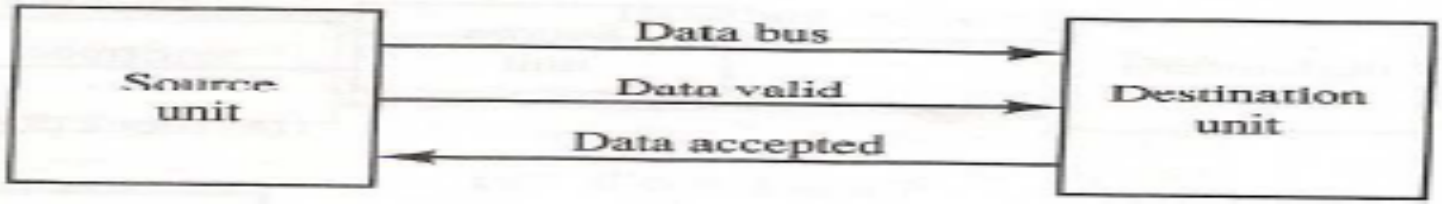
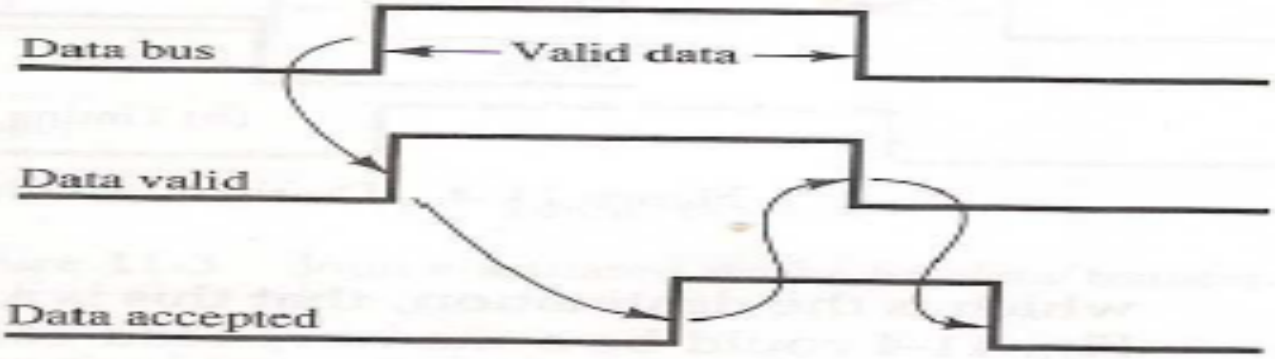


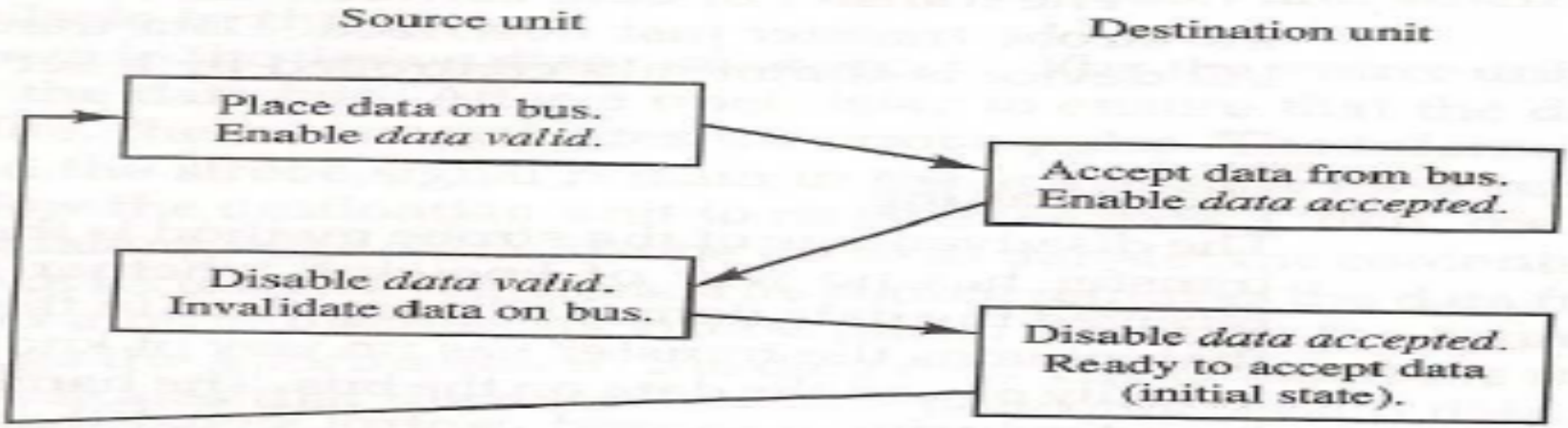
Figure 11-4 Destination-initiated strobe for data transfer.



(a) Block diagram



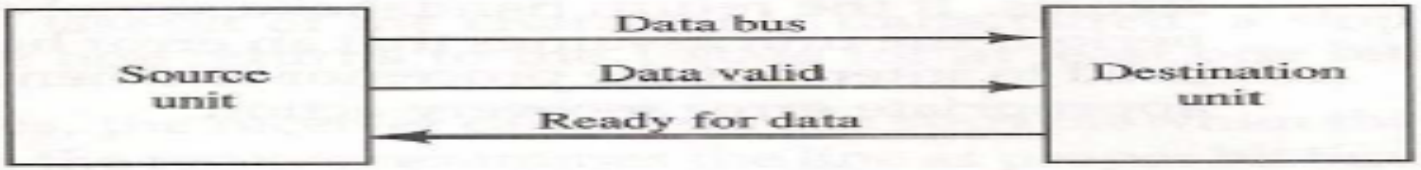
(b) Timing diagram



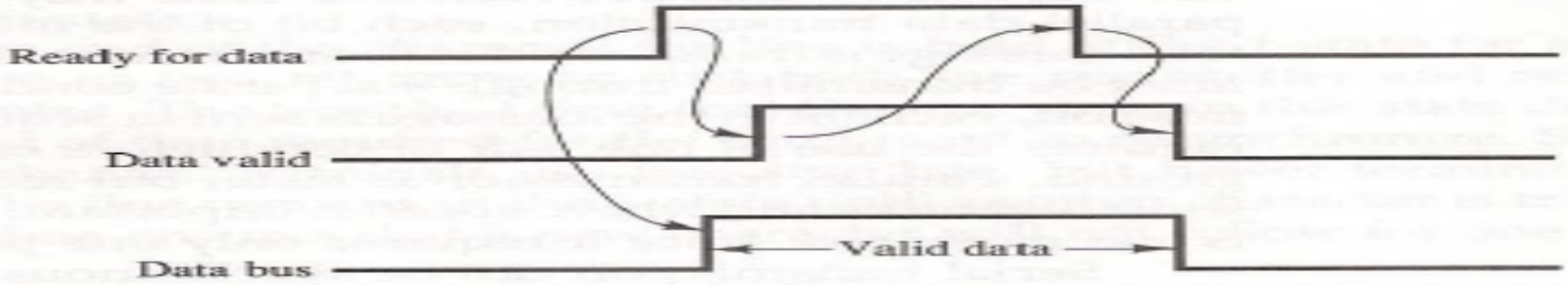
(c) Sequence of events

Figure 11-5 Source-initiated transfer using handshaking.

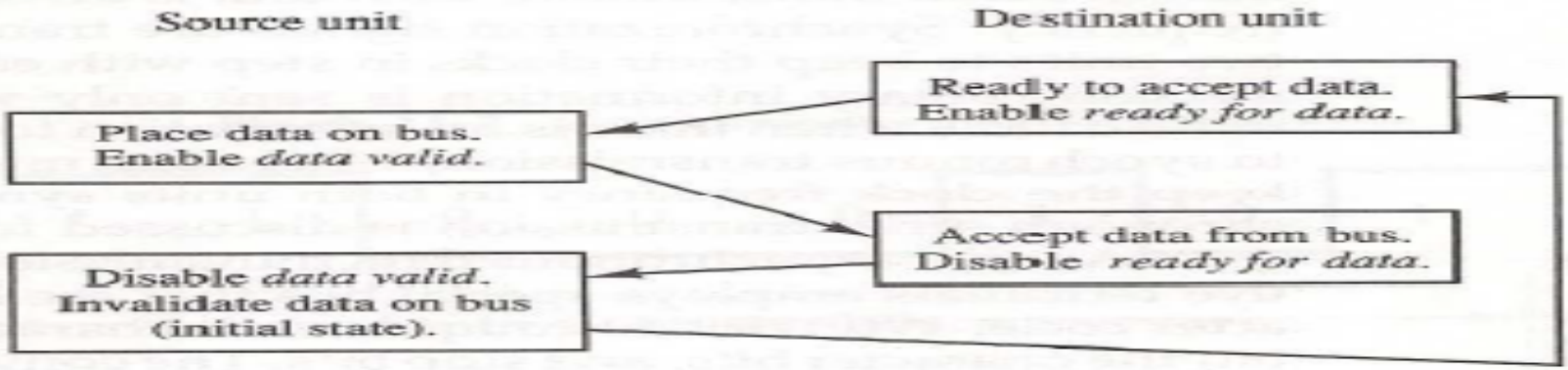
Figure 11-6 Destination-initiated transfer using handshaking.



(a) Block diagram



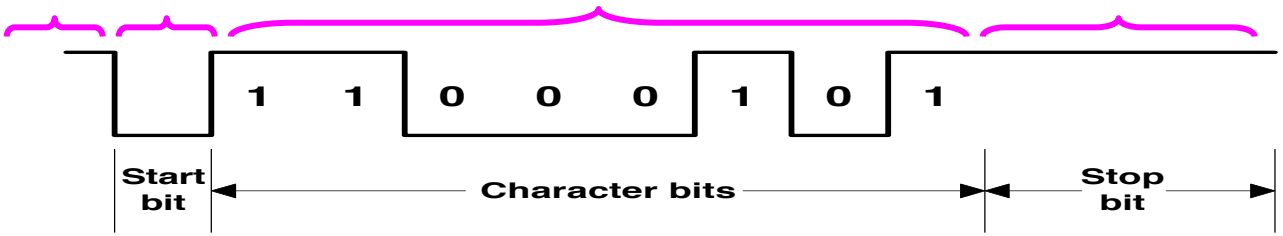
(b) Timing diagram



(c) Sequence of events

Timeout : If the return handshake signal does not respond within a given time period, the unit assumes that an error has occurred.

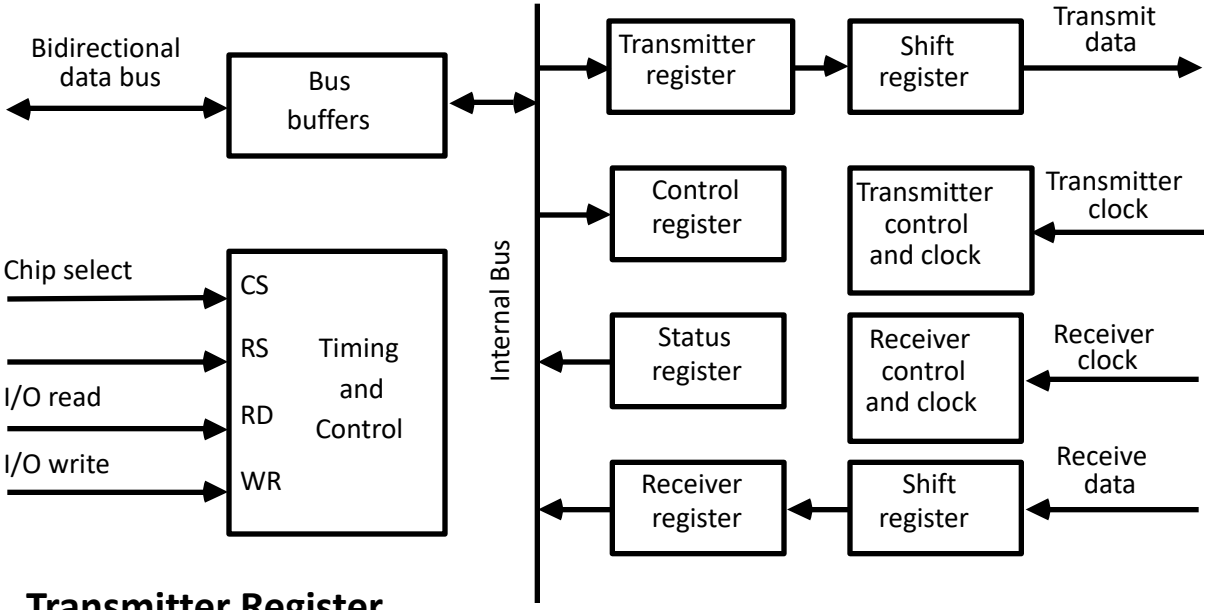
- Asynchronous Serial Transfer
  - Synchronous transmission :
    - The two unit share a common clock frequency
    - Bits are transmitted continuously at the rate dictated by the clock pulses
  - Asynchronous transmission :
    - Binary information sent only when it is available and line remain idle otherwise
    - Special bits are inserted at both ends of the character code
    - Each character consists of three parts :
      - 1) start bit : always “0”, indicate the beginning of a character
      - 2) character bits : data
      - 3) stop bit : always “1”



- Asynchronous transmission rules :
  - ① When a character is not being sent, the line is kept in the 1-state
  - ② The initiation of a character transmission is detected from the start bit, which is always “0”
  - ③ The character bits always follow the start bit
  - ④ After the last bit of the character is transmitted, a stop bit is detected when the line returns to the 1-state for at least one bit time
- Baud Rate : Signals transfer per second
  - 10 character per second with 11 bit format = 110 bit per second

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 Register

- 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

- UART stands for?

a. Universal Asynchronous Sender Receiver
b. Universal Asynchronous Receiver Transmitter
c. Universal Synchronous Receiver Transmitter
d. None Of These.

Overview

- Peripheral Devices
- Input-Output Interface
- Asynchronous Data Transfer
- **Modes of Transfer**
- **Priority Interrupt**
- Direct Memory Access
- Input-Output Processor

Modes of Transfer

➢ Binary information received from external device is usually stored in memory.

➢ Information transferred from central computer into an external device originates in the memory unit.

➢The CPU merely execute I/O instructions and may accept data temporarily but ultimate source or destination is the Memory Unit.

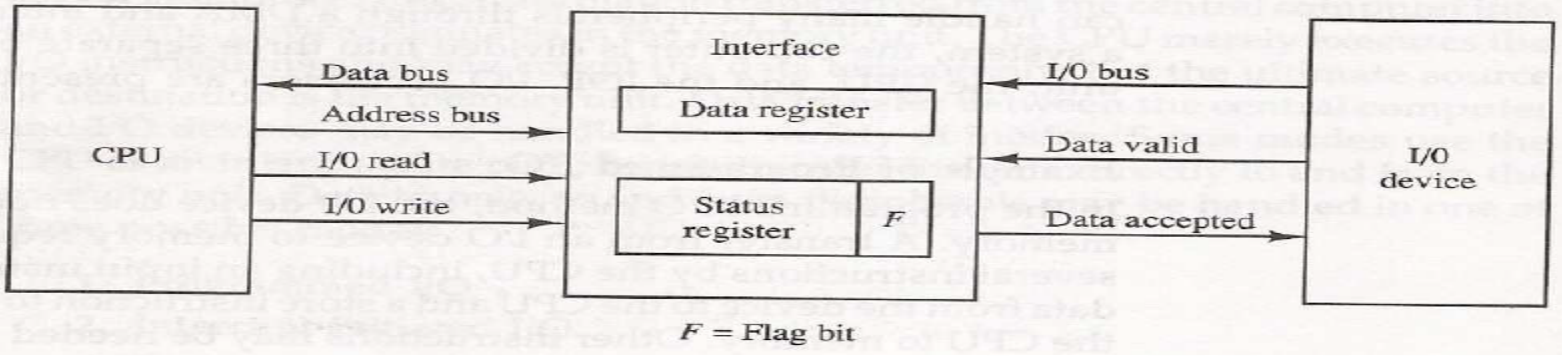
➢Data transfer between central computer and I/O devices may be handled in a variety of modes. Some modes use CPU as intermediate path and others transfer data directly to and from memory unit.

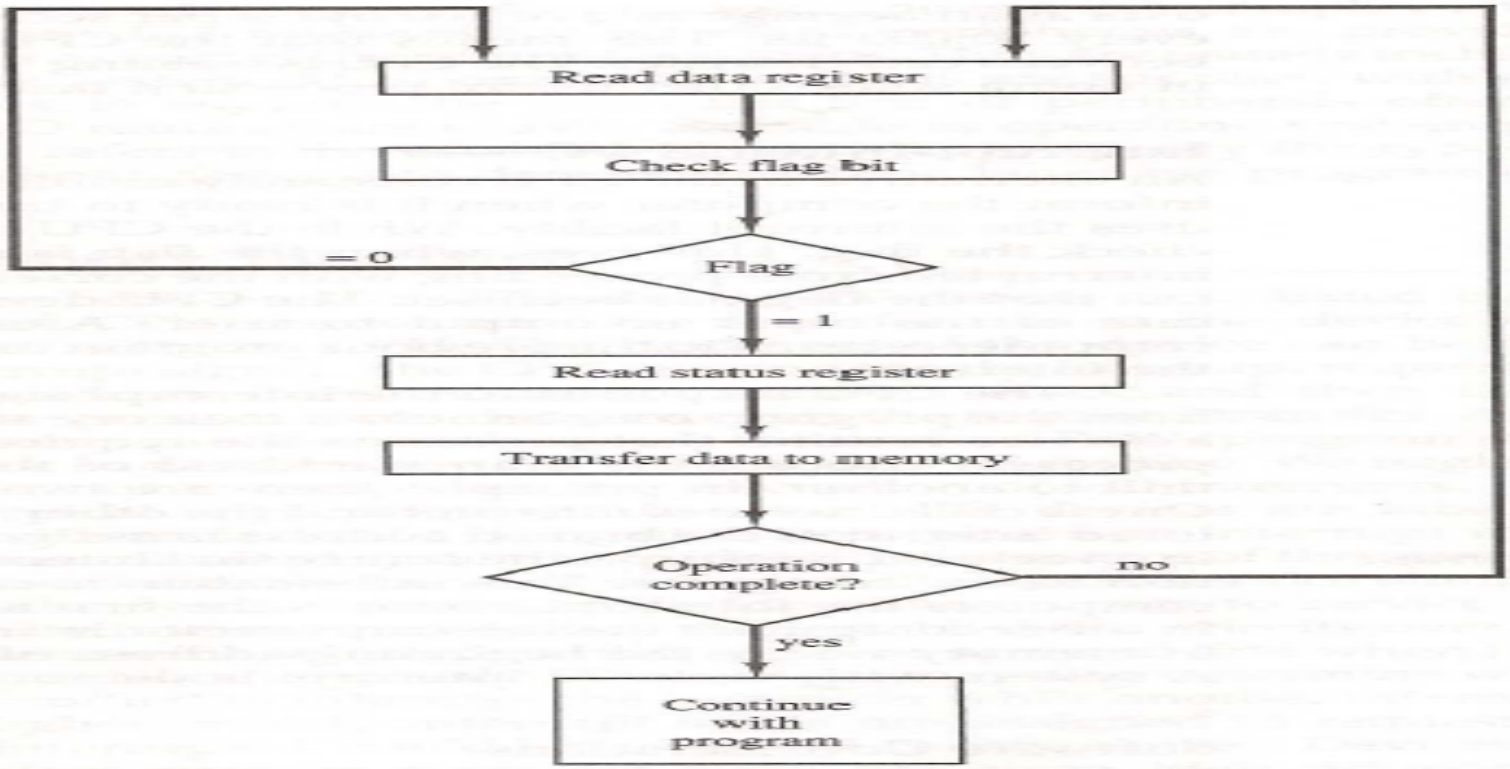
➢Data Transfer to or from peripheral can be handled in one of three possible modes :

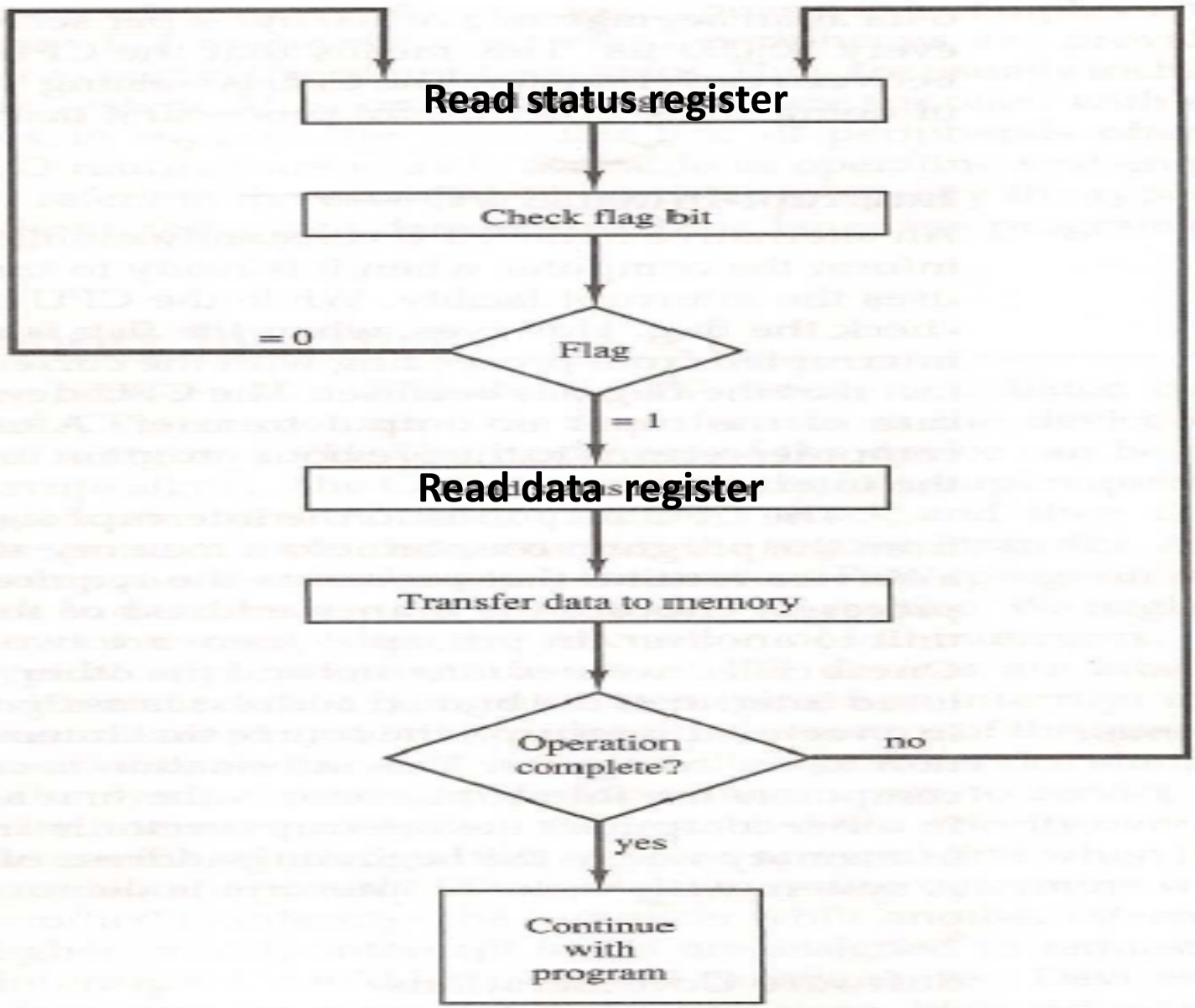
- ➢ **Programmed I/O**
- ➢ **Interrupt-Initiated I/O**
- ➢ **Direct Memory Access (DMA)**

Modes of Transfer – Programmed I/O

Figure 11-10 Data transfer from I/O device to CPU.






Read status register
Check flag bit
Flag
= 0
= 1
Read data register
Transfer data to memory
Operation complete?
no
yes
Continue with program

Programmed I/O

-Programmed I/O operations are the result of I/O Instructions written in computer program.

-Each data item transfer is initiated by an instruction in the program.

- Usually, transfer is to and from a CPU register to peripheral. Other instructions are needed to transfer data to and from CPU and Memory
- Transferring data under program control requires constant monitoring of the peripheral by CPU.

- In programmed I/O method, CPU stays in a program loop until the I/O unit indicated that it is ready for data transfer.
- This is a time consuming process since it keeps the processor busy needlessly.
- It can be avoided by using **Interrupt** facility and special commands to inform the interface to issue an interrupt request signal when data are available for the device.

Interrupted I/O

In the programmed I/O method, 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 since it keeps the processor busy needlessly. It can be avoided by using an interrupt facility and special commands to inform the interface to issue an interrupt request signal when the data are available from the device. In the meantime the CPU can proceed to execute another program. The interface meanwhile keeps monitoring the device. When the interface determines that the device is ready for data transfer, it generates an interrupt request to the computer. Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to a service program to process the I/O transfer, and then returns to the task it was originally performing.

DMA

Transfer of data under programmed I/O is between CPU and peripheral. In direct memory access (DMA), the interface transfers data into and out of the memory unit through the memory bus. The CPU initiates the transfer by supplying the interface with the starting address and the number of words needed to be transferred and then proceeds to execute other tasks. When the transfer is made, the DMA requests memory cycles through the memory bus. When the request is granted by the memory controller, the DMA transfers the data directly into memory. The CPU merely delays its memory access operation to allow the direct memory I/O transfer. Since peripheral speed is usually slower than processor speed, I/O-memory transfers are infrequent compared to processor access to memory. DMA transfer is discussed in more detail in

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

A priority interrupt is a system that establishes priority over the various sources to determine

- **which condition is to serviced first when two or more requests arrive simultaneously**
- **which conditions are permitted to interrupt the computer while another request is being serviced**

Priority Interrupts

Priority Interrupt by Software **(Polling)**

Polling procedure is used to identify highest priority source by software means

- common branch address for all the interrupts

- **Priority is established by the order of polling the devices(interrupt sources)**
  - highest priority device is tested first and if interrupt is on , control branches to service routine for this source otherwise next lower priority source is tested

- Flexible since it is established by software
- Low cost since it needs a very little hardware

- Very slow
- if there are many interrupt time required to poll may exceed time available to service IO device

Priority Interrupts

Priority Interrupt by Hardware

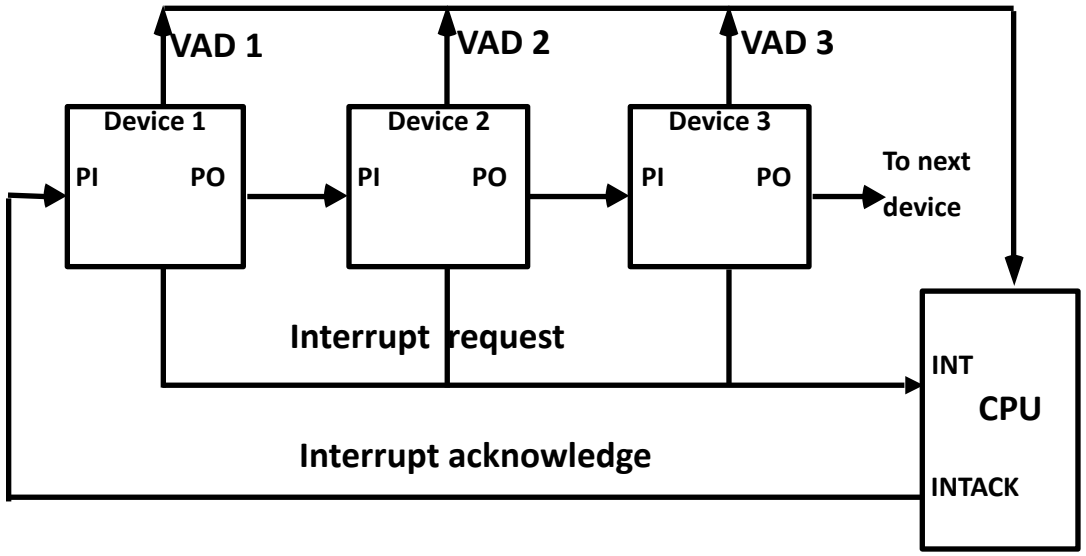
- Require a **priority interrupt manager** which accepts all the interrupt requests to determine the highest priority request

- 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

- Can be addressed using serial or parallel connection of interrupt lines. Example of serial is Daisy chaining Priority

Hardware Priority Interrupts – Daisy Chain

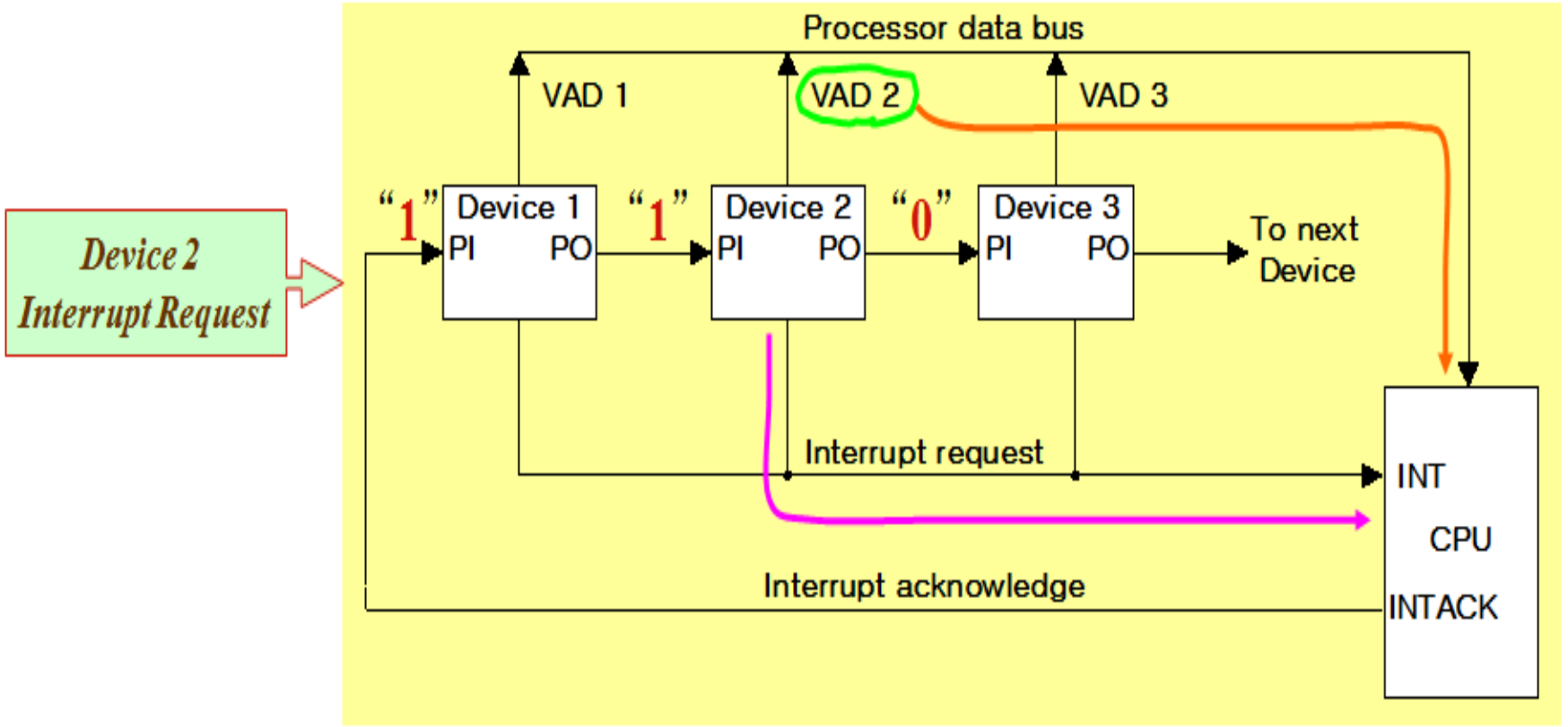


*** Serial hardware priority function**
*** Interrupt Request Line**
- Single common line
*** Interrupt Acknowledge Line**
- Daisy-Chain

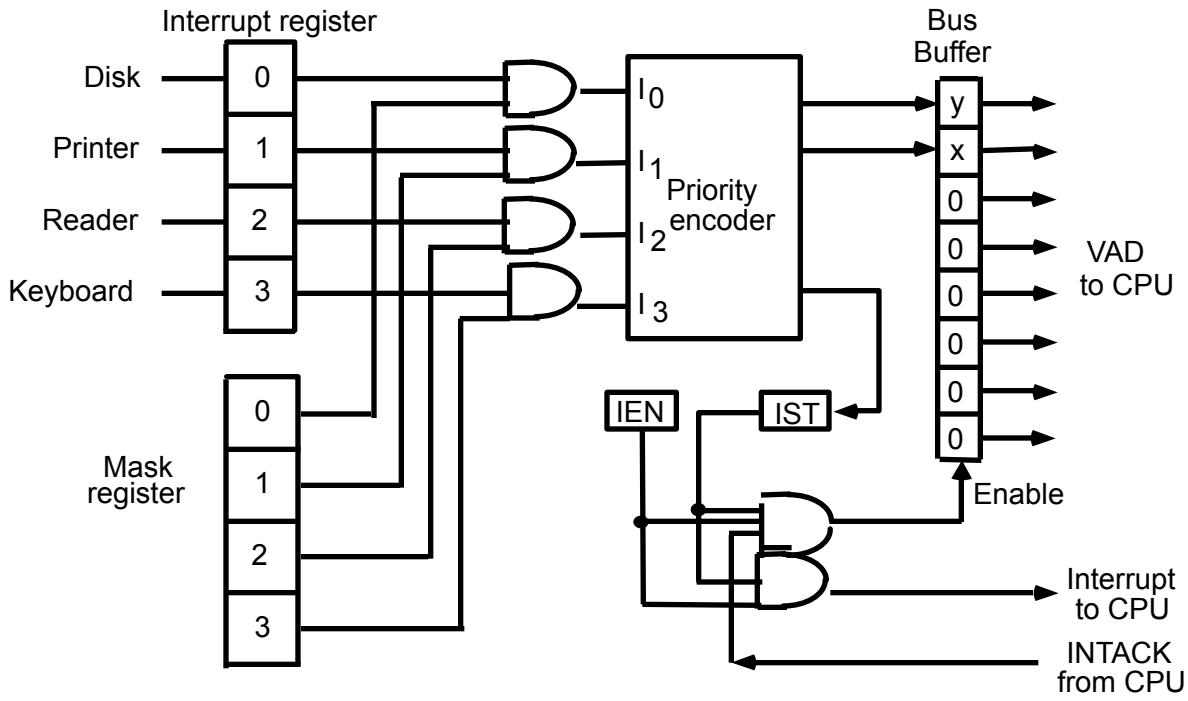
-Serial connection of all device that request an interrupt
-Device with highest priority placed in first position followed by devices with lower priority and so on.
-Interrupt generated by any device → signals low state interrupt line
-CPU responds by enabling interrupt acknowledgement (INTACK) line.
- device receives PI=1 and passes to next only when not requesting else PI=0
-Thus device with PI=1 and PO=0 is one with highest priority requesting interrupt

Hardware Priority Interrupts – Daisy Chain

Example: Daisy chain working



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**
- **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			Boolean functions
I_0	I_1	I_2	I_3	x	y	IST	
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

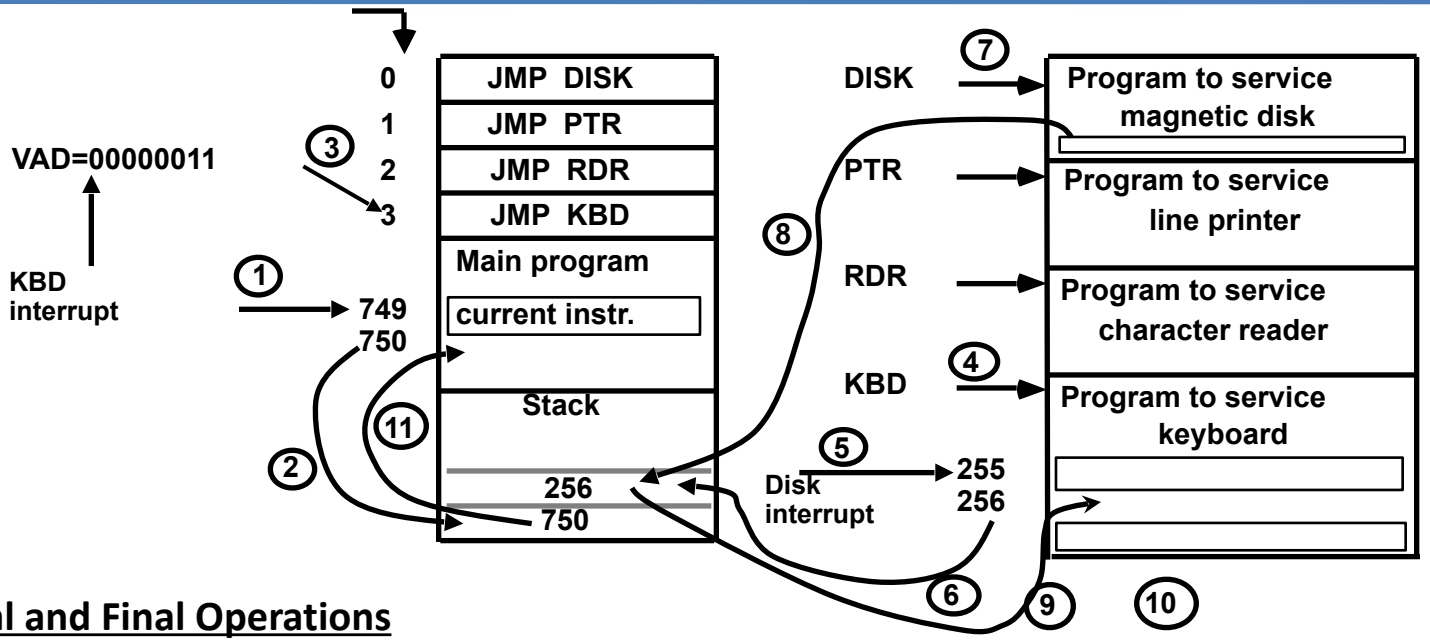
At the end of each Instruction cycle

- 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

In Daisy Chaining Priority if the device does not have any pending interrupt requests then the value of PI and PO will be

a. PI=0, PO=0
b. PI=0, PO=1
c. PI=1, PO=0
d. PI=1, PO=1

Overview

- Peripheral Devices
- Input-Output Interface
- Asynchronous Data Transfer
- Modes of Transfer
- Priority Interrupt
- **Direct Memory Access**
- **Input-Output Processor**

Direct Memory Access

* Block of data transfer between high speed devices like Disk and Memory
* DMA controller - Interface which takes over the buses to manage the transfer directly between Memory and I/O Device, freeing CPU for other tasks
* CPU initializes DMA Controller by sending memory address and the block size (number of words)

Fig 1: CPU bus signals for DMA transfer

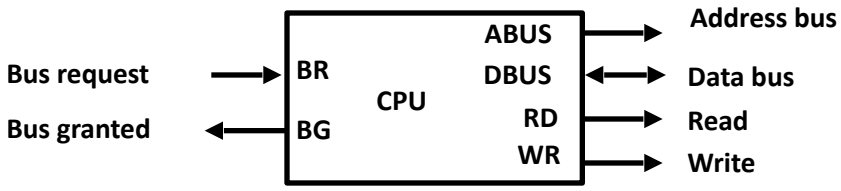
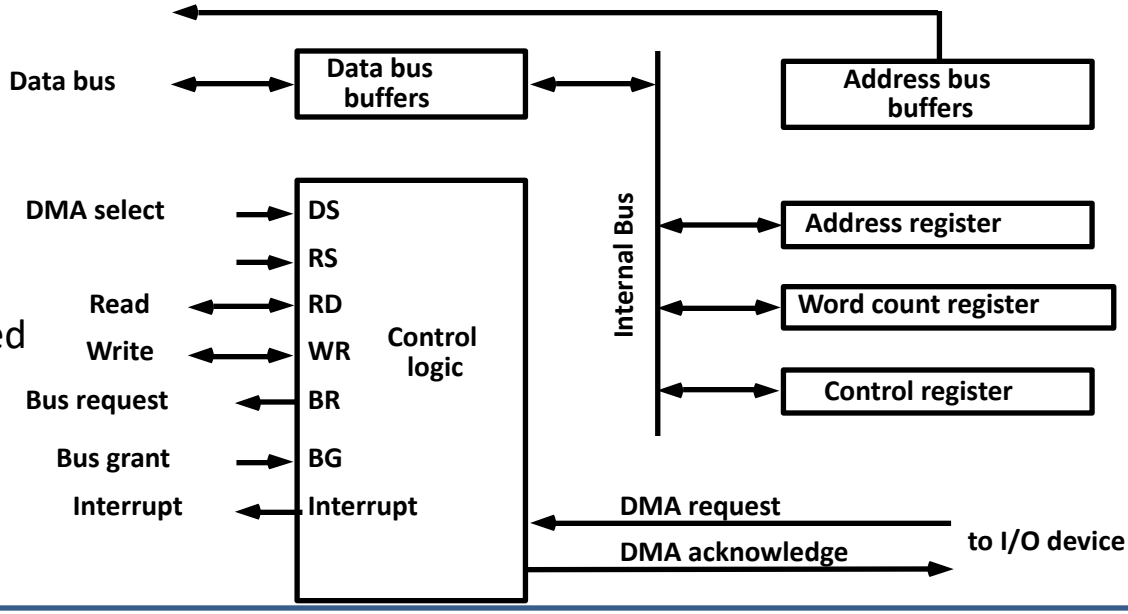


Fig 2: Block diagram of DMA controller



Address register:

Contains an address to specify Desired location in memory

Word count register

Holds no. of words to be transferred

Control register

Specifies the mode of transfer

DMA I/O Operation

DMA is first initialized by CPU. After that DMA starts and continues to transfer data between memory and peripheral unit until an entire block is transferred.

CPU initializes the DMA by sending following information through data bus:

(1) Starting address of the memory block (for read/write)

(2) Word Count (no. of words in memory block)

(3) Control to specify mode of transfer (E.g. read/write)

(4) A control to start DMA Transfer

Direct Memory Access

RD and WR is bidirectional

When BG=0 CPU can communicate with DMA Register
When BG=1 CPU left the buses and DMA can communicate directly with memory

DMA Transfer can be made in several ways

(1)Burst Transfer : a block sequence consisting of memory words is transferred in continuous burst while the DMA controller is master of memory bus

- This mode of transfer is needed for fast devices such as magnetic disk where data transmission cannot be stopped or slowed down until an entire block is transferred

(2) Cycle stealing : Alternative technique called cycle stealing allows DMA controller to transfer one data word at time after which it must return control of the buses to the CPU.

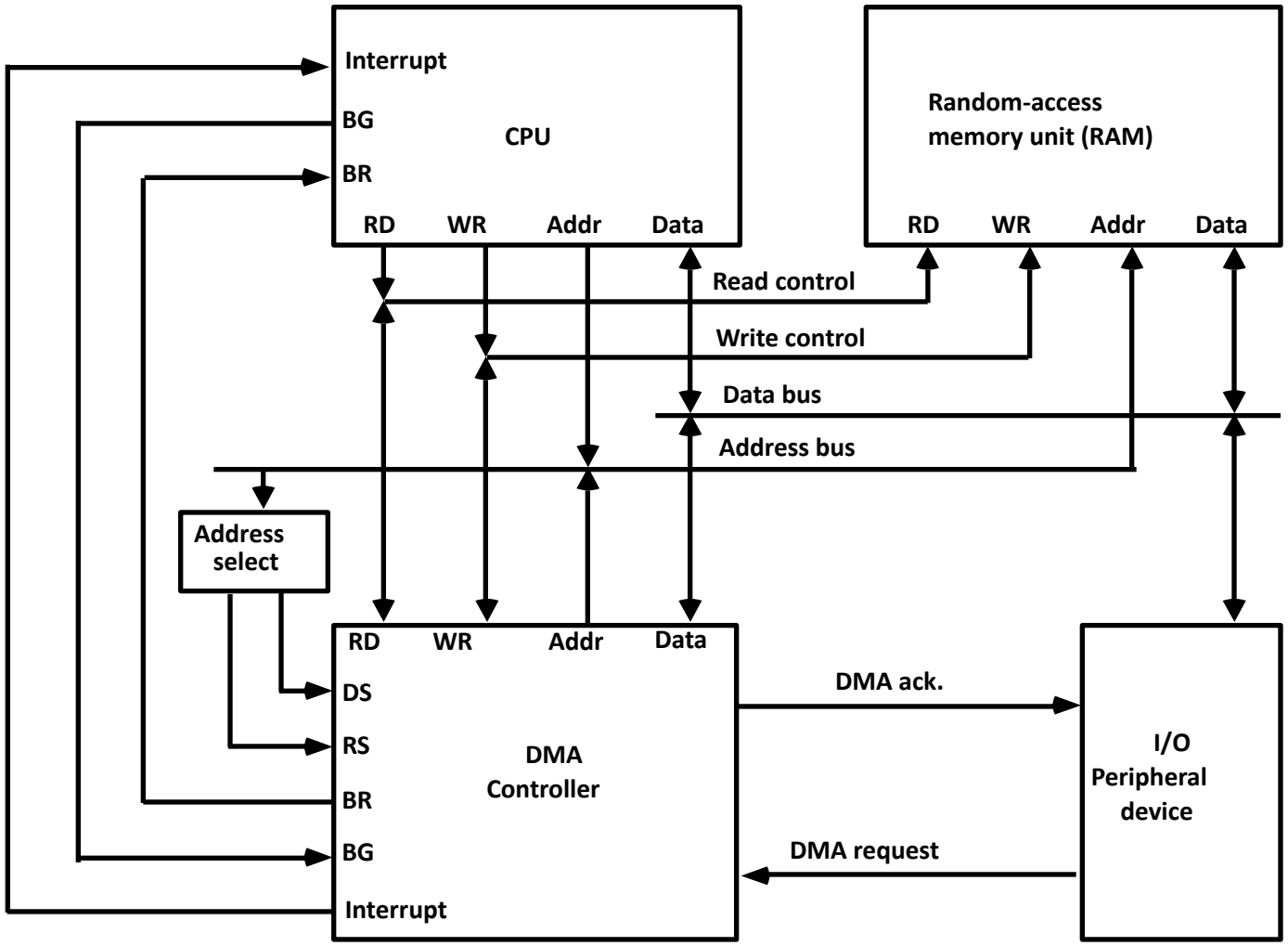
- CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to "steal" one memory cycle

Q—The technique where the controller is given complete access to main memory is _____

a) Cycle stealing
b) Memory stealing
c) Memory Con
d) Burst mode

Answer: d
Explanation: The controller is given full control of the memory access cycles and can transfer blocks at a faster rate.

DMA Transfer



After the completion of the DMA transfer, the processor is notified by _____

a) Acknowledge signal
b) Interrupt signal
c) WMFC signal
d) None of the mentioned

Answer: b
Explanation: The controller raises an interrupt signal to notify the processor that the transfer was complete.

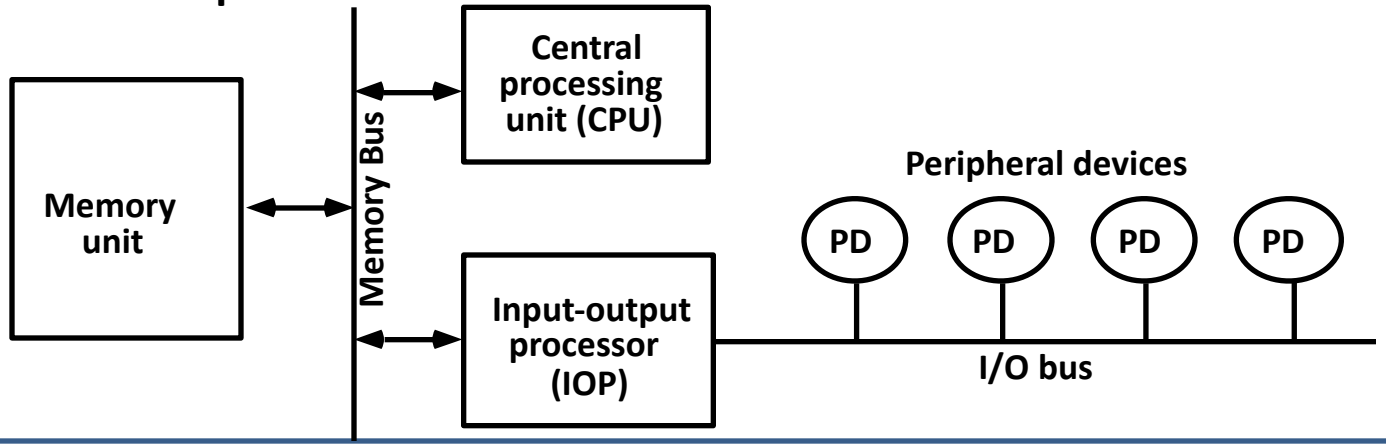
I/O Processor - Channel

Channel

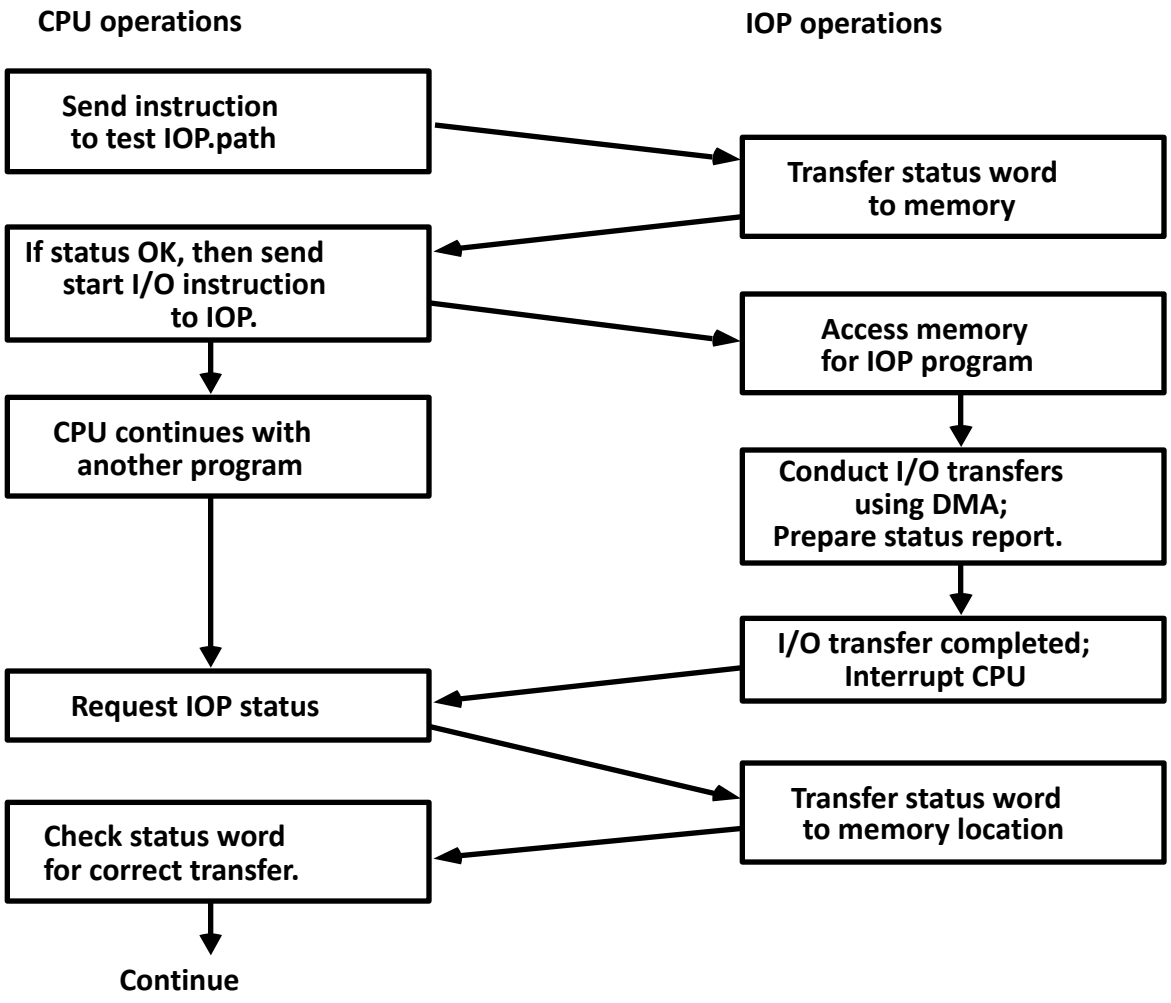
- **Processor with direct memory access capability that communicates with I/O devices**
- **Channel accesses memory by cycle stealing**
- **Unlike DMA Controller, IOP can fetch and execute its own instruction**
- **IOP Instructions (Commands) specially designed to facilitate I/O transfer.**

- **Data gathered in IOP at device rate and bit capacity while CPU executing own program**
- **Transfer between IOP and Device similar to Programmed I/O and transfer between IOP and Memory similar to DMA**
- **CPU is master while IOP is slave processor**

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



Channel CPU Communication

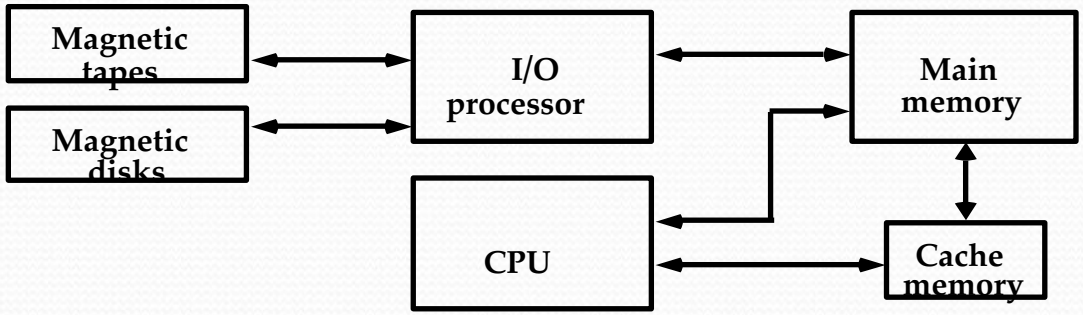


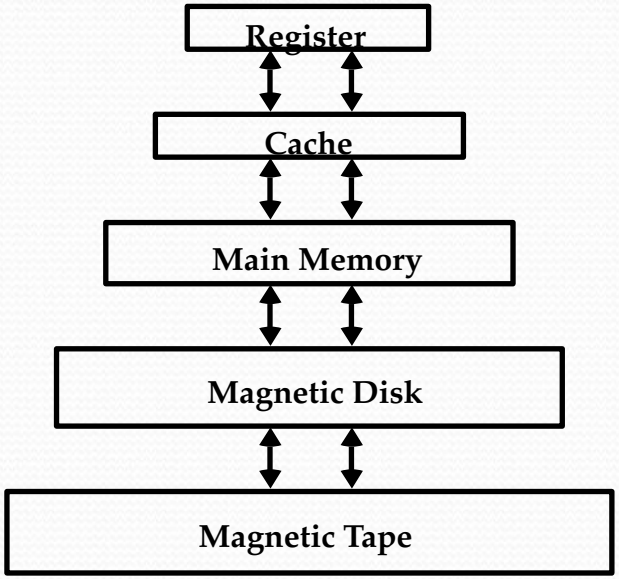
UNIT-5

- Memory Hierarchy
- **Main Memory**
- **Auxiliary Memory**
- **Associative Memory**
- Cache Memory
- Virtual Memory

Memory Hierarchy

Memory Hierarchy is to obtain the highest possible access speed while minimizing the total cost of the memory system





The last on the hierarchy scale of memory devices is _____

a) Main memory

b) Secondary memory

c) cache

d) Flash drives

Memory Access Methods

.Following are the methods to access information from memory locations:

1. **Random Access**: Main memories are random access memories, in which each memory location has a unique address. Using this unique address any memory location can be reached in the same amount of time in any order.
2. **Sequential Access**: This methods allows memory access in a sequence or in order.
3. **Direct Access**: In this mode, information is stored in tracks, with each track having a separate read/write head.

Main Memory

It is the central storage unit of the computer system. It is a large and fast memory used to store data during computer operations. Main memory is made up of **RAM** and **ROM**, with RAM integrated circuit chips holing the major share.

- **RAM: Random Access Memory**
  - **DRAM**: Dynamic RAM, is made of capacitors and transistors, and must be refreshed every 10~100 ms. It is slower and cheaper than SRAM.
  - **SRAM**: Static RAM, has a six transistor circuit in each cell and retains data, until powered off.
  - **NVRAM**: Non-Volatile RAM, retains its data, even when turned off. Example: Flash memory.

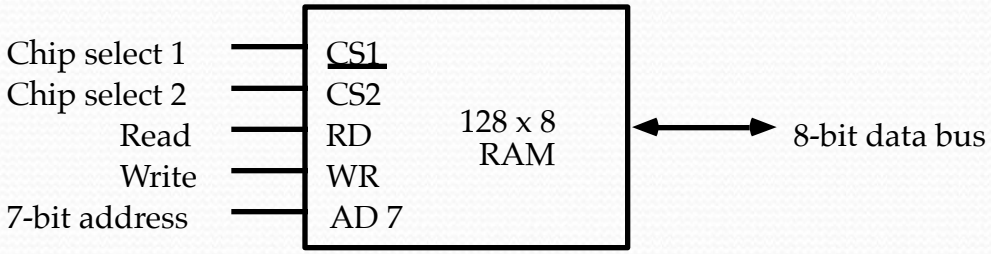
Main Memory

- **ROM: Read Only Memory,** is non-volatile and is more like a permanent storage for information.
- It also stores the **bootstrap loader** program, to load and start the operating system when computer is turned on.
- **PROM**(Programmable ROM), **EPROM**(Erasable PROM) and **EEPROM**(Electrically Erasable PROM) are some commonly used ROMs.

Main Memory

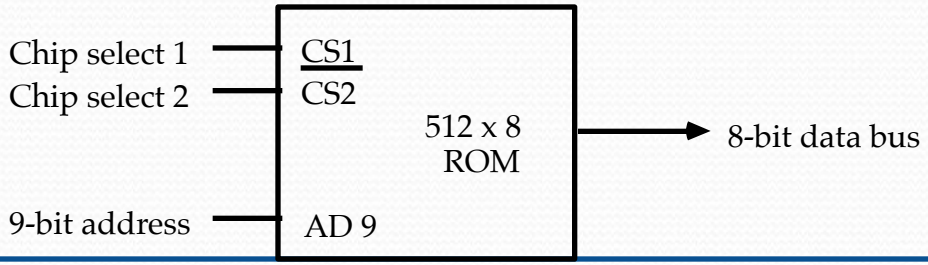
RAM and ROM Chips

Typical RAM chip



CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	x	x	Inhibit	High-impedence
0	1	x	x	Inhibit	High-impedence
1	0	0	0	Inhibit	High-impedence
1	0	0	1	Write	Input data to RAM
1	0	1	x	Read	Output data from RAM
1	1	x	x	Inhibit	High-impedence

Typical ROM chip



Memory Address Map

Address space assignment to each memory chip

Example: 512 bytes RAM and 512 bytes ROM

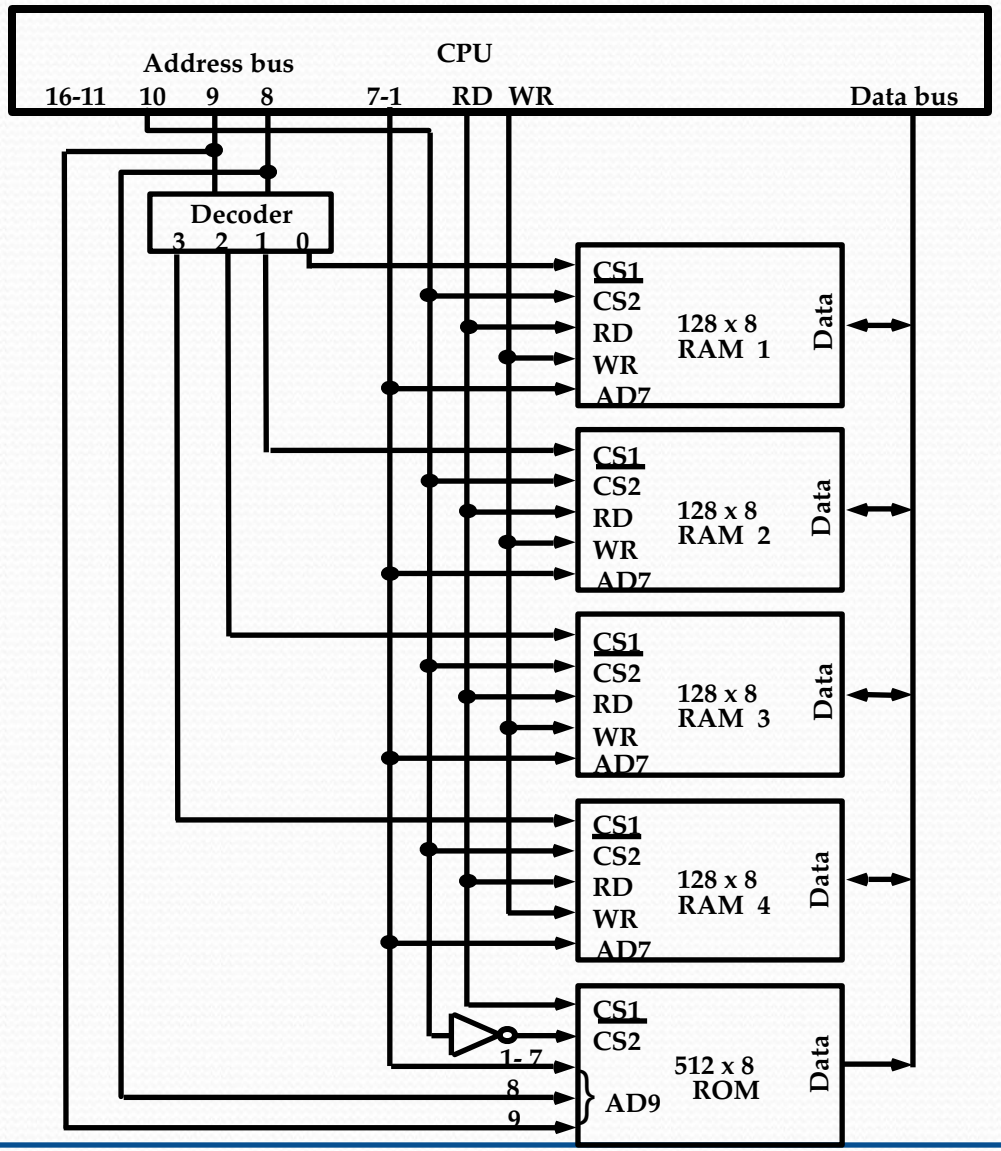
Component	Hexa address	Address bus 10 9 8 7 6 5 4 3 2 1
RAM 1	0000 - 007F	0 0 0 x x x x x x x
RAM 2	0080 - 00FF	0 0 1 x x x x x x x
RAM 3	0100 - 017F	0 1 0 x x x x x x x
RAM 4	0180 - 01FF	0 1 1 x x x x x x x
ROM	0200 - 03FF	1 x x x x x x x x x

Memory Connection to CPU

-RAM and ROM chips are connected to a CPU through the data and address buses

-- The low-order lines in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs

Connection of Memory to CPU



Q-Which of the following is the fastest means of memory access for CPU?

a) Registers
b) Cache
c) Main memory
d) Virtual Memory

Q-(A)- How many 128*8 RAM chips are needed to provide a memory capacity of 2048 bytes ?

(B)- How many lines of the address bus must be used to access 2048 bytes of memory ?How many of these lines will be common to all chips ?

(C)- How many lines must be decoded for chip select ? Specify the size of the decoders.

(a) $\frac{2048}{128} = 16 \text{ chips}$

(b) $2048 = 2^{11}$

$128 = 2^{7}$

11 lines to address 2078 bytes.
7 lines to address each chip
4 lines to decoder for selecting 16 chips

(c) 4×16 decoder

Auxiliary Memory

- Auxiliary memory is much larger in size than main memory but is slower. It normally stores system programs, instruction and data files. It is also known as secondary memory. Characteristics of Auxiliary Memory are following −

- **Non-volatile memory** − Data is not lost when power is cut off.

- **Reusable** − The data stays in the secondary storage on permanent basis until it is not overwritten or deleted by the user.

- **Reliable** − Data in secondary storage is safe because of high physical stability of secondary storage device.

- **Convenience** − With the help of a computer software, authorised people can locate and access the data quickly.

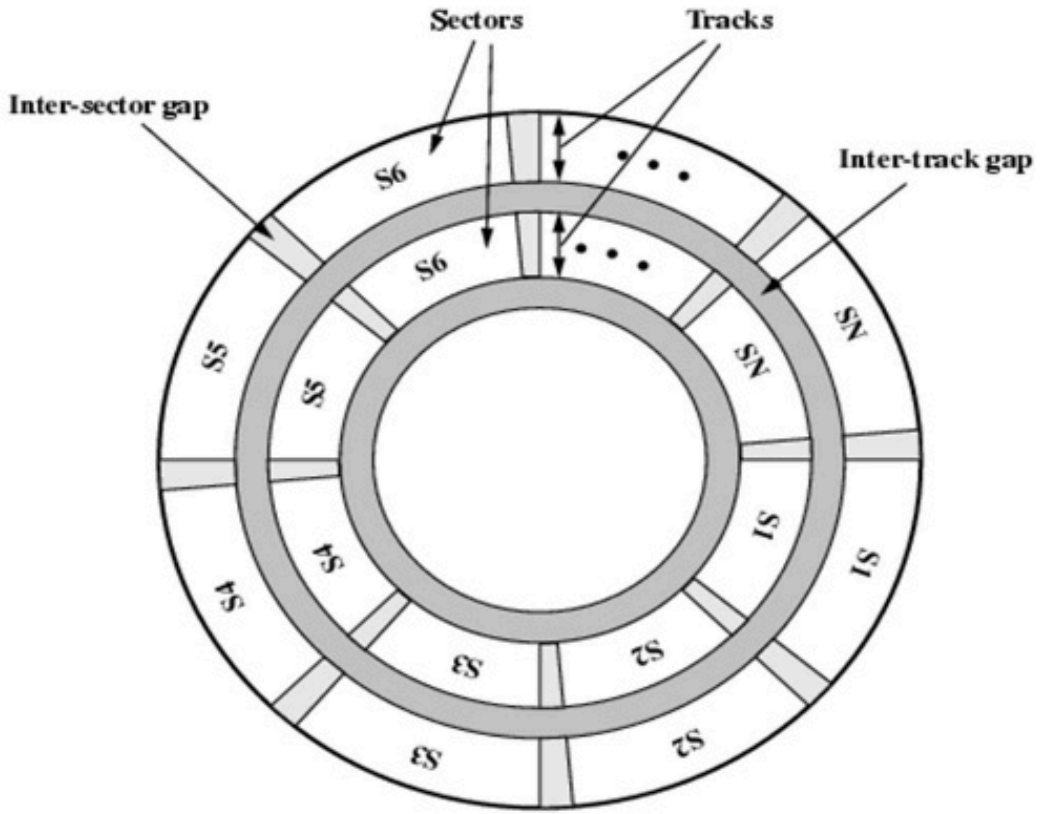
- **Capacity** − Secondary storage can store large volumes of data in sets of multiple disks.

- **Cost** − It is much lesser expensive to store data on a tape or disk than primary memory.

Magnetic tape contains thin plastic ribbon is used for storing data. It is a sequential access memory. So the data read/write speed is slower. It is mainly used for data backups.

Magnetic Disk contains circular disk made of metal or plastic. Both side of disk is usually used for storing data. The disk is coated by magnetic oxide. The disk is divided into multiple concentric circles known as **track** and tracks are divided into **sectors** in which data are stored.

Disk data layout



- The memory bits are stored in the magnetized surface in spots along the concentric circles called **tracks**.
- The concentric circles (tracks) are commonly divided into sections called **sectors**.

Q- Magnetic disk is a sequential access device.

a) True
b) False

1. **Seek time –** The time taken by the R-W head to reach the desired track from it's current position.
2. **Rotational latency –** Time taken by the sector to come under the R-W head.
3. **Data transfer time –** Time taken to transfer the required amount of data. It depends upon the rotational speed.
4. **Controller time –** The processing time taken by the controller.
5. **Average Access time –** seek time + Average Rotational latency + data transfer time + controller time

Q-What is the average access time for transferring 512 bytes of data with the following specifications:-

Average seek time= 5 msec

Data rate= 40KB/sec

Average rotational latency=5 msec

A—22.8 msec

B—22.5 msec

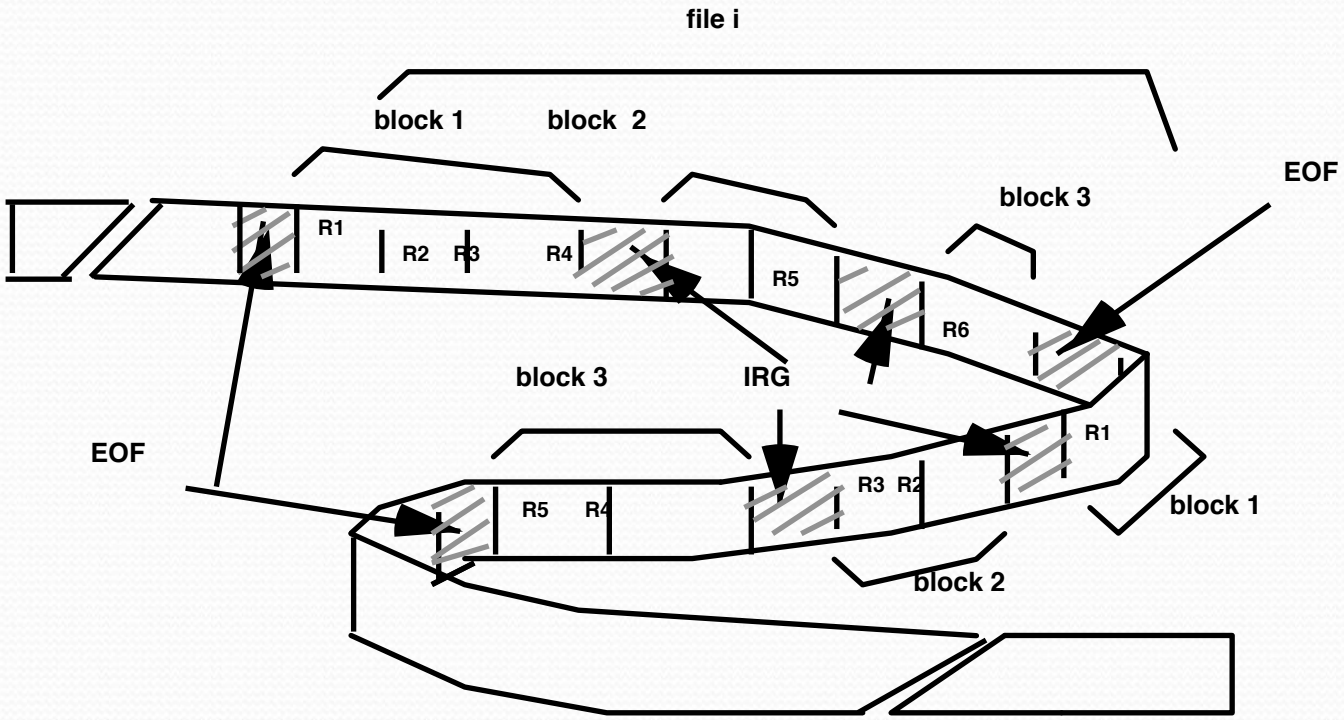
C—22.8sec

D—27.5msec

Magnetic Tape

- Magnetic tape is a storage medium that allows data archiving, collection, and backup for different kinds of data.
- The magnetic tape is constructed using a plastic strip coated with a magnetic recording medium.
- The bits are recorded as magnetic spots on the tape along several tracks.
- Magnetic tape units can be halted, started to move forward or in reverse, or can be rewound.
- However, they cannot be started or stopped fast enough between individual characters. For this reason, information is recorded in blocks referred to as records.

Information Organization on Magnetic Tapes



Which of the following places the common data elements in order from smallest to largest?

(A) Character, File, Record, Field, Database, File
(B)Character, Record, Field, Database, File
(C)byte,Character, Field, Record, File, Database
(D)Bit, Byte, Character, Record, Field, File, Database
(E)None of these

Consider a hard disk with:

4 surfaces

128 tracks/surface

256 sectors/track

512 bytes/sector

1. **What is the capacity of the hard disk?**

   Disk capacity = surfaces * tracks/surface * sectors/track * bytes/sector

   Disk capacity = _____bytes

   Disk capacity = _____ MB

2. **The disk is rotating at 3600 RPM, what is the data transfer rate?**

   60 sec -> 3600 rotations

   1 sec -> 60 rotations

   Data transfer rate = number of rotations per second * track capacity * number of surfaces (since 1 R-W head is used for each surface)

   Data transfer rate = _____**bytes/sec**

   Data transfer rate = _____ MB/sec

3- The disk is rotating at 3600 RPM, what is the average access time?

3- The disk is rotating at 3600 RPM, what is the average access time?

Since, seek time, controller time and the amount of data to be transferred is not given, we consider all the three terms as 0.

Therefore, Average Access time = Average rotational delay
Rotational latency => 60 sec -> 3600 rotations
1 sec -> 60 rotations
Rotational latency = (1/60) sec = 16.67 msec.

Average Rotational latency = (16.67)/2. = 8.33 msec.
Thus, on average, the rotational latency is half the time it takes the disk to make a complete revolution.

Average Access time = 0+0+8.33+0=8.33 msec.

Q__Disk access time does not depends on which of the following factors _____

a) Seek time
b) Latency
c) Transfer rate
d) Arrival rate

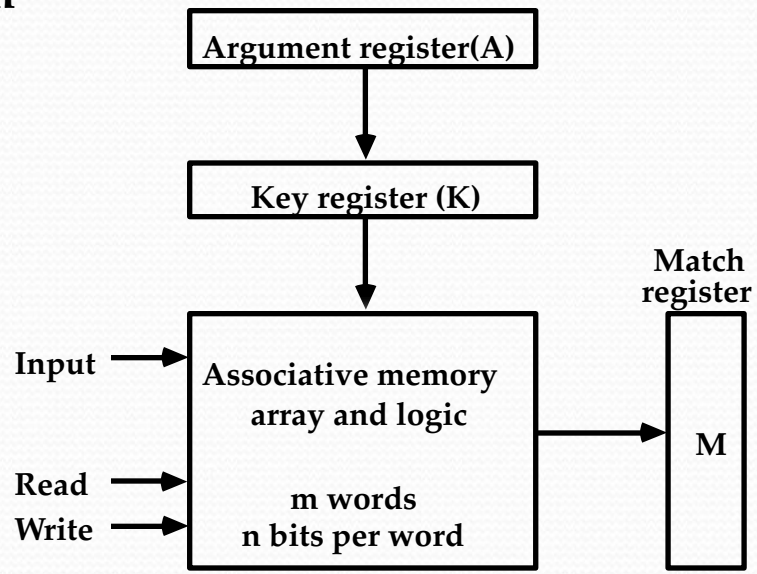
Disk access time does not depends on Arrival rate.

The disk access time depends on the seek time, latency, and transfer rate. Wherein, seek time is the time required to position the read/write head over the desired track.

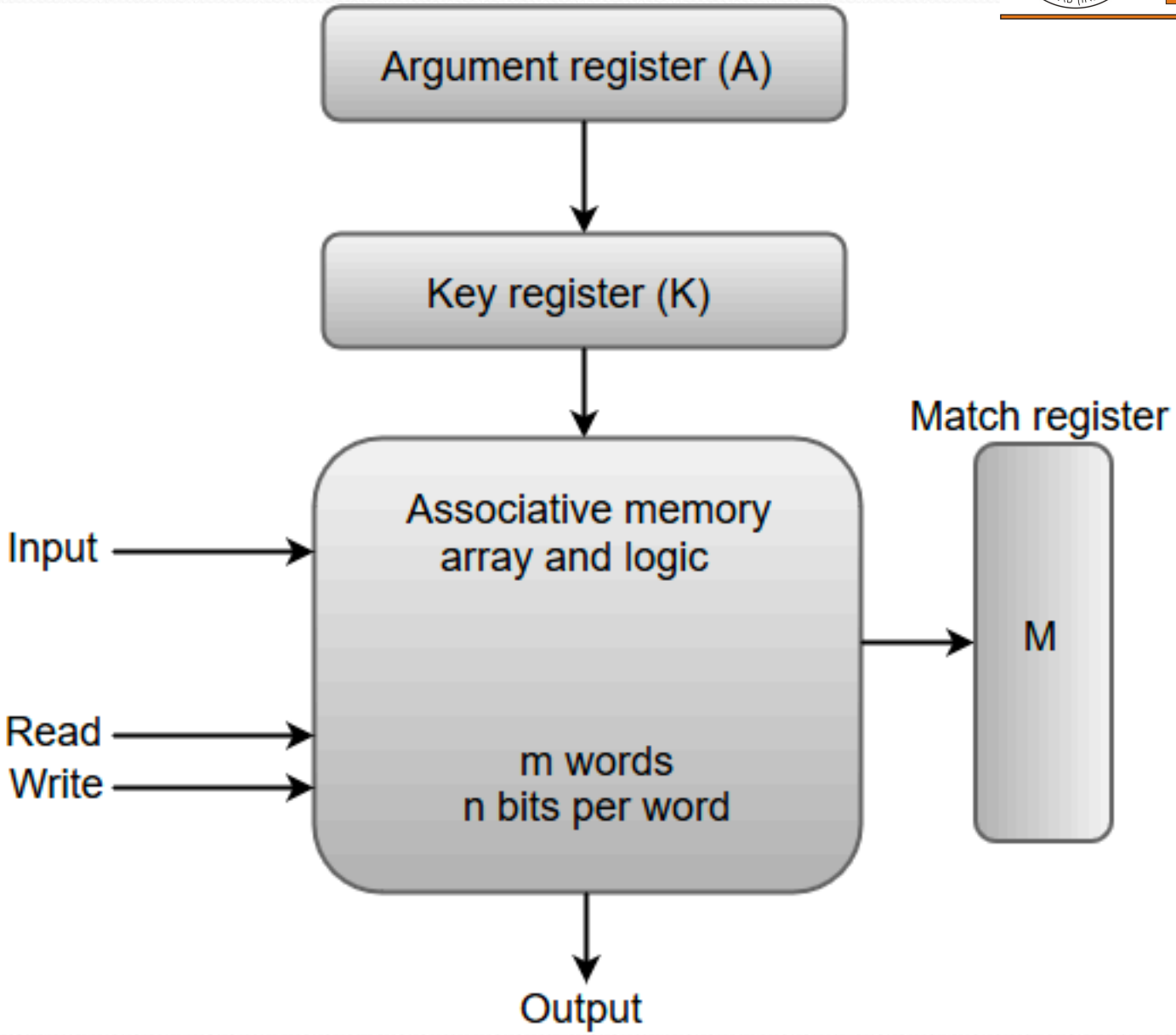
Associative Memory

- Accessed by the content of the data rather than by an address
- Also called Content Addressable Memory (CAM)

Hardware Organization



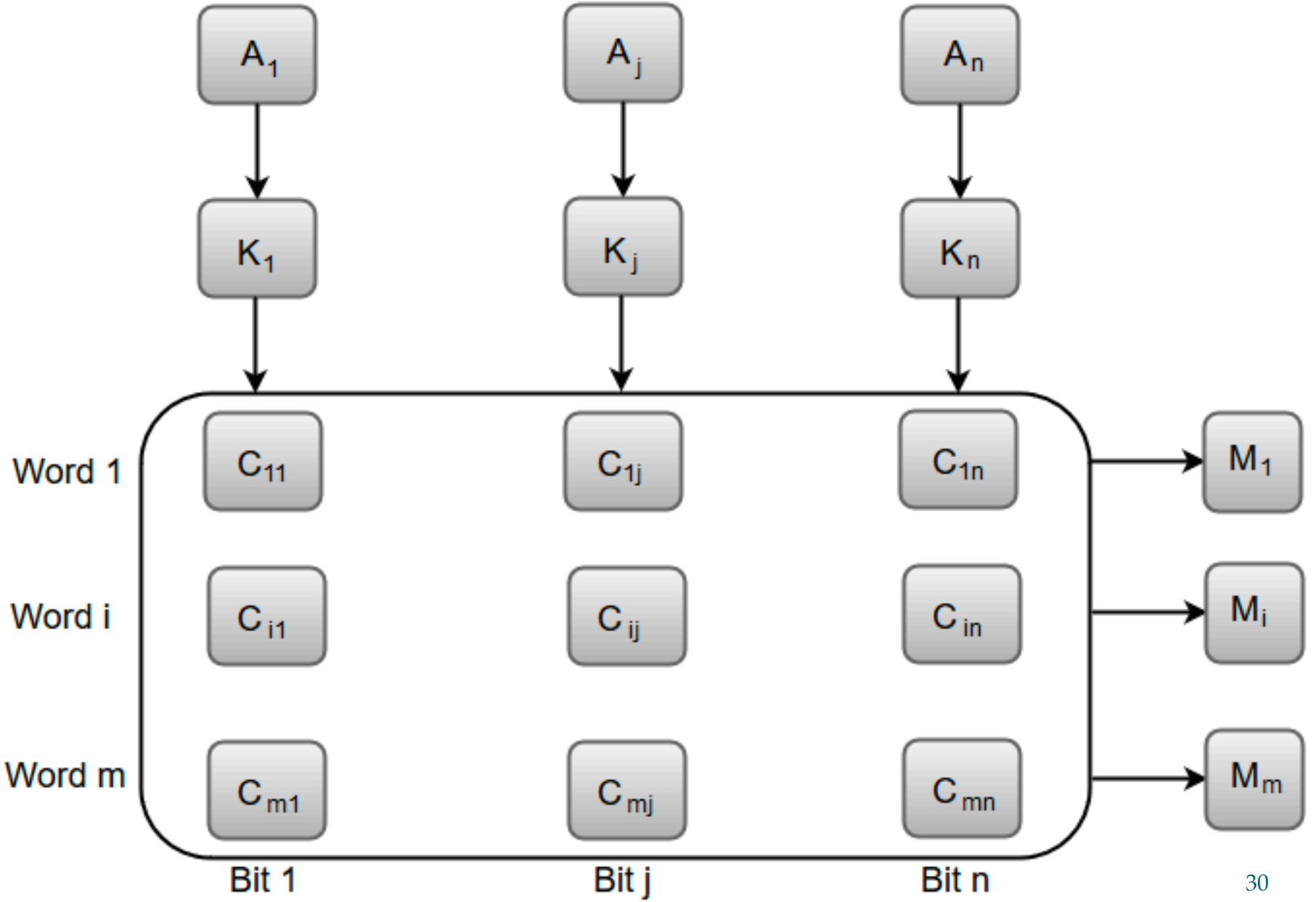
A	101 111100	
K	111 000000	
Word 1	100 111100	no match
Word 2	101 000001	match


Argument register (A)
Key register (K)
Match register
Associative memory
array and logic
Input
M
Read
Write
m words
n bits per word
Output

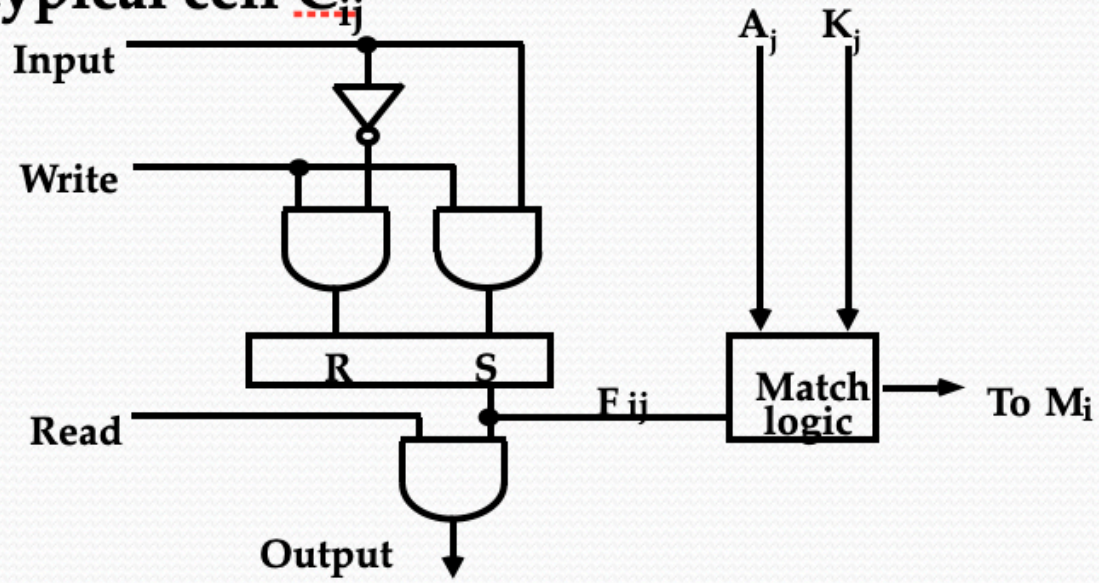
- From the block diagram, we can say that an associative memory consists of a memory array and logic for 'm' words with 'n' bits per word.
- The functional registers like the argument register **A** and key register **K** each have **n** bits, one for each bit of a word.
- The match register **M** consists of **m** bits, one for each memory word.
- The words which are kept in the memory are compared in parallel with the content of the argument register.

The key register (K) provides a mask for choosing a particular field or key in the argument word. If the key register contains a binary value of all 1's, then the entire argument is compared with each memory word. Otherwise, only those bits in the argument that have 1's in their corresponding position of the key register are compared. Thus, the key provides a mask for identifying a piece of information which specifies how the reference to memory is made.

Associative memory of m word, n cells per word:



Internal organization of a typical cell C_{ij}



- The cells present inside the memory array are marked by the letter C with two subscripts.
- The first subscript gives the word number and the second specifies the bit position in the word. For instance, the cell C_{ij} is the cell for bit **j** in word **I**
- A bit A_j in the argument register is compared with all the bits in column **j** of the array provided that $K_j = 1$. This process is done for all columns **j** = 1, 2, 3....., n.
- If a match occurs between all the unmasked bits of the argument and the bits in word **i**, the corresponding bit M_i in the match register is set to 1.
- If one or more unmasked bits of the argument and the word do not match, M_i is cleared to 0

Advantages of Associative memory :-

1. It is used where search time needs to be less or short.
2. It is suitable for parallel searches.
3. It is often used to speedup databases.
4. It is used in page tables used by the virtual memory and used in neural networks.

Disadvantages of Associative memory :-

1. It is more expensive than RAM.
2. Each cell must have storage capability and logical circuits for matching its content with external argument.

Which memory has largest storage capacity among all?

(A) Auxiliary memory
(B) RAM
(C) Associative memory
(D) Cache memory

Q-What is the average access time for transferring 512 bytes of data with the following specifications-

- Average seek time = 5 msec
- Disk rotation = 6000 RPM
- Data rate = 40 KB/sec
- Controller overhead = 0.1 msec

Time taken for one full rotation = **60/6000 sec**

= 1/100 sec = 0.01 sec = 10 msec

Average rotational delay = **½ x Time taken for one full rotation**

= ½ x 10 msec = 5 msec

Transfer time = **(512bytes / 40 kb) sec**

= 0.0125 sec = 12.5 msec

Average access time = **Avg. seek time + Avg. rotational delay + Transfer time + Controlled overhead**

= 5 msec + 5 msec + 12.5 msec + 0.1 msec

= 22.6 msec

LOVELY PROFESSIONAL UNIVERSITY

UNIT-5

- Memory Hierarchy
- Main Memory
- Auxiliary Memory
- Associative Memory
- **Cache Memory**
- Virtual Memory

Levels of memory:

- **Level 1 or Register –**
  It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.

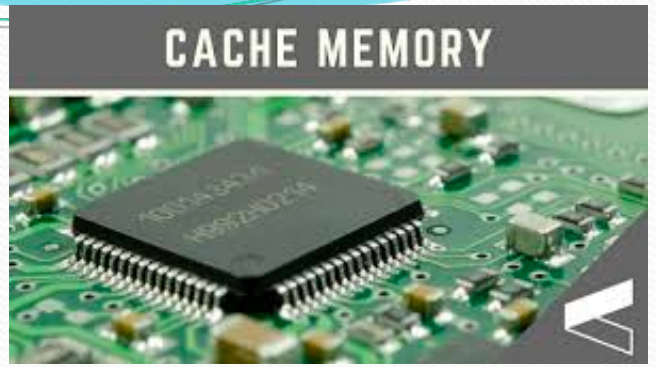
- **Level 2 or Cache memory –**
  It is the fastest memory which has faster access time where data is temporarily stored for faster access.

- **Level 3 or Main Memory –**
  It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory

- **Level 4 or Secondary Memory –**
  It is external memory which is not as fast as main memory but data stays permanently in this memory.

LOVELY PROFESSIONAL UNIVERSITY

Cache Memory

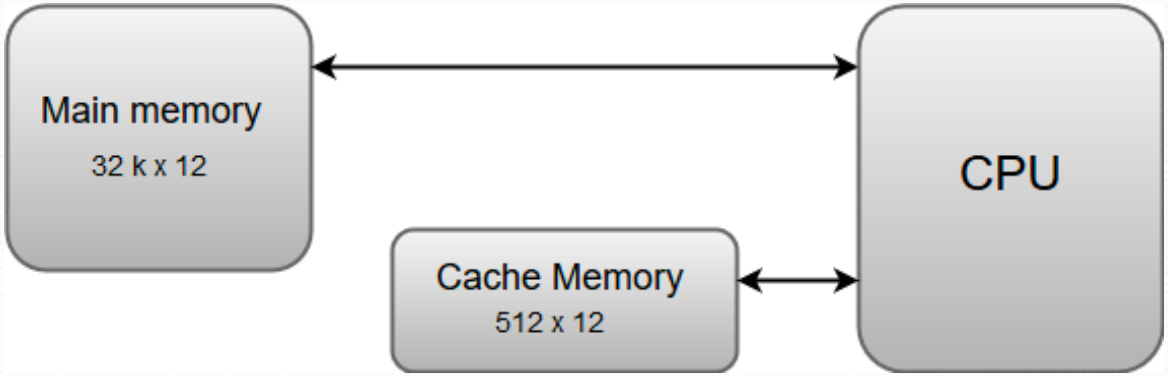


- **Cache Memory** is a very high-speed memory.
- It is used to speed up and synchronising with high-speed CPU.
- Cache memory is costlier than main memory or disk memory.
- Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU.
- It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.

Cache Memory

- Cache memory is used to reduce the average time to access data from the Main memory.
- The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations.
- The Cache memory is implemented using the SRAM chips and not the DRAM chips. SRAM stands for Static RAM.
- It is faster and is expensive.

Cache Memory



- The data or contents of the main memory that are used frequently by CPU are stored in the cache memory so that the processor can easily access that data in a shorter time.
- Whenever the CPU needs to access memory, it first checks the cache memory.
- If the data is not found in cache memory, then the CPU moves into the main memory.
- Cache memory is placed between the CPU and the main memory.

Advantages:-

- Cache memory is faster than main memory.
- It consumes less access time as compared to main memory.
- It stores the program that can be executed within a short period of time.
- It stores data for temporary use.
- Cache memory is a Random Access Memory.
- The main advantage of cache memory is its very fast speed.
- It can be accessed by the CPU at much faster speed than main memory

Disadvantages:-

- Cache memory has limited capacity.
- It is very expensive.

Locality of reference –

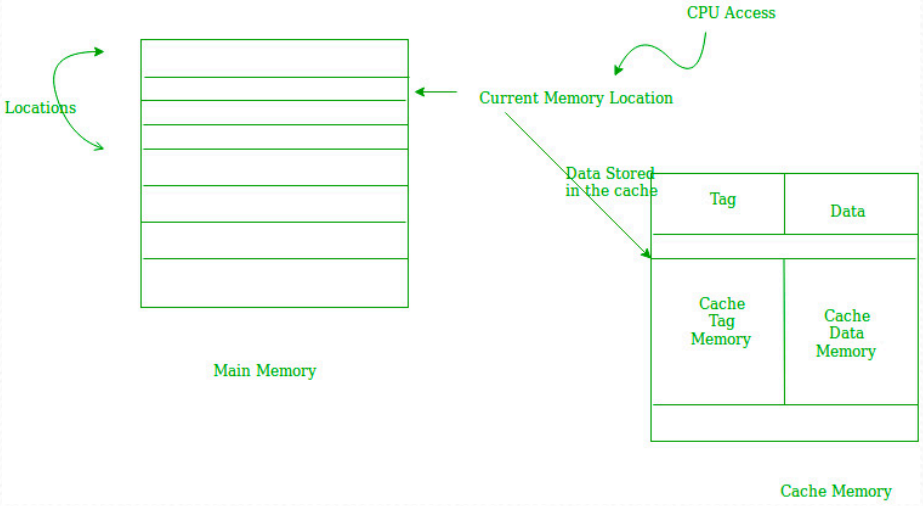
Since size of cache memory is less as compared to main memory. So to check which part of main memory should be given priority and loaded in cache is decided based on locality of reference.

Types of Locality of reference

1. **Spatial Locality of reference**
   This says that there is a chance that element will be present in the close proximity to the reference point and next time if again searched then more close proximity to the point of reference.If a word is accessed, adjacent(near) words are likely accessed soon.

2. **Temporal Locality of reference**

Temporal locality means current data or instruction that is being fetched may be needed soon. So we should store that data or instruction in the cache memory so that we can avoid again searching in main memory for the same data.



Cache Performance:

- When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.
- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache.
- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred.
- For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

```
Hit ratio = hit / (hit + miss) =  no. of hits/total accesses
```

```
Hit + Miss  = Total CPU Reference
```

Performance of Cache

Performance of Cache Memory System

Hit Ratio - % of memory accesses satisfied by Cache memory system

Te: Effective memory access time in Cache memory system

Tc: Cache access time

Tm: Main memory access time

$$Te = Tc + (1 - h)\, Tm$$

Example: Tc = 0.4 μs, Tm = 1.2μs, h = 0.85%

$$Te = 0.4 + (1 - 0.85) * 1.2 = 0.58\mu s$$

Performance of Cache

Average access time of any memory system consists of two levels:

Cache and Main Memory.

If ***Tc*** is time to access cache memory and ***Tm*** is the time to access main memory then we can write:

```
Tavg = Average time to access memory
Tavg = h * Tc + (1-h)*(Tm + Tc)
```

Q- What is the Hit ratio of cache memory, If cache access time is 30 ns, main memory access time is 150 ns and average access time is 42 ns .

```
Average time to access memory =
Hit ratio * cache access time + miss ratio
(cache access time + main memory access time)

Tavg = h * Tc + (1-h)*(Tm + Tc)
```

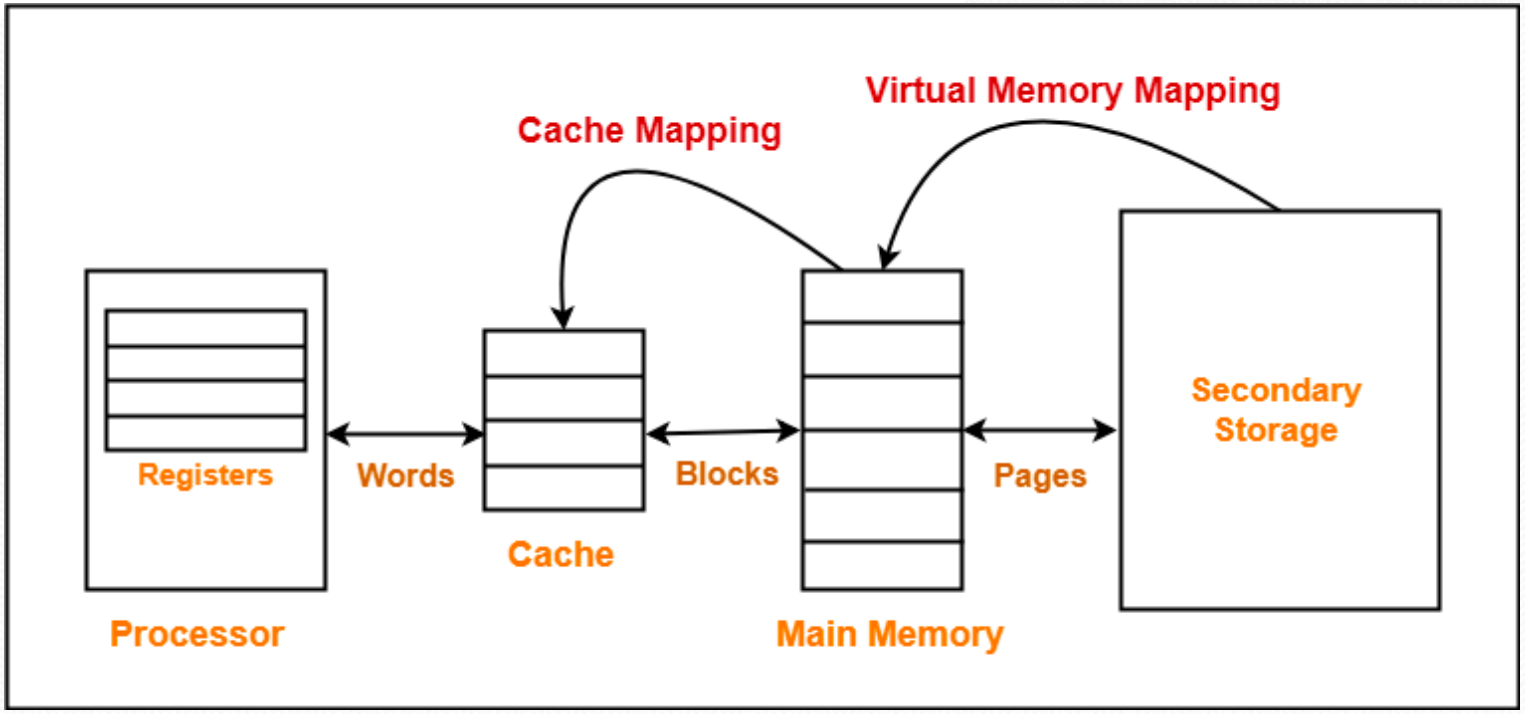
Cache Mapping-

- Cache mapping defines how a block from the main memory is mapped to the cache memory in case of a cache miss.

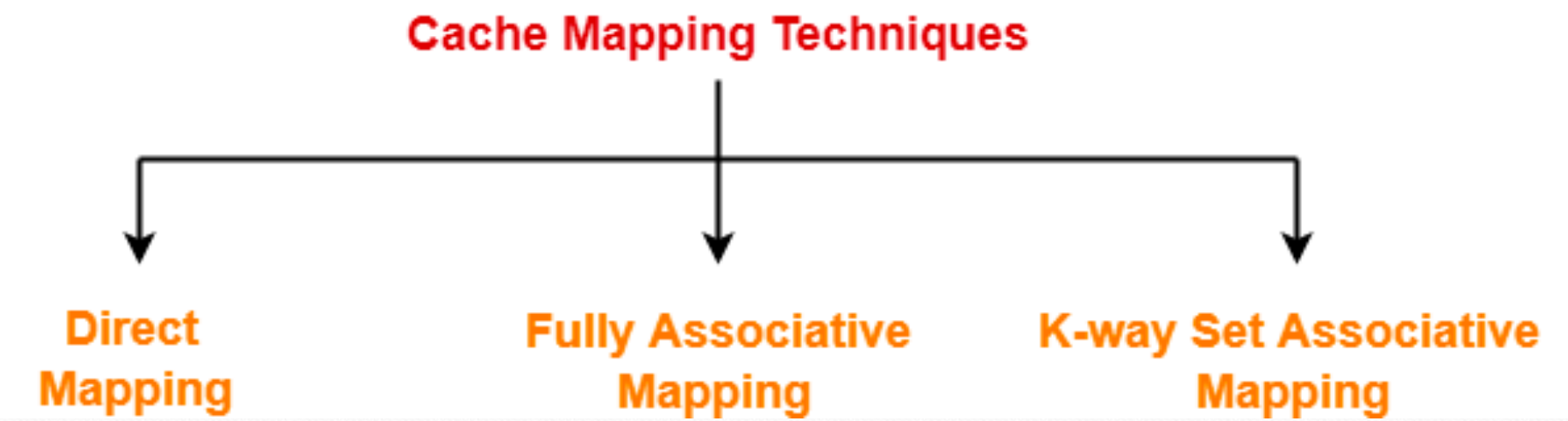
OR

- Cache mapping is a technique by which the contents of main memory are brought into the cache memory.

The following diagram illustrates the mapping process-



- A mapping function is the method used to locate a memory address within a cache
- It is used when copying a block from main memory to the cache and it is used again when trying to retrieve data from the cache
- There are three kinds of mapping functions
  - Direct
  - Associative
  - Set Associative



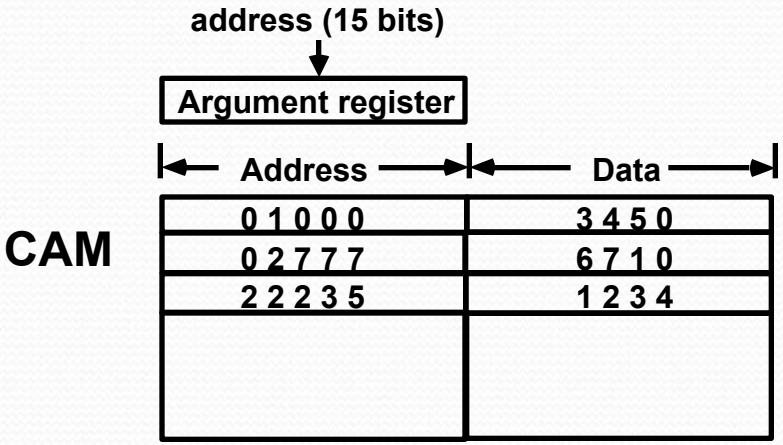
Associative mapping:-

- In this type of mapping the associative memory is used to store content and addresses both of the memory word.
- This enables the placement of the any word at any place in the cache memory.
- It is considered to be the fastest and the most flexible mapping form.

Memory and Cache Mapping

Associative Mapping

- Any block location in Cache can store any block in memory
  -> Most flexible
- Mapping Table is implemented in an associative memory
  -> Fast, very Expensive
- Mapping Table
  Stores both address and the content of the memory word



- Need of Replacement Algorithm-

In fully associative mapping,

- A replacement algorithm is required.
- Replacement algorithm suggests the block to be replaced if all the cache lines are occupied.
- Thus, replacement algorithm like FCFS Algorithm, LRU Algorithm etc is employed.

Division of Physical Address-

In fully associative mapping, the physical address is divided as-

Division of Physical Address in Fully Associative Mapping

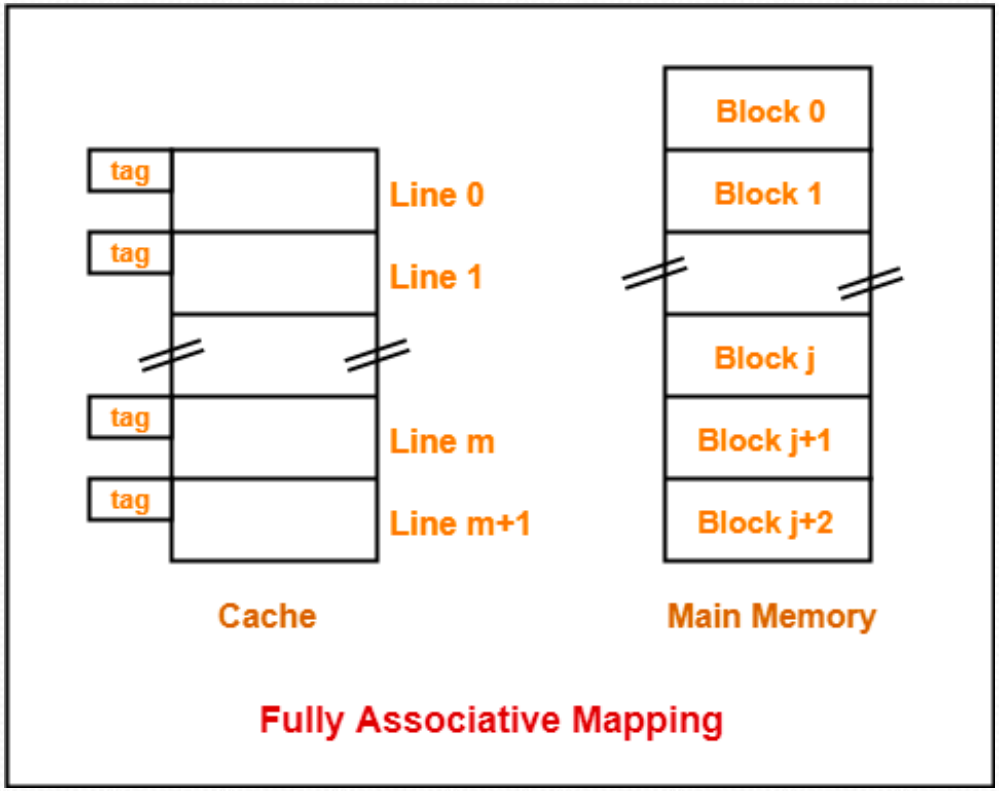
Associative Mapping-

In fully associative mapping,

- A block of main memory can map to any line of the cache that is freely available at that moment.
- This makes fully associative mapping more flexible than direct mapping.

Example-

Consider the following scenario-



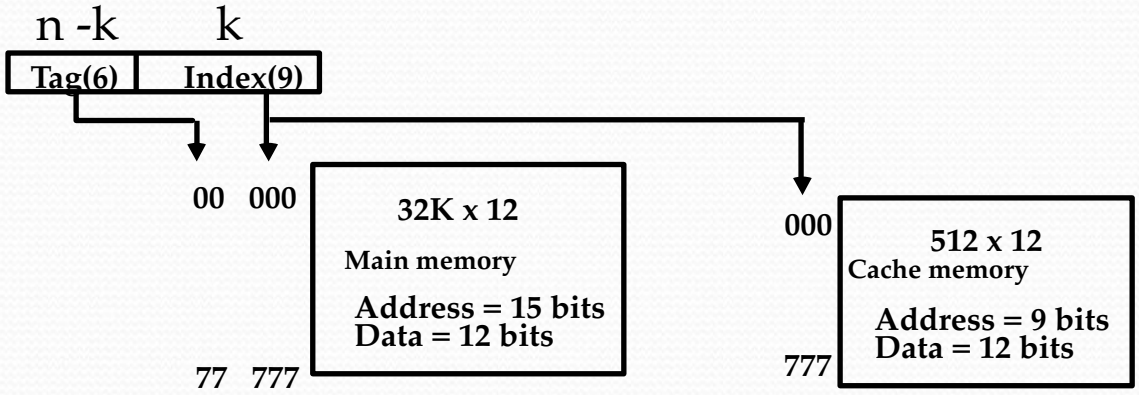
Cache Mapping – direct mapping

- In Direct mapping, assign each memory block to a specific line in the cache.
- If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed.
- An address space is split into two parts index field and a tag field. The cache is used to store the tag field whereas the rest is stored in the main memory.

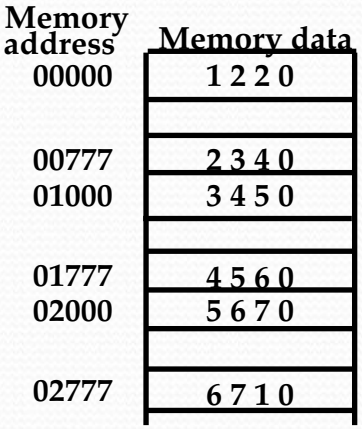
- Each memory block has only one place to load in Cache
- Mapping Table is made of RAM instead of CAM
- n-bit memory address consists of 2 parts; k bits of Index field and n-k bits of Tag field
- n-bit addresses are used to access main memory and k-bit Index is used to access the Cache

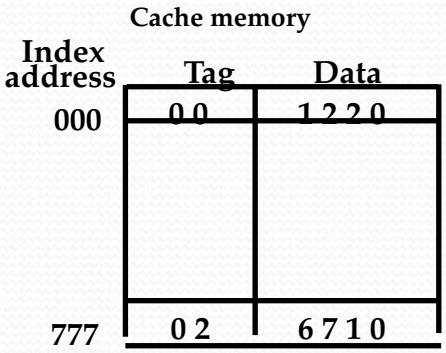
Cache Mapping – direct mapping

Addressing Relationships



Direct Mapping Cache Organization





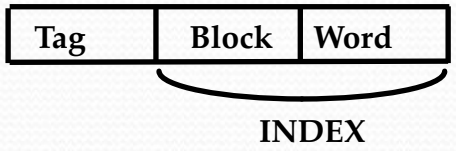
Cache Mapping – direct mapping

Operation

- CPU generates a memory request with (TAG;INDEX)
- Access Cache using INDEX ; (tag; data)
  Compare TAG and tag
- If matches -> Hit
  Provide Cache(data) to CPU
- If not match -> Miss
  Search main memory and replace the block from cache memory

Direct Mapping with block size of 8 words

Block 0	000 007	0 1 0 1	3 4 5 0 6 5 7 8
Block 1	010 017		
Block 63	770 777	0 2 0 2	 6 7 1 0

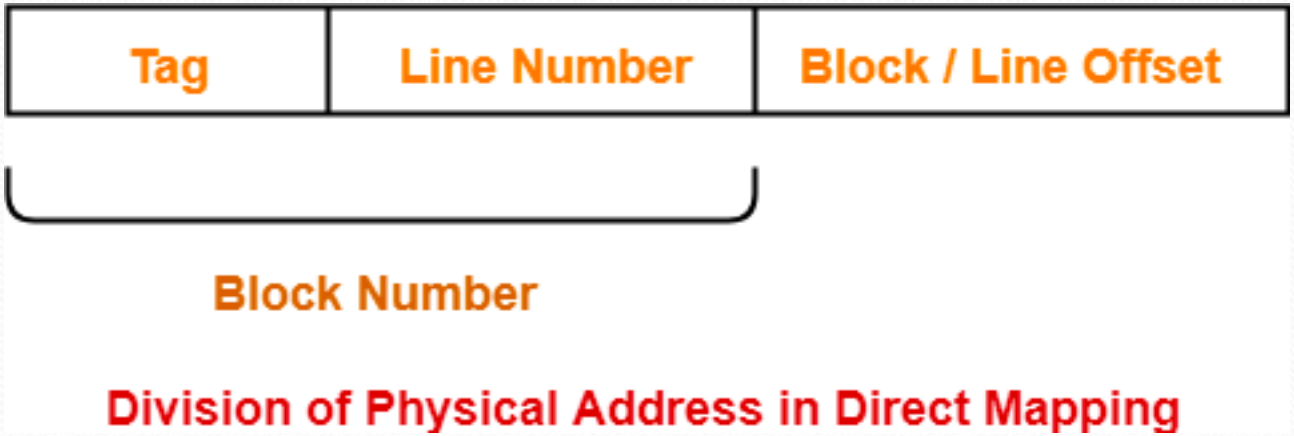


In direct mapping,

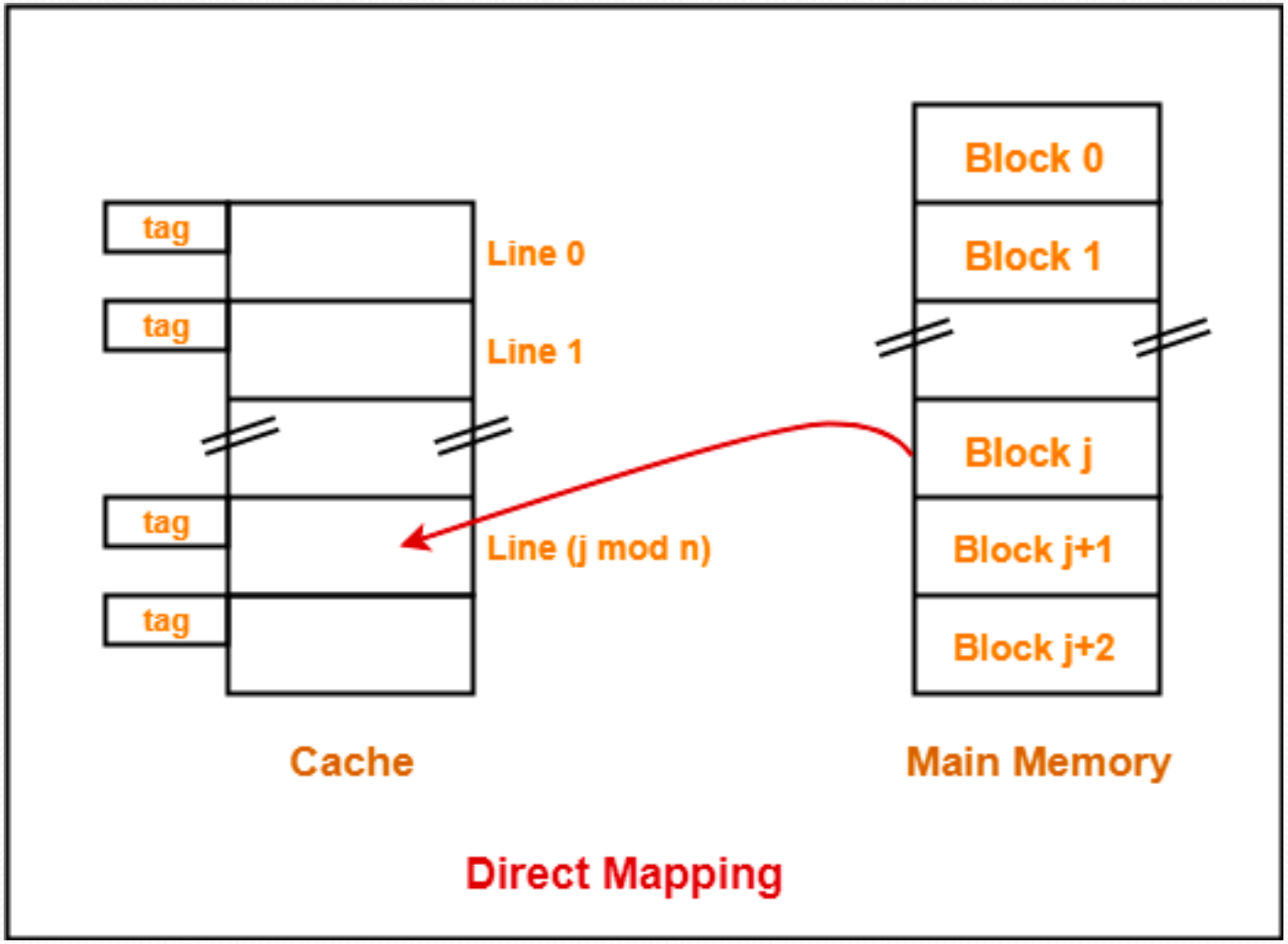
- There is no need of any replacement algorithm.
- This is because a main memory block can map only to a particular line of the cache.
- Thus, the new incoming block will always replace the existing block (if any) in that particular line.

Division of Physical Address-

In direct mapping, the physical address is divided as-



Division of Physical Address in Direct Mapping


tag
tag
tag
tag
Line 0
Line 1
Line (j mod n)
Block 0
Block 1
Block j
Block j+1
Block j+2
Cache
Main Memory
Direct Mapping

Cache Mapping – Set Associative Mapping

- This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed.
- Set associative addresses instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a ***set***.
- Then a block in memory can map to any one of the lines of a specific set..
- Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address.
- Set associative cache mapping combines the best of direct and associative cache mapping techniques.

Cache Mapping – Set Associative Mapping

- Each memory block has a set of locations in the Cache to load

Set Associative Mapping Cache with set size of two

Index	Tag	Data	Tag	Data
000	01	3450	02	5670
777	02	6710	00	2340

Cache Write

Write Through

When writing into memory

If Hit, both Cache and memory is written in parallel
If Miss, Memory is written
For a read miss, missing block may be overloaded onto a cache block.

Memory is always updated
-> Important when CPU and DMA I/O are both executing

Slow, due to the memory access time

Cache Write

Write-Back (Copy-Back)

When writing into memory

If Hit, only Cache is written
If Miss, missing block is brought to Cache and write into Cache
For a read miss, candidate block must be written back to the memory

Memory is not up-to-date, i.e., the same item in
Cache and memory may have different value

Q- Assume a memory access to main memory takes **30 ns** and a memory access to the cache on a cache "hit" takes **3 ns**. If 80% of the processor's memory requests result in a hit ratio, what is the average memory access time?

a.) 8.4nS

b.) 33nS

c.) 7.8nS

d.) 27.0nS

e.) 9nS

```
Tavg = h * Tc + (1-h)*(Tm + Tc)
```

LOVELY PROFESSIONAL UNIVERSITY

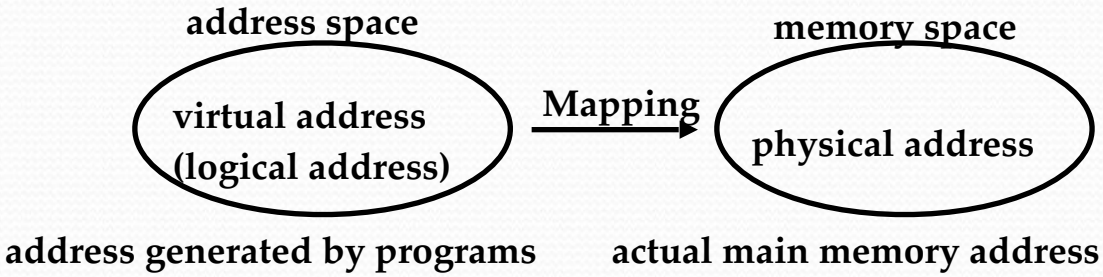
UNIT-5

- Memory Hierarchy
- Main Memory
- Auxiliary Memory
- Associative Memory
- Cache Memory
- **Virtual Memory**

Virtual Memory

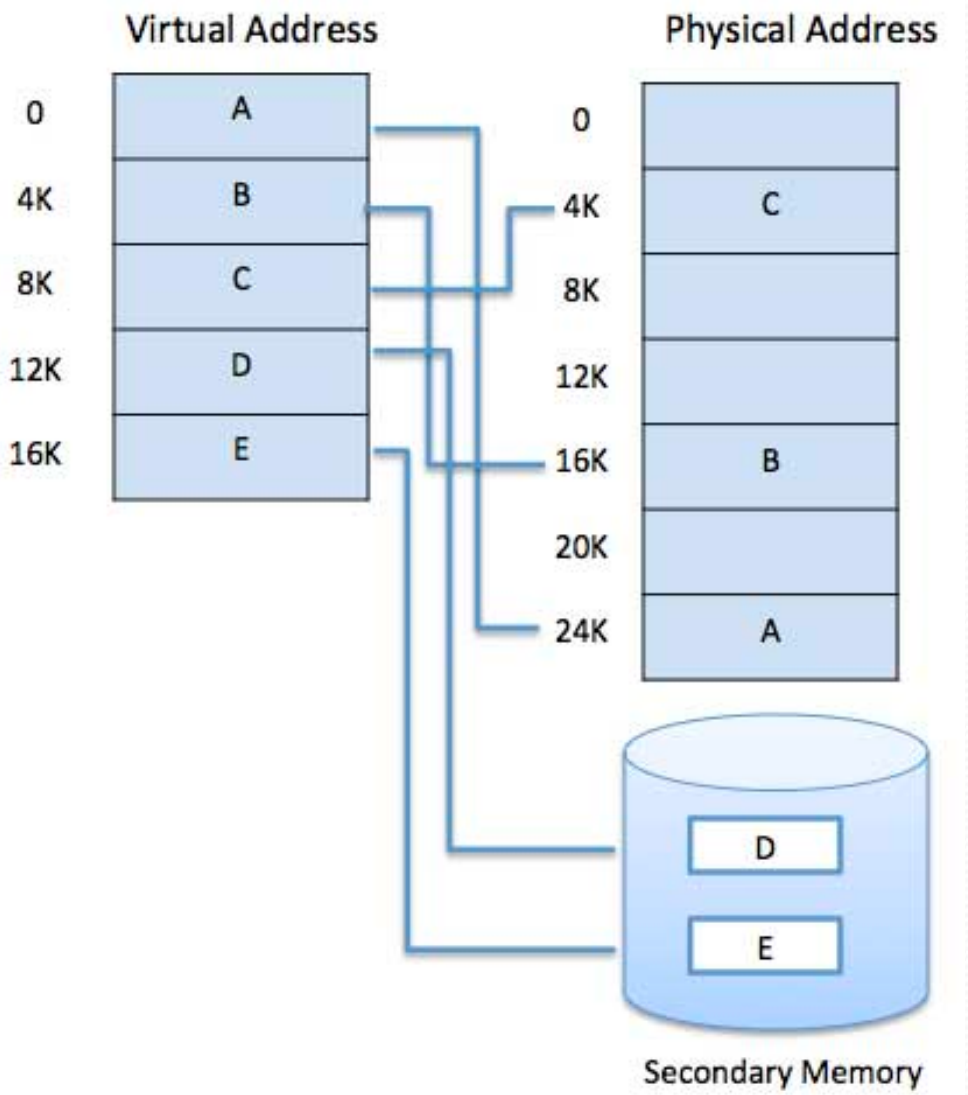
Give the programmer the illusion that the system has a very large memory, even though the computer actually has a relatively small main memory.

Address Space(Logical) and Memory Space(Physical)



- Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware.

- The MMU's job is to translate virtual addresses into physical addresses.



Paging

- A computer can address more memory than the amount physically installed on the system.
- This extra memory is actually called **virtual memory.**
- Paging technique plays an important role in implementing virtual memory.
- Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2).
- The size of the process is measured in the number of pages.
- Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** .

- Main memory is divided into small fixed-sized blocks of (physical) memory called_____

- **pages**
- **frames**
- **words**
- **Chunks**

Address Translation

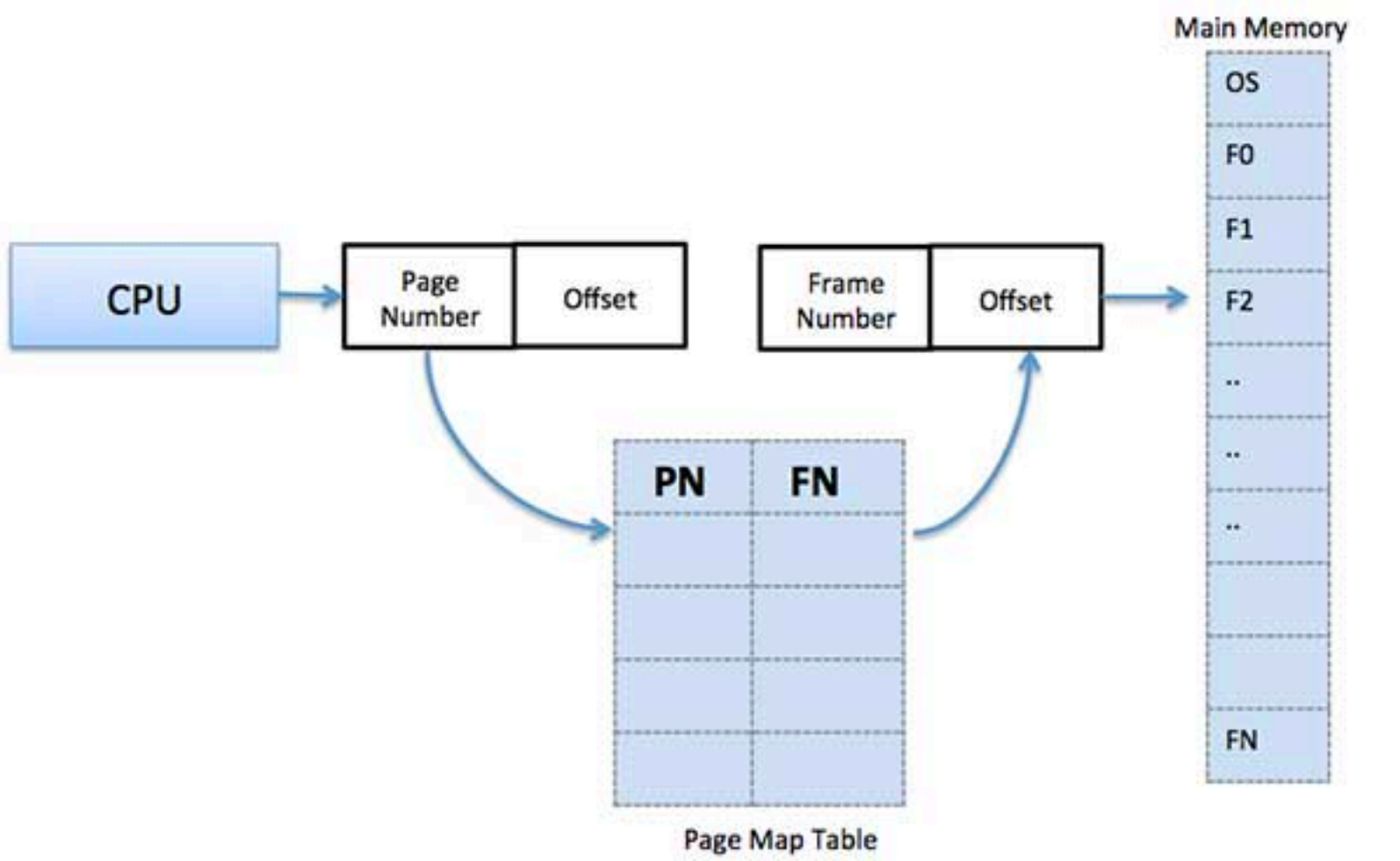
Page address is called **logical address** and represented by **page number** and the **offset**.

Logical Address = Page number + page offset

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

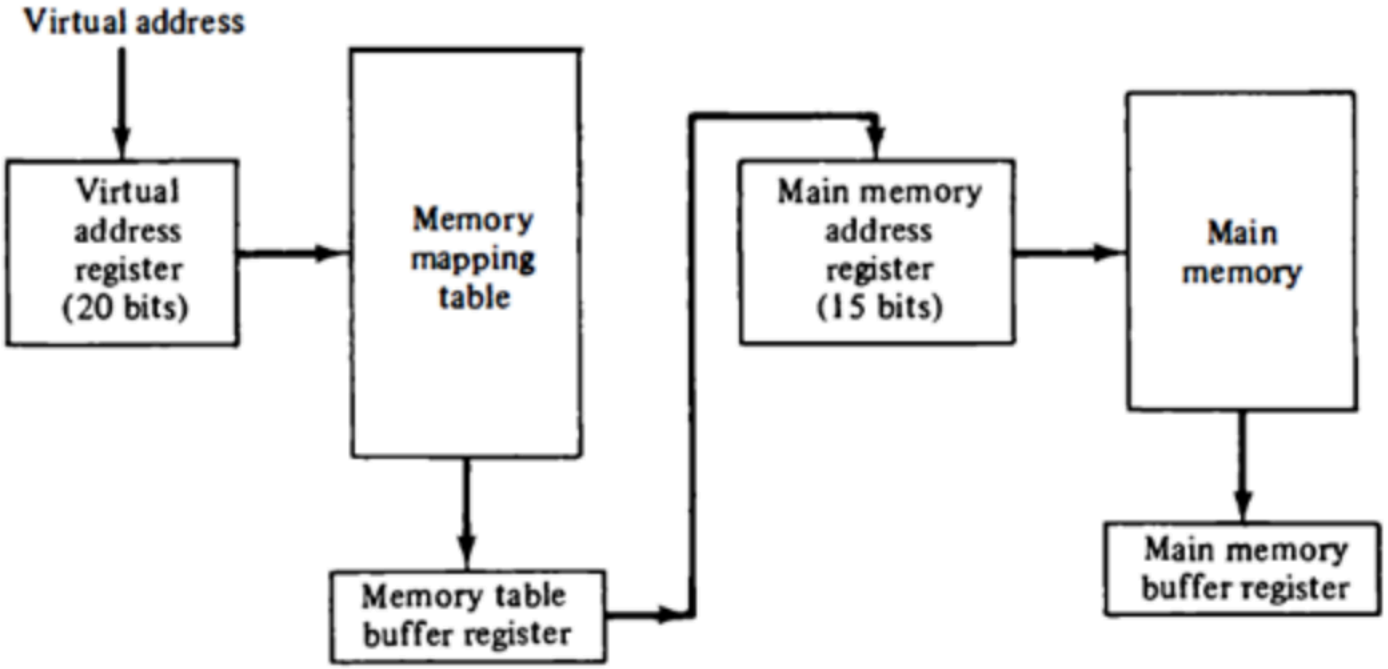
Physical Address = Frame number + page offset

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.


Main Memory
OS
F0
F1
F2
..
..
..
FN
CPU
Page Number
Offset
Frame Number
Offset
PN
FN
Page Map Table

Address Mapping

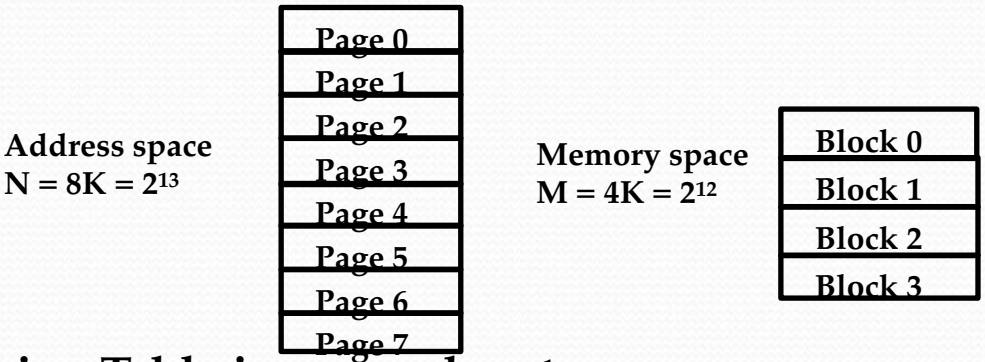
Memory *Mapping Table* for *Virtual Address -> Physical Address*



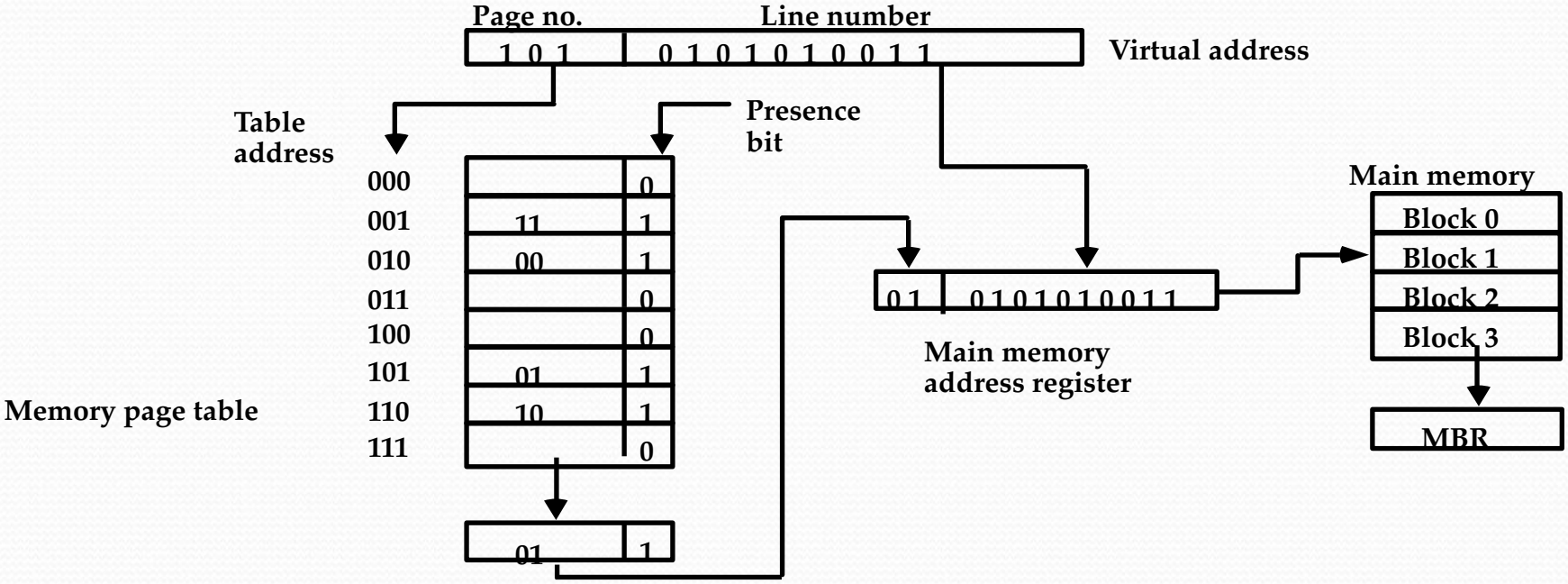
Address Mapping

Address Space and Memory Space are each divided into fixed size group of words called *pages*

1K words group



Organization of memory Mapping Table in a paged system



Associative Memory Page Table

Assume that

Number of Blocks in memory = m

Number of Pages in Virtual Address Space = n

Page Table

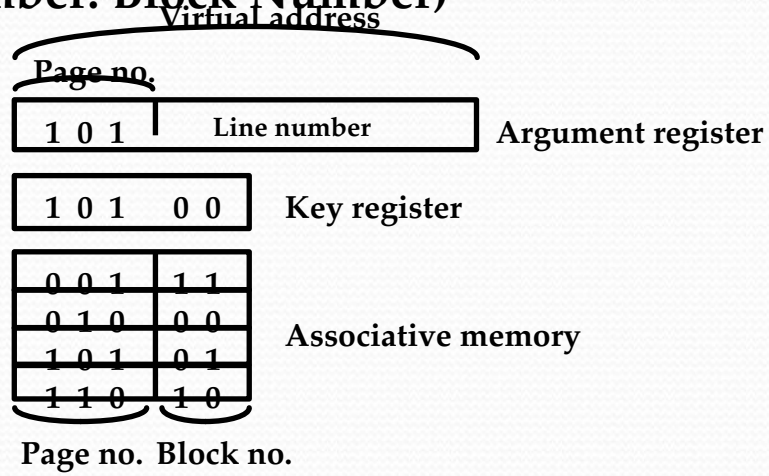
- Straight forward design -> n entry table in memory Inefficient storage space utilization

<- n-m entries of the table is empty

- More efficient method is m-entry Page Table

Page Table made of an Associative Memory

m words; (Page Number: Block Number)



Page Fault

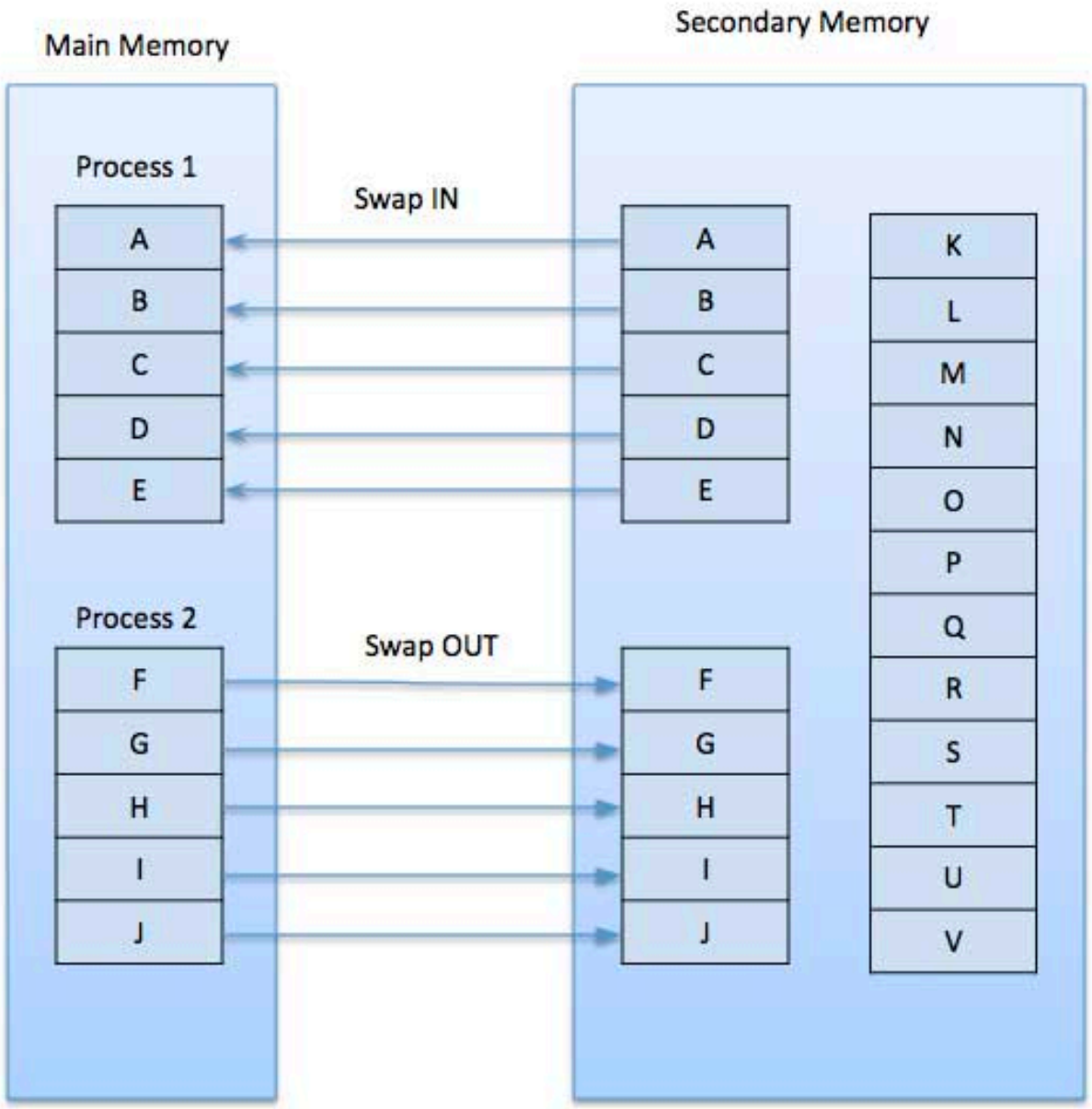
Page number cannot be found in the Page Table

- **If the Page number cannot be found in the Page Table, it is counted as:-**

A- Page lost
B- Page fault
C- Demand page
D- Page miss

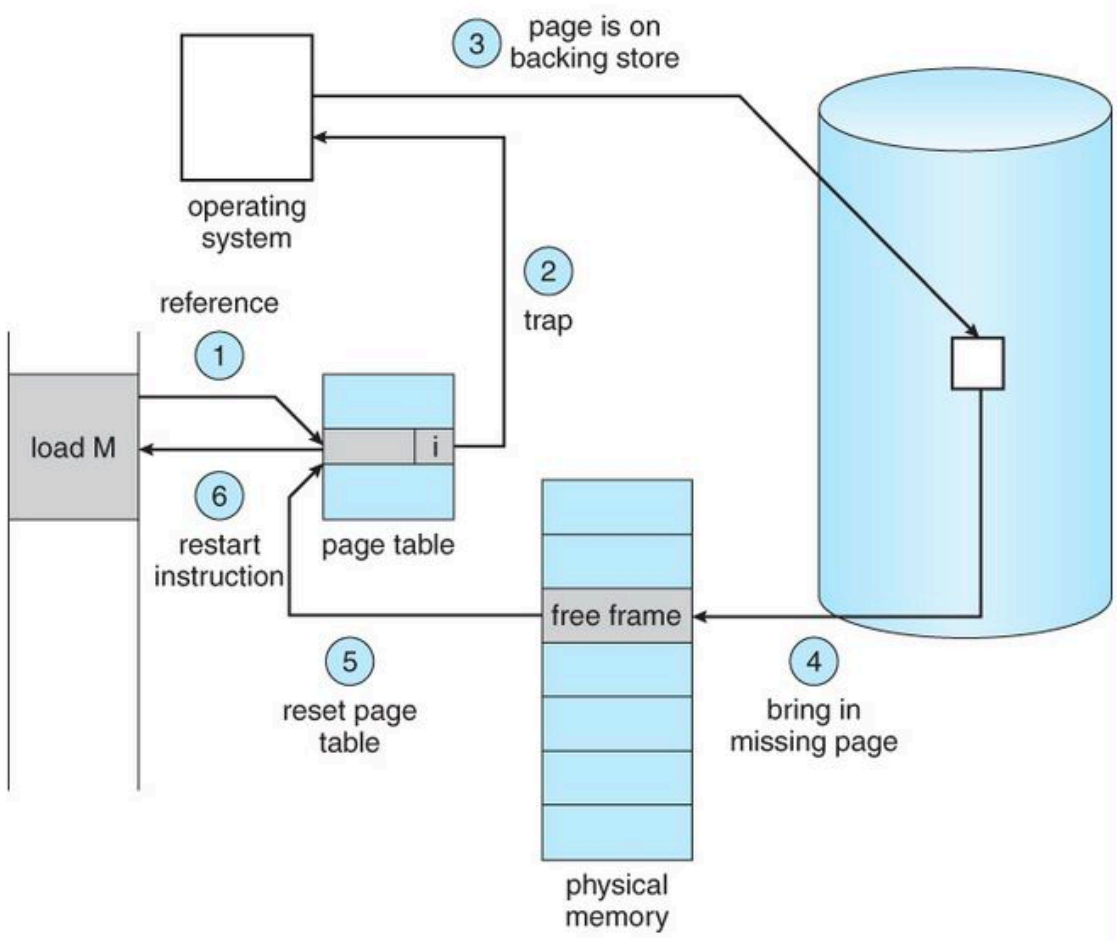
Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance.



Page fault

A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. So when page fault occurs then following sequence of events happens :



- The computer hardware traps to the kernel and program counter (PC) is saved on the stack.
- Current instruction state information is saved in CPU registers..
- Operating system finds that a **page fault has occurred** and tries to find out which virtual page is needed.
- Some times hardware register contains this required information.
- If not, the operating system must retrieve PC, fetch instruction and find out what it was doing when the fault occurred.
- Once virtual address caused page fault is known, system checks to see if address is valid and checks if there is no protection access problem.

- If the virtual address is valid, the system checks to see if a page frame is free.
- If no frames are free, the page replacement algorithm is run to remove a page.
- When disk interrupt indicates page has arrived, page tables are updated to reflect its position, and frame marked as being in normal state.
- Faulting instruction is backed up to state it had when it began and PC is reset.
- Faulting is scheduled, operating system returns to routine that called it.

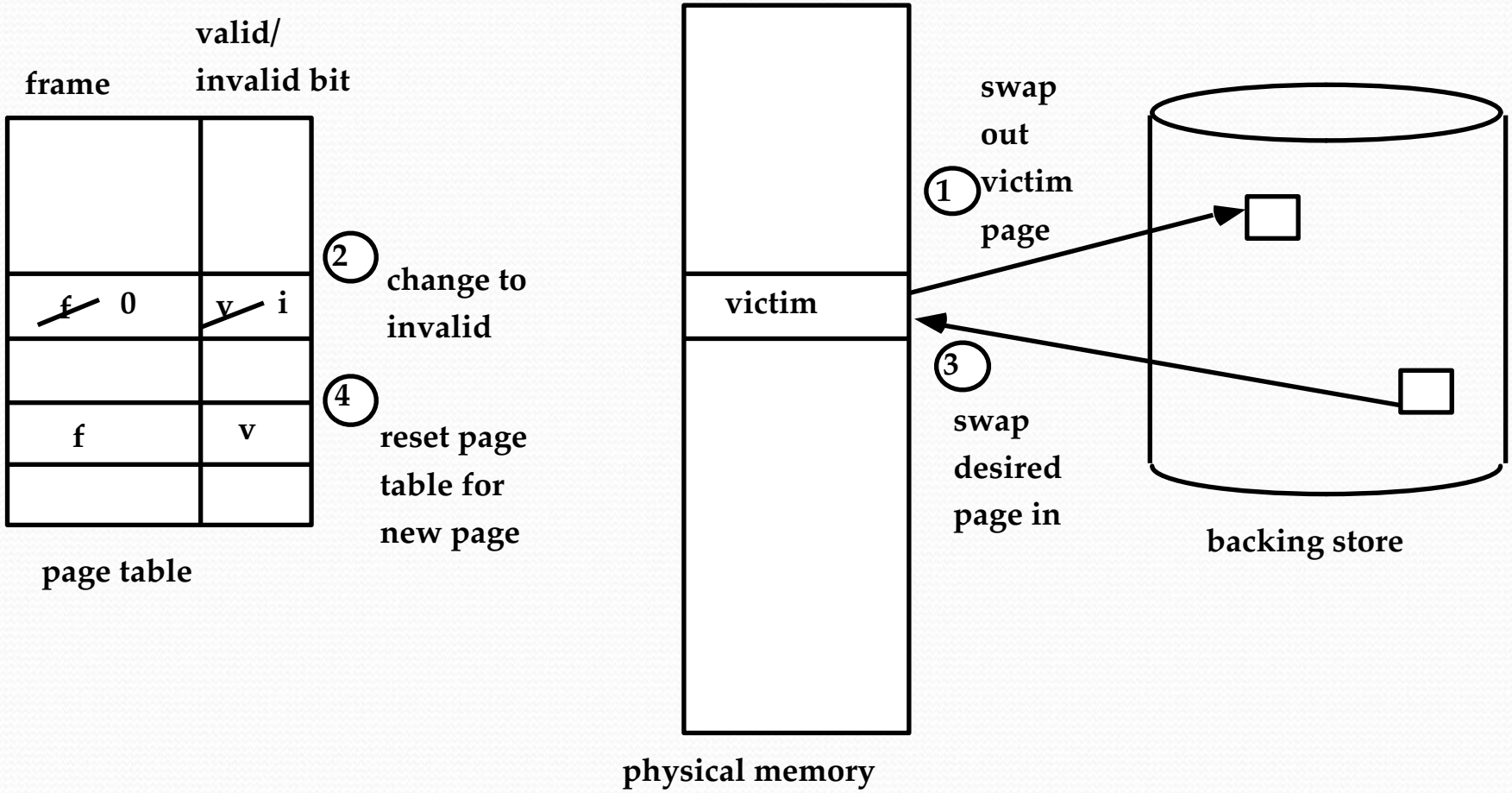
Page Replacement

Decision on which page to displace to make room for an incoming page when no free frame is available

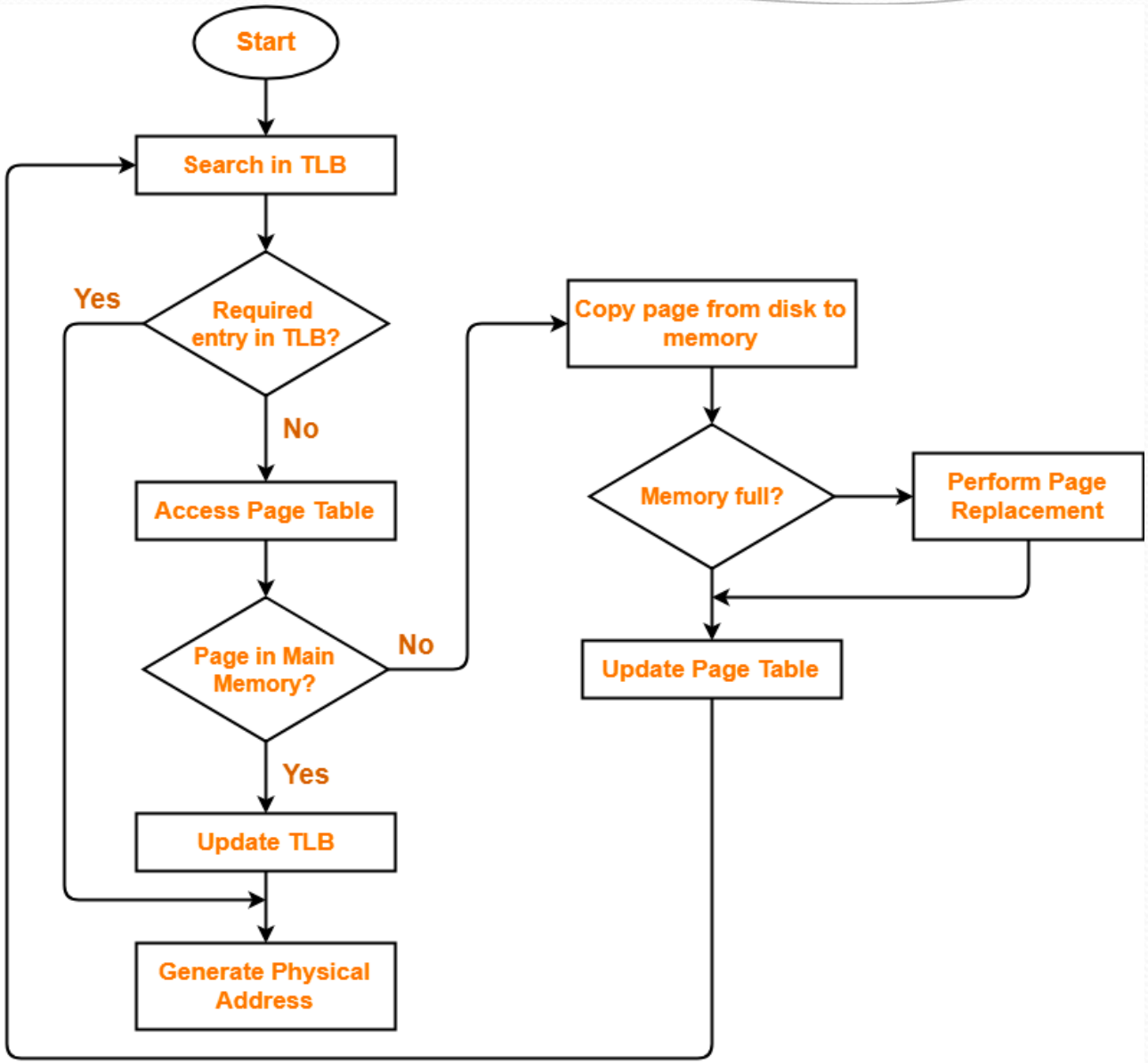
Modified page fault service routine

1. Find the location of the desired page on the backing store
2. Find a free frame
   - If there is a free frame, use it
   - Otherwise, use a page-replacement algorithm to select a *victim* frame
   - Write the victim page to the backing store
3. Read the desired page into the (newly) free frame
4. Restart the user process

Page Replacement



Page Fault/ Page replacement



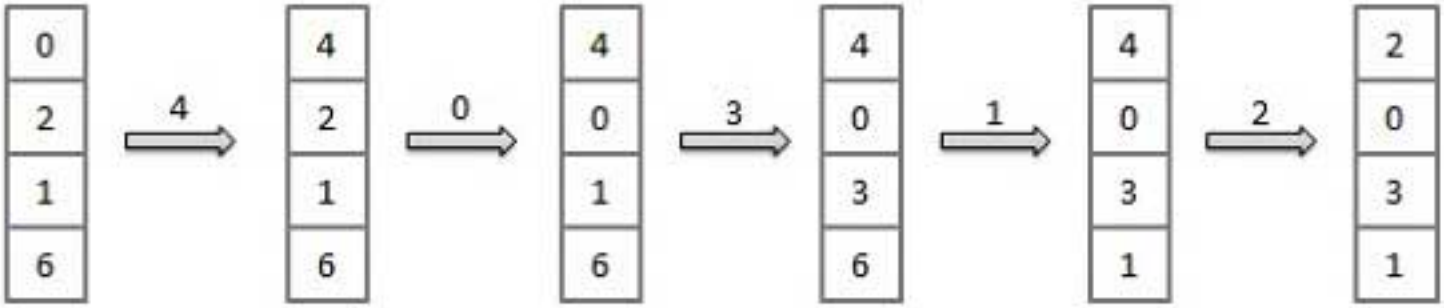
Flowchart

First In First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

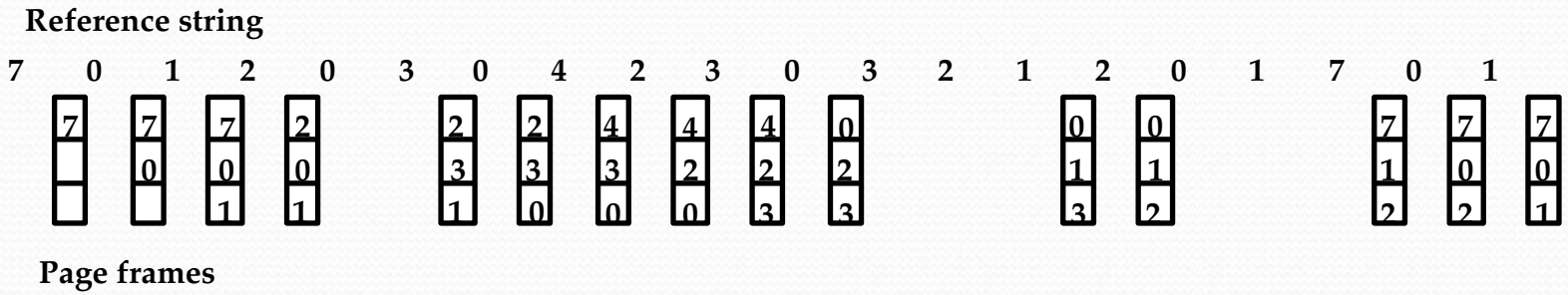
Misses : x x x x x x x x x



Fault Rate = 9 / 12 = 0.75

Page Replacement Algorithms

FIFO



FIFO algorithm selects the page that has been in memory the longest time

Using a queue - every time a page is loaded, its

- identification is inserted in the queue

Easy to implement

May result in a frequent page fault

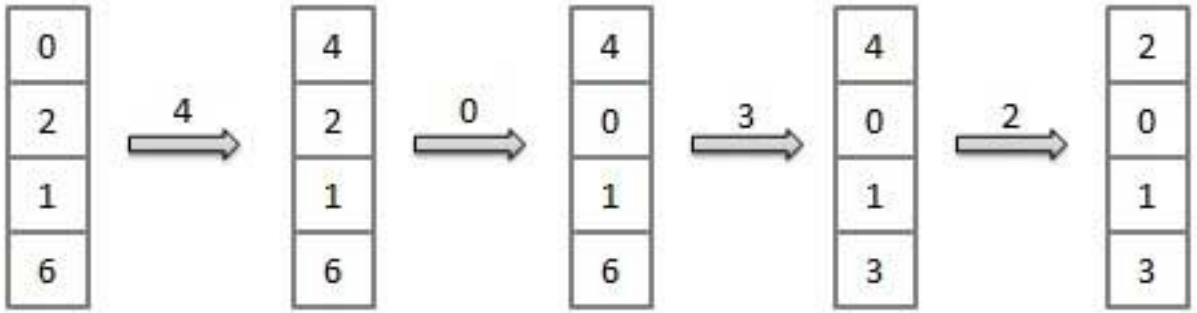
Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.

- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



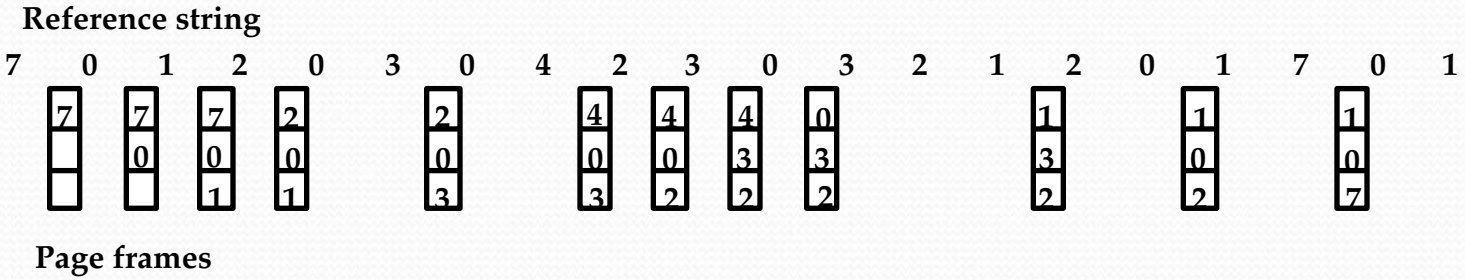
Fault Rate = 8 / 12 = 0.67

Page Replacement Algorithms

LRU

- LRU uses the recent past as an approximation of near future.

Replace that page which has not been used for the longest period of time



- LRU may require substantial hardware assistance
- The problem is to determine an order for the frames defined by the time of last use

Which algorithm chooses the page that has not been used for the longest period of time whenever the page required to be replaced?

a) first in first out algorithm
b) additional reference bit algorithm
c) least recently used algorithm
d) counting based page replacement algorithm

Page reference stream:

1	2	3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4	3
1	1	1	1	1	2	2	3	5	1	6	6	2	5	5	3	3	1	6	2
	2	2	2	2	3	3	5	1	6	2	2	5	3	3	1	1	6	2	4
		3	3	3	5	5	1	6	2	5	5	3	1	1	6	6	2	4	3
*	*	*			*		*	*	*	*		*	*		*		*	*	*

FIFO

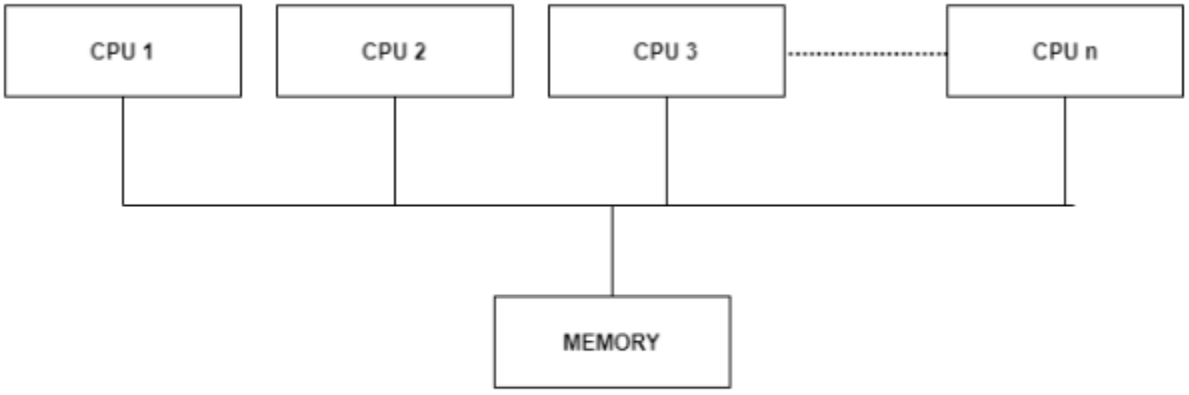
Total 14 page faults

UNIT 6

- **Multiprocessors**
- **Interconnection Structures**
- Parallel Processing
- Pipelining
- Introduction to GPU
- Latest technology and trends in computer architecture
- Multi-cores processor
- Next generation processors architecture
- Microarchitecture
- Latest processor for smartphone or tablet and desktop

Multiprocessors

A shared-memory multiprocessor (or just multiprocessor henceforth) is a computer system in which two or more CPUs share full access to a common RAM.



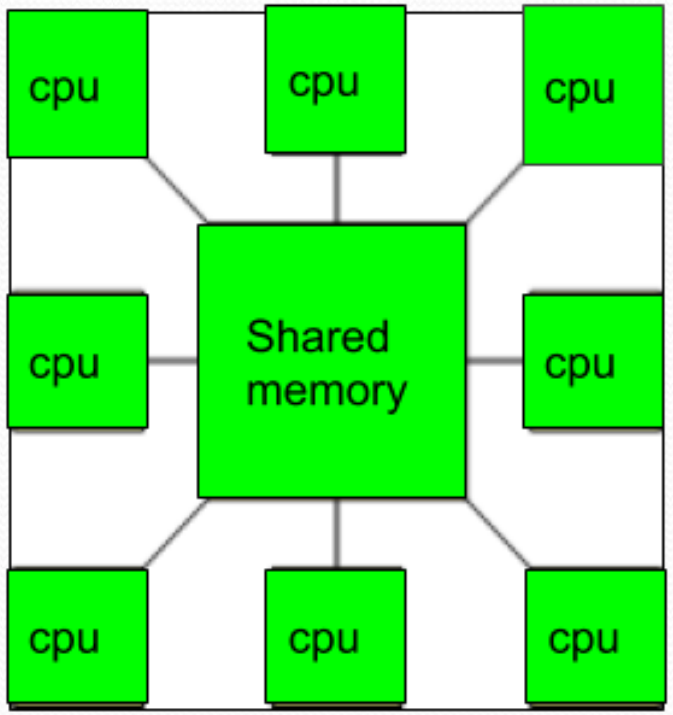
Multiprocessing Architecture

Multiprocessors

A Multiprocessor is a computer system with two or more central processing units (CPUs) share full access to a common RAM.

The main objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.

In shared memory multiprocessors, all the CPUs shares the common memory but in a distributed memory multiprocessor, every CPU has its own private memory.



What you the difference between Multiprocessor and Multicomputer?

Multiprocessor

- A multiprocessor system is simply a computer that has more than one CPU on its motherboard.
- Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system.

Multicomputer

- A computer made up of several computers. The term generally refers to an architecture in which each processor has its own memory rather than multiple processors with a shared memory

The main objective in building the multimicroprocessor is

a.	greater throughput
b.	enhanced fault tolerance
c.	greater throughput and enhanced fault tolerance
d.	none of the mentioned

Types of Multiprocessors

•Symmetric Multiprocessors

➢In these types of systems, each processor contains a similar copy of the operating system and they all communicate with each other.

➢All the processors are in a peer to peer relationship i.e. **no master - slave relationship** exists between them.

➢An example of the symmetric multiprocessing system is the Encore version of Unix for the Multi-max Computer.

•Asymmetric Multiprocessors

➢In asymmetric systems, each processor is given a predefined task.

➢There is a master processor that gives instruction to all the other processors.

➢Asymmetric multiprocessor system **contains a master slave relationship**.

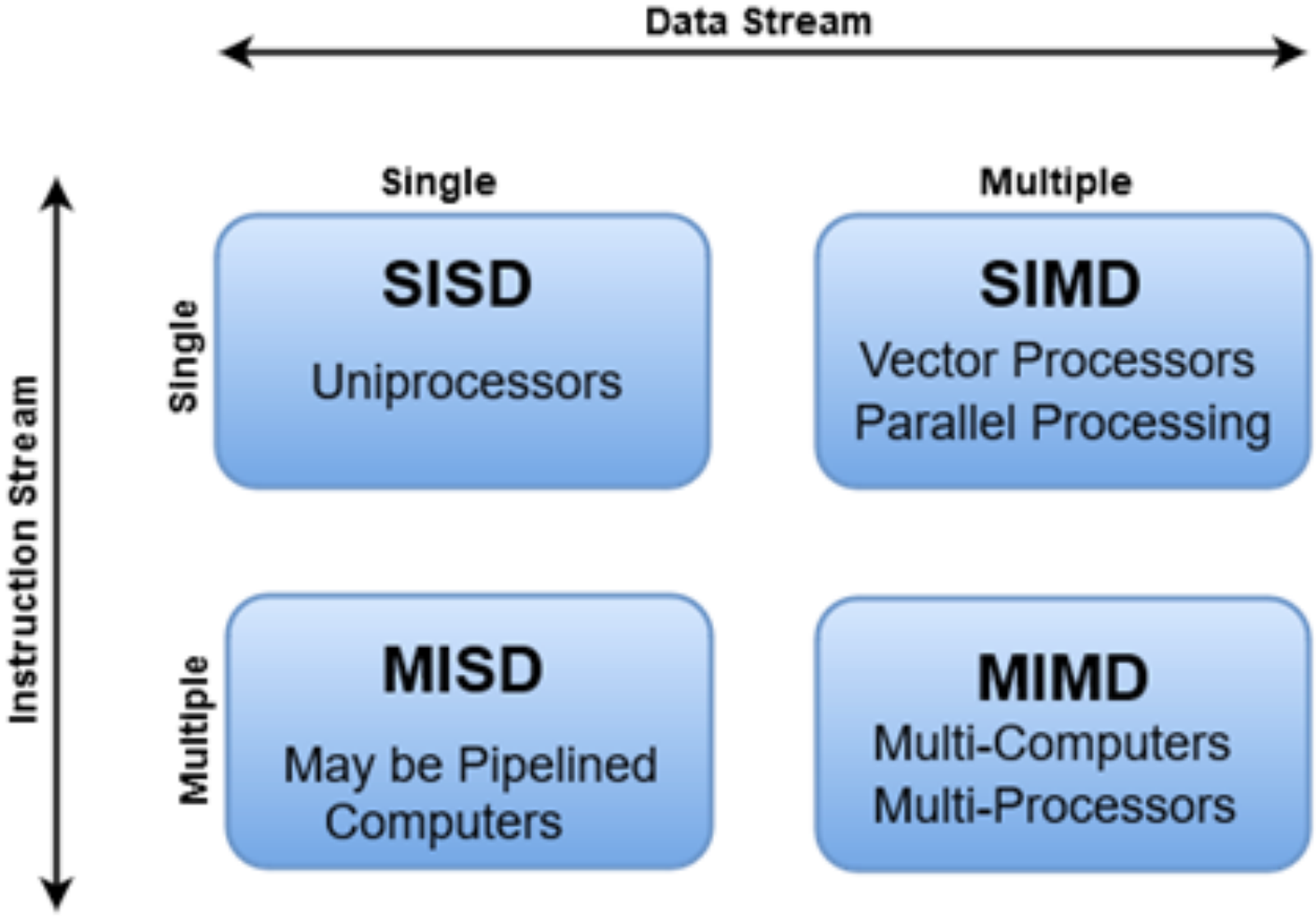
Applications of Multiprocessor –

1. As a uniprocessor, such as single instruction, single data stream (SISD).
2. As a multiprocessor, such as single instruction, multiple data stream (SIMD), which is usually used for vector processing.
3. Multiple series of instructions in a single perspective, such as multiple instruction, single data stream (MISD), which is used for describing hyper-threading or pipelined processors.
4. Inside a single system for executing multiple, individual series of instructions in multiple perspectives, such as multiple instruction, multiple data stream (MIMD).

- Which application of multiprocessor is used for describing hyper-threading or pipelined processors.

A- SISD
B- SIMD
C- MISD
D- MIMD

Flynn's Classification of Computers



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

- **SISD** stands for ***'Single Instruction and Single Data Stream'***.
- It represents the organisation of a single computer containing a control unit, a processor unit, and a memory unit.
- Instructions are executed sequentially, and the system may or may not have internal parallel processing capabilities.
- Most conventional computers have SISD architecture like the traditional Von-Neumann computers.

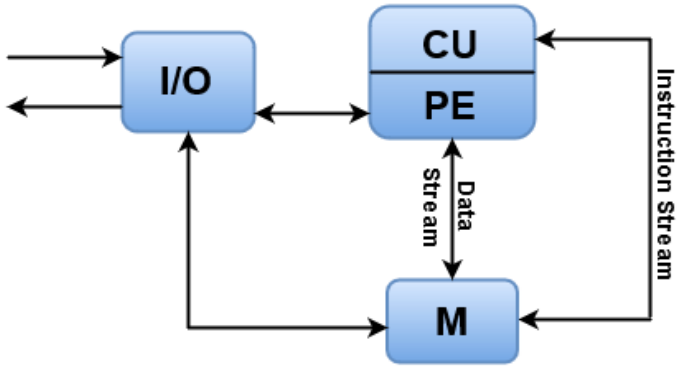
Instructions are decoded by the Control Unit and then the Control Unit sends the instructions to the processing units for execution.

Data Stream flows between the processors and memory bi-directionally.

Examples:

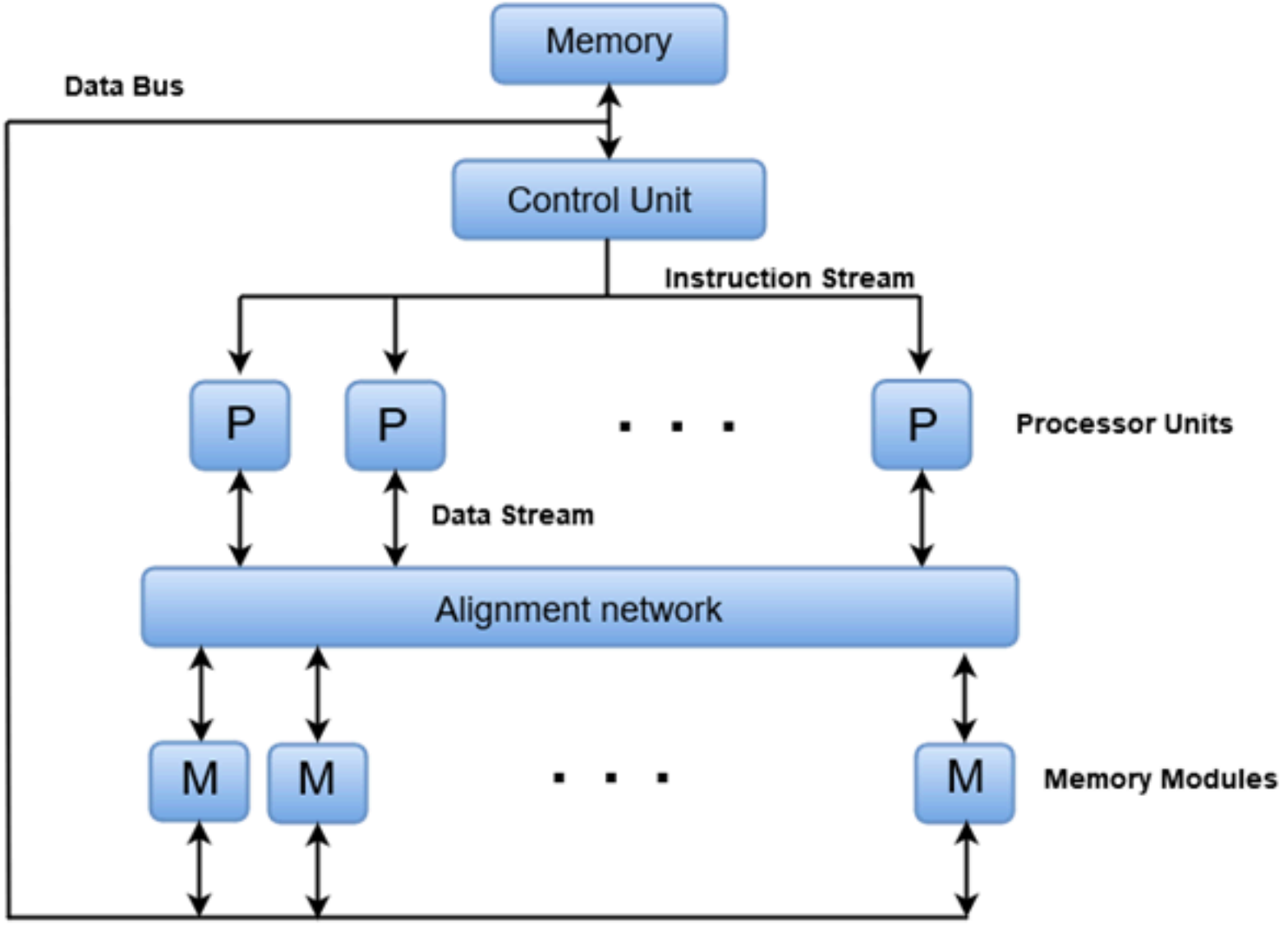
Older generation computers, minicomputers, and workstations

SISD:

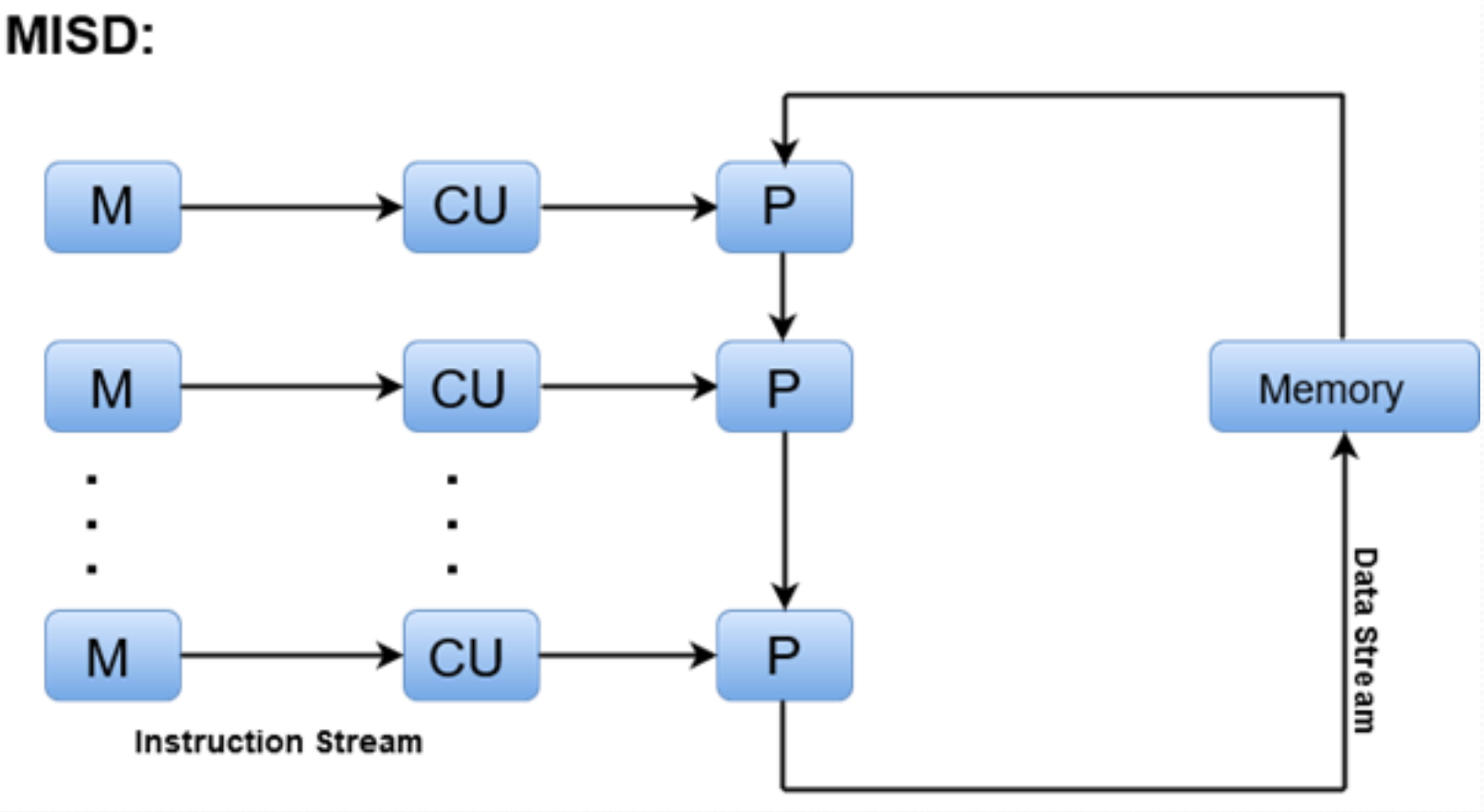


1. Where, CU = Control Unit, PE = Processing Element, M = Memory

SIMD:

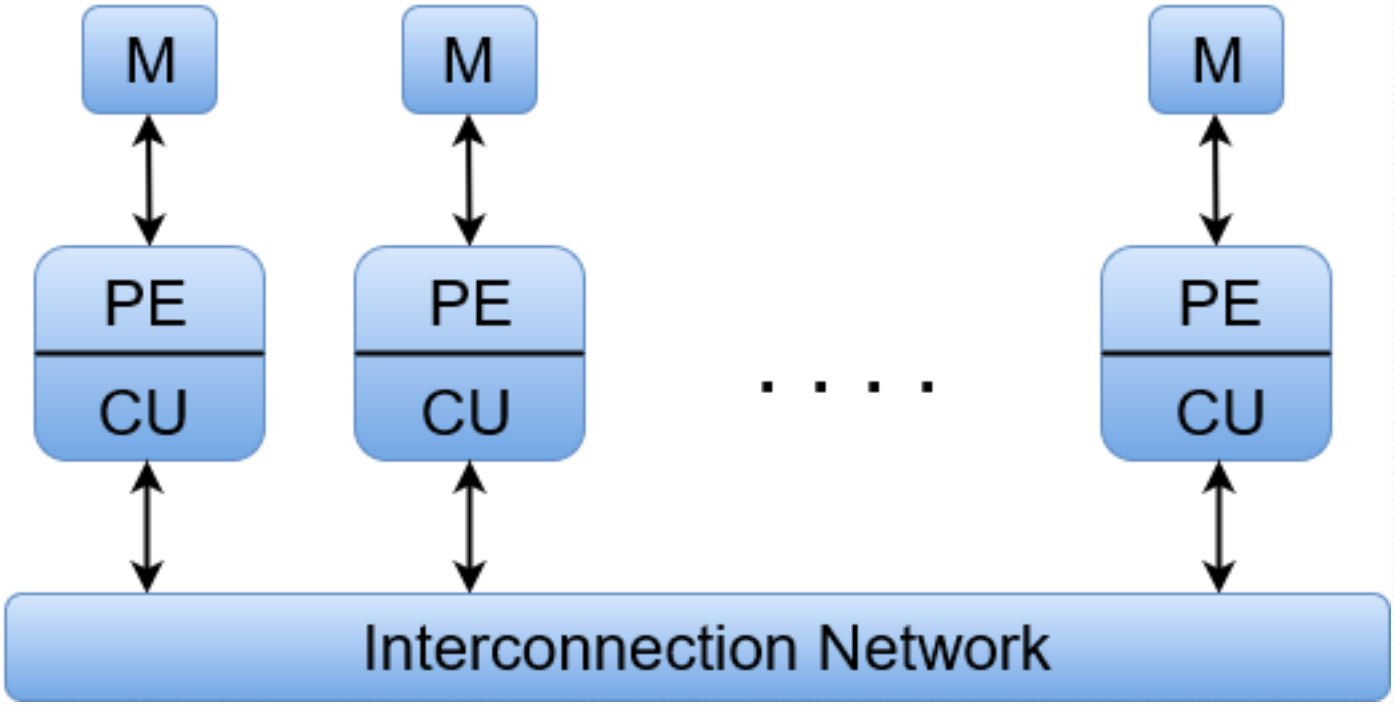

Memory
Data Bus
Control Unit
Instruction Stream
P
P
P
Processor Units
Data Stream
Alignment network
M
M
M
Memory Modules

MISD:



1. Where, M = Memory Modules, CU = Control Unit, P = Processor Units

MIMD:



Where, M = Memory Module, PE = Processing Element, and CU = Control Unit

Which computer system is type of Older generation computers, minicomputers, and workstations?

A-SIMD

B-SISD

C-MISD

D-MIMD

Advantages of Multiprocessor Systems

- **More reliable Systems:-** In a multiprocessor system, even if one processor fails, the system will not halt. This ability to continue working despite hardware failure is known as graceful degradation.

- **Enhanced Throughput:-** If multiple processors are working in tandem, then the throughput of the system increases i.e. number of processes getting executed per unit of time increase. If there are N processors then the throughput increases by an amount just under N.

- **More Economic Systems:-** Multiprocessor systems are cheaper than single processor systems in the long run because they share the data storage, peripheral devices, power supplies etc.

Disadvantages of Multiprocessor Systems

❑**Increased Expense:-** Even though multiprocessor systems are cheaper in the long run than using multiple computer systems, still they are quite expensive. It is much cheaper to buy a simple single processor system than a multiprocessor system.

❑**Complicated Operating System Required:-** There are multiple processors in a multiprocessor system that share peripherals, memory etc. So, it is much more complicated to schedule processes and impart resources to processes, than in single processor systems. Hence, a more complex and complicated operating system is required in multiprocessor systems.

❑**Large Main Memory Required:-** All the processors in the multiprocessor system share the memory. So a much larger pool of memory is required as compared to single processor systems.

How multiprocessor are classified?

- Multiprocessor are classified by the way their memory is organized, mainly it is classified into two types

1. Tightly coupled multiprocessor
2. Loosely coupled multiprocessor

Coupling of Processors

Tightly Coupled System

- Tasks and/or processors communicate in a highly synchronized fashion
- Communicates through a common shared memory
- Shared memory system

Loosely Coupled System

- Tasks or processors do not communicate in a synchronized fashion
- Communicates by message passing packets
- Overhead for data exchange is high
- Distributed memory system

Tightly coupled Multiprocessor

- A **multiprocessor** is a tightly coupled computer system having two or more processing units (**Multiple Processors**) each sharing main memory and peripherals, in order to simultaneously process programs

- Tightly coupled Multiprocessor is also know as shared memory system

Loosely-coupled multiprocessor

- Loosely-coupled multiprocessor systems (often referred to as clusters) are based on multiple standalone single or dual processor commodity computers interconnected via a high speed communication system.
- Loosely-coupled multiprocessor is also known as distributed memory.
- Example

  A Linux beowulf cluster

Difference b/w Tightly coupled and Loosely coupled multiprocessor

Tightly coupled

- Tightly-coupled systems physically smaller than loosely-coupled system.
- More expensive .

Loosely coupled

- It is just opposite of tightly coupled system.
- Less expensive.

4. In tightly coupled systems, the microprocessors share

a) common clock

b) bus control logic

c) common clock and bus control logic

d) none of the mentioned

What is another name of tightly coupled multiprocessor?

(A) Distributed memory processors
(B) Mutually coupled processors
(C) Binding memory processors
(D) Shared memory processors

Interconnection Structure

- The components that form a multiprocessor system are CPUs, IOPs connected to input-output devices, and a memory unit.
- The interconnection between the components can have different physical configurations, depending on the number of transfer paths that are available between the processors and memory in a shared memory system .

There are several physical forms available for establishing an interconnection network.

1o Time-shared common bus
2o Multiport memory
3o Crossbar switch
4o Multistage switching network
5o Hypercube system

In shared bus architecture, the required processor(s) to perform a bus cycle, for fetching data or instructions is

a) one processor
b) two processors
c) more than two processors
d) none of the mentioned

1.Time Shared Common Bus

A common-bus multiprocessor system consists of a number of processors connected through a common path to a memory unit.

o Only one processor can communicate with the memory or another processor at any given time.

o As a consequence, the total overall transfer rate within the system is limited by the speed of the single path

A more economical implementation of a dual bus structure is depicted in Fig. below.

Part of the local memory may be designed as a *cache memory attached to the CPU.*

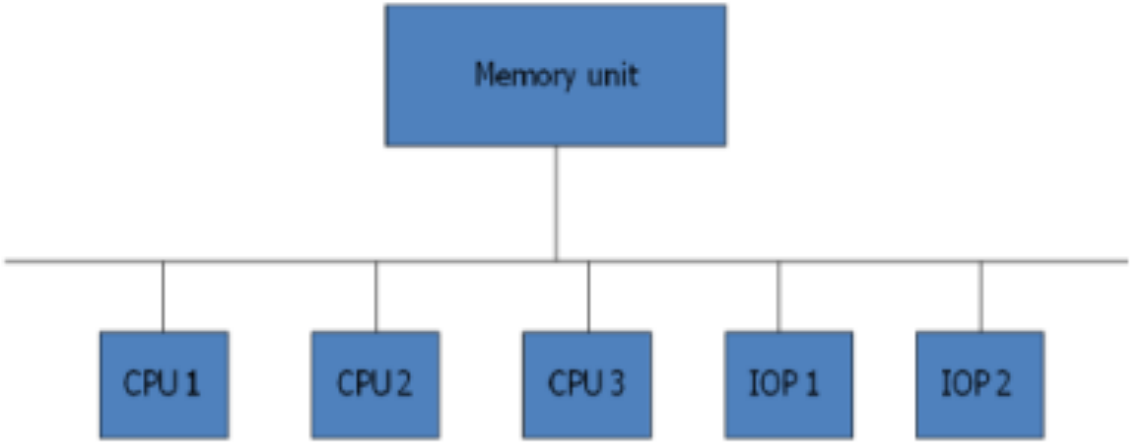


Fig: Time shared common bus organization

2.Multiport Memory

A multiport memory system employs separate buses between each memory module and each CPU.

The module must have internal control logic to determine which port will have access to memory at any given time.

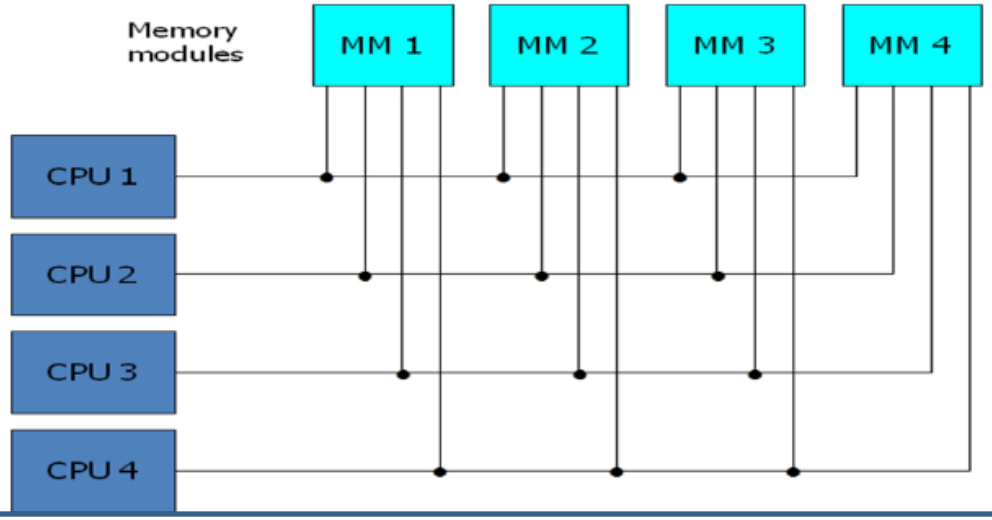
Memory access conflicts are resolved by assigning fixed priorities to each memory port.

Adv.:

o The high transfer rate can be achieved because of the multiple paths.

Disadv.:

o It requires expensive memory control logic and a large number of cables and connections



3.Cross Bar Switch

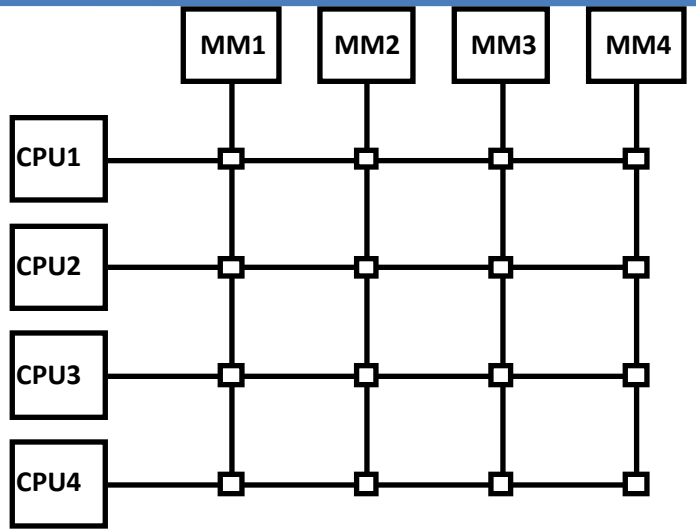


fig. shows the functional design of a crossbar switch connected to one memory module.

Consists of a number of *crosspoints that are placed at intersections between processor buses and memory module paths.*

The small square in each crosspoint is a *switch that determines the path from a processor to a memory module.*

Adv.:

o Supports simultaneous transfers from all memory modules

Disadv.:

o The hardware required to implement the switch can become quite large and complex.

In Cross bar switch , it consists of a number of _____ *that are placed at intersections between processor buses and memory module paths.*

A- BAR points

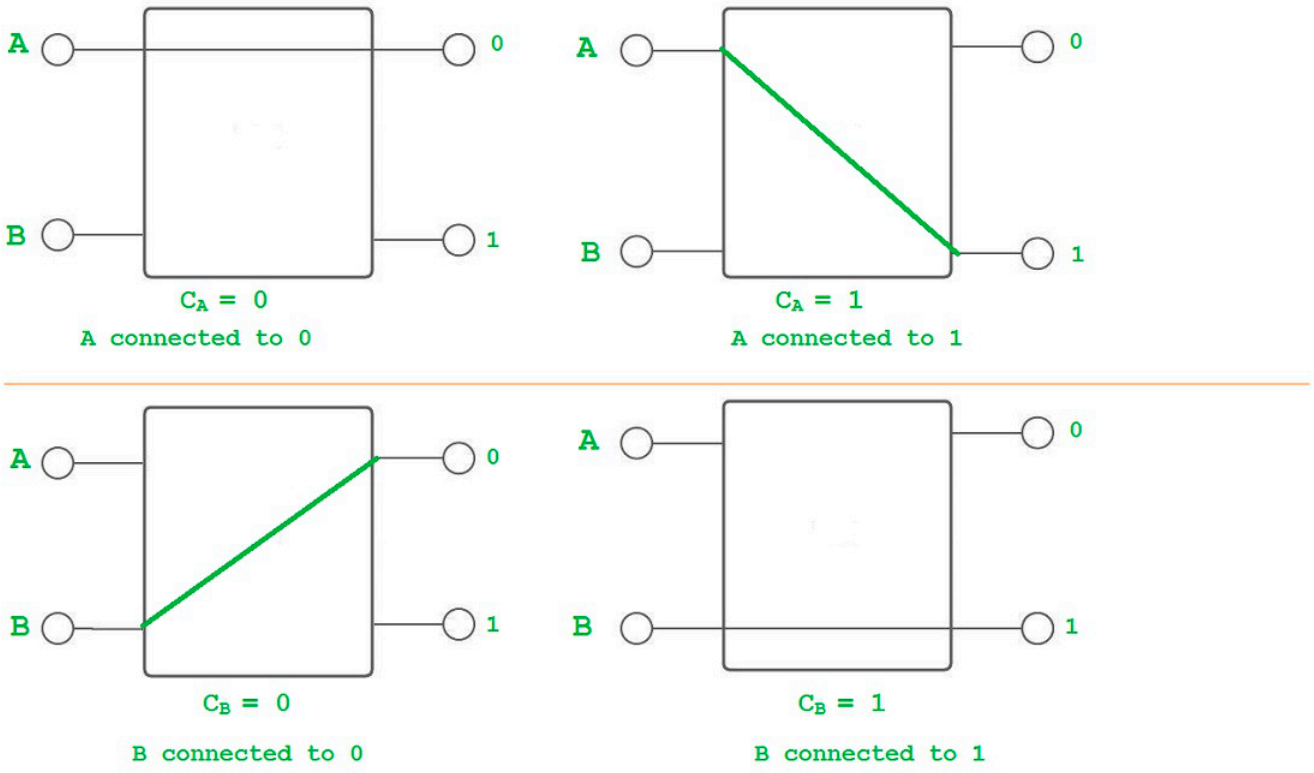
B- cross points

C- cross nets

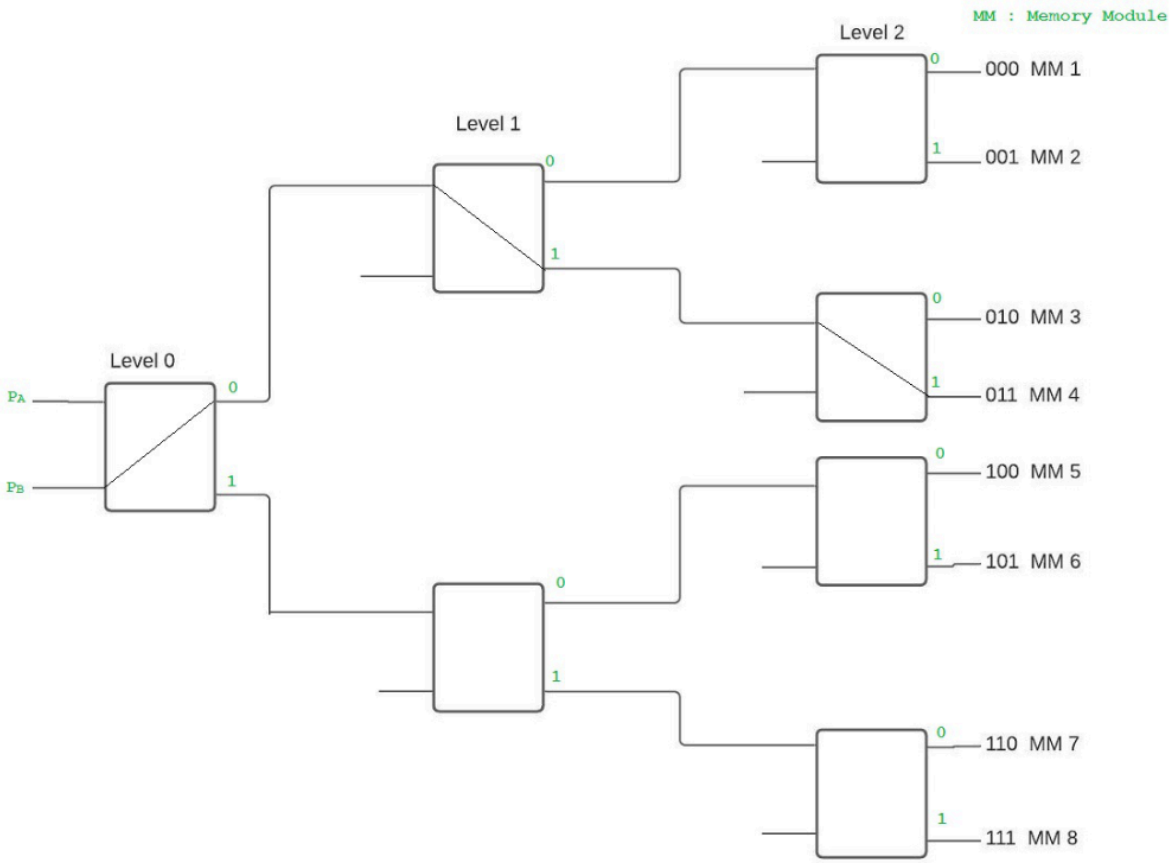
D- path points

Multistage Switching Network :

The 2×2 crossbar switch is used in the multistage network. It has 2 inputs (A & B) and 2 outputs (0 & 1). To establish the connection between the input & output terminals, the control inputs C_A & C_B are associated.



- In the diagram, P_A & P_B are 2 processors, and they are connected to 8 memory modules in a binary way from 000(0) to 111(7) through switches. Three levels are there from a source to a destination. To choose output in a level, one bit is assigned to each of the 3 levels. There are 3 bits in the destination number: 1st bit determines the output of the switch in 1st level, 2nd bit in 2nd level & 3rd bit in the 3rd level.

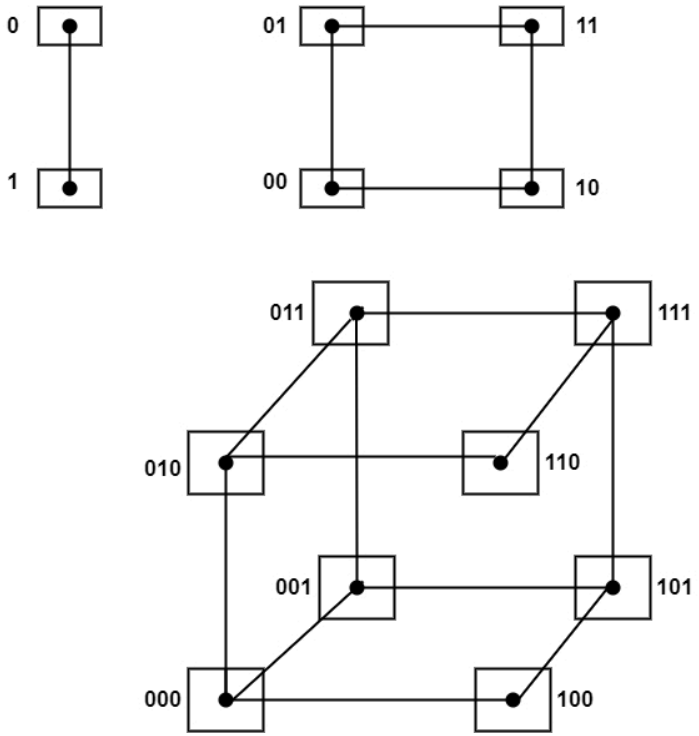


*1 to 8 way switch using 2*2 Switch*

The hypercube interconnection is also defined as a binary n-cube multiprocessor. The hypercube is treated to be a loosely coupled system. This system is composed of $N = 2^n$ processors that are linked in an n-dimensional binary cube. Each processor denotes a node of the cube.

A one-cube structure includes $n = 1$ and $2^n = 2$. It has two processors that are connected by an individual route. A two-cube structure includes $n = 2$ and $2^n = 4$. It has four nodes that are linked as a square. There are eight nodes associated as a cube in a three-cube structure.

Hypercube Structures for n = 1, 2, and 3



Multiprocessor architectures

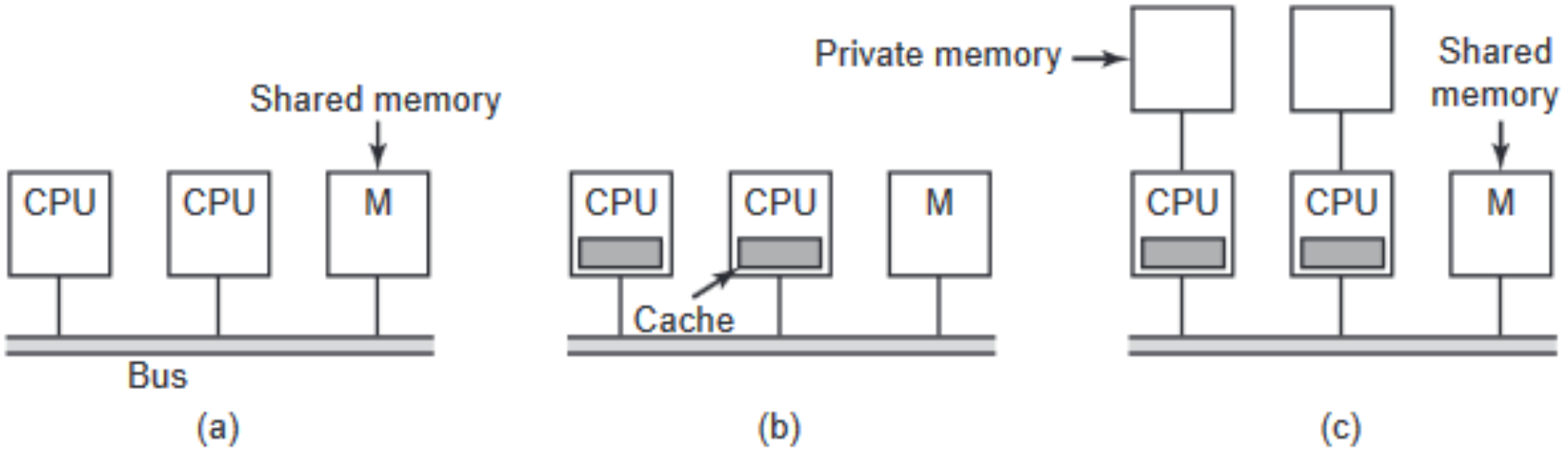


Figure 8-1. Three bus-based multiprocessors. (a) Without caching. (b) With caching. (c) With caching and private memories.

(a) Two or more CPUs and one or more memory modules all use the same bus for communication. When a CPU wants to read a memory word, it first checks to see if the bus is busy. If the bus is idle, the CPU puts the address of the word it wants on the bus, asserts a few control signals, and waits until the memory puts the desired word on the bus.

(b) Problem to previous approach, If the bus is busy when a CPU wants to read or write memory, the CPU just waits until the bus becomes idle. The cache can be inside the CPU chip, next to the CPU chip, on the processor board, or some combination of all three. Since many reads can now be satisfied out of the local cache, there will be much less bus traffic, and the system can support more CPUs.

(c) In this each CPU has not only a cache, but also a local, private memory which it accesses over a dedicated (private) bus. To use this configuration optimally, the compiler should place all the program text, strings, constants and other read-only data, stacks, and local variables in the private memories. The shared memory is then only used for writable shared variables.

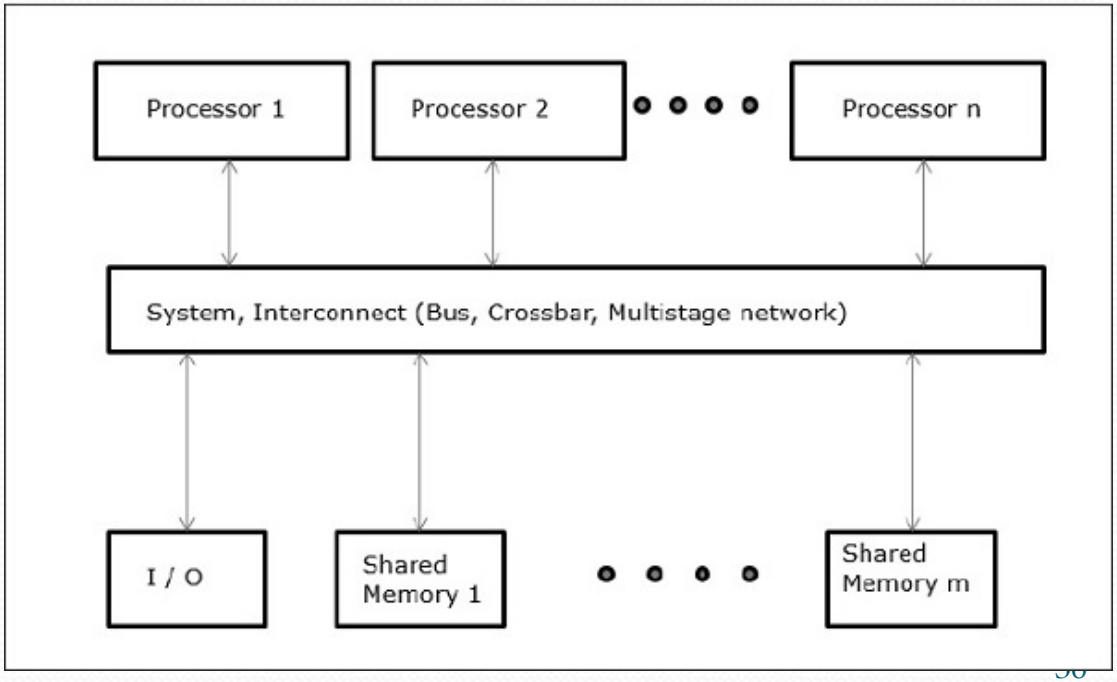
Shared-Memory Multicomputers

Three most common shared memory multiprocessors models are –

Uniform Memory Access (UMA)

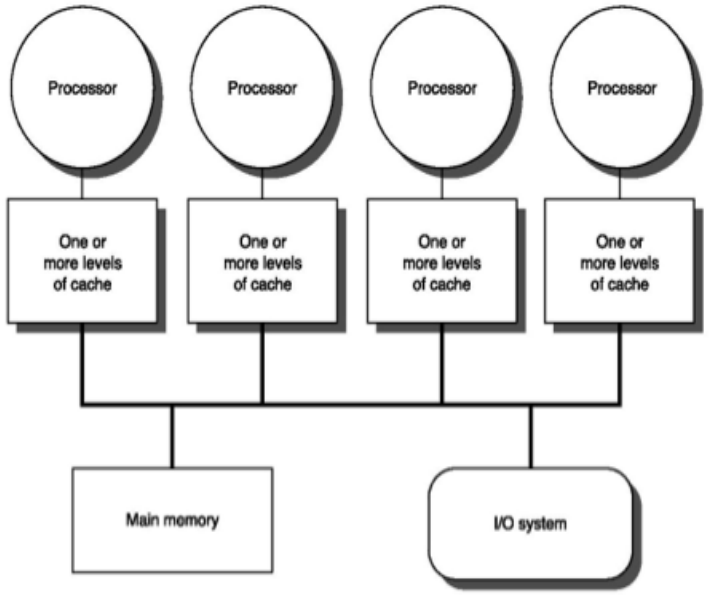
In this model, all the processors share the physical memory uniformly. All the processors have equal access time to all the memory words. Each processor may have a private cache memory. Same rule is followed for peripheral devices.

When all the processors have equal access to all the peripheral devices, the system is called a **symmetric multiprocessor**. When only one or a few processors can access the peripheral devices, the system is called an **asymmetric multiprocessor**.



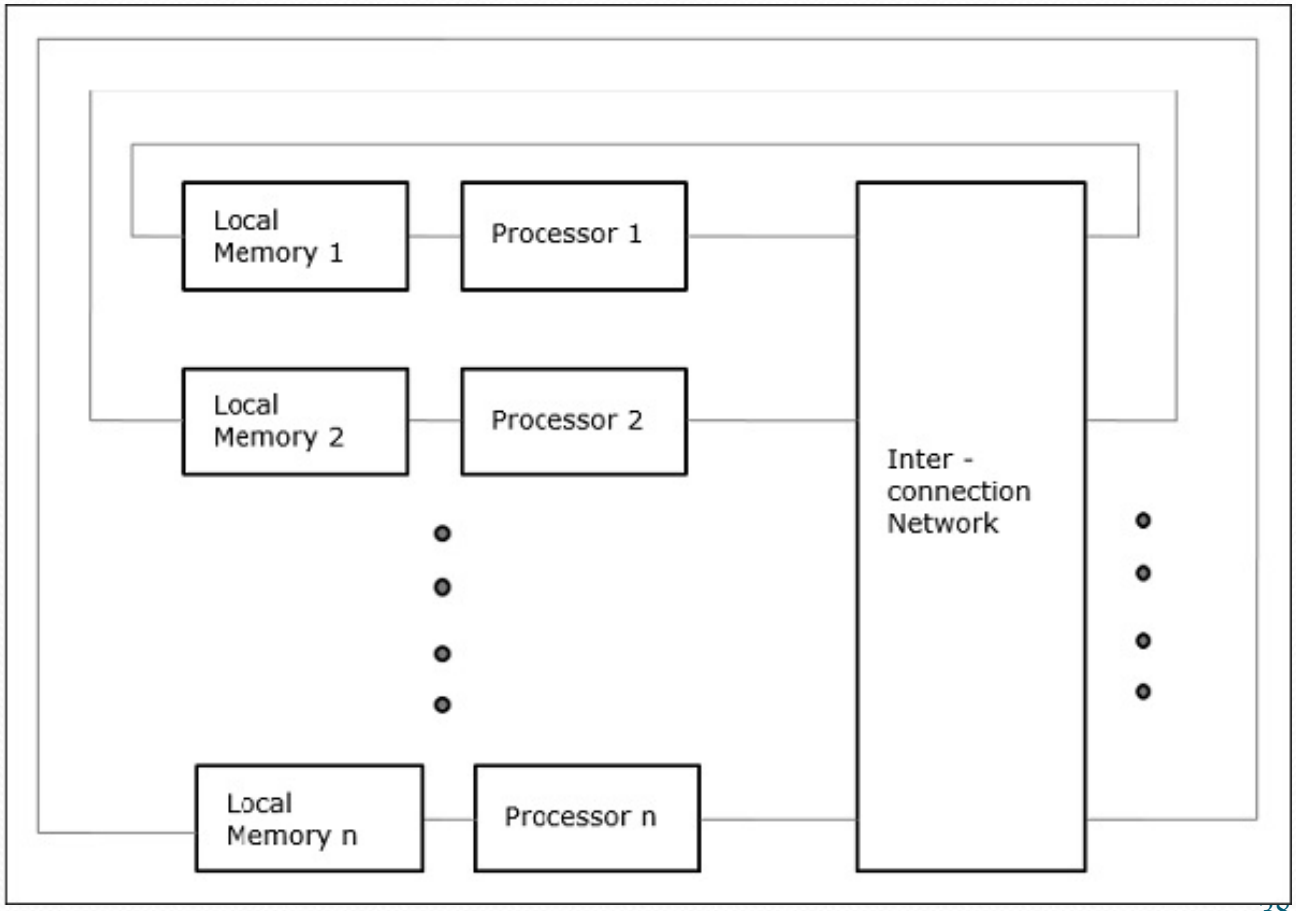
1. **Uniform Memory Access (UMA)**: the name of this type of architecture hints to the fact that all processors share a unique centralized primary memory, so *each CPU has the same memory access time.*

- Owing to this architecture, these systems are also called *Symmetric Shared-memory Multiprocessors (SMP)* (Hennessy-patterson, Fig. 6.1)



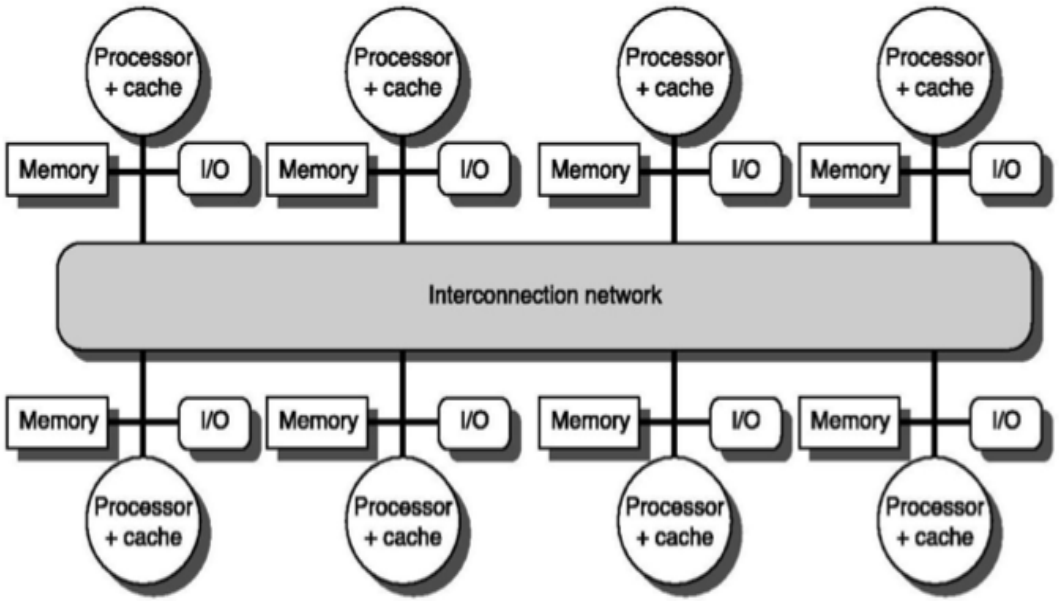
Non-uniform Memory Access (NUMA)

In NUMA multiprocessor model, the access time varies with the location of the memory word. Here, the shared memory is physically distributed among all the processors, called local memories. The collection of all local memories forms a global address space which can be accessed by all the processors.



2. **Non Uniform Memory Access (NUMA)**: these systems have a shared logical address space, but physical memory is *distributed* among CPUs, so *that access time to data depends on data position, in local or in a remote memory* (thus the NUMA denomination)

- These systems are also called *Distributed Shared Memory (DSM)* architectures (Hennessy-Patterson, Fig. 6.2)



In which of the following shared memory multiprocessors models, All the processors have equal access time to all the memory words.

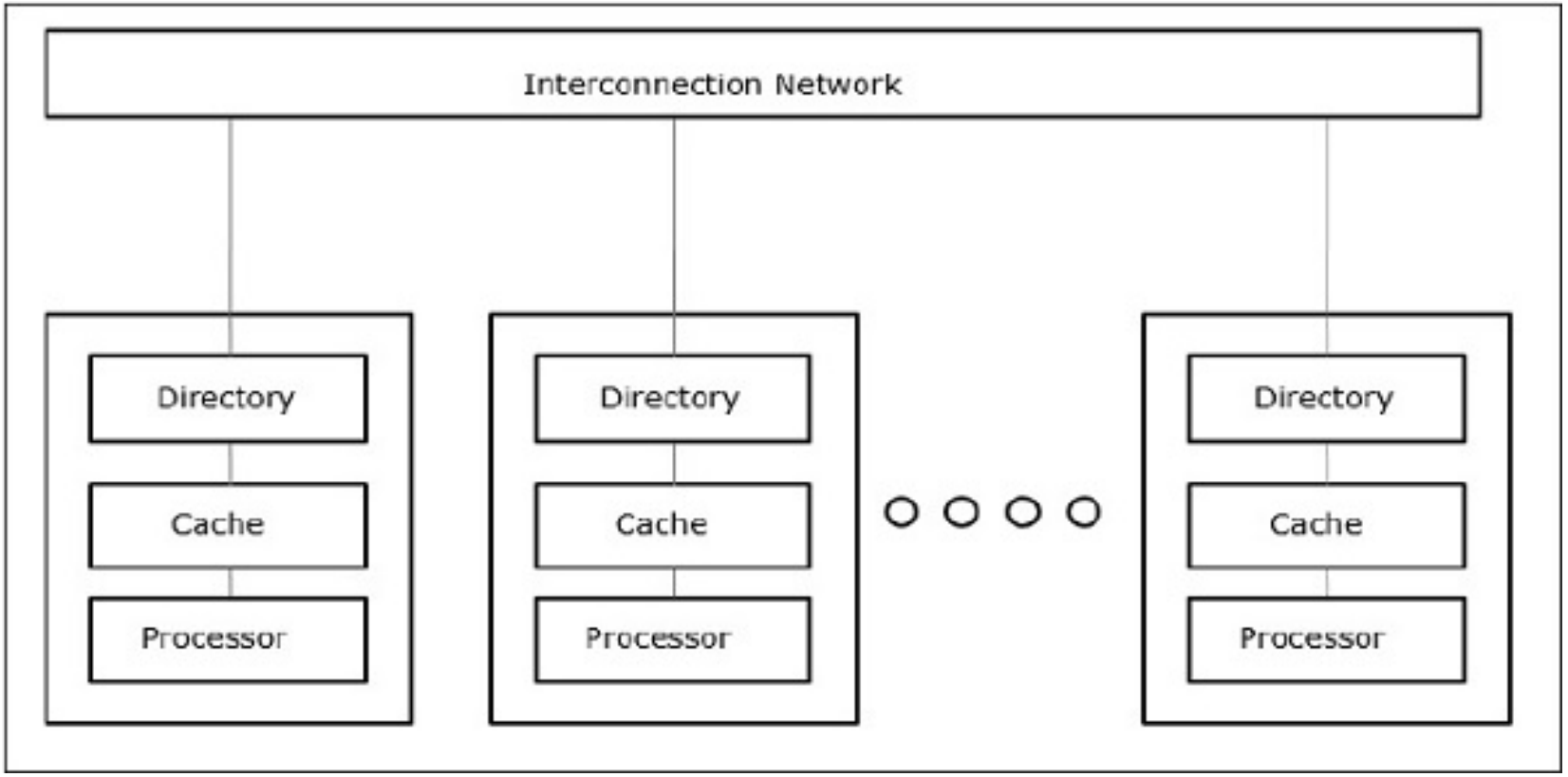
A- NUMA
B-COMA
C-UMA
D- None of these

Comparison Chart

Basis for comparison	UMA	NUMA
Basic	Uses a single memory controller	Multiple memory controller
Type of buses used	Single, multiple and crossbar.	Tree and hierarchical
Memory accessing time	Equal	Changes according to the distance of microprocessor.
Suitable for	General purpose and time-sharing applications	Real-time and time-critical applications
Speed	Slower	Faster
Bandwidth	Limited	More than UMA.

Cache Only Memory Architecture (COMA)

The COMA model is a special case of the NUMA model. Here, all the distributed main memories are converted to cache memories.



3. **Cache Only Memory Access (COMA)**: data have no specific "permanent" location (no specific memory address) where they stay and whence they can be read (copied into local caches) and/or modified (first in the cache and then updated at their "permanent" location).

- Data can migrate and/or can be replicated in the various memory banks of the central main memory.

In which of the shared memory multiprocessor , memory is physically distributed among all the processors, called local memories.

A- UMA
B- NUMA
C- COMA
D- None of these

UNIT 6

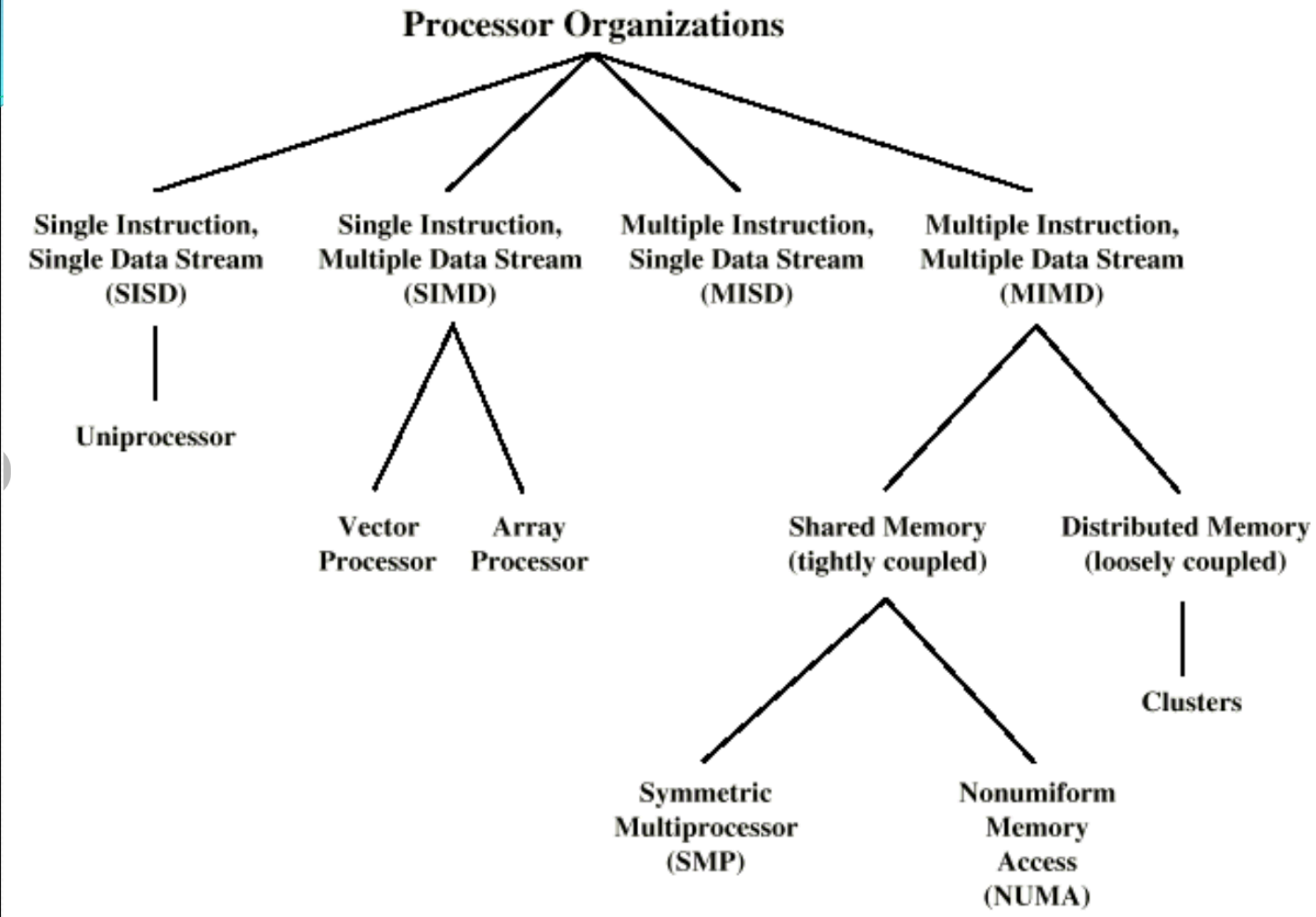
- **Parallel Processing**
- **Pipelining**

Several instructions execution simultaneously in _____

a) processing
b) parallel processing
c) serial processing
d) multitasking

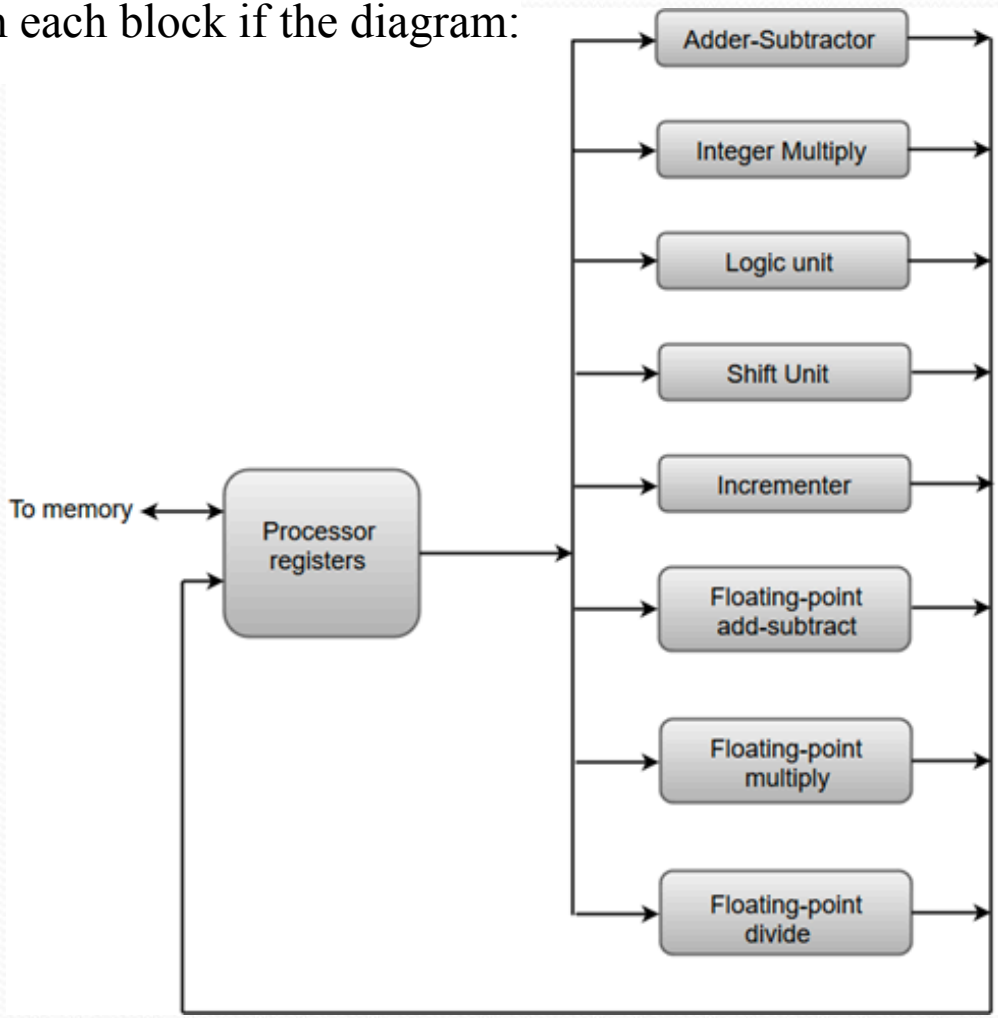
•Parallel Processing

- **Parallel processing** is a method in **computing** of running two or more **processors** (CPUs) to handle separate parts of an overall task.
- Breaking up different parts of a task among multiple **processors** will help reduce the amount of time to run a program.
- Instead of processing each instruction sequentially, a **parallel processing** system provides concurrent data processing to increase the execution time.
- In this the system may have two or more ALU's and should be able to execute two or more instructions at the same time.
- The purpose of parallel processing is to speed up the computer processing capability and increase its throughput.
- **NOTE: Throughput** is the number of instructions that can be executed in a unit of time.


Processor Organizations
Single Instruction, Single Data Stream (SISD)
Single Instruction, Multiple Data Stream (SIMD)
Multiple Instruction, Single Data Stream (MISD)
Multiple Instruction, Multiple Data Stream (MIMD)
Uniprocessor
Vector Processor
Array Processor
Shared Memory (tightly coupled)
Distributed Memory (loosely coupled)
Clusters
Symmetric Multiprocessor (SMP)
Nonumiform Memory Access (NUMA)

•Parallel Processing

- The following diagram shows one possible way of separating the execution unit into eight functional units operating in parallel .The operation performed in each functional unit is indicated in each block if the diagram:



•Parallel Processing

- The **adder and integer multiplier** performs the arithmetic operation with integer numbers.
- The **floating-point operations** are separated into three circuits operating in parallel.
- The **logic, shift, and increment operations** can be performed concurrently on different data.
- All units are independent of each other, so one number can be shifted while another number is being incremented.

Types of parallelism :—

Data Parallelism

Data Parallelism means concurrent execution of the same task on each multiple computing core.

Task Parallelism

Task Parallelism means concurrent execution of the different task on multiple computing cores.

Bit-level parallelism

E.g., consider a case where an 8-bit processor must add two 16-bit integers. First the 8 lower-order bits from each integer were must added by processor, then add the 8 higher-order bits, and then two instructions to complete a single operation. A processor with 16-bit would be able to complete the operation with single instruction.

Instruction-level parallelism

Instruction-level parallelism means the simultaneous execution of multiple instructions from a program.

Pipelining

To improve the performance of a CPU we have two options:

1) Improve the hardware by introducing faster circuits.

2) Arrange the hardware such that more than one operation can be performed at the same time.

Since, there is a limit on the speed of hardware and the cost of faster circuits is quite high, we have to adopt the 2nd option.

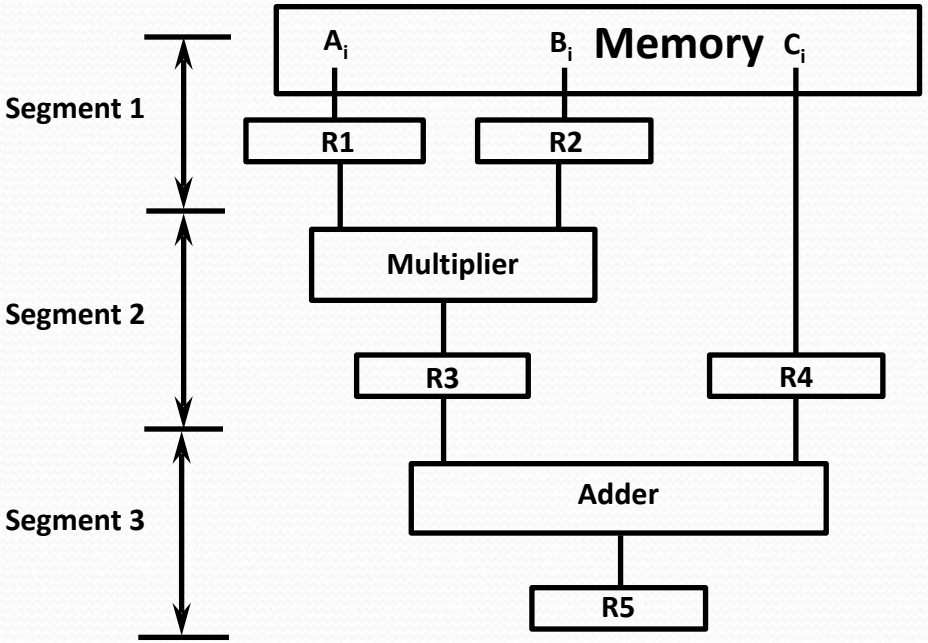
Pipelining : Pipelining is a process of arrangement of hardware elements of the CPU such that its overall performance is increased. Simultaneous execution of more than one instruction takes place in a pipelined processor.

A technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.

Pipelining

Simplest way to understand pipelining is to imagine that each segment consist of input register followed by combinational circuit. The o/p of combinational circuit in a segment is applied to i/p register of next segment

$A_i * B_i + C_i$ for i = 1, 2, 3, ... , 7



$R1 \leftarrow A_i$, $R2 \leftarrow B_i$	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

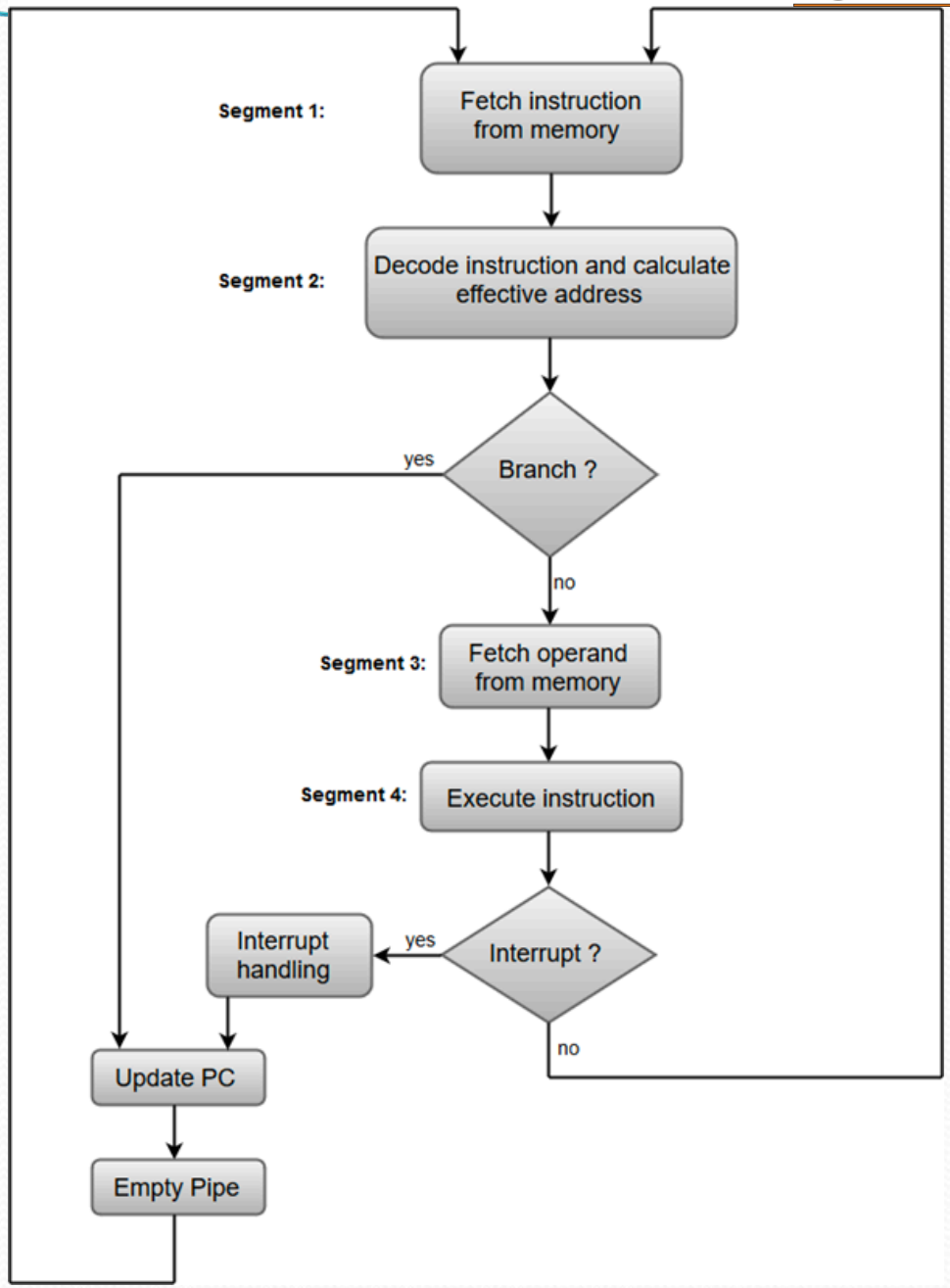
Instruction Pipeline

- Pipeline processing can occur not only in the data stream but in the instruction stream as well.
- Most of the digital computers with complex instructions require instruction pipeline to carry out operations like fetch, decode and execute instructions.
- In general, the computer needs to process each instruction with the following sequence of steps:-

1. **Fetch instruction from memory.**
2. **Decode the instruction.**
3. **Calculate the effective address.**
4. **Fetch the operands from memory.**
5. **Execute the instruction.**
6. **Store the result in the proper place.**

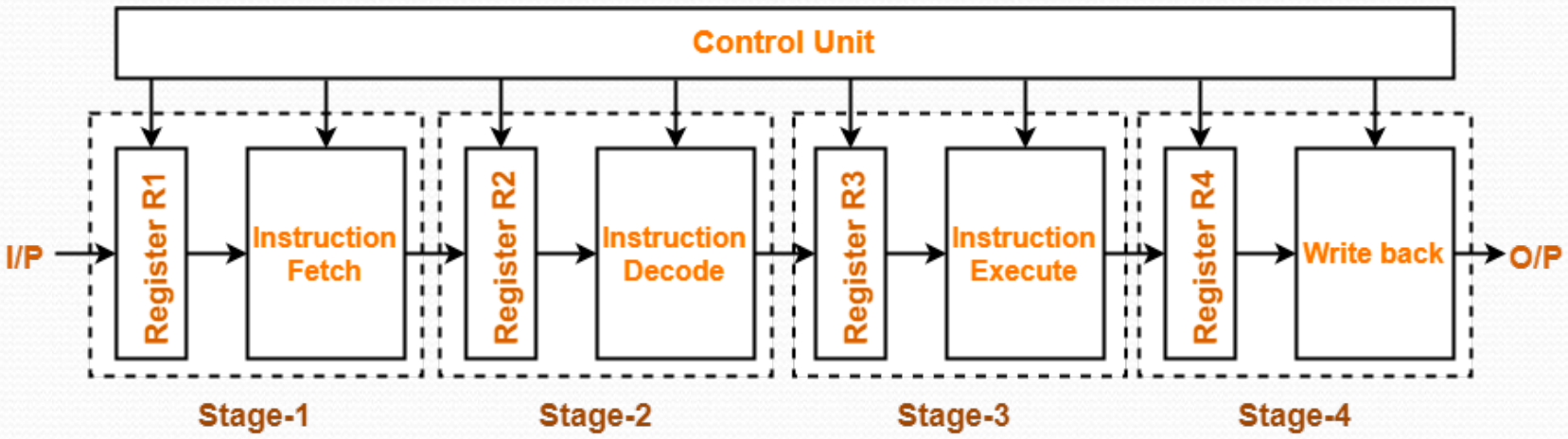
The organisation of an instruction pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

One of the most common examples of this type of organisation is a **Four-segment instruction pipeline.**



In four stage pipelined architecture, the execution of each instruction is completed in following 4 stages:-

- Instruction fetch (IF)
- Instruction decode (ID)
- Instruction Execute (IE)
- Write back (WB)
- **To implement four stage pipeline,**
- The hardware of the CPU is divided into four functional units.
- Each functional unit performs a dedicated task.



Four Stage Instruction Pipeline

Stage-01:

At stage-01,

- First functional unit performs instruction fetch.
- It fetches the instruction to be executed.

Stage-02:

At stage-02,

- Second functional unit performs instruction decode.
- It decodes the instruction to be executed.

Stage-03:

At stage-03,

- Third functional unit performs instruction execution.
- It executes the instruction.

Stage-04:

At stage-04,

- Fourth functional unit performs write back.
- It writes back the result so obtained after executing the instruction.

Pipeline implement

A. fetch instruction
B. decode instruction
C. fetch operand
D. calculate operand
E. execute instruction
F. all of above

Execution in a pipelined processor

Execution sequence of instructions in a pipelined processor can be visualized using a space-time diagram. For example, consider a processor having 4 stages and let there be 2 instructions to be executed. We can visualize the execution sequence through the following space-time diagrams:

Non overlapped execution:

Total time = 8 Cycle

STAGE / CYCLE	1	2	3	4	5	6	7	8
S1	I_1				I_2			
S2		I_1				I_2		
S3			I_1				I_2	
S4				I_1				I_2

Execution-

In pipelined architecture,

- Instructions of the program execute parallelly.
- When one instruction goes from n^{th} stage to $(n+1)^{th}$ stage, another instruction goes from $(n-1)^{th}$ stage to n^{th} stage.

Overlapped execution:

Total time = 5 Cycle

STAGE / CYCLE	1	2	3	4	5
S1	I_1	I_2			
S2		I_1	I_2		
S3			I_1	I_2	
S4				I_1	I_2

Pipeline Stages

RISC processor has 5 stage instruction pipeline to execute all the instructions in the RISC instruction set. Following are the 5 stages of RISC pipeline with their respective operations:

Stage 1 (Instruction Fetch)

In this stage the CPU reads instructions from the address in the memory whose value is present in the program counter.

Stage 2 (Instruction Decode)

In this stage, instruction is decoded and the register file is accessed to get the values from the registers used in the instruction.

Stage 3 (Instruction Execute)

In this stage, ALU operations are performed.

Stage 4 (Memory Access)

In this stage, memory operands are read and written from/to the memory that is present in the instruction.

Stage 5 (Write Back)

In this stage, computed/fetched value is written back to the register present in the instruction.

In non-pipelined architecture,

Time taken to execute six instructions would be

= 6 x Time taken to execute one instruction

= 6*4 clock cycles

= 24 clock cycles

Performance of a pipelined processor

Calculating Cycle Time-

In pipelined architecture,

- Cycle time is the value of one clock cycle.

There are two cases possible-

Case-01: All the stages offer same delay-

If all the stages offer same delay, then-

Cycle time = Delay offered by one stage including the delay due to its register

Case-02: All the stages do not offer same delay-

If all the stages do not offer same delay, then-

Cycle time = Maximum delay offered by any stage including the delay due to its register

Calculating Frequency Of Clock-

Frequency of the clock (f) = 1 / Cycle time

Calculating Non-Pipelined Execution Time-

In non-pipelined architecture,

- The instructions execute one after the other.
- The execution of a new instruction begins only after the previous instruction has executed completely.
- So, number of clock cycles taken by each instruction = k clock cycles

Thus,

Non-pipelined execution time

= Total number of instructions x Time taken to execute one instruction

= n x k clock cycles

Calculating Pipelined Execution Time-

In pipelined architecture,

- Multiple instructions execute parallelly.
- Number of clock cycles taken by the first instruction = k clock cycles
- After first instruction has completely executed, one instruction comes out per clock cycle.
- So, number of clock cycles taken by each remaining instruction = 1 clock cycle

Thus,

Pipelined execution time

= Time taken to execute first instruction + Time taken to execute remaining instructions

= 1 x k clock cycles + (n-1) x 1 clock cycle

= (k + n – 1) clock cycles

Calculating Speed Up-

speedup (S) of the pipelined processor over non-pipelined processor, when 'n' tasks are executed on the same processor is:

S = Performance of pipelined processor / Performance of Non-pipelined processor.

As the performance of a processor is inversely proportional to the execution time, we have

Speed up

= Non-pipelined execution time / Pipelined execution time

= n x k clock cycles / (k + n – 1) clock cycles

= n x k / (k + n – 1)

When the number of tasks 'n' are significantly larger than k, that is, n >> k i.e F o r very large number of instructions, n→∞. Thus, speed up = k.

S = n * k / n

S = k

where 'k' are the number of stages in the pipeline.

Also, **Efficiency** = Given speed up / Max speed up = S / Smax

We know that, Smax = k

So, **Efficiency = S / k**

Throughput = Number of instructions / Total time to complete the instructions

So, **Throughput** = n / (k + n – 1) * Tp

Throughput is defined as number of instructions executed per unit time.

Q—-The 5 stages of the processor have the following latencies:

FETCH	DECODE	EXECUTE	MEMORY	WRITEBACK
300s	400s	350s	550s	100s
200s	150s	100s	190s	140s

- Assume that when pipelining, each pipeline stage costs 20ps extra for the registers between pipeline stages.
- **Non-pipelined processor:** what is the cycle time? What is the latency of an instruction? What is the throughput?
- **Pipelined processor:** What is the cycle time? What is the latency of an instruction? What is the throughput?

Non-pipelined processor:

what is the cycle time? What is the latency of an instruction? What is the throughput?

Because there is no pipelining, the cycle time must allow an instruction to go through all stages in one cycle. The latency is the same as cycle time since it takes the instruction one cycle to go from the beginning of fetch to the end of writeback. The throughput is defined as 1/CT inst/s.

- CT = 300 + 400 + 350 + 550 + 100 = 1700s
- Latency = 1700s
- Throughput = 1/1700 inst/s

- CT = 200 + 150 + 100 + 190 + 140 = 780s
- Latency = 780s
- Throughput = 1/780 inst/s

Pipelined processor:

What is the cycle time? What is the latency of an instruction? What is the throughput?

Pipelining reduces the cycle time to the length of the longest stage plus the register delay. Latency becomes CT*N where N is the number of stages as one instruction will need to go through each of the stages and each stage takes one cycle. The throughput formula remains the same.

- CT = 550 + 20 = 570 s
- Latency = 5 * 570 = 2850s
- Throughput = 1/570 inst/s

- CT = 200 + 20 = 220 s
- Latency = 5 * 220 = 1100s
- Throughput = 1/220 inst/s

Pipelining problems

- Consider a pipeline having 4 phases with duration 60, 50, 90 and 80 ns. Given latch delay is 10 ns. Calculate-

1. Pipeline cycle time
2. Non-pipeline execution time
3. Speed up ratio
4. Pipeline time for 1000 tasks
5. Sequential time for 1000 tasks
6. Throughput

1. Cycle time

= Maximum delay due to any stage + Delay due to its register

= Max { 60, 50, 90, 80 } + 10 ns

= 90 ns + 10 ns

= 100 ns

2.

Non-pipeline execution time for one instruction

= 60 ns + 50 ns + 90 ns + 80 ns

= 280 ns

3.

Speed up

= Non-pipeline execution time / Pipeline execution time

= 280 ns / Cycle time

= 280 ns / 100 ns

= 2.8

4.

Pipeline time for 1000 tasks

= Time taken for 1st task + Time taken for remaining 999 tasks

= 1 x 4 clock cycles + 999 x 1 clock cycle

= 4 x cycle time + 999 x cycle time

= 4 x 100 ns + 999 x 100 ns

= 400 ns + 99900 ns

= 100300 ns

5.

Non-pipeline time for 1000 tasks

= 1000 x Time taken for one task

= 1000 x 280 ns

= 280000 ns

6.

Throughput for pipelined execution

= Number of instructions executed per unit time

= 1000 tasks / 100300 ns

- Consider a pipeline having 4 phases with duration 40, 50, 70 and 80 ns. Given latch delay is 20 ns. Calculate-

1. Pipeline cycle time
2. Non-pipeline execution time
3. Speed up ratio
4. Pipeline time for 5000 tasks
5. Sequential time for 5000 tasks
6. Throughput