UNIT – 6

PART – 1 : INPUT-OUTPUT ORGANIZATION

Introduction:

Input/Output Subsystem:

The I/O subsystem of a computer provides an efficient mode of communication between the central system and the outside environment. It handles all the input-output operations of the computer system.

Peripheral Devices:

Input or output devices that are connected to computer are called **peripheral devices**. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be the part of computer system. These devices are also called **peripherals**.

For example: *Keyboards*, *display units* and *printers* are common peripheral devices.

There are three types of peripherals:

1. **Input peripherals**: Allows user input, from the outside world to the computer.

   Example: Keyboard, Mouse etc.

2. **Output peripherals**: Allows information output, from the computer to the outside world.

   Example: Printer, Monitor etc

3. **Input-Output peripherals**: Allows both input (from outside world to computer) as well as, output (from computer to the outside world).

   Example: Touch screen etc.

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

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

Interfaces:

Interface is a shared boundary between two separate components of the computer system which can be used to attach two or more components to the system for communication purposes.

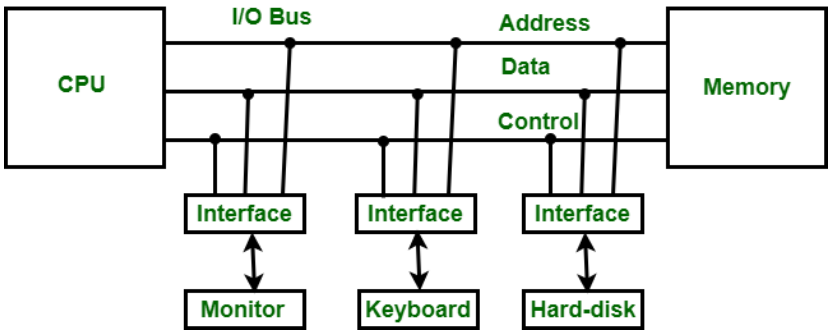
There are two types of interface:

1. CPU Inteface
2. I/O Interface

Let's understand the I/O Interface in details,

Input-Output Interface:

Input-Output Interface is used as a method which helps in transferring of information between the internal storage devices i.e. memory and the external peripheral device. A peripheral device is that which provide input and output for the computer, it is also called Input-Output devices. For Example: A keyboard and mouse provide Input to the computer are called input devices while a monitor and printer that provide output to the computer are called output devices. Just like the external hard-drives, there is also availability of some peripheral devices which are able to provide both input and output.



Input-Output Interface

In micro-computer base system, the only purpose of peripheral devices is just to provide **special communication links** for the interfacing them with the CPU. To resolve the differences between peripheral devices and CPU, there is a special need for communication links.

The major differences are as follows:

1. The nature of peripheral devices is electromagnetic and electro-mechanical. The nature of the CPU is electronic. There is a lot of difference in the mode of operation of both peripheral devices and CPU.
2. There is also a synchronization mechanism because the data transfer rate of peripheral devices are slow than CPU.

3. In peripheral devices, data code and formats are differ from the format in the CPU and memory.
4. The operating mode of peripheral devices are different and each may be controlled so as not to disturb the operation of other peripheral devices connected to CPU.

There is a special need of the additional hardware to resolve the differences between CPU and peripheral devices to supervise and synchronize all input and output devices.

Functions of Input-Output Interface:

1. It is used to synchronize the operating speed of CPU with respect to input-output devices.
2. It selects the input-output device which is appropriate for the interpretation of the input-output signal.
3. It is capable of providing signals like control and timing signals.
4. In this data buffering can be possible through data bus.
5. There are various error detectors.
6. It converts serial data into parallel data and vice-versa.
7. It also convert digital data into analog signal and vice-versa.

Structure of Input-Output Interface:

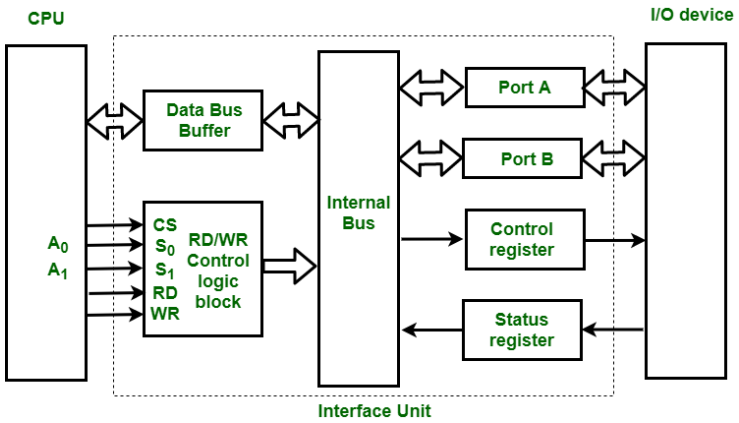
The block diagram of an Input-Output Interface unit contains the following blocks:

1. Data Bus Buffer

2. Read/Write Control Logic

3. Port A, Port B register

4. Control and Status register



These are explained as following below -

Data Bus Buffer: The bus buffer use bi-directional data bus to communicate with CPU. All control word data and status information between interface unit and CPU are transferred through data bus.

Port A and Port B: Port A and Port B are used to transfer data between Input-Output device and Interface Unit. Each port consist of bi-directional data input buffer and bi-directional data output buffer. Interface unit connect directly with an input device and output disk or with device that require both input and output through Port A and Port B i.e. modem, external hard-drive, magnetic disk.

Control and Status Register: CPU gives control information to control register on basis of control information. Interface unit control input and output operation between CPU and input-output device. Bits which are present in status register are used for checking of status conditions. Status register indicate status of data register, port A, port B and also record error that may be occur during transfer of data.

Read/Write Control Logic: This block generates necessary control signals for overall device operations. All commands from CPU are accepted through this block. It also allow status of interface unit to be transferred onto data bus through this block accept CS, read and write control signal from system bus and S_0 , S_1 from system address bus. Read and Write signal are used to define direction of data transfer over data bus.

```
Read Operation: CPU <---- I/O device
Write Operation: CPU ----> I/O device
```

The read signal direct data transfer from interface unit to CPU and write signal direct data transfer from CPU to interface unit through data bus.

Address bus is used to select to interface unit. Two least significant lines of address bus (A_0 , A_1) are connected to select lines S_0, S_1. This two select input lines are used to select any one of four registers in interface unit. The selection of interface unit is according to the following criteria:

Read state:

Chip Select	**Operation**		**Select lines**		**Selection of Interface unit**
CS	**Read**	**Write**	**S0**	**S1**	
0	0	1	0	0	Port A
0	0	1	0	1	Port B
0	0	1	1	0	Control Register
0	0	1	1	1	Status Register

Write State:

Chip Select	Operation		Select lines		Selection of Interface unit
CS	**Read**	**Write**	**S0**	**S1**	
0	1	0	0	0	Port A
0	1	0	0	1	Port B
0	1	0	1	0	Control Register
0	1	0	1	1	Status Register

Example:

- If S_0, S_1 = 0 1, then Port B data register is selected for data transfer between CPU and I/O device.
- If S_0, S_1 = 1 0, then Control register is selected and store the control information send by the CPU.

Modes of Transfer:

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes. Data transfer to and from the peripherals may be done in any of the three possible ways

1. Programmed I/O.
2. Interrupt- initiated I/O.
3. Direct memory access (DMA).

Now let's discuss each mode one by one.

1. Programmed I/O:

Programmed I/O uses the I/O instructions written in the computer program. The instructions in the program initiate every data item transfer. Usually, the data transfer is from a memory and CPU register. This case requires constant monitoring by the peripheral device's CPU.

Advantages:

- Programmed I/O is simple to implement.
- It requires very little hardware support.
- CPU checks status bits periodically.

Disadvantages:

- The processor has to wait for a long time for the I/O module to be ready for either transmission or reception of data.
- The performance of the entire system is severely degraded.

2. Interrupt- Initiated I/O:

In the above section, we saw that the CPU is kept busy unnecessarily. We can avoid this situation by using an interrupt-driven method for data transfer. The interrupt facilities and special commands inform the interface for issuing an interrupt request signal as soon as the data is available from any device. In the meantime, the CPU can execute other programs, and the interface will keep monitoring the i/O device. Whenever it determines that the device is ready for transferring data interface initiates an interrupt request signal to the CPU. As soon as the CPU detects an external interrupt signal, it stops the program it was already executing, branches to the service program to process the I/O transfer, and returns to the program it was initially running.

Working of CPU in terms of interrupts:

- CPU issues read command.
- It starts executing other programs.
- Check for interruptions at the end of each instruction cycle.
- On interruptions:-
    - Process interrupt by fetching data and storing it.
    - See operation system notes.
- Starts working on the program it was executing.

Advantages:

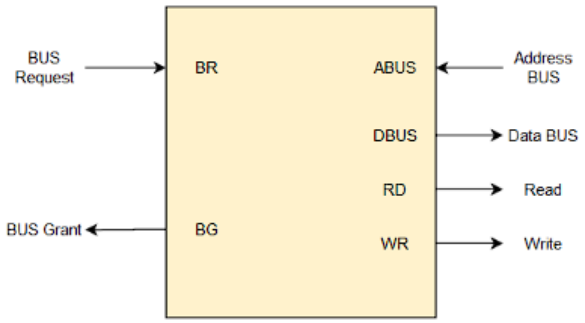
- It is faster and more efficient than Programmed I/O.
- It requires very little hardware support.
- CPU does not check status bits periodically.

Disadvantages:

- It can be tricky to implement if using a low-level language.
- It can be tough to get various pieces of work well together.
- The hardware manufacturer / OS maker usually implements it, e.g., Microsoft.

3. Direct Memory Access (DMA):

The data transfer between any fast storage media like a memory unit and a magnetic disk gets limited with the speed of the CPU. Thus, it will be best to allow the peripherals to directly communicate with the storage using the memory buses by removing the intervention of the CPU. This mode of transfer of data technique is known as Direct Memory Access (DMA). During Direct Memory Access, the CPU is idle and has no control over the memory buses. The DMA controller takes over the buses and directly manages data transfer between the memory unit and I/O devices.



CPU Bus Signal for DMA transfer

Bus Request - We use bus requests in the DMA controller to ask the CPU to relinquish the control buses.

Bus Grant - CPU activates bus grant to inform the DMA controller that DMA can take control of the control buses. Once the control is taken, it can transfer data in many ways.

Types of DMA transfer using DMA controller:

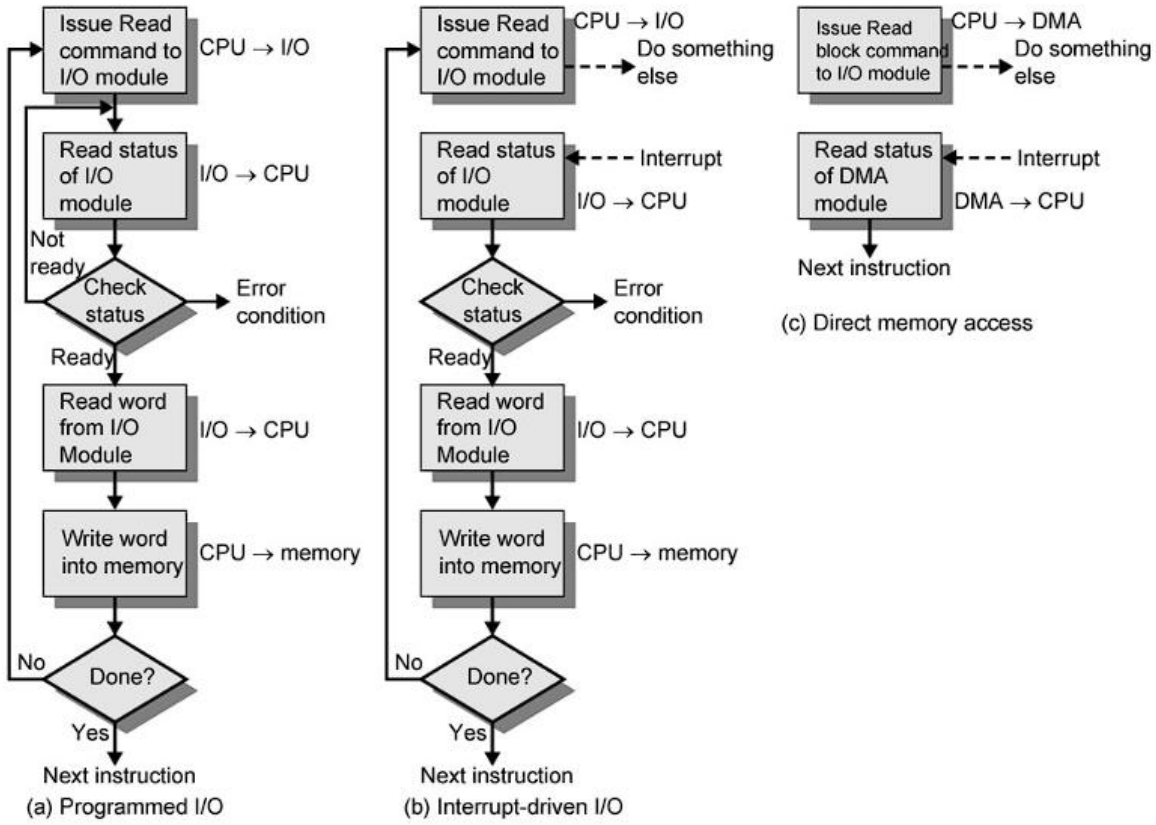
- **Burst Transfer:** In this transfer, DMA will return the bus control after the complete data transfer. A register is used as a byte count, which decrements for every byte transfer, and once it becomes zero, the DMA Controller will release the control bus. When the DMA Controller operates in burst mode, the CPU is halted for the duration of the data transfer.
- **Cyclic Stealing:** It is an alternative method for data transfer in which the DMA controller will transfer one word at a time. After that, it will return the control of the buses to the CPU. The CPU operation is only delayed for one memory cycle to allow the data transfer to “steal” one memory cycle.

Advantages:

- It is faster in data transfer without the involvement of the CPU.
- It improves overall system performance and reduces CPU workload.
- It deals with large data transfers, such as multimedia and files.

Disadvantages:

- It is costly and complex hardware.
- It has limited control over the data transfer process.
- Risk of data conflicts between CPU and DMA.



Priority Interrupt:

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

Priority Interrupt by Software (Polling):

- Priority is established by the order of polling the devices (interrupt sources), that is identify the highest-priority source by software means
- One common branch address is used for all interrupts
- Program polls the interrupt sources in sequence
- The highest-priority source is tested first
- Flexible since it is established by software
- Low cost since it needs a very little hardware
- Very slow

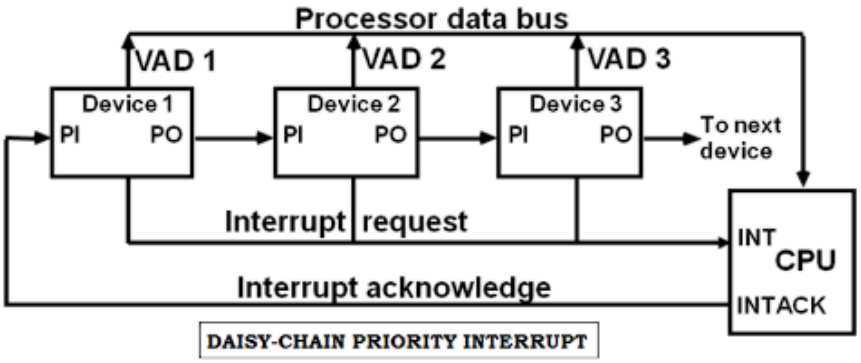
Priority Interrupt by Hardware:

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

1. Daisy Chain Priority (Serial):

The daisy-chaining method involves connecting all the devices that can request an interrupt in a serial manner. This configuration is governed by the priority of the devices. The device with the highest priority is placed first followed by the second highest priority device and so on.

The given figure depicts this arrangement -



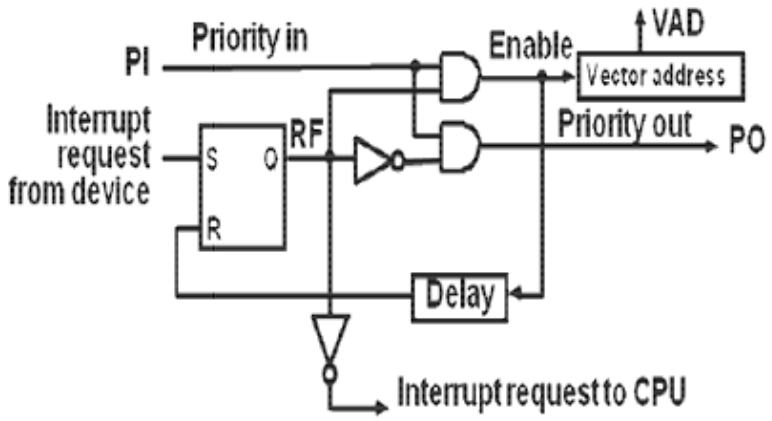
DAISY-CHAIN PRIORITY INTERRUPT

WORKING:

There is an interrupt request line which is common to all the devices and goes into the CPU.

- When no interrupts are pending, the line is in HIGH state. But if any of the devices raises an interrupt, it places the interrupt request line in the LOW state.
- The CPU acknowledges this interrupt request from the line and then enables the interrupt acknowledge line in response to the request.
- This signal is received at the PI(Priority in) input of device 1.
- If the device has not requested the interrupt, it passes this signal to the next device through its PO(priority out) output. (PI = 1 & PO = 1)
- However, if the device had requested the interrupt, (PI =1 & PO = 0)
    - The device consumes the acknowledge signal and block its further use by placing 0 at its PO(priority out) output.
    - The device then proceeds to place its interrupt vector address(VAD) into the data bus of CPU.
    - The device puts its interrupt request signal in HIGH state to indicate its interrupt has been taken care of.
- If a device gets 0 at its PI input, it generates 0 at the PO output to tell other devices that acknowledge signal has been blocked. (PI = 0 & PO = 0)

Hence, the device having PI = 1 and PO = 0 is the highest priority device that is requesting an interrupt. Therefore, by daisy chain arrangement we have ensured that the highest priority interrupt gets serviced first and have established a hierarchy. The farther a device is from the first device, the lower its priority.



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

Fig: One stage of Daisy chain priority arrangement

2. Parallel Priority:

- The parallel priority interrupt method uses a register whose bits are set separately by the interrupt signal from each device.
- Priority is established according to the position of the bits in the register.
- In addition to the interrupt register, the circuit may include a mask register whose purpose is to control the status of each interrupt request.

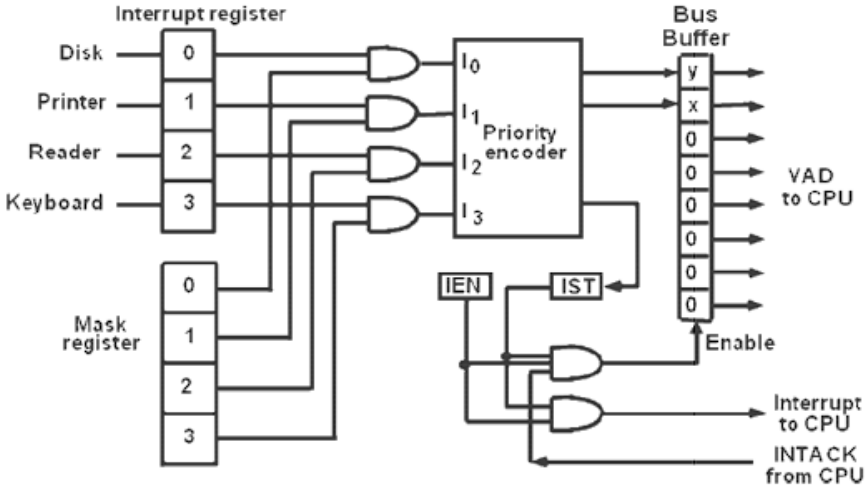


Fig: Parallel priority interrupts hardware

- 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:

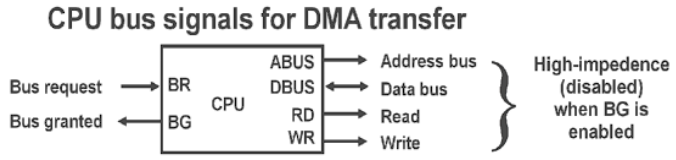
- Determines the highest priority interrupt when more than one interrupts take place

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	$(IST) = I_0 + I_1 + I_2 + I_3$

Fig: Priority Encoder Truth Table

Direct Memory Access (DMA):

- The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called **direct memory access (DMA)**.
- During DMA transfer, the CPU is idle and has no control of the memory buses.
- A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.
- The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessors is to disable the buses through special control signals.



The two control signals Bus Request and Bus Grant are used to fascinate the DMA transfer. The bus request input is used by the DMA controller to request the CPU for the control of the buses. When BR signal is high, the CPU terminates the execution of the current instructions and then places the address, data, read and write lines to the high impedance state and sends the bus grant signal. The DMA controller now takes the control of the buses and transfers the data directly between memory and I/O without processor interaction. When the transfer is completed, the bus request signal is made low by DMA. In response to which CPU disables the bus grant and again CPU takes the control of address, data, read and write lines.

What is a DMA Controller?

Direct Memory Access uses hardware for accessing the memory, that hardware is called a DMA Controller. It has the work of transferring the data between Input Output devices and main memory with very less interaction with the processor. The direct Memory Access Controller is a control unit, which has the work of transferring data.

DMA Controller Diagram:

DMA Controller is a type of control unit that works as an interface for the data bus and the I/O Devices. As mentioned, DMA Controller has the work of transferring the data without the intervention of the processors, processors can control the data transfer. DMA Controller also contains an address unit, which generates the address and selects an I/O device for the transfer of data. Here we are showing the block diagram of the DMA Controller.

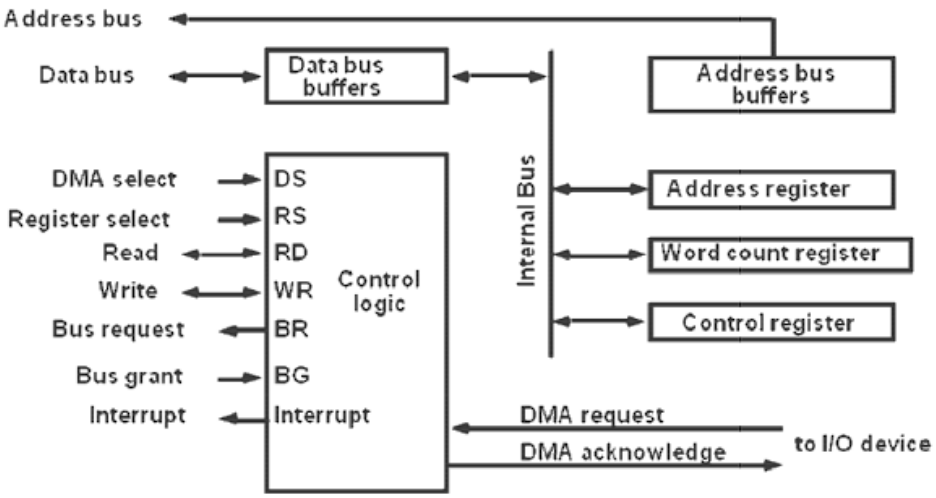


Fig: Block diagram of DMA controller

Types of Direct Memory Access (DMA):

There are four popular types of DMA –

1. Single-Ended DMA: Single-Ended DMA Controllers operate by reading and writing from a single memory address. They are the simplest DMA.

2. Dual-Ended DMA: Dual-Ended DMA controllers can read and write from two memory addresses. Dual-ended DMA is more advanced than single-ended DMA.

3. Arbitrated-Ended DMA: Arbitrated-Ended DMA works by reading and writing to several memory addresses. It is more advanced than Dual-Ended DMA.

4. Interleaved DMA: Interleaved DMA are those DMA that read from one memory address and write from another memory address.

DMA Transfer:

The data transfer can be showed as below sequences:

1. The DMA request line is used to request a DMA transfer.

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

3. The CPU activates the bus grant (BG) output to inform the external DMA that its buses are in a high-impedance state (so that they can be used in the DMA transfer.)

4. The address bus is used to address the DMA controller and memory at given location

5. The Device select (DS) and register select (RS) lines are activated by addressing the DMA controller.

6. The RD and WR lines are used to specify either a read (RD) or write (WR) operation on the given memory location.

7. The DMA acknowledge line is set when the system is ready to initiate data transfer.

8. The data bus is used to transfer data between the I/O device and memory.

9. When the last word of data in the DMA transfer is transferred, the DMA controller informs the termination of the transfer to the CPU by means of the interrupt line.

The diagrammatic representation can be showed as below:

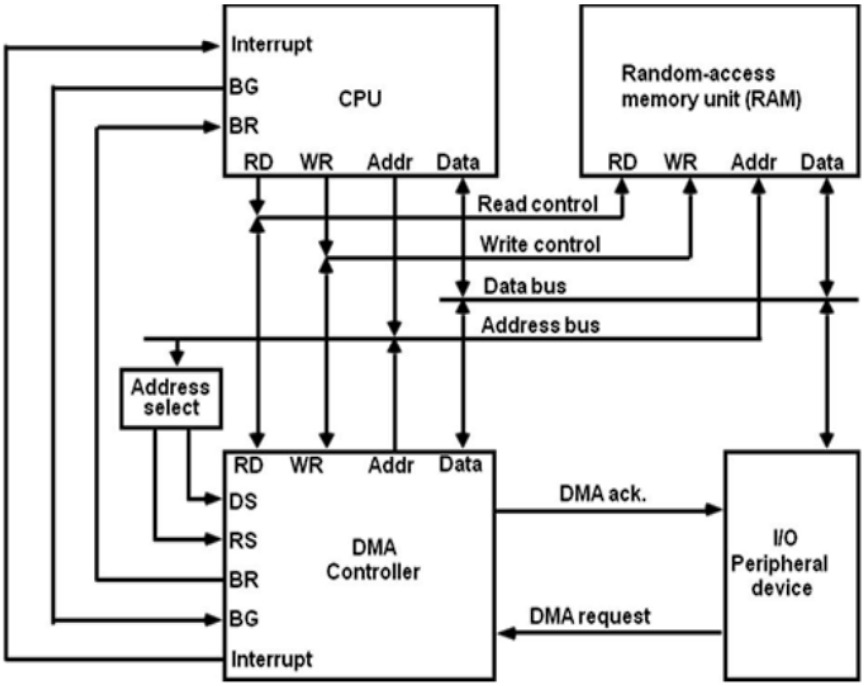


Fig: DMA transfer in a Computer System

Modes of DMA Transfer:

Now after getting some brief idea about DMA and its working it's the time to analyze Modes of DMA Transfer.

- During the DMA Transfer CPU can perform only those operation in which it doesn't require the access of System Bus which means mostly CPU will be in blocked state.
- For how much time CPU remains in the blocked state or we can say for how much time CPU will give the control of DMAC of system buses will actually depend upon the following modes of DMA Transfer and after that CPU will take back control of system buses from DMAC.

Types of DMA transfer using DMA controller:

1. Burst Transfer: DMA returns the bus after complete data transfer. A register is used as a byte count, being decremented for each byte transfer, and upon the byte count reaching zero, the DMAC will release the bus. When the DMAC operates in burst mode, the CPU is halted for the duration of the data transfer.

Steps involved are:

1. Bus grant request time.
2. Transfer the entire block of data at transfer rate of device because the device is usually slow than the speed at which the data can be transferred to CPU.
3. Release the control of the bus back to CPU
4. So, total time taken to transfer the **N bytes = Bus grant request time + (N) * (memory transfer rate) + Bus release control time**.

% CPU idle (Blocked)=(Y/X+Y)*100

% CPU Busy=(X/X+Y)*100

Where, X µsec = data transfer time or preparation time (words/block)

Y µsec = memory cycle time or cycle time or transfer time (words/block)

2. Cyclic Stealing: An alternative method in which DMA controller transfers one word at a time after which it must return the control of the buses to the CPU. The CPU delays its operation only for one memory cycle to allow the direct memory I/O transfer to "steal" one memory cycle.

Steps Involved are:

1. Buffer the byte into the buffer
2. Inform the CPU that the device has 1 byte to transfer (i.e. bus grant request)
3. Transfer the byte (at system bus speed)
4. Release the control of the bus back to CPU.

In cycle stealing mode we always follow pipelining concept that when one byte is getting transferred then Device is parallel preparing the next byte. "The fraction of CPU time to the data transfer time" if asked then cycle stealing mode is used.

% CPU idle (Blocked) =(Y/X)*100

% CPU busy=(X/Y)*100

Where, X µsec = data transfer time or preparation time (words/block)

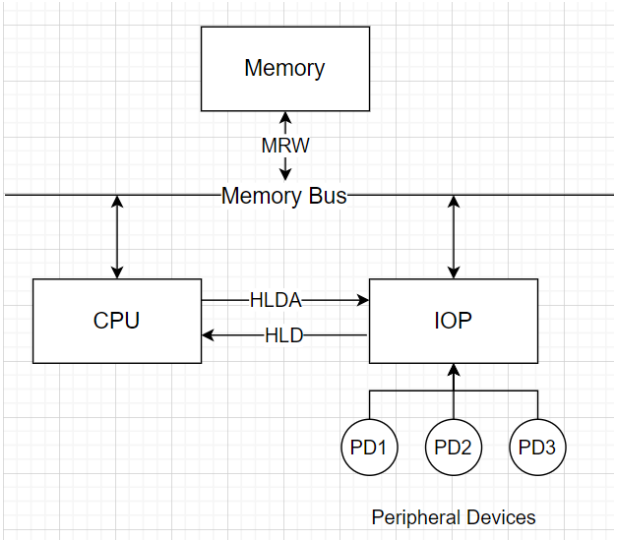
Y µsec = memory cycle time or cycle time or transfer time (words/block)

3. Interleaved/ Transparent Mode: In this technique, the DMA controller takes over the system bus when the microprocessor is not using it. An alternate half cycle i.e. half cycle DMA + half cycle processor.

Input-Output Processor:

The input-output operations are generally the slowest ones. Consider the tasks when frequent I/O operations are required. This will greatly reduce the processor's efficiency as most of its time will be spent in handling the I/O instructions. To handle this, we use the DMA mode of data transfer.

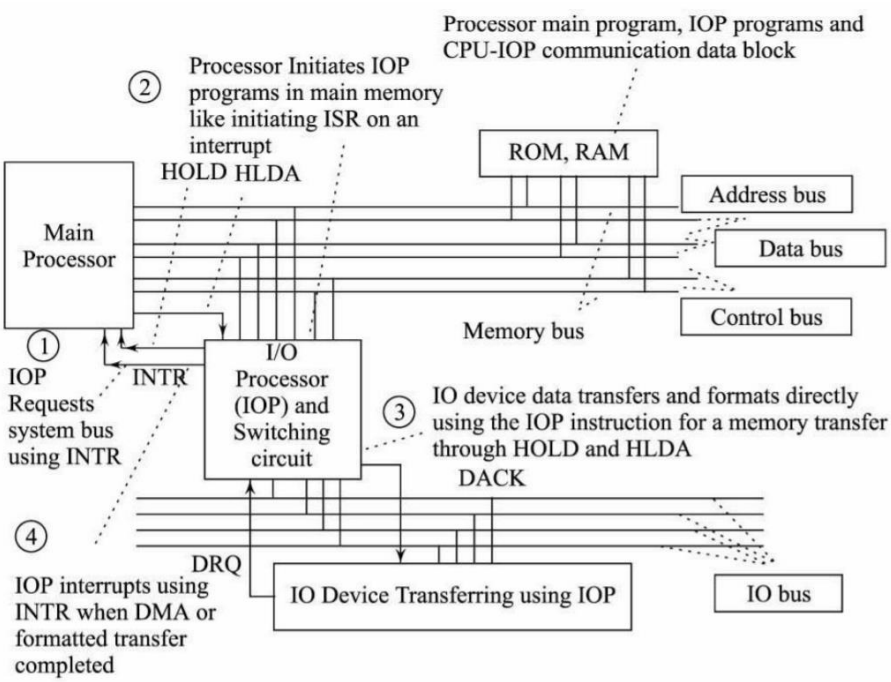
An input-output processor (IOP) is a special processor used in DMA that can directly access and store data into memory. It is similar to a CPU, except that it only handles the tasks of I/O processing. It acts as an interface between the computer and the I/O devices.



While the CPU solves the computational tasks, the input-output processor provides a path for the transfer of data between the peripheral devices and the memory. The CPU assigns the task of I/O transfer to the input-output processor, and the input-output processor executes the instructions.

Working of Input-Output Processor:

1. When an I/O device has to access or store data to memory, it requests the DMA controller. The DMA controller then sends a Hold request (HLD) to the CPU to hold.
2. The CPU, on receiving the Hold request, sends Hold acknowledgment (HLDA) to the DMA controller.
3. After receiving the HLDA, the DMA controller allows the data transfer and takes charge of the system bus.
4. On completion of the data transfer, the DMA again issues an interrupt, letting the CPU know that the data transfer has finished.



Features of an Input-Output Processor:

The features of an Input-Output Processor (IOP) are:

- **Interface Management**: IOPs manage the communication between the CPU and various input/output devices. They handle the protocols and data formats required for communication.
- **Data Buffering**: IOPs often have dedicated memory buffers to store data temporarily during input/output operations. This buffering helps in managing data transfer rates between devices with varying speeds.
- **Interrupt Handling**: IOPs handle interrupts generated by input/output devices. They notify the CPU when a device needs attention, allowing the CPU to respond promptly without polling each device continuously.
- **Data Conversion**: IOPs may perform data format conversions between the format used by the device and the format expected by the CPU. This includes tasks like data encoding/decoding, byte swapping, and data compression/decompression.
- **Error Handling**: IOPs are responsible for detecting and handling errors that occur during input/output operations. They may implement error detection mechanisms such as checksums or parity bits and take appropriate actions in case of errors.
- **Bus Arbitration**: In systems with multiple devices sharing the same bus, IOPs manage bus arbitration to ensure fair access to the bus for all devices.

Applications of I/O Processors:

- **Disk Controllers**: IOPs are commonly used as disk controllers to manage data transfer between the CPU and disk drives. They handle tasks such as reading/writing data, managing disk caches, and handling error recovery.
- **Network Interface Controllers (NICs)**: IOPs are used in network interface controllers to manage communication between the computer and the network. They handle tasks such as packet transmission/reception, protocol processing, and error detection/correction.
- **USB Controllers**: IOPs are used in USB controllers to manage communication between the computer and USB devices. They handle tasks such as device enumeration, data transfer, and power management.
- **RAID Controllers**: IOPs are used in RAID (Redundant Array of Independent Disks) controllers to manage data redundancy and performance optimization across multiple disk drives.
- **Graphics Processors**: In some systems, IOPs are used as graphics processors to offload graphics processing tasks from the CPU. They handle tasks such as rendering, texture mapping, and image manipulation.

Advantages of Input-Output processor:

1. Speeds up the read-write operations as it does not involve a processor.
2. I/O devices can access the main memory directly.
3. Reduces the CPU overhead of waiting for I/O to complete.
4. By offloading input/output operations from the CPU, an I/O processor allows the CPU to focus on computation-intensive tasks, thus improving overall system performance.
5. With an I/O processor handling input/output operations independently, the CPU and I/O processor can work in parallel, enabling concurrent execution of tasks and reducing system idle time.
6. I/O processors can be optimized for specific input/output tasks, allowing for more efficient and dedicated handling of those operations compared to a general-purpose CPU.

Disadvantages of Input-Output processor:

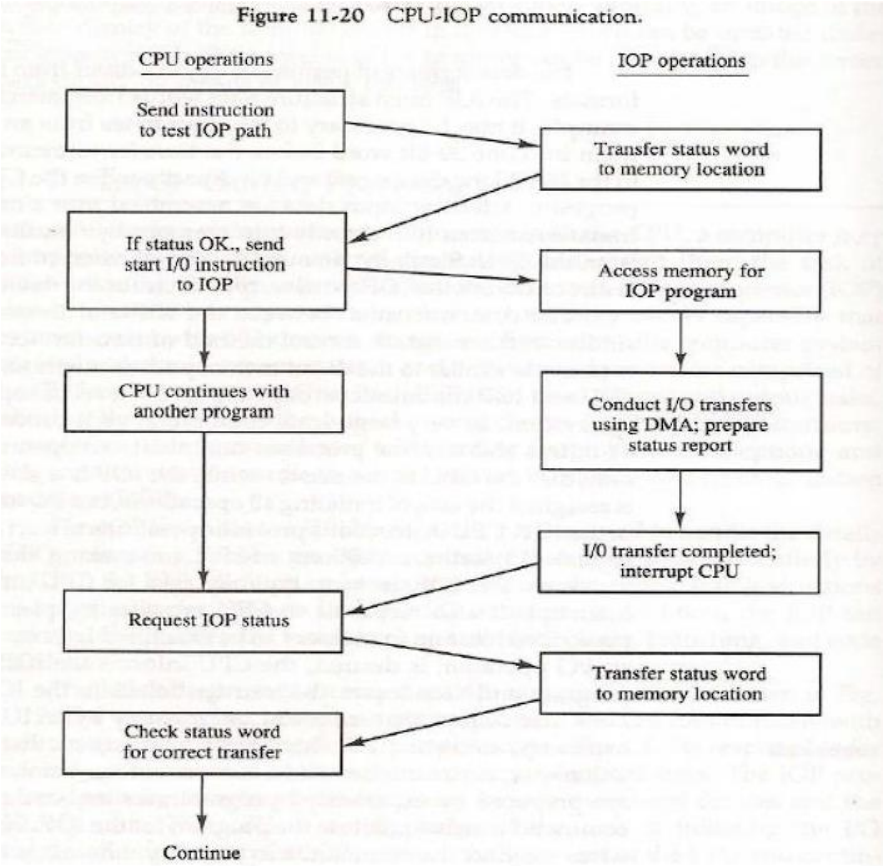
1. Increases the hardware cost.

2. Cache coherence problem (keeping multiple local caches synchronized) can occur.
3. Implementing dedicated input/output processors adds to the overall hardware cost of the system, particularly in terms of additional processors, memory, and interconnects required to support them.
4. While I/O processors can reduce CPU overhead for input/output operations, they can become potential bottlenecks if not properly designed or if the input/output workload exceeds the processing capacity of the I/O processors.
5. In systems with multiple I/O processors or distributed input/output subsystems, maintaining cache coherence and synchronization among the processors can introduce additional overhead and complexity.

CPU-IOP Communication:

There is a communication channel between IOP and CPU to perform task which come under computer architecture. This channel explains the commands executed by IOP and CPU while performing some programs. The CPU don't execute the instructions but it assigns the task of initiating operations, the instructions are executed by IOP. I/O transfer is instructed by CPU. The IOP asks for CPU through interrupt. This channel starts by CPU, by giving "test IOP path" instruction to IOP and then the communication begins as shown in diagram:

Figure 11-20 CPU-IOP communication.



Whenever CPU gets interrupt from IOP to access memory, it sends test path instruction to IOP. IOP executes and check for status, if the status given to CPU is OK, then CPU gives start instruction to IOP and gives it some control and get back to some another (or same) program, after that IOP is able to access memory for its program. Now IOP start controlling I/O transfer using DMA and create another status report as well. AS soon as this I/O transfer completes IOP once again send interrupt to CPU, CPU again request for status of IOP, IOP check status word from memory location and gives it to CPU. Now CPU check the correctness of status and continues with the same process.

PART – 2 : MEMORY ORGANIZATION

Introduction:

The memory is divided into cells; each of them is identified by a unique number called an address. The CPU generates the memory request in order to access the instruction. When the CPU wants to read or write an address, it generates control signals such as "read" and "write," which each cell can identify. Because the program is present in memory, the instruction must be transferred from memory to the CPU whenever the CPU executes the program.

Memory Request:

The address and control signals are included in the memory request. When inserting data and information into the stack, each block uses memory (in the form of RAM), and the number of memory cells is determined by the memory chip's capacity.

Example:

We will find the number of cells present in the 64k*8 memory chips.

Here, the size of each cell = 8

And the number of bytes present in 64k = $(2^6)*(2^{10})$

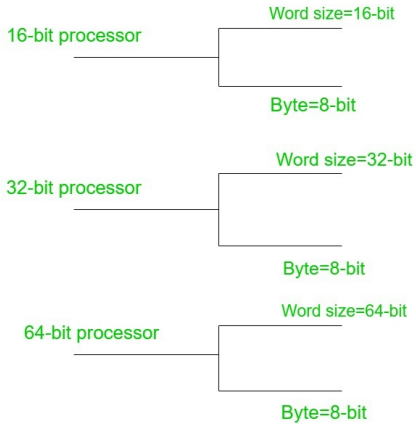
Thus,

Total number of cells here = 2^{16} cells

Along with the total number of cells, the total number of address lines that are needed to enable any cell can be determined here.

Word Size:

It refers to the maximum amount of bits a CPU can handle at one time, and it fluctuates by the processor. The hardware or the instruction set of a processor handles a fixed amount of data as a unit called **word size**.

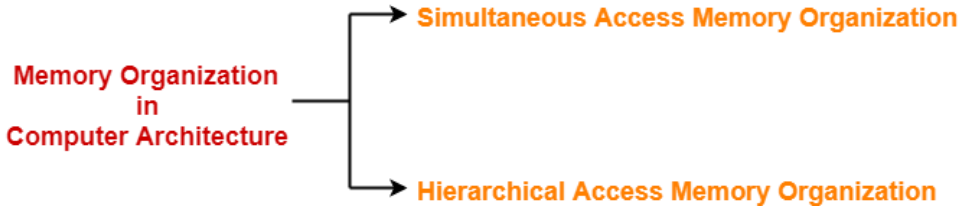


Because of generation and current technology, word size varies based on processor architectures; it could be as small as 4-bits or as large as 64-bits, depending on what a given processor can handle. Addresses, Registers, Fixed-point numbers, and Floating-point numbers are all examples of notions that require word size.

Classification of Memory Organization:

In a computer,

- Memory is organized at different levels.
- CPU may try to access different levels of memory in different ways.
- On this basis, the memory organization is broadly divided into two types-



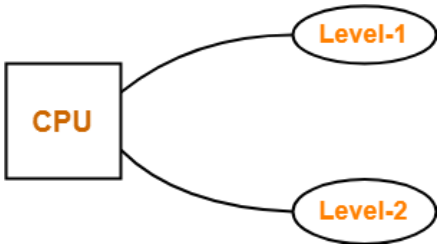
1. Simultaneous Access Memory Organization:

In this memory organization,

- All the levels of memory are directly connected to the CPU.
- Whenever CPU requires any word, it starts searching for it in all the levels simultaneously.

Example:

Consider the following simultaneous access memory organization-



Here, two levels of memory are directly connected to the CPU.

Let-

- T1 = Access time of level L1
- S1 = Size of level L1
- C1 = Cost per byte of level L1
- H1 = Hit rate of level L1

Similar are the notations for level L2.

Average Memory Access Time-

Average time required to access memory per operation

= H1 x T1 + (1 – H1) x H2 x T2

= H1 x T1 + (1 – H1) x 1 x T2

= H1 x T1 + (1 – H1) x T2

Important Note:

In any memory organization,

- The data item being searched will definitely be present in the last level.
- Thus, hit rate for the last level is always 1.

Average Cost Per Byte-

Average cost per byte of the memory

= { C1 x S1 + C2 x S2 } / { S1 + S2 }

Similarly,

Average Memory Access Time for 3 Levels-

Average time required to access memory per operation

= H1 x T1 + (1 – H1) x H2 x T2 + (1 – H1) x (1 – H2) x H3 x T3

= H1 x T1 + (1 – H1) x H2 x T2 + (1 – H1) x (1 – H2) x 1 x T3

= H1 x T1 + (1 – H1) x H2 x T2 + (1 – H1) x (1 – H2) x T3

Average Cost Per Byte for 3 Levels -

Average cost per byte of the memory

= { C1 x S1 + C2 x S2 + C3 x S3 } / { S1 + S2 + S3 }

2. Hierarchical Access Memory Organization:

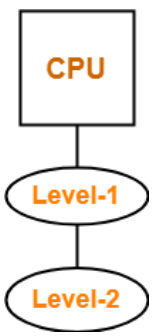
In this memory organization, memory levels are organized as-

- Level-1 is directly connected to the CPU.
- Level-2 is directly connected to level-1.
- Level-3 is directly connected to level-2 and so on.

Whenever CPU requires any word,

- It first searches for the word in level-1.
- If the required word is not found in level-1, it searches for the word in level-2.
- If the required word is not found in level-2, it searches for the word in level-3 and so on.

Example: Consider the following hierarchical access memory organization-



Here, two levels of memory are connected to the CPU in a hierarchical fashion.

Let-

- T1 = Access time of level L1
- S1 = Size of level L1
- C1 = Cost per byte of level L1
- H1 = Hit rate of level L1

Similar are the notations for level L2.

Average Memory Access Time-

Average time required to access memory per operation

= H1 x T1 + (1 – H1) x H2 x (T1 + T2)

= H1 x T1 + (1 – H1) x 1 x (T1 + T2)

= H1 x T1 + (1 – H1) x (T1 + T2)

Average Cost Per Byte-

Average cost per byte of the memory

= { C1 x S1 + C2 x S2 } / { S1 + S2 }

Similarly,

Average Memory Access Time for 3 Levels -

Average time required to access memory per operation

= H1 x T1 + (1 – H1) x H2 x (T1 + T2) + (1 – H1) x (1 – H2) x H3 x (T1 + T2 + T3)

= H1 x T1 + (1 – H1) x H2 x (T1 + T2) + (1 – H1) x (1 – H2) x 1 x (T1 + T2 + T3)

= H1 x T1 + (1 – H1) x H2 x (T1 + T2) + (1 – H1) x (1 – H2) x (T1 + T2 + T3)

Average Cost Per Byte for 3 Levels -

Average cost per byte of the memory

= { C1 x S1 + C2 x S2 + C3 x S3 } / { S1 + S2 + S3 }

PRACTICE PROBLEM BASED ON MEMORY ORGANIZATION-

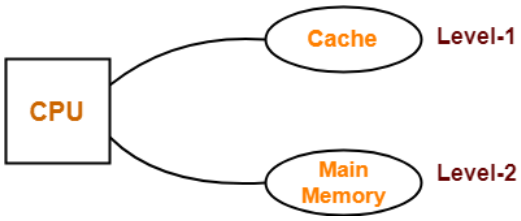
Problem: What is the average memory access time for a machine with a cache hit rate of 80% and cache access time of 5 ns and main memory access time of 100 ns when-

1. Simultaneous access memory organization is used.
2. Hierarchical access memory organization is used.

Solution-

Part-01: Simultaneous Access Memory Organization-

The memory organization will be as shown-



Average memory access time

= H1 x T1 + (1 – H1) x H2 x T2

= 0.8 x 5 ns + (1 – 0.8) x 1 x 100 ns

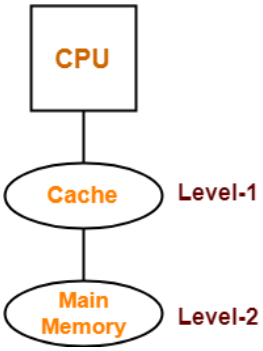
= 4 ns + 0.2 x 100 ns

= 4 ns + 20 ns

= 24 ns

Part-02: Hierarchical Access Memory Organization-

The memory organization will be as shown-



Average memory access time

= H1 x T1 + (1 – H1) x H2 x (T1 + T2)

= 0.8 x 5 ns + (1 – 0.8) x 1 x (5 ns + 100 ns)

= 4 ns + 0.2 x 105 ns

= 4 ns + 21 ns

= 25 ns

Important Note:

While solving numerical problems**,** If the kind of memory organization used is not mentioned, **assume Hierarchical Access Memory Organization**.

Memory Hierarchy:

Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as **locality of references**. The figure below clearly demonstrates the different levels of the memory hierarchy.

Why Memory Hierarchy is Required in the System?

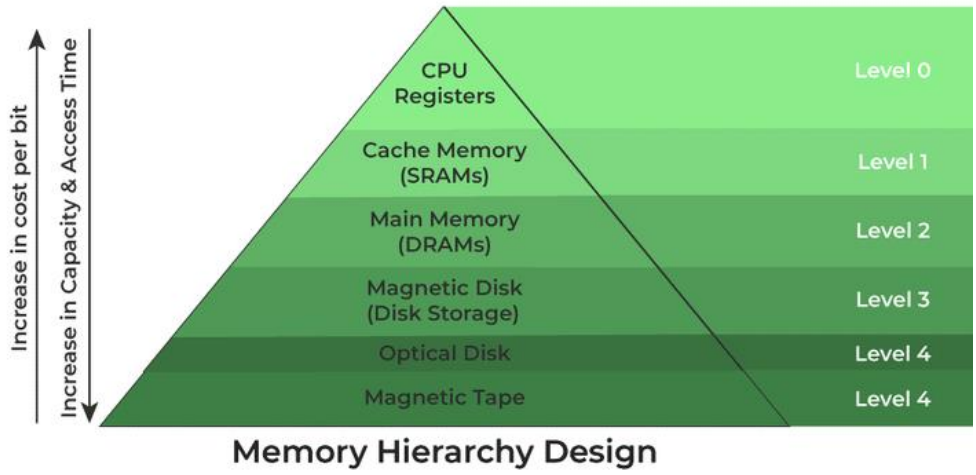
Memory Hierarchy is one of the most required things in Computer Memory as it helps in optimizing the memory available in the computer. There are multiple levels present in the memory, each one having a different size, different cost, etc. Some types of memory like cache, and main memory are faster as compared to other types of memory but they are having a little less size and are also costly whereas some memory has a little higher storage value, but they are a little slower. Accessing of data is not similar in all types of memory, some have faster access whereas some have slower access.

Types of Memory Hierarchy:

This Memory Hierarchy Design is divided into 2 main types:

- **External Memory or Secondary Memory:** Comprising of Magnetic Disk, Optical Disk, and Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via an I/O Module.
- **Internal Memory or Primary Memory:** Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

Memory Hierarchy Design:



Memory Hierarchy Design

1. Registers: Registers are small, high-speed memory units located in the CPU. They are used to store the most frequently used data and instructions. Registers have the fastest access time and the smallest storage capacity, typically ranging from 16 to 64 bits.

2. Cache Memory: Cache memory is a small, fast memory unit located close to the CPU. It stores frequently used data and instructions that have been recently accessed from the main memory. Cache memory is designed to minimize the time it takes to access data by providing the CPU with quick access to frequently used data.

3. Main Memory: Main memory, also known as RAM (Random Access Memory), is the primary memory of a computer system. It has a larger storage capacity than cache memory, but it is slower. Main memory is used to store data and instructions that are currently in use by the CPU.

Types of Main Memory -

- **Static RAM:** Static RAM stores the binary information in flip flops and information remains valid until power is supplied. It has a faster access time and is used in implementing cache memory.
- **Dynamic RAM:** It stores the binary information as a charge on the capacitor. It requires refreshing circuitry to maintain the charge on the capacitors after a few milliseconds. It contains more memory cells per unit area as compared to SRAM.

4. Secondary Storage: Secondary storage, such as hard disk drives (HDD) and solid-state drives (SSD), is a non-volatile memory unit that has a larger storage capacity than main memory. It is used to store data and instructions that are not currently in use by the CPU. Secondary storage has the slowest access time and is typically the least expensive type of memory in the memory hierarchy.

5. Magnetic Disk: Magnetic Disks are simply circular plates that are fabricated with either a metal or a plastic or a magnetized material. The Magnetic disks work at a high speed inside the computer and these are frequently used.

6. Magnetic Tape: Magnetic Tape is simply a magnetic recording device that is covered with a plastic film. It is generally used for the backup of data. In the case of a magnetic tape, the access time for a computer is a little slower and therefore, it requires some amount of time for accessing the strip.

Characteristics of Memory Hierarchy:

- **Capacity:** It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.
- **Access Time:** It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.
- **Performance:** Earlier when the computer system was designed without a Memory Hierarchy design, the speed gap increased between the CPU registers and Main Memory due to a large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.
- **Cost Per Bit:** As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

Advantages of Memory Hierarchy:

- It helps in removing some destruction, and managing the memory in a better way.
- It helps in spreading the data all over the computer system.
- It saves the consumer's price and time.

System-Supported Memory Standards:

According to the memory Hierarchy, the system-supported memory standards are defined below:

Level	1	2	3	4
Name	Register	Cache	Main Memory	Secondary Memory
Size	<1 KB	less than 16 MB	<16GB	>100 GB
Implementation	Multi-ports	On-chip/SRAM	DRAM (capacitor memory)	Magnetic
Access Time	0.25ns to 0.5ns	0.5 to 25ns	80ns to 250ns	50 lakh ns
Bandwidth	20000 to 1 lakh MB	5000 to 15000	1000 to 5000	20 to 150
Managed by	Compiler	Hardware	Operating System	Operating System
Backing Mechanism	From cache	from Main Memory	from Secondary Memory	from ie

Main Memory:

Main Memory is also known as the **Primary storage or memory**, which is the part of the computer that stores current data, programs, and instructions. Primary storage is stored in the motherboard which results in the data from and to primary storage can be read and written at a very good pace.

What is Main Memory?

Main memory is a segment of computer memory that can be accessed directly by the processor. In a hierarchy of memory, Main memory has access time less than secondary memory and greater than cache memory. Generally, Main memory has a storage capacity lesser than secondary memory and greater than cache memory.

Need of Main memory:

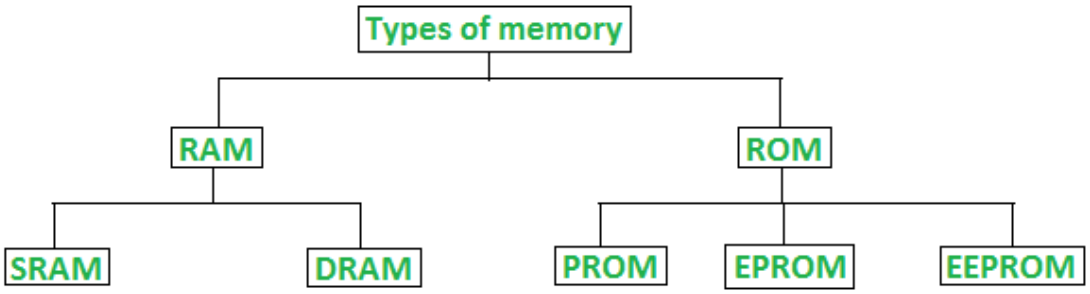
In order to enhance the efficiency of the system, memory is organized in such a way that access time for the ready process is minimized. The following approach is followed to minimize access time for the ready process.

- All programs, files, and data are stored in secondary storage that is larger and hence has greater access time.
- Secondary memory cannot be accessed directly by a CPU or processor.
- In order, to execute any process operating system loads the process in primary memory which is smaller and can be accessed directly by the CPU.
- Since only those processes are loaded in primary memory which is ready to be executed, the CPU can access those processes efficiently and this optimizes the performance of the system.

Classification of Main Memory:

Main memory can be broadly classified into two parts:

1. **R**ead-**O**nly **M**emory (ROM)
2. **R**andom **A**ccess **M**emory (RAM)



Read-Only Memory (ROM):

Any data which need not be altered are stored in ROM. ROM includes those programs which run on booting of the system *(known as* a ***bootstrap program*** that initializes *OS)* along with data like algorithm required by OS. Anything stored in ROM cannot be altered or changed.

Types of ROM:

ROM can be broadly classified into 4 types based on their behavior:

- **MROM:** *Masked ROM* is hardwired and pre-programmed ROM. Any content that is once written cannot be altered anyhow.
- **PROM:** *Programmable ROM* can be modified once by the user. The user buys a blank PROM and writes the desired content but once written content cannot be altered.
- **EPROM:** *Erasable and Programmable ROM* Content can be changed by erasing the initial content which can be done by exposing EPROM to UV radiation. This exposure to ultra-violet light dissipates the charge on ROM and content can be rewritten on it.
- **EEPROM:** *Electrically Erasable and Programmable ROM* Content can be changed by erasing the initial content which could be easily erased electrically. However, one byte can be erased at a time instead of deleting in one go. Hence, reprogramming of EEPROM is a slow process.

Random Access Memory (RAM):

Any process in the system which needs to be executed is loaded in RAM which is processed by the CPU as per Instructions in the program. Like if we click on applications like Browser, firstly browser code will be loaded by the Operating system into the RAM after which the CPU will execute and open up the Browser.

Types of RAM:

RAM can be broadly classified into SRAM (Static RAM) and DRAM (Dynamic RAM) based on their behavior:

- **DRAM:** Dynamic RAM or DRAM needs to periodically refresh in a few milliseconds to retain data. DRAM is made up of capacitors and transistors and electric charge leaks from capacitors and DRAM needs to be charged periodically. DRAM is widely used in home PCs and servers as it is cheaper than SRAM.
- **SRAM:** Static RAM or SRAM keeps the data as long as power is supplied to the system. SRAM uses Sequential circuits like a flip-flop to store a bit and hence need not be periodically refreshed. SRAM is expensive and hence only used where speed is the utmost priority.

Main Memory is volatile in nature.

Content of primary memory may or may not vanish when power is lost depending on if it is stored in RAM or ROM.

- The content of ROM is non-volatile in nature, they are stored even when power is lost.
- The content of RAM is volatile in nature, it vanishes when power is lost.

The primary technology used for the main memory is based on semiconductor integrated circuits. The integrated circuits for the main memory are classified into two major units.

1. RAM (Random Access Memory) integrated circuit chips
2. ROM (Read Only Memory) integrated circuit chips

1. RAM integrated circuit chips:

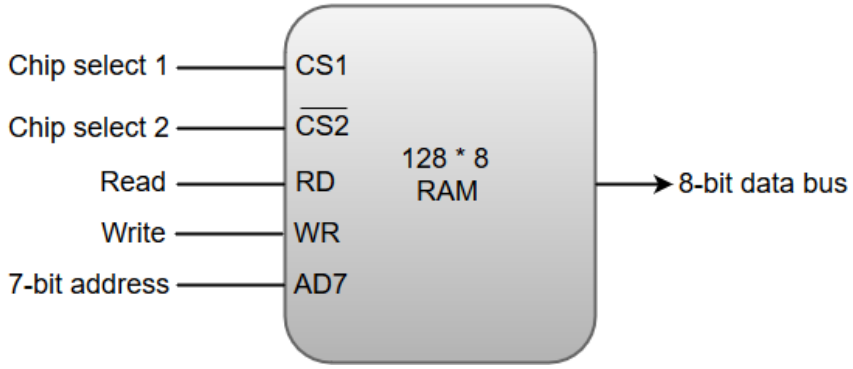
The RAM integrated circuit chips are further classified into two possible operating modes, **static** and **dynamic**.

The primary compositions of a static RAM are flip-flops that store the binary information. The nature of the stored information is volatile, i.e. it remains valid as long as power is applied to the system. The static RAM is easy to use and takes less time performing read and write operations as compared to dynamic RAM.

The dynamic RAM exhibits the binary information in the form of electric charges that are applied to capacitors. The capacitors are integrated inside the chip by MOS transistors. The dynamic RAM consumes less power and provides large storage capacity in a single memory chip.

RAM chips are available in a variety of sizes and are used as per the system requirement. The following block diagram demonstrates the chip interconnection in a 128 * 8 RAM chip.

Typical RAM chip:



- A 128 * 8 RAM chip has a memory capacity of 128 words of eight bits (one byte) per word. This requires a 7-bit address and an 8-bit bidirectional data bus.
- The 8-bit bidirectional data bus allows the transfer of data either from memory to CPU during a **read** operation or from CPU to memory during a **write** operation.
- The **read** and **write** inputs specify the memory operation, and the two chip select (CS) control inputs are for enabling the chip only when the microprocessor selects it.
- The bidirectional data bus is constructed using **three-state buffers**.
- The output generated by three-state buffers can be placed in one of the three possible states which include a signal equivalent to logic 1, a signal equal to logic 0, or a high-impedance state.

Note: The logic 1 and 0 are standard digital signals whereas the high-impedance state behaves like an open circuit, which means that the output does not carry a signal and has no logic significance.

The following function table specifies the operations of a 128 * 8 RAM chip.

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

From the functional table, we can conclude that the unit is in operation only when CS1 = 1 and CS2 = 0. The bar on top of the second select variable indicates that this input is enabled when it is equal to 0.

2. ROM Integrated Circuit:

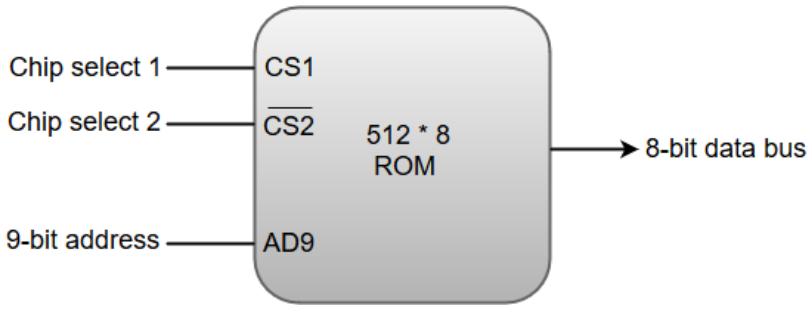
The primary component of the main memory is RAM integrated circuit chips, but a portion of memory may be constructed with ROM chips.

A ROM memory is used for keeping programs and data that are permanently resident in the computer.

Apart from the permanent storage of data, the ROM portion of main memory is needed for storing an initial program called a **bootstrap loader**. The primary function of the **bootstrap loader** program is to start the computer software operating when power is turned on.

ROM chips are also available in a variety of sizes and are also used as per the system requirement. The following block diagram demonstrates the chip interconnection in a 512 * 8 ROM chip.

Typical ROM chip:



- A ROM chip has a similar organization as a RAM chip. However, a ROM can only perform read operation; the data bus can only operate in an output mode.
- The 9-bit address lines in the ROM chip specify any one of the 512 bytes stored in it.
- The value for chip select 1 and chip select 2 must be 1 and 0 for the unit to operate. Otherwise, the data bus is said to be in a high-impedance state.

Memory Address Map:

The system designer must calculate the amount of memory required for a given application and assign it to RAM or ROM.

The interconnection between the processor and the memory is established by knowing the size of memory required and the type of ROM and RAM chips available. The addressing of memory can be established using a table that specifies the memory address assigned to each chip.

The table is called the **Memory address map**.

For example, 512 bytes RAM and 512 bytes ROM.

		ADDRESS BUS									
COMPONENT	HEXA DECIMAL ADDRESS	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:

- The data and address buses are used to connect RAM and ROM chips to a CPU.
- The low-order lines in the address bus choose the byte within the chips, while the other lines in the address bus select a particular chip through its chip select inputs.

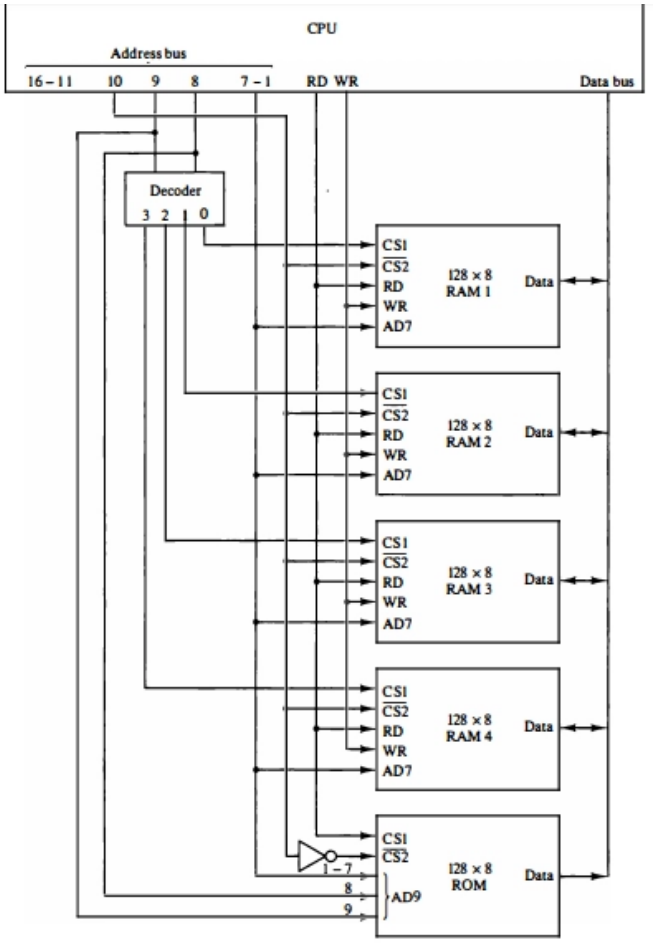


Figure 4 Memory connection to the CPU.

Auxiliary Memory:

An Auxiliary memory also known as secondary storage or external memory is known as the lowest-cost, highest-capacity and slowest-access storage in a computer system. It is where programs and data are kept for long-term storage or when not in immediate use. The most common examples of auxiliary memories are magnetic tapes and magnetic disks.

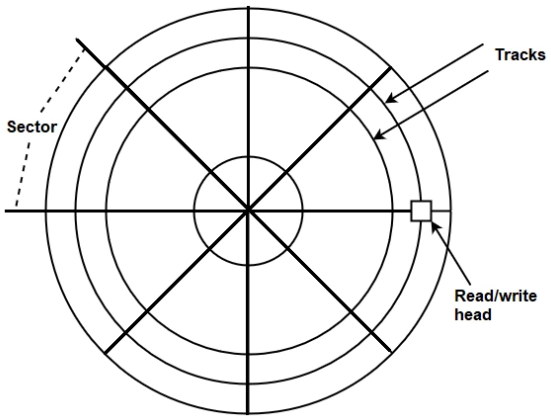
Types of Auxiliary Memory:

Magnetic Disks:

A magnetic disk is a round plate forged of metal or plastic coated with magnetized material. We use both sides of the disk and stack multiple disks on one spindle with read/write heads available on each surface.

All disks spin together at high speed and are not stopped or initiated for access. Memory Bits are saved in the magnetized surface in marks along concentric circles known as tracks. We divide the concentric circles (tracks) into regions known as sectors.

In this system, the lowest quantity of data that can be sent is a sector. The subdivision of one disk surface into tracks and sectors is displayed in the figure.



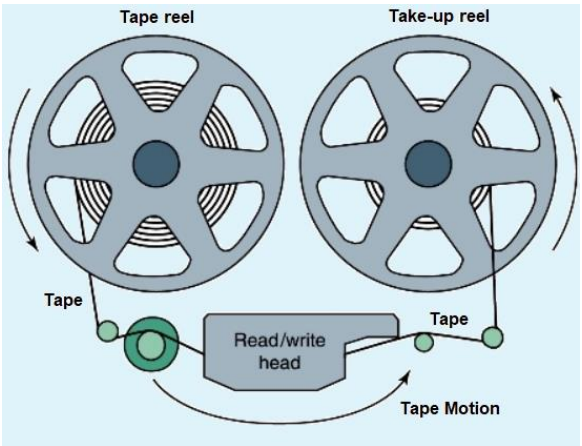
Magnetic Tape:

Magnetic tape transport includes the robotic, mechanical, and electronic components to support the methods and control structure for a magnetic tape unit. The tape is a layer of plastic coated with a magnetic documentation medium.

Bits are listed as a magnetic stain on the tape along various tracks. There are seven or nine bits are recorded together to form a character together with a parity bit. Read/write heads are mounted one in each track therefore that information can be recorded and read as a series of characters.

Magnetic tape units can be stopped, initiated to move forward, or in the opposite, or it can be reversed. However, they cannot be initiated or stopped fast enough between single characters. For this reason, data is recorded in blocks defined as records. Gaps of unrecorded tape are added between records where the tape can be stopped.

The tape begins affecting while in a gap and achieves its permanent speed by the time it arrives at the next record. Each record on tape has a recognition bit design at the starting and end. By reading the bit design at the starting, the tape control recognizes the data number.



Flash Memory:

Flash memory is a non-volatile memory used to store and transfer data between a personal computer (PC) and digital devices. We can electronically reprogram and erase it. We can often see it in USB flash drives, MP3 players, digital cameras, and solid-state drives.

Flash memory is an electronically programmable, erasable, and read-only memory (EEPROM), but it can also act as standalone storage devices such as USB drives.

Optical Disk:

An optical disk is any computer disk that uses optical storage techniques and technology to read and write data. It is a storage device using optical (light) energy. It is a computer storage disk that stores data digitally and uses laser beams to read and write data. It uses optical technology in which laser light is central to the spinning disks.

Associative Memory:

Associative memory is also known as content addressable memory (CAM) or associative storage or associative array. It is a special type of memory that is optimized for performing searches through data, as opposed to providing a simple direct access to the data based on the address.

it can store the set of patterns as memories when the associative memory is being presented with a key pattern, it responds by producing one of the stored pattern which closely resembles or relates to the key pattern.

it can be viewed as **data correlation** here. input data is correlated with that of stored data in the CAM.

it forms of two types:

1. **auto associative memory network:** An auto-associative memory network, also known as a recurrent neural network, is a type of associative memory that is used to

recall a pattern from partial or degraded inputs. In an auto-associative network, the output of the network is fed back into the input, allowing the network to learn and remember the patterns it has been trained on. This type of memory network is commonly used in applications such as speech and image recognition, where the input data may be incomplete or noisy.

2. **hetero associative memory network:** A hetero-associative memory network is a type of associative memory that is used to associate one set of patterns with another. In a hetero-associative network, the input pattern is associated with a different output pattern, allowing the network to learn and remember the associations between the two sets of patterns. This type of memory network is commonly used in applications such as data compression and data retrieval.

Associative memory of conventional semiconductor memory (usually RAM) with added comparison circuity that enables a search operation to complete in a single clock cycle. It is a hardware search engine, a special type of computer memory used in certain very high searching applications.

How Does Associative Memory Work?

In conventional memory, data is stored in specific locations, called addresses, and retrieved by referencing those addresses. In associative memory, data is stored together with additional tags or metadata that describe its content. When a search is performed, the associative memory compares the search query with the tags of all stored data, and retrieves the data that matches the query.

Associative memory is designed to quickly find matching data, even when the search query is incomplete or imprecise. This is achieved by using parallel processing techniques, where multiple search queries can be performed simultaneously. The search is also performed in a single step, as opposed to conventional memory where multiple steps are required to locate the data.

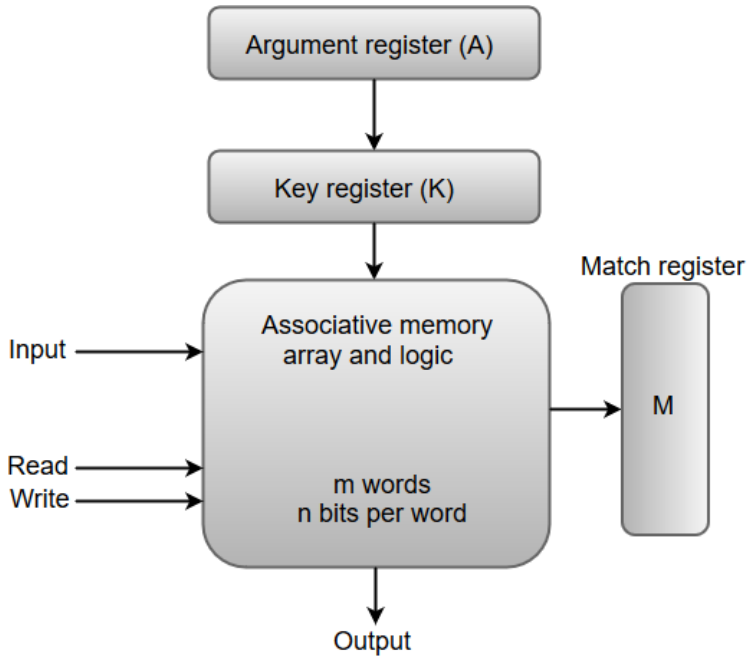
Applications of Associative memory:

1. It can be only used in memory allocation format.
2. It is widely used in the database management systems, etc.
3. Networking: Associative memory is used in network routing tables to quickly find the path to a destination network based on its address.
4. Image processing: Associative memory is used in image processing applications to search for specific features or patterns within an image.

5. Artificial intelligence: Associative memory is used in artificial intelligence applications such as expert systems and pattern recognition.
6. Database management: Associative memory can be used in database management systems to quickly retrieve data based on its content.

Hardware Organization of Associative Memory:

The following diagram shows the block representation of an Associative memory -



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.

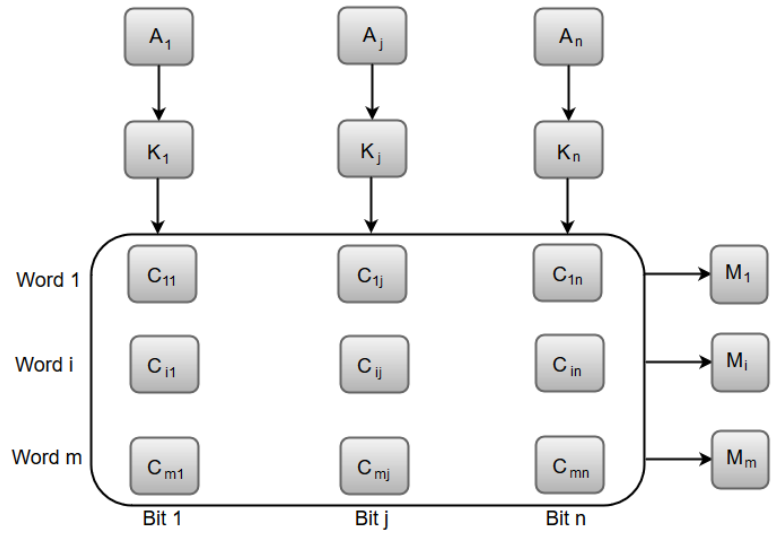
The words which are kept in the memory are compared in parallel with the content of the argument register.

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.

Relationship between the memory array and the external registers in associative memory:

The following diagram can represent the relation between the memory array and the external registers in an associative memory -

Associative memory of m word, n cells per word:



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.

Read-Write operation in Associative Memory:

1. Read Operation:

- In an associative memory, a read operation involves presenting a search key or a portion of data to the memory.
- The memory compares the presented key with all stored entries simultaneously, searching for a match.
- If a match is found, the memory returns the associated data or information corresponding to the matched entry.
- Associative memory allows for extremely fast retrieval of data since it can search the entire memory in parallel rather than sequentially.

2. **Write Operation:**

- In associative memory, a write operation involves storing data along with its associated key.
- When writing data into associative memory, both the data and its corresponding key are presented to the memory.
- The memory then compares the presented key with all stored keys simultaneously to determine if there is a match.
- If a match is found, the memory updates the associated data with the newly provided data.
- If no match is found, the memory creates a new entry and stores the provided data along with its associated key.
- It's worth noting that some associative memory systems may have limitations on write operations, such as allowing updates only to existing entries or requiring specific protocols for adding new entries.

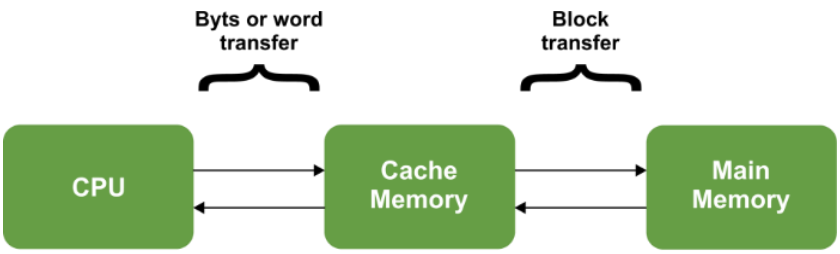
Cache Memory:

Cache memory is a type of high-speed volatile computer memory that is used to temporarily store frequently accessed data and instructions. It serves as a buffer between the CPU and the slower main memory (RAM), providing faster access to commonly used data and instructions and reducing the average time to access memory.

Cache memory operates on the principle of temporal and spatial locality, which states that programs tend to access the same memory locations repeatedly and neighboring memory locations tend to be accessed together. By storing copies of frequently accessed data and instructions in the cache, the CPU can quickly retrieve them without having to access the slower main memory every time.

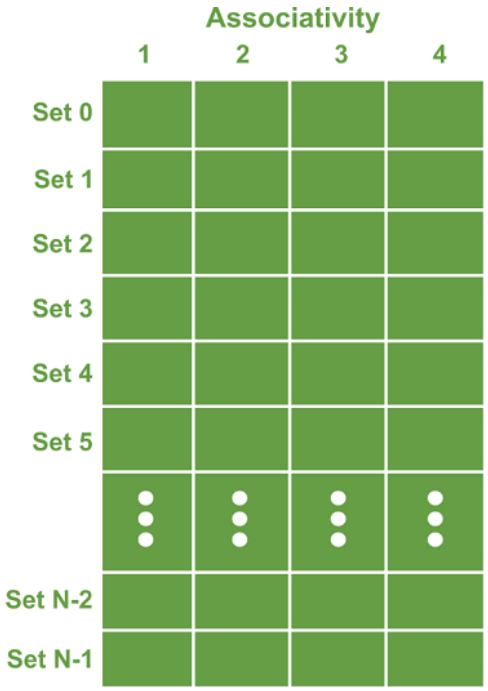
The data or contents of the main memory that are used frequently by CPU are stored in the cache memory so that the processor can easily access that data in a shorter time. Whenever the CPU needs to access memory, it first checks the cache memory. If the data is not found in cache memory, then the CPU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

Cache memory is placed between the CPU and the main memory. The block diagram for a cache memory can be represented as:



The cache is the fastest component in the memory hierarchy and approaches the speed of CPU components.

Cache memory is organised as distinct set of blocks where each set contains a small fixed number of blocks.



Logical organisation of a 4-way set associate cache

As shown in the above sets are represented by the rows. The example contains N sets and each set contains four blocks. Whenever an access is made to cache, the cache controller does not search the entire cache in order to look for a match. Rather, the controller maps the address to a particular set of the cache and therefore searches only the set for a match.

If a required block is not found in that set, the block is not present in the cache and cache controller does not search it further. This kind of cache organisation is called set associative because the cache is divided into distinct sets of blocks. As each set contains four blocks the cache is said to be four way set associative.

The basic operation of a cache memory is as follows:

- When the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the fast memory.
- If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word.
- A block of words one just accessed is then transferred from main memory to cache memory. The block size may vary from one word (the one just accessed) to about 16 words adjacent to the one just accessed.

- The performance of the cache memory is frequently measured in terms of a quantity called **hit ratio**.
- When the CPU refers to memory and finds the word in cache, it is said to produce a **hit**.
- If the word is not found in the cache, it is in main memory and it counts as a **miss**.
- The ratio of the number of hits divided by the total CPU references to memory (hits plus misses) is the hit ratio.

Cache Memory Levels and Types:

There are several levels of cache memory in modern computer systems:

1. **L1 Cache**: This is the smallest and fastest cache memory located directly on the CPU chip itself. It is divided into separate instruction cache (L1i) and data cache (L1d) components. L1 cache has extremely fast access times but limited capacity.
2. **L2 Cache**: L2 cache is located on the CPU chip or very close to it. It is larger but slower than L1 cache and typically serves as a secondary cache layer. Some CPUs have multiple levels of L2 cache.
3. **L3 Cache**: L3 cache is shared among multiple CPU cores in a multi-core processor or across multiple CPU chips in a multi-processor system. It has larger capacity but slower access times compared to L1 and L2 cache.

Types of Cache:

There are mainly two types of Cache, and we are going to discuss each of them below.

1. **Primary Cache:** It is the type of Cache always located on the processor chip. Its access time is significantly less than the processor register and is very small in size.
2. **Secondary Cache:** It is placed between the rest of the memory and the cache memory. It is also known as Level 2 cache, and it is also placed on the processor.

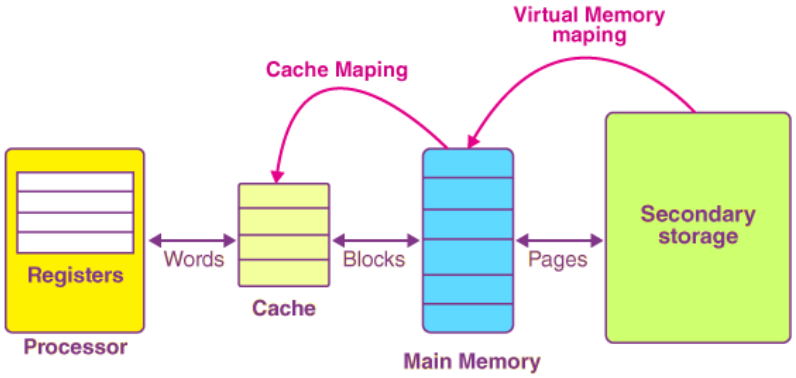
Cache Mapping:

- 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 –

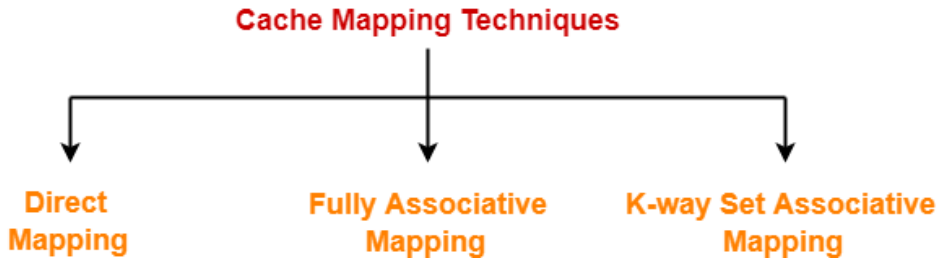


Now, before we proceed ahead, it is very crucial that we note these points:

Important Note:

- The main memory gets divided into multiple partitions of equal size, known as the frames or blocks.
- The cache memory is actually divided into various partitions of the same sizes as that of the blocks, known as lines.
- The main memory block is copied simply to the cache during the process of cache mapping, and this block isn't brought at all from the main memory.

Techniques of Cache Mapping:



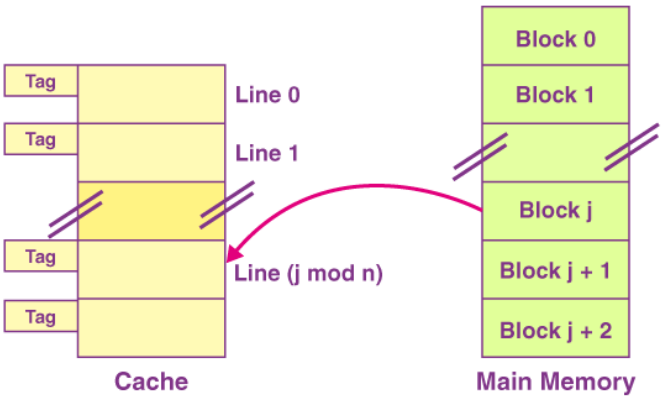
1. Direct Mapping:

In the case of direct mapping, a certain block of the main memory would be able to map a cache only up to a certain line of the cache. The total line numbers of cache to which any distinct block can map are given by the following:

Cache line number = (Address of the Main Memory Block) Modulo (Total number of lines in Cache)

For example,

- Let us consider that particular cache memory is divided into a total of ‘n’ number of lines.
- Then, the block ‘j’ of the main memory would be able to map to line number only of the cache (j mod n).



Direct Mapping

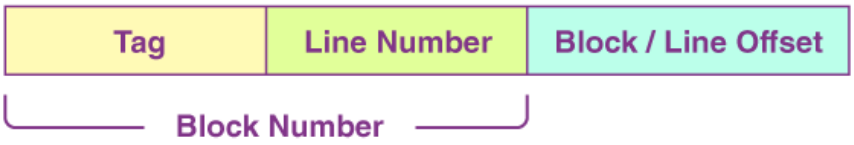
The Need for Replacement Algorithm:

In the case of direct mapping,

- There is no requirement for a replacement algorithm.
- It is because the block of the main memory would be able to map to a certain line of the cache only.
- Thus, the incoming (new) block always happens to replace the block that already exists, if any, in this certain line.

Division of Physical Address:

In the case of direct mapping, the division of the physical address occurs as follows:



Division of Physical Address in Direct Mapping

Hit latency:

The time taken to find out whether the required word is present in the **Cache Memory** or not is called as **hit latency**.

For direct mapped cache,

Hit latency = Multiplexer latency + Comparator latency

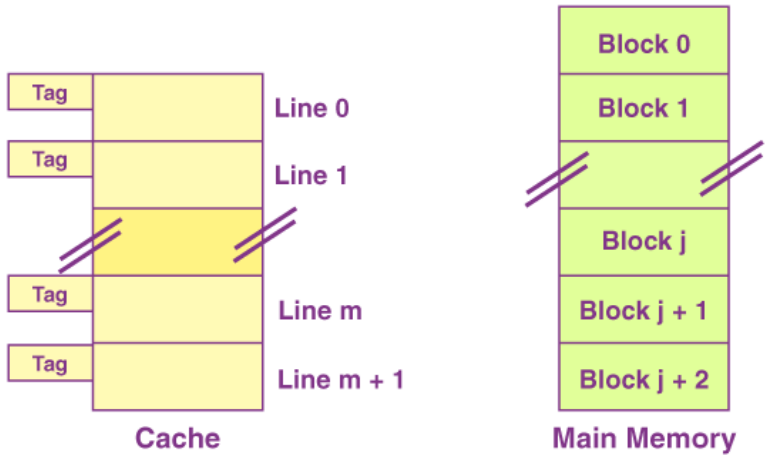
2. Fully Associative Mapping:

In the case of fully associative mapping,

- The main memory block is capable of mapping to any given line of the cache that's available freely at that particular moment.
- It helps us make a fully associative mapping comparatively more flexible than direct mapping.

For Example:

Let us consider the scenario given as follows:



Fully Associative Mapping

Here, we can see that,

- Every single line of cache is available freely.
- Thus, any main memory block can map to a line of the cache.
- In case all the cache lines are occupied, one of the blocks that exists already needs to be replaced.

The Need for Replacement Algorithm

In the case of fully associative mapping,

- The replacement algorithm is always required.
- The replacement algorithm suggests a block that is to be replaced whenever all the cache lines happen to be occupied.
- So, replacement algorithms such as LRU Algorithm, FCFS Algorithm, etc., are employed.

Division of Physical Address

In the case of fully associative mapping, the division of the physical address occurs as follows:

Division of Physical Address in Fully Associative Mapping

3. K-way Set Associative Mapping:

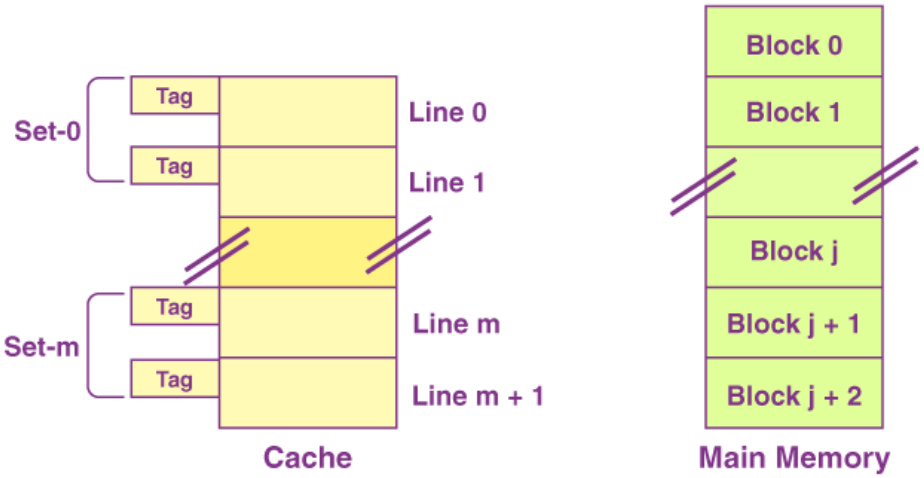
In the case of k-way set associative mapping,

- The grouping of the cache lines occurs into various sets where all the sets consist of k number of lines.
- Any given main memory block can map only to a particular cache set.
- However, within that very set, the block of memory can map any cache line that is freely available.
- The cache set to which a certain main memory block can map is basically given as follows:

Cache set number = (Block Address of the Main Memory) Modulo (Total Number of sets present in the Cache)

For Example:

Let us consider the example given as follows of a two-way set-associative mapping:



2-Way Set Associative Mapping

In this case,

- k = 2 would suggest that every set consists of two cache lines.

- Since the cache consists of 6 lines, the total number of sets that are present in the cache = 6 / 2 = 3 sets.
- The block ‘j’ of the main memory is capable of mapping to the set number only (j mod 3) of the cache.
- Here, within this very set, the block ‘j’ is capable of mapping to any cache line that is freely available at that moment.
- In case all the available cache lines happen to be occupied, then one of the blocks that already exist needs to be replaced.

The Need for Replacement Algorithm:

In the case of k-way set associative mapping,

- The k-way set associative mapping refers to a combination of the direct mapping as well as the fully associative mapping.
- It makes use of the fully associative mapping that exists within each set.
- Therefore, the k-way set associative mapping needs a certain type of replacement algorithm.

Division of Physical Address:

In the case of fully k-way set mapping, the division of the physical address occurs as follows:

Division of Physical Address in K-way Set Associative Mapping

Hit latency:

For set associative mapping,

Hit latency = Multiplexer latency + Comparator latency + OR Gate latency

Special Cases:

- In case k = 1, the k-way set associative mapping would become direct mapping. Thus,

  Direct Mapping = one-way set associative mapping

- In the case of k = The total number of lines present in the cache, then the k-way set associative mapping would become **fully associative mapping**.

Locality of Reference:

There are mainly two localities of reference, and here we are going to discuss both of them.

1. **Temporal Locality of Reference:** This LRU (Least Recent Algorithm) will be used. Whenever a web page fault happens inside a phrase, it will no longer best load phrase in the primary reminiscence. However, the entire web page fault might be loaded because the spatial locality of reference rule says that if you are referring to any phrase, the subsequent phrase might be referred to in its check-in, that's why we load the entire web page desk so the entire block might be loaded.
2. **Spatial Locality of Reference:** In this, there is a chance that an element can be present in the close proximity of the reference point and will become closer and closer when searched again.

Virtual Memory:

Virtual memory is a memory management technique used by computer operating systems to provide an illusion of a larger, contiguous, and more accessible memory space than the physical RAM (Random Access Memory) available on a system.

Purpose of Virtual Memory:

1. Extending physical memory capabilities to run larger programs and handle larger data sets.
2. Providing memory isolation and protection between different processes.
3. Simplifying memory management by presenting a large, contiguous virtual address space.
4. Improving memory utilization by loading only required portions of memory and swapping out unused parts.
5. Enabling features like demand paging, copy-on-write, memory mapping, and memory sharing.
6. Leveraging locality and caching to improve performance by keeping frequently accessed data in memory.

Address Space and Memory Space:

An address used by a programmer will be termed as a virtual address and set of such addresses is the address space.

• An address in main memory is known as a physical address.

• The set of these locations is termed as memory space. So address space is set of addresses produced by programs as they reference instructions and data, memory space comprises actual main memory locations directly addressable for processing.

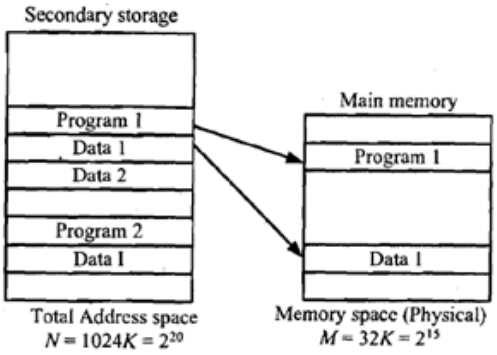


Diagram of Address Space and Memory Space:

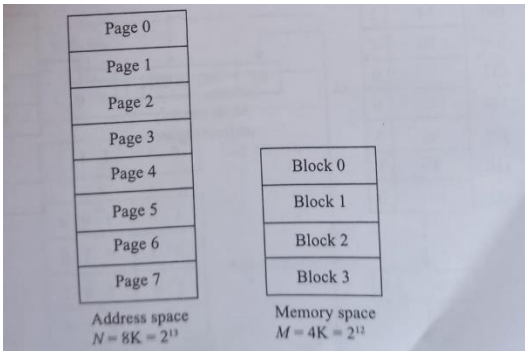
• Supposes a computer which has a main-memory capacity of 64K words (K=1024). 16-bits are required to specify a physical address in memory because $64K = 2^{16}$. Assume that computer has auxiliary memory for storing information equivalent to capacity of 16 main memories. Let's signify address space by N and memory space by M we then have for this illustration: N = 16 × 64 K = 1024K and M = 64K.

• In a multiprogramming computer system, data and programs are transferred to and from auxiliary memory and main memory based on demands obliged by CPU. Suppose program 1 is currently being executed in CPU. Program 1 and a part of its associated data are moved from secondary memory in the main memory as displayed in Figure below. Parts of programs and data require not being in contiguous locations in memory because information is being moved in and out and empty spaces may be available in scattered locations in memory.

Address Mapping Using Pages:

A physical memory is broken down into groups of equal size called blocks or page frame and the groups of address space of the same size as block is called pages.

Consider a computer a with address space =8K and memory space=4K



Associative Memory Page Table:

An associative memory page table is a data structure used in computer operating systems to translate virtual addresses generated by the CPU into physical addresses in main memory (RAM).

Unlike traditional page tables that use hierarchical structures like tables or trees, an associative memory page table employs associative memory (also known as content-addressable memory or CAM) to enable rapid address translation.

Here's how an associative memory page table works:

1. **Translation Process:**
   - When a CPU generates a virtual address, it needs to be translated into a physical address to access data in main memory.
   - The virtual address typically consists of a page number and an offset within the page.
   - The associative memory page table stores mappings between virtual page numbers and corresponding physical page frame numbers.
2. **Content-Addressable Memory (CAM):**
   - Associative memory allows for parallel search and retrieval based on content rather than addresses.
   - In the context of a page table, each entry in the associative memory contains a virtual page number (or a portion thereof) and the corresponding physical page frame number.
   - When the CPU generates a virtual address, the virtual page number is presented to the associative memory, which searches for a match.
   - If a match is found, the associated physical page frame number is retrieved, enabling the CPU to access the corresponding data in main memory.
3. **Rapid Address Translation:**
   - The use of associative memory for page table lookup enables rapid address translation, as the entire memory can be searched in parallel for a matching virtual page number.
   - This approach eliminates the need for sequential searches typically associated with hierarchical page table structures, resulting in faster address translation and reduced memory access latency.

4. Scalability and Efficiency:

- Associative memory page tables can efficiently handle large address spaces, as the search time remains constant regardless of the number of entries in the page table.
- Additionally, associative memory page tables are particularly suitable for systems with variable memory allocation requirements, as they can dynamically adapt to changes in the mapping between virtual and physical addresses.

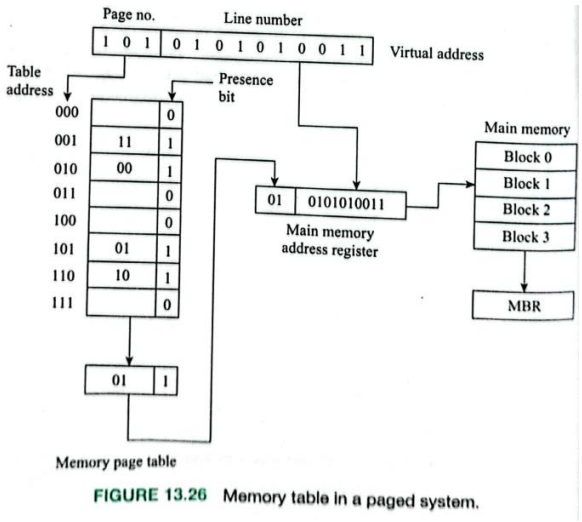


FIGURE 13.26 Memory table in a paged system.

Paging:

Paging is another implementation of Virtual Memory. The logical storage is marked as Pages of some size, say 4KB. The MM is viewed and numbered as page frames. Each page frame equals the size of Pages. The Pages from the logical view are fitted into the empty Page Frames in MM. This is synonymous to placing a book in a bookshelf. Also, the concept is similar to cache blocks and their placement.

Below Figure explains how two program's pages are fitted in Page Frames in MM. As you see, any page can get placed into any available Page Frame. Unallotted Page Frames are shown in white.

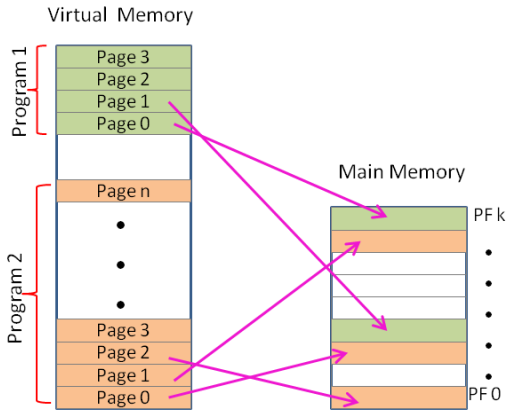


Figure : Virtual Memory Pages to MM Page Frame Mapping

This mapping is necessary to be maintained in a Page Table. The mapping is used during address translation. Typically a page table contains virtual page address, corresponding physical frame number where the page is stored, Presence bit, Change bit and Access rights (Refer figure19.6). This Page table is referred to check whether the desired Page is available in the MM. The Page Table resides in a part of MM. Thus every Memory access requested by CPU will refer memory twice – once to the page table and second time to get the data from accessed location. This is called the Address Translation Process and is detailed in figure19.7.

Virtual Page Address	Page Frame Number	Presence bit P	Change bit C	Access Rights
A	0004000	1	0	R, X
B	0645728	0	1	R, W, X
D	0010234	1	1	R, W, X
F	0060216	0	0	R

Figure 19.6 Typical Page Table Entry

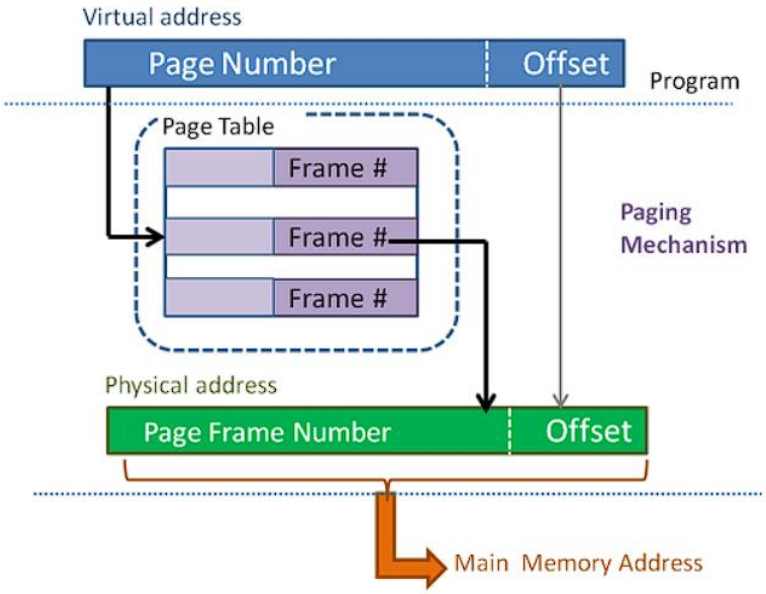


Figure 19.7 Virtual Memory Address Translation in Paging Implementation

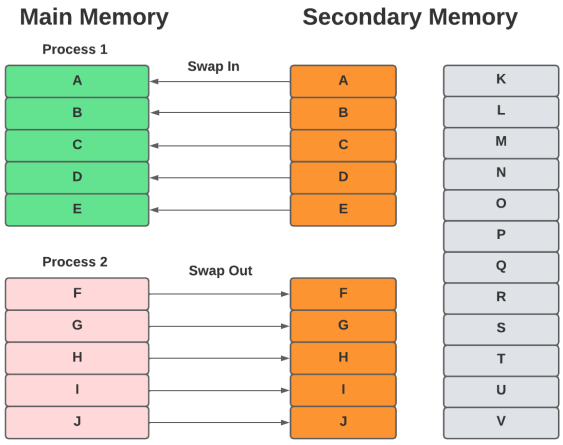
Page size determination is an important factor to obtain Maximum Page Hits and Minimum Thrashing. Thrashing is very costly in VM as it means getting data from Disk, which is 1000 times likely to be slower than MM.

In the Paging Mechanism, Page Frames of fixed size are allotted. There is a possibility that some of the pages may have contents less than the page size, as we have in our printed books. This causes unutilized space (fragment) in a page frame. By no means, this unutilized space is usable for any other purpose. Since these fragments are inside the allotted Page Frame, it is called ***Internal Fragmentation*.**

Demand Paging:

A demand paging mechanism is similar to a paging system with swapping, except that processes are maintained in secondary memory, and pages are loaded only when needed, rather than in preparation.

As a result, when a context switch occurs, the OS does not copy any of the old program's pages from disks or any of the new program's pages into the main memory. Instead, after loading the first page, it will begin executing the new program and fetch the program's pages, which are referenced.



If a program addresses a page that may not be available in the main memory because it was swapped during execution, the processor treats it as an invalid memory reference. This is because page faults and transfers send the control back to the OS, which requests that the page be stored back into memory.

Common Terms in Demand Paging:

Following are some common terms in demand paging operating systems:

1. Page Fault: There will be a miss if the referenced page is not present in the main memory; this is known as a page miss or page fault.

The CPU must look up the missing page in secondary memory. When the number of page faults is significant, the system's effective access time increases dramatically.

2. Swapping: Swapping comprises either erasing all of the process's pages from memory or marking the pages so that we can remove them via the page replacement method.

When a process is suspended, it indicates it is unable to run. However, we can change the process for a while. The system can swap the process from secondary memory to primary memory over a period of time. Thrashing describes a condition in which a process is busy, and the pages are swapped in and out of it.

3. Thrashing: The effective access time will be the time needed by the CPU to read one word from the secondary memory if the number of page faults is equal to the number of referred pages or if the number of page faults is so high that the CPU is only reading pages from the secondary memory. This is known as Thrashing.

If the page fault rate is PF%, the time spent retrieving a page from secondary memory and resuming is S (service time), and the memory access time is "ma", the effective access time may be calculated as follows:

$$EAT = PF \times S + (1 - PF) \times (ma)$$

Translation Look-aside Buffer (TLB):

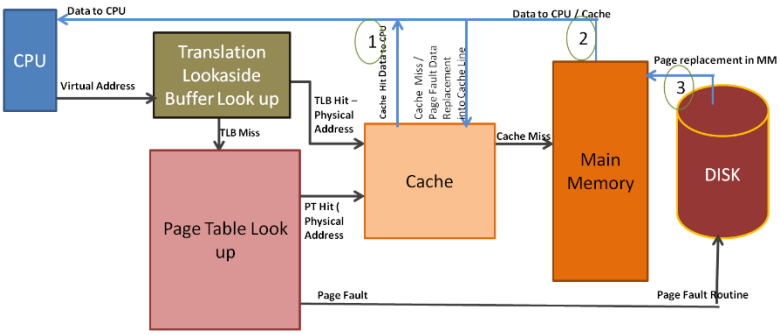
Every Virtual address Translation requires two memory references,

- once to read the segment/page table and
- once more to read the requested memory word.

TLB is a hardware functionality designed to speed up Page Table lookup by reducing one extra access to MM. A TLB is a fully associative cache of the Page Table. The entries in TLB correspond to the recently used translations. TLB is sometimes referred to as **address cache**. TLB is part of the Memory Management Unit (MMU) and MMU is present in the CPU block.

TLB entries are similar to that of Page Table. With the inclusion of TLB, every virtual address is initially checked in TLB for address translation. If it is a TLB Miss, then the page table in MM is looked into. Thus, a TLB Miss does not cause Page fault. Page fault will be generated only if it is a miss in the Page Table too but not otherwise. Since TLB is an associative address cache in CPU, TLB hit provides the fastest possible address translation; Next best is the page hit in Page Table; worst is the page fault.

Having discussed the various individual Address translation options, it is to be understood that in a Multilevel Hierarchical Memory all the functional structures coexist. i.e. TLB, Page Tables, Segment Tables, Cache (Multiple Levels), Main Memory and Disk. Page Tables can be many and many levels too, in which case, few Page tables may reside in Disk. In this scenario, what is the hierarchy of verification of tables for address translation and data service to the CPU? Refer figure19.8.



Address Translation verification sequence starts from the lowest level i.e.

TLB -> Segment / Page Table Level 1 -> Segment / Page Table Level n

Once the address is translated into a physical address, then the data is serviced to CPU. Three possibilities exist depending on where the data is.

Case 1 - TLB or PT hit and also Cache Hit - Data returned from CPU to Cache

Case 2 - TLB or PT hit and Cache Miss - Data returned from MM to CPU and Cache

Case 3 - Page Fault - Data from disk loaded into a segment / page frame in MM; MM returns data to CPU and Cache

It is simple, in case of Page hit either Cache or MM provides the Data to CPU readily. The protocol between Cache and MM exists intact. If it is a Segment/Page fault, then the routine is handled by OS to load the required data into Main Memory. In this case, data is not in the cache too. Therefore, while returning data to CPU, the cache is updated treating it as a case of Cache Miss.

Fragmentation:

Fragmentation occurs when we break the memory into small-sized blocks. Furthermore, fragmentations are non-contiguous in nature. Hence, we can't allocate them to processes. As a result, memory will have multiple unused blocks while processes are pending in the queue and waiting for the memory. **Fragmentation can be either internal or external.**

Internal fragmentation occurs when we split the physical memory into contiguous mounted-sized blocks and allocate memory for a process that can be larger than the memory requested. Hence, the unused allocated space is left and can't be used by other processes. Best fit block search is the solution for internal fragmentation.

External fragmentation occurs when total unused memory space is enough to answer all the allocation requests. Here the memory is non-contiguous. Therefore, the memory has empty blocks scattered all over, which are insufficient to be allocated to other programs. Compaction is the solution for external fragmentation.

Page Replacement:

Page replacement is a crucial aspect of virtual memory management in operating systems.

When a program requires more memory than is physically available, virtual memory allows the operating system to use disk storage as an extension of RAM.

However, since physical memory is limited, the operating system must decide which pages of memory to keep in RAM and which ones to swap out to disk. Page replacement algorithms govern this decision-making process.

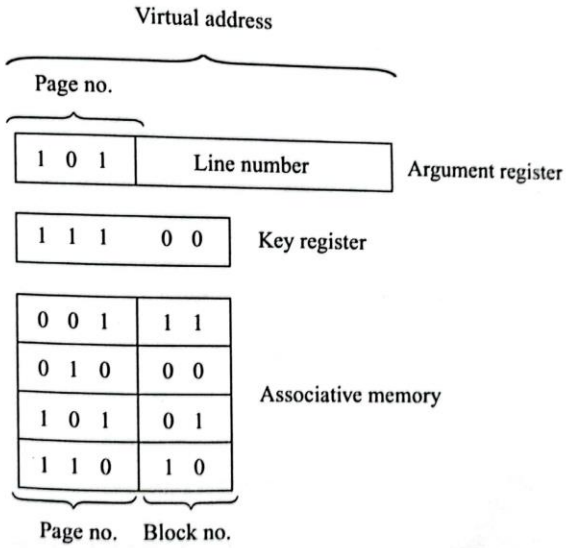


FIGURE 13.27 An associative memory page table.

Here's how page replacement in virtual memory typically works:

1. **Page Fault:**
   - When a program attempts to access a page that is not currently in RAM, a page fault occurs.
   - The operating system intercepts this page fault and triggers a page replacement algorithm to select a page to evict from RAM to make space for the requested page.
2. **Page Replacement Algorithms:**
   - Various page replacement algorithms exist, each with its own criteria for selecting which page to replace.
   - Common page replacement algorithms include:
     - **Optimal:** Replaces the page that will not be used for the longest period in the future. This algorithm provides the best possible theoretical performance but is impractical to implement as it requires knowledge of future memory accesses.
     - **FIFO (First-In-First-Out):** Replaces the oldest page in memory, i.e., the page that has been in memory the longest. It's simple to implement but may suffer from the "Belady's anomaly," where increasing the number of frames can lead to an increase in page faults.

- **LRU (Least Recently Used):** Replaces the page that has not been accessed for the longest period of time. This algorithm is more complex to implement but generally performs well in practice.
- **LFU (Least Frequently Used):** Replaces the page with the fewest accesses since it was loaded into memory. This algorithm considers the frequency of page accesses.
- **Clock (Second-Chance):** Similar to FIFO but incorporates a "use bit" for each page to determine whether it has been accessed since the last examination. Pages are evicted if their use bit is not set.
- **LRU Approximation:** Provides an approximation of LRU using additional data structures or algorithms that are simpler to implement but still aim to evict the least recently used pages.

3. **Page Replacement Process:**

- When a page is selected for replacement, the operating system writes the page's contents back to disk if it has been modified (dirty), updating the corresponding page table entry to indicate that the page is no longer in memory.
- The operating system then loads the requested page from disk into the newly vacated memory frame and updates the page table accordingly.
- Finally, the program's execution resumes, and the requested memory access is allowed to proceed.