Tulsiramji Gaikwad-Patil College Engineering and Technology
Department of Computer Science and Engineering
Semester: B.E. III Semester
Subject: Computer Architecture and organization
Solution Set: Summer 2018

Q1 a) State & explain the various addressing modes. Give one example for each of the addressing modes.

Ans-

1. **Register mode -** The operand is the contents of a processor register; the name (address) of the register is given in the instruction.
2. **Absolute mode –** The operand is in a memory location; the address of this location is given explicitly in the instruction. (In some assembly languages, this mode is called Direct).

   The instruction

   Move LOC, R2

   Processor registers are used as temporary storage locations where the data is a register are accessed using the Register mode. The Absolute mode can represent global variables in a program. A declaration such as

   Integer A, B;

3. **Immediate mode –** The operand is given explicitly in the instruction.

   For example, the instruction

   Move $200_{immediate}$, R0

   Places the value 200 in register R0. Clearly, the Immediate mode is only used to specify the value of a source operand. Using a subscript to denote the Immediate mode is not appropriate in assembly languages. A common convention is to use the sharp sign (#) in front of the value to indicate that this value is to be used as an immediate operand. Hence, we write the instruction above in the form

   Move #200, R0

INDIRECTION AND POINTERS:-

In the addressing modes that follow, the instruction does not give the operand or its address explicitly; instead, it provides information from which the memory

address of the operand can be determined. We refer to this address as the effective address (EA) of the operand.

1. **Indirect mode –** The effective address of the operand is the contents of a register or memory location whose address appears in the instruction.
   To execute the Add instruction in fig (a), the processor uses the value which is in register R1, as the effective address of the operand. It requests a read operation from the memory to read the contents of location B. the value read is the desired operand, which the processor adds to the contents of register R0. Indirect addressing through a memory location is also possible as shown in fig (b). In this case, the processor first reads the contents of memory location A, then requests a second read operation using the value B as an address to obtain the operand

The register or memory location that contains the address of an operand is called a pointer. Indirection and the use of pointers are important and powerful concepts in programming.

In the program shown Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2. The initialization section of the program loads the counter value n from memory location N into R1 and uses the immediate addressing mode to place the address value NUM1, which is the address of the first number in the list, into R2. Then it clears R0 to 0. The first two instructions in the loop implement the unspecified instruction block starting at LOOP. The first time through the loop, the instruction **Add (R2), R0** fetches the operand at location NUM1 and adds it to R0. The second Add instruction adds 4 to the contents of the pointer R2, so that it will contain the address value NUM2 when the above instruction is executed in the second pass through the loop.

Where B is a pointer variable. This statement may be compiled

into Move B, R1

Move (R1), A

Using indirect addressing through memory, the same action can be achieved with Move (B), A

Indirect addressing through registers is used extensively. The above program shows the flexibility it provides. Also, when absolute addressing is not available, indirect addressing through registers makes it possible to access global variables by first loading the operand's address in a register.

INDEXING AND ARRAYS:-

A different kind of flexibility for accessing operands is useful in dealing with lists and arrays.

Index mode – the effective address of the operand is generated by adding a constant value to the contents of a register.

The register use may be either a special register provided for this purpose, or, more commonly, it may be any one of a set of general-purpose registers in the processor. In either case, it is referred to as index register. We indicate the Index mode symbolically as

X (Ri)

Where X denotes the constant value contained in the instruction and Ri is the name of the register involved. The effective address of the operand is given by

EA = X + [Rj]

The contents of the index register are not changed in the process of generating the effective address. In an assembly language program, the constant X may be given either as an explicit number or as a symbolic name representing a numerical value.

Fig a illustrates two ways of using the Index mode. In fig a, the index register, R1, contains the address of a memory location, and the value X defines an offset (also called a displacement) from this address to the location where the operand is found. An alternative use is illustrated in fig b. Here, the constant X corresponds to a memory address, and the contents of the index register define the offset to the operand. In either case, the effective address is the sum of two values; one is given explicitly in the instruction, and the other is stored in a register.

In the most basic form of indexed addressing several variations of this basic form provide a very efficient access to memory operands in practical programming situations. For example, a second register may be used to contain the offset X, in which case we can write the Index mode as

(Ri, Rj)

The effective address is the sum of the contents of registers Ri and Rj. The second register is usually called the base register. This form of indexed addressing provides more flexibility in accessing operands, because both components of the effective address can be changed.

Another version of the Index mode uses two registers plus a constant, which can be denoted as

X(Ri, Rj)

In this case, the effective address is the sum of the constant X and the contents of registers Ri and Rj. This added flexibility is useful in accessing multiple components inside each item in a record, where the beginning of an item is specified by the (Ri, Rj) part of the addressing mode. In other words, this mode implements a three-dimensional array.

RELATIVE ADDRESSING:-

We have defined the Index mode using general-purpose processor registers. A useful version of this mode is obtained if the program counter, PC, is used instead of a general purpose register. Then, X(PC) can be used to address a memory location that is X bytes away from the location presently pointed to by the program counter.

Relative mode – The effective address is determined by the Index mode using the program counter in place of the general-purpose register Ri.

This mode can be used to access data operands. But, its most common use is to specify the target address in branch instructions. An instruction such as

Branch > 0 LOOP

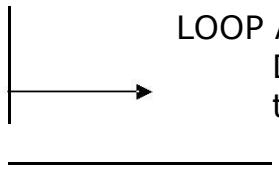
Causes program execution to go to the branch target location identified by the name LOOP if the branch condition is satisfied. This location can be computed by specifying it as an offset from the current value of the program counter. Since the branch target may be either before or after the branch instruction, the offset is given as a signed number.

Autoincrement mode – the effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically to point to the next item in a list.

(Ri)+

Autodecrement mode – the contents of a register specified in the instruction are first automatically decremented and are then used as the effective address of the operand.

-(Ri)



```
        Move      N, R1
                  #NUM1,
        Move      R2
        Clear     R0
                  (R2)+,
LOOP    Add       R0
        Decremen
        t         R1

        Branch>0  LOOP
        Move      R0, SUM
```

Fig : The Autoincrement addressing mode used in the program

Q2 a) Differentiate between Hardwired control unit & microprogrammed control unit.

Ans-

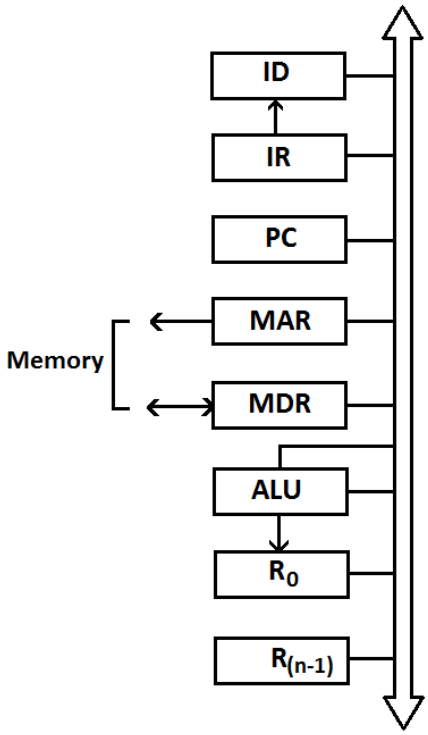
Hardwired control unit	Microprogrammed control unit
1) Speed is fast.	1) Speed is slow.
2) More costlier.	2) Cheaper.
3) Occurrence of error is more	3) Occurrence of error is less
4) Control functions implemented in hardware	4) Control functions implemented in software
5) Not flexible to accommodate new system specification or new instruction redesign is required.	5) More flexible to accommodate new system specification or new instructions.
6) Difficult to handle complex instruction sets.	6) Easier to handle complex instruction sets
7) Complicated design process.	7) Orderly, systematic and simple design process.
8) Complex decoding and sequencing logic.	8) Easier decoding and sequencing logic
9) More chip area.	9) Less chip area.
10) Application: Mostly RISC microprocessor.	10) Application: Mainframes some microprocessors

Q2 b) Explain single bus structure & generate the control signal with the help of suitable example & diagram.

Ans-

In single bus structure inside the CPU, different components are linked by a single bus. The various components available inside CPU in this architecture includes **Instruction Register (IR), Instruction Decoder (ID), Program Counter (PC), Memory Address Register (MAR), Memory Data Register (MDR), Arithmetic and Logic Unit (ALU)** and **General purpose Register.**

In single bus structure, the CPU uses **Instruction Register (IR), Instruction Decoder (ID), Program Counter (PC), Memory Address Register (MAR), Memory Data Register (MDR)** and **General purpose Register** during processing to perform different operations. Let's discuss all these registers along with ALU and how they work together in the single bus structure.



Instruction Register (IR):

Instruction Register holds the current instruction that is being executed. It doesn't store any address but stores the instruction. If we require which instruction is currently being executed by the CPU, then we will look at the contents of IR which tells us the currently executing instruction.

Instruction Decoder (ID):

Instruction Decoder decodes the opcodes of the instruction to generate the necessary signal.

Program Counter (PC):

Program Counter keeps the tracks of instruction that is to be executed next to the competition of the current instruction. So the program counter is a type of register which doesn't store any instruction but stores the address of next instruction to be executed. When the instruction is fetched, the value of PC is automatically incremented and it points to the address of next instruction.

Memory Address Register (MAR):

Memory Address Register holds the address of active memory location. When CPU wants to store or read data from memory, CPU stores the required address of memory location in MAR.

Memory Data Register (MDR):

Memory Data Register holds data. The primary job of MDR is to handle the data transfer between the main memory and the processor. That means it holds the contents of location read from or written in the memory.

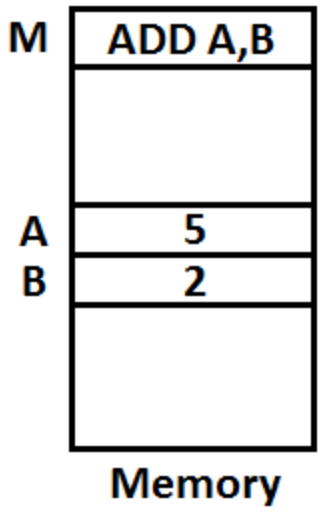
Arithmetic & Logic Unit (ALU):

Arithmetic & Logic Unit is the component where actual execution is performed. ALU is one of the significant parts of the computer system which is capable of performing different calculations. Depending on the design of ALU it makes the CPU more powerful, efficient as well as faster.

General Purpose Register:

The General Purpose Registers are used for different purposes including holding of intermediate results.

Now to understand how the single bus structure in computer organisation works, let us consider an instruction **ADD A, B** which is in location **M** of memory.



The address M is first placed in MAR and a read signal is generated by CPU. The content of location M i.e. ADD A, B instruction is placed in MDR which is then placed to Instruction Register (IR). This instruction is decoded by the Instruction Decoder (ID) and required signal components are activated to perform the addition operation.

First Operand address A is placed in MAR and read signal is generated. The operand content 5 is fetched from memory and placed in MDR. This data 5 is sent to ALU by the single bus. Second operand address B is placed in MAR and read signal is generated. Operand data 2 is fetched and placed in MDR. This data, 2 is then sent to ALU by the single bus. Once two operands are available inside the ALU, arithmetic addition is performed due to the signals generated by the system. The result 7 is temporarily placed in the general purpose register R_0.

It is then sent to MDR by the single bus. An address of memory where the result data should be stored is placed in MAR. Once, the result data and address are placed in MDR and MAR respectively, write signal is generated which fetches the result data from CPU to proper memory position.

The considered instruction ADD A, B is in 2-address instruction format. In this format, 2nd operand source is destination also. Hence, MAR will hold B and result data 7 is placed to location B of memory.

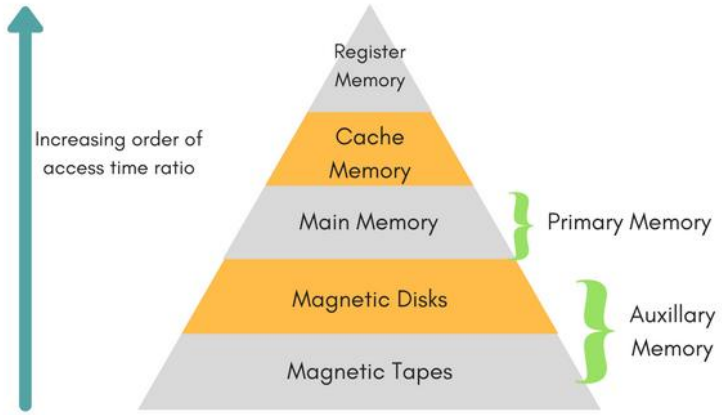
Q5) Explain the concept of memory hierarchy & characteristics of memory systems.

Ans-

A memory unit is the collection of storage units or devices together. The memory unit stores the binary information in the form of bits. Generally, memory/storage is classified into 2 categories:

- **Volatile Memory**: This loses its data, when power is switched off.

- **Non-Volatile Memory**: This is a permanent storage and does not lose any data when power is switched off.



The total memory capacity of a computer can be visualized by hierarchy of components. The memory hierarchy system consists of all storage devices contained in a computer system from the slow Auxiliary Memory to fast Main Memory and to smaller Cache memory.

Auxillary memory access time is generally **1000 times** that of the main memory, hence it is at the bottom of the hierarchy.

The **main memory** occupies the central position because it is equipped to communicate directly with the CPU and with auxiliary memory devices through Input/output processor (I/O).

When the program not residing in main memory is needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space in main memory for other programs that are currently in use.

The **cache memory** is used to store program data which is currently being executed in the CPU. Approximate access time ratio between cache memory and main memory is about **1 to 7~10**

Memory Access Methods

Each memory type, is a collection of numerous memory locations. To access data from any memory, first it must be located and then the data is read from the memory location. Following are the methods to access information from memory locations:

1. **Random Access**: Main memories are random access memories, in which each memory location has a unique address. Using this unique address any memory location can be reached in the same amount of time in any order.
2. **Sequential Access**: This methods allows memory access in a sequence or in order.

3. **Direct Access**: In this mode, information is stored in tracks, with each track having a separate read/write head.

Main Memory

The memory unit that communicates directly within the CPU, Auxillary memory and Cache memory, is called main memory. It is the central storage unit of the computer system. It is a large and fast memory used to store data during computer operations. Main memory is made up of **RAM** and **ROM**, with RAM integrated circuit chips holing the major share.

- RAM: Random Access Memory
  - **DRAM**: Dynamic RAM, is made of capacitors and transistors, and must be refreshed every 10~100 ms. It is slower and cheaper than SRAM.
  - **SRAM**: Static RAM, has a six transistor circuit in each cell and retains data, until powered off.
  - **NVRAM**: Non-Volatile RAM, retains its data, even when turned off. Example: Flash memory.
- ROM: Read Only Memory, is non-volatile and is more like a permanent storage for information. It also stores the **bootstrap loader** program, to load and start the operating system when computer is turned on. **PROM**(Programmable ROM), **EPROM**(Erasable PROM) and **EEPROM**(Electrically Erasable PROM) are some commonly used ROMs.

Auxiliary Memory

Devices that provide backup storage are called auxiliary memory. **For example:** Magnetic disks and tapes are commonly used auxiliary devices. Other devices used as auxiliary memory are magnetic drums, magnetic bubble memory and optical disks.

It is not directly accessible to the CPU, and is accessed using the Input/Output channels.

Cache Memory

The data or contents of the main memory that are used again and again by CPU, are stored in the cache memory so that we can easily access that data in 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 onto the main memory. It also transfers block of recent data into the cache and keeps on deleting the old data in cache to accomodate the new one.

Hit Ratio

The performance of cache memory is 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 cache, it is in main memory then it counts as a **miss**.

The ratio of the number of hits to the total CPU references to memory is called hit ratio.

Hit Ratio = Hit/(Hit + Miss)

Associative Memory

It is also known as **content addressable memory (CAM)**. It is a memory chip in which each bit position can be compared. In this the content is compared in each bit cell which allows very fast table lookup. Since the entire chip can be compared, contents are randomly stored without considering addressing scheme. These chips have less storage capacity than regular memory chips.

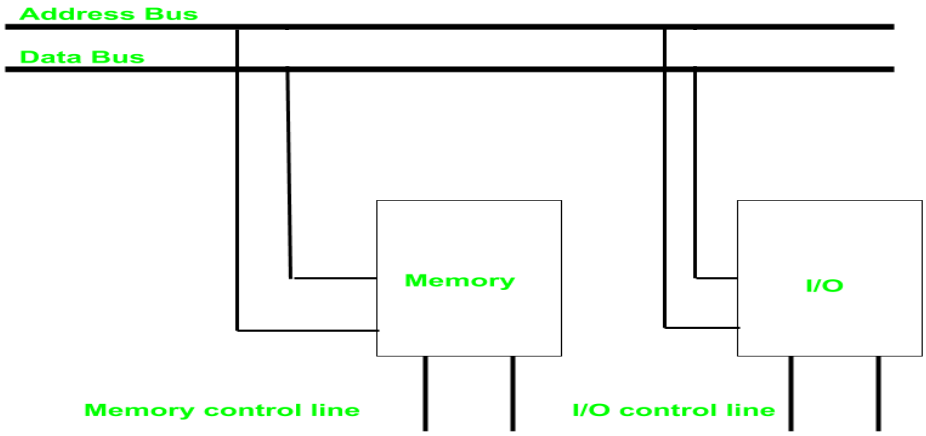
Q7 a) Differentiate between IO mapped I/O & memory mapped I/O devices.

Ans-

Isolated I/O	Memory Mapped I/O
Memory and I/O have separate address space	Both have same address space
All address can be used by the memory	Due to addition of I/O addressable memory become less for memory
Separate instruction control read and write operation in I/O and Memory	Same instructions can control both I/O and Memory
In this I/O address are called ports.	Normal memory address are for both
More efficient due to separate buses	Lesser efficient
Larger in size due to more buses	Smaller in size
It is complex due to separate logic is used to control both.	Simpler logic is used as I/O is also treated as memory only.

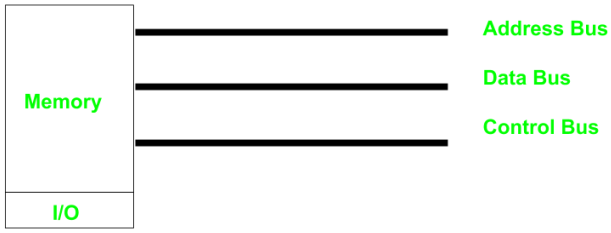
Isolated I/O –

Then we have Isolated I/O in which we Have common bus(data and address) for I/O and memory but separate read and write control lines for I/O. So when CPU decode instruction then if data is for I/O then it places the address on the address line and set I/O read or write control line on due to which data transfer occurs between CPU and I/O. As the address space of memory and I/O is isolated and the name is so. The address for I/O here is called ports. Here we have different read-write instruction for both I/O and memory.



Memory Mapped I/O –

In this case every bus in common due to which the same set of instructions work for memory and I/O. Hence we manipulate I/O same as memory and both have same address space, due to which addressing capability of memory become less because some part is occupied by the I/O.



Q7 b) Explain magnetic disk in detail.

Ans-

A magnetic disk primarily consists of a rotating magnetic surface and a mechanical arm that moves over it. The mechanical arm is used to read from and write to the disk. The data on a magnetic disk is read and written using a magnetization process. Data is organized on the disk in the form of tracks and sectors, where tracks are the circular divisions of the disk. Tracks are further divided into sectors that contain blocks of data. All read and write operations on the magnetic disk are performed on the sectors.

Magnetic disks have traditionally been used as primary storage in computers. With the advent of solid-state drives (SSDs), magnetic disks are no longer considered the only option, but are still commonly used.

Tracks and Spots

The disk surface is divided into concentric tracks (circles within circles). The thinner the tracks, the more storage. The data bits are recorded as tiny magnetic spots on the

tracks. The smaller the spot, the more bits per inch and the greater the storage.

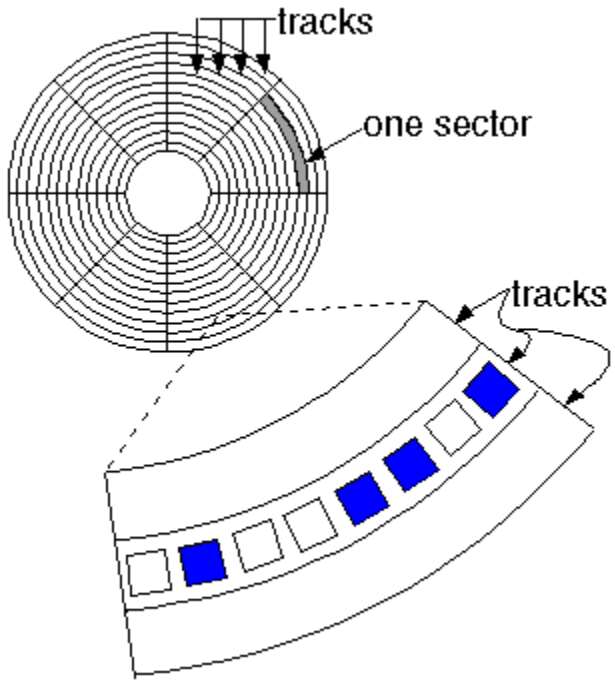
Sectors

Tracks are further divided into sectors, which hold a block of data that is read or written at one time; for example, READ SECTOR 782, WRITE SECTOR 5448. In order to update the disk, one or more sectors are read into the computer, changed and written back to disk. The operating system figures out how to fit data into these fixed spaces.

Modern disks have more sectors in the outer tracks than the inner ones because the outer radius of the platter is greater than the inner radius (see CAV). See magnetic tape and optical disc.

Tracks and Sectors

Tracks are concentric circles on the disk, broken up into storage units called "sectors." The sector, which is typically 512 bytes, is the smallest unit that can be read or written.



Q8 a) Differentiate between synchronous & Asynchronous data transfer.

Ans-

Point of Comparison	Synchronous Transmission	Asynchronous Transmission

Definition	Transmits data in the form of chunks or frames	Transmits 1 byte or character at a time
Speed of Transmission	Quick	Slow
Cost	Expensive	Cost-effective
Time Interval	Constant	Random
With gap between the data?	Yes	None
Examples	Chat Rooms, Telephonic Conversations, Video Conferencing	Email, Forums, Letters

Synchronous vs. Asynchronous Transmission

1. In synchronous transmission data is transmitted in the form of chunks, while in asynchronous transmission data is transmitted one byte at a time.
2. Synchronous transmission needs a clock signal between the source and target to let the target know of the new byte. While in asynchronous transmission, a clock signal is not needed because of the parity bit attached to the data sent which serves as a start indicator of the new byte.
3. Data transfer rate of synchronous transmission is faster since it transmits in chunks of data, compared to asynchronous transmission which transmits one byte at a time.
4. Asynchronous transmission is straightforward and cost-effective while synchronous transmission is complicated and pricey.
5. Synchronous transmission is systematic and needs lower overhead compared to asynchronous transmission.

Both **synchronous and asynchronous** transmission have their benefits and limitations. Asynchronous is used for sending a small amount of data while synchronous transmission is used for sending bulk of data. Therefore, that both synchronous and asynchronous transmissions are essential for data transmission.

Q8 b) Explain the concept of Interrupt with all its types

Ans-

Interrupt is a signal which has highest priority from hardware or software which processor should process its signal immediately.

Types of Interrupts:

Although interrupts have highest priority than other signals, there are many type of interrupts but basic type of interrupts are

1. **Hardware Interrupts:** If the signal for the processor is from external device or hardware is called hardware interrupts. Example: from keyboard we will press the key to do some action this pressing of key in keyboard will generate a signal which is given to the processor to do action, such interrupts are called hardware interrupts. Hardware interrupts can be classified into two types they are
    - **Maskable Interrupt:** The hardware interrupts which can be delayed when a much highest priority interrupt has occurred to the processor.
    - **Non Maskable Interrupt:** The hardware which cannot be delayed and should process by the processor immediately.
2. **Software Interrupts:** Software interrupt can also divided in to two types. They are
    - **Normal Interrupts:** the interrupts which are caused by the software instructions are called software instructions.
    - **Exception:** unplanned interrupts while executing a program is called Exception. For example: while executing a program if we got a value which should be divided by zero is called a exception.

Classification of Interrupts According to Periodicity of Occurrence:

1. **Periodic Interrupt:** If the interrupts occurred at fixed interval in timeline then that interrupts are called periodic interrupts
2. **Aperiodic Interrupt:** If the occurrence of interrupt cannot be predicted then that interrupt is called aperiodic interrupt.

Classification of Interrupts According to the Temporal Relationship with System Clock:

1. **Synchronous Interrupt:** The source of interrupt is in phase to the system clock is called synchronous interrupt. In other words interrupts which are dependent on the system clock. Example: timer service that uses the system clock.
2. **Asynchronous Interrupts:** If the interrupts are independent or not in phase to the system clock is called asynchronous interrupt.

Q9 a) What is data hazards? Explain in detail with example

Ans-

Data hazards occur when instructions that exhibit data dependence, modify data in different stages of a pipeline. Hazard cause delays in the pipeline. There are mainly three types of data hazards:

1) RAW (Read after Write) [Flow dependency]
2) WAR (Write after Read) [Anti-Data dependency]
3) WAW (Write after Write) [Output dependency]

Let there be two instructions I and J, such that J follow I. Then,

- RAW hazard occurs when instruction J tries to read data before instruction I writes it.
  Eg:
  I: R2 <- R1 + R3
  J: R4 <- R2 + R3
- WAR hazard occurs when instruction J tries to write data before instruction I reads it.
  Eg:
  I: R2 <- R1 + R3
  J: R3 <- R4 + R5
- WAW hazard occurs when instruction J tries to write output before instruction I writes it.
  Eg:
  I: R2 <- R1 + R3
  J: R2 <- R4 + R5

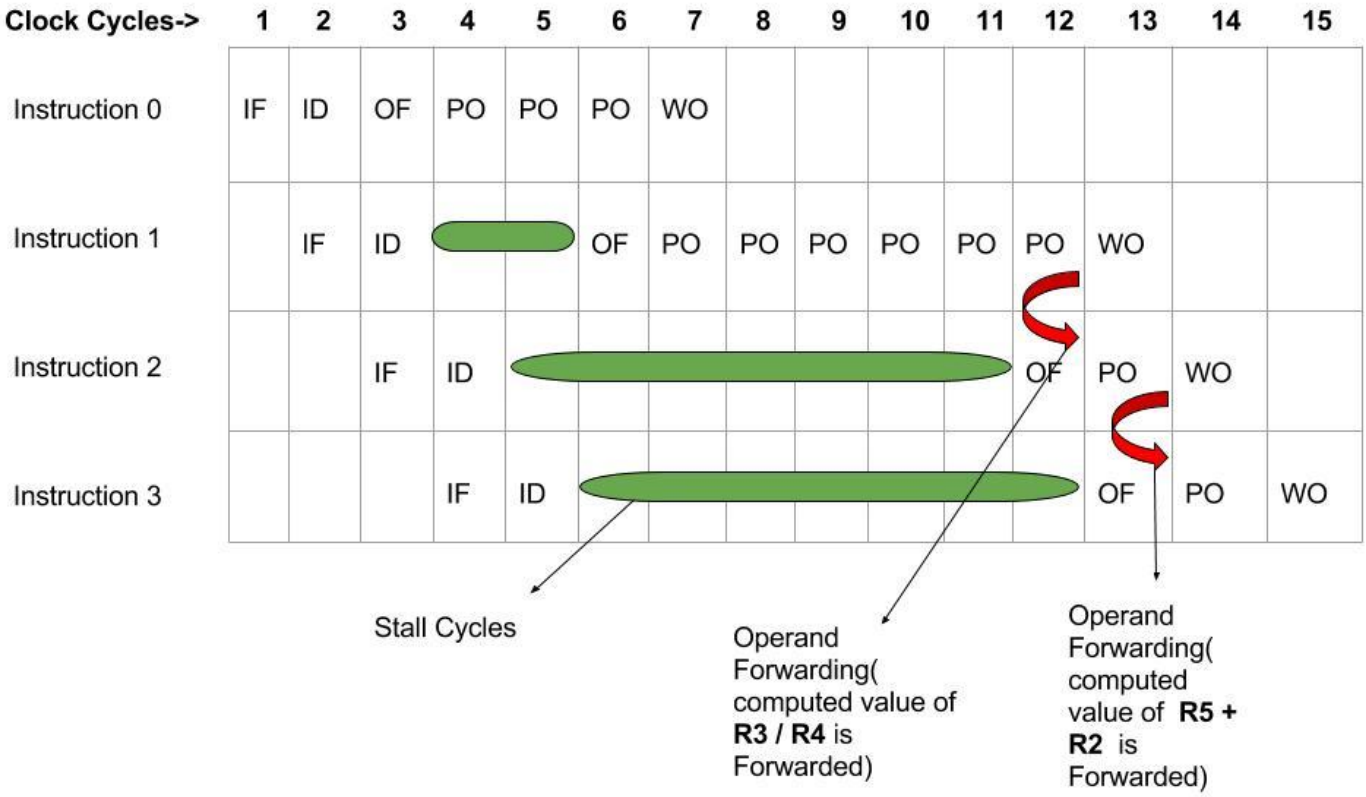
WAR and WAW hazards occur during the out-of-order execution of the instructions.

Q9 b) What is the concept of operand forwarding.

Ans-

Operand Forwarding : In this technique the value of operand is given to the concerned stage of dependent instruction before it is stored.

In the above question, I2 is dependent on I0 and I1, and I3 is dependent on I2.



The above is a space-time diagram representing the pipeline in which the instructions gets executed.

Instruction 0 is a MUL operation which take 3 clock cycles of CPU in the PO stage, and at any other stage it takes only 1 cycle.

Instruction 1 is a DIV operation which take 6 clock cycles of CPU in the PO stage, and at any other stage it takes only 1 cycle.

It can be noticed here that even when the OF stage was free in the 4th clock cycle, then also the instruction 1 was not given to it. This is a design issue. The operands should be fetched only if they are going to get operated or executed in the next cycle, else there is a possibility of data corruption. As PO stage was not free in the next cycle hence OF was delayed and was done for instruction 1 only just before 1 cycle of going to PO stage.

Instruction 2 is an ADD operation which take 1 clock cycles of CPU in all stages. But it is a dependent operation. it needs the operands which are provided by Instruction 0 and 1.

Instruction 2 needs R5 and R2 to add, it gets R2 on time, because till the time Instruction 2 reaches its PO stage R2 would have been stored in memory. Now R5 is also needed, but Instruction 2's PO and Instruction 1's WO are in parallel. That means Instruction 2 can't take the value of R5 before it is stored by Instruction 1. So here comes the concept of Operand Forwarding. Before Instruction 1 store it's result/value which is R5, it can first forward it to instruction 2's Fetch-Execute Buffer, so that Instruction 2 can also use it in parallel to Instruction's WO stage. This will save extra clock cycles required(if Operand forwarding is not used, and R5 need to be taken from memory).

In Instruction 3, same operand forwarding concept is applied for the value of R2 which is computed by Instruction 2.

Hence operand forwarding saved 2 extra clock cycles here. (1 cycle in Instruction 2 and 1 cycle in Instruction 3).

So the total no of cycles are 15, which can be seen from the diagram, each instance of the stage represents 1 clock cycle. So total 15.

Q10 a) Explain the concept of delayed branch with 2-stage pipeline.

Ans-

Q10 b) Explain the concept of branch prediction with the help of neat diagram.

Ans-

Q11 a) What is the need of parallel processing? Explain the classification of parallel architecture.

Ans-

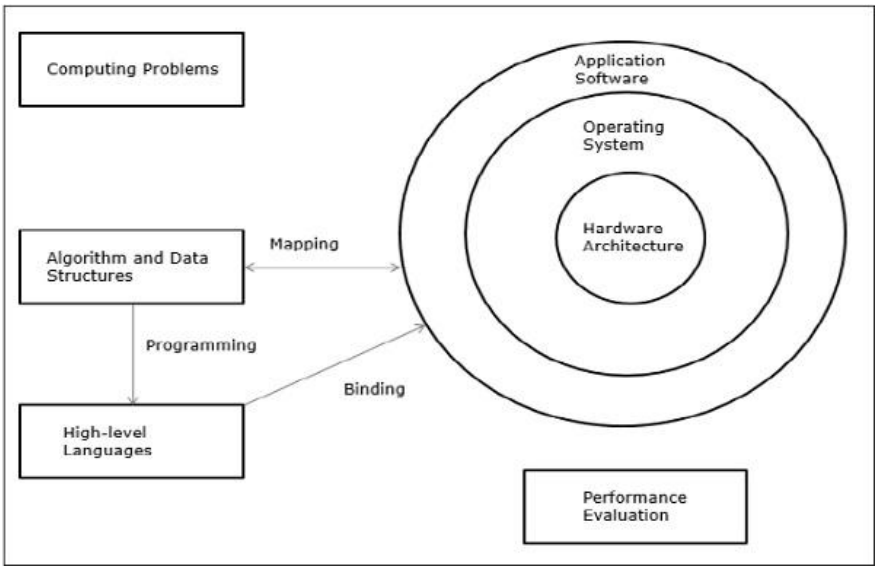
Parallel processing has been developed as an effective technology in modern computers to meet the demand for higher performance, lower cost and accurate results in real-life applications. Concurrent events are common in today's computers due to the practice of multiprogramming, multiprocessing, or multicomputing.

Modern computers have powerful and extensive software packages. To analyze the development of the performance of computers, first we have to understand the basic development of hardware and software.

- **Computer Development Milestones** − There is two major stages of development of computer - **mechanical** or **electromechanical** parts. Modern computers evolved after the

introduction of electronic components. High mobility electrons in electronic computers replaced the operational parts in mechanical computers. For information transmission, electric signal which travels almost at the speed of a light replaced mechanical gears or levers.

- **Elements of Modern computers** − A modern computer system consists of computer hardware, instruction sets, application programs, system software and user interface.



The computing problems are categorized as numerical computing, logical reasoning, and transaction processing. Some complex problems may need the combination of all the three processing modes.

- **Evolution of Computer Architecture** − In last four decades, computer architecture has gone through revolutionary changes. We started with Von Neumann architecture and now we have multicomputers and multiprocessors.
- **Performance of a computer system** − Performance of a computer system depends both on machine capability and program behavior. Machine capability can be improved with better hardware technology, advanced architectural features and efficient resource management. Program behavior is unpredictable as it is dependent on application and run-time conditions

two types of parallel computers −

- Multiprocessors
- Multicomputers

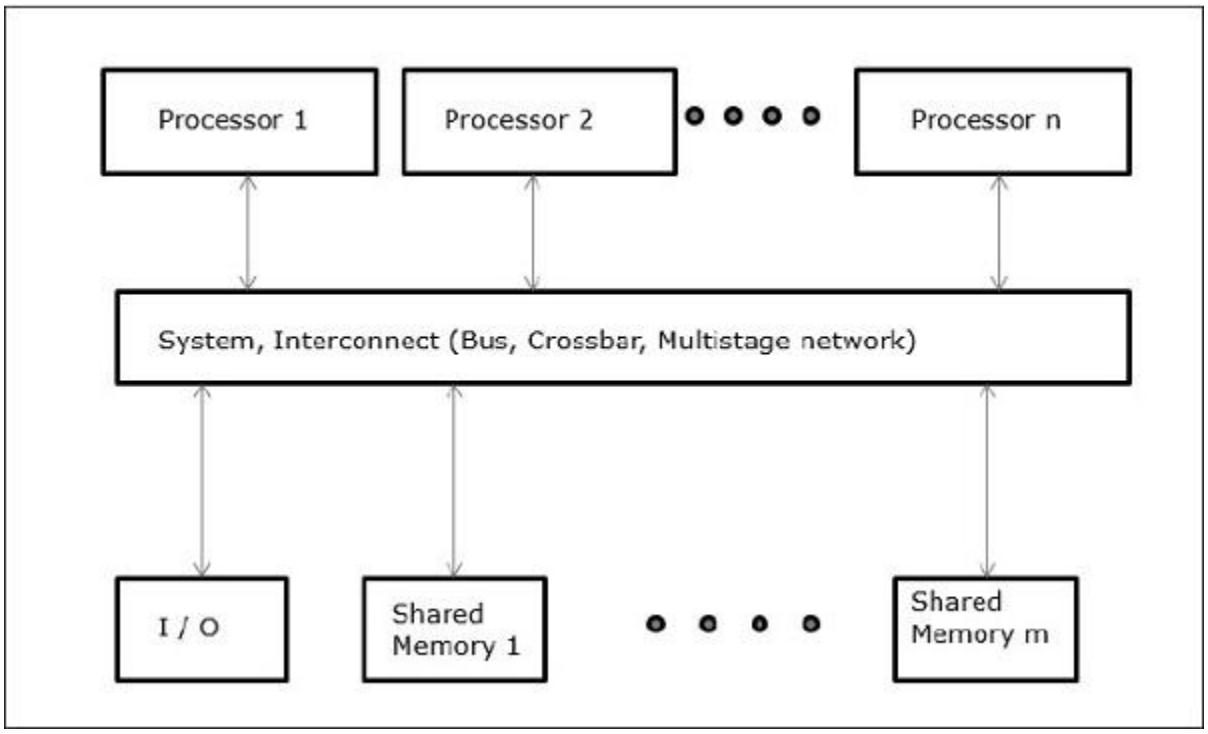
Shared-Memory Multicomputers

Three most common shared memory multiprocessors models are −

Uniform Memory Access (UMA)

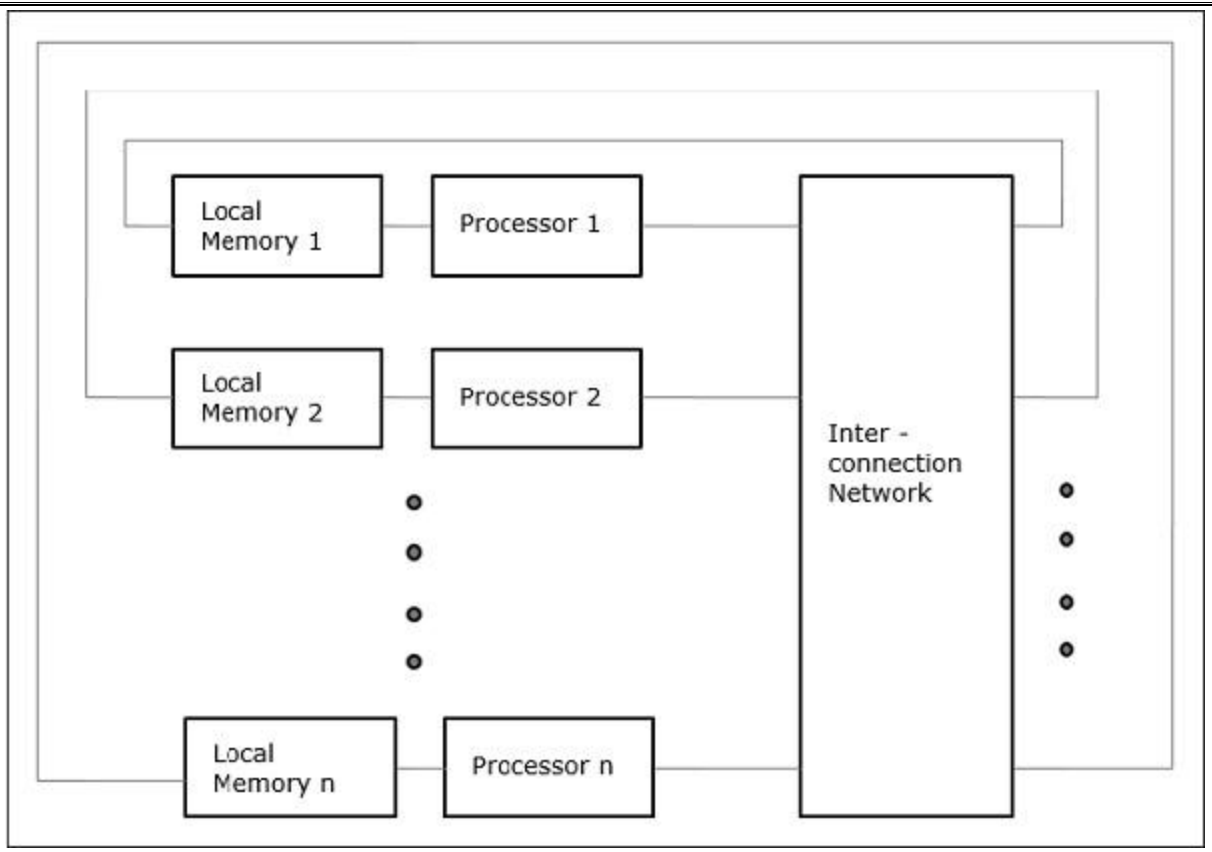
In this model, all the processors share the physical memory uniformly. All the processors have equal access time to all the memory words. Each processor may have a private cache memory. Same rule is followed for peripheral devices.

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



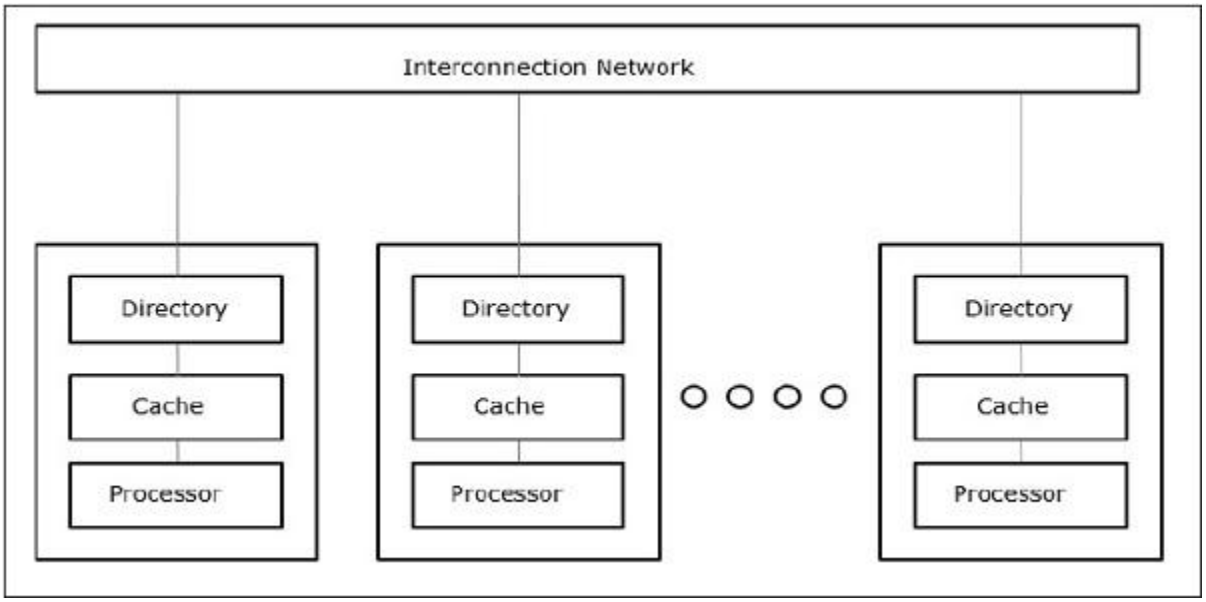
Non-uniform Memory Access (NUMA)

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

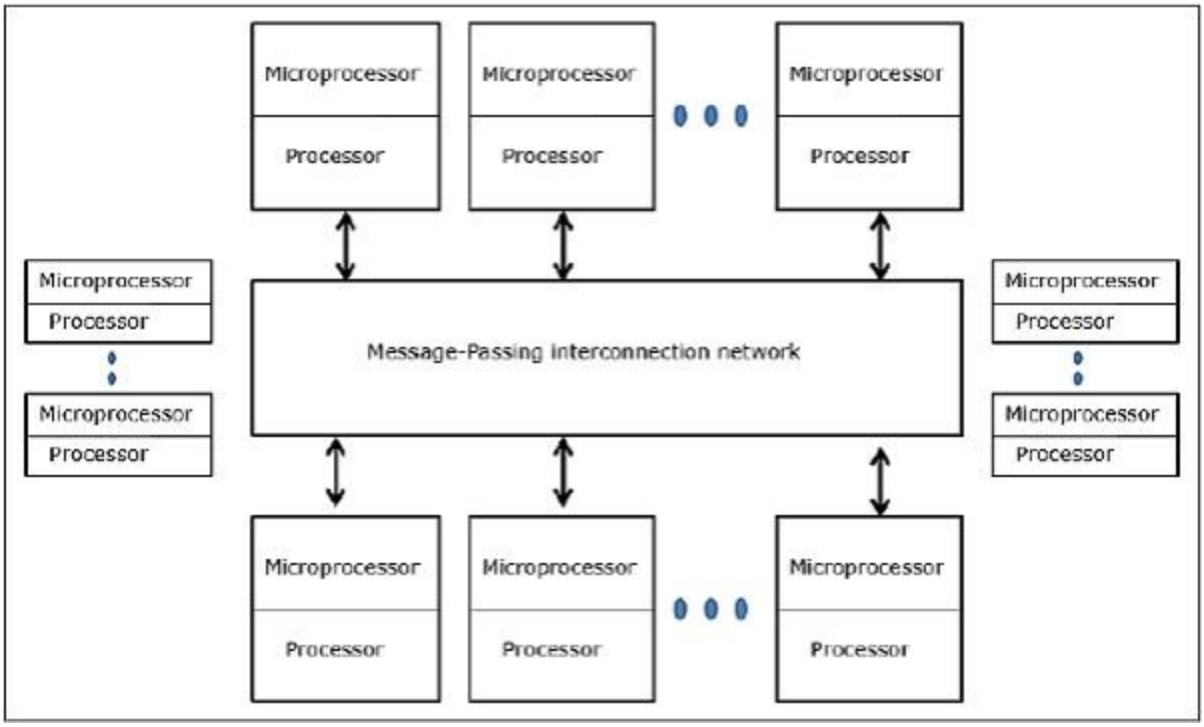


Cache Only Memory Architecture (COMA)

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



- **Distributed - Memory Multicomputers** − A distributed memory multicomputer system consists of multiple computers, known as nodes, inter-connected by message passing network. Each node acts as an autonomous computer having a processor, a local memory and sometimes I/O devices. In this case, all local memories are private and are accessible only to the local processors. This is why, the traditional machines are called **no-remote-memory-access (NORMA)** machines.



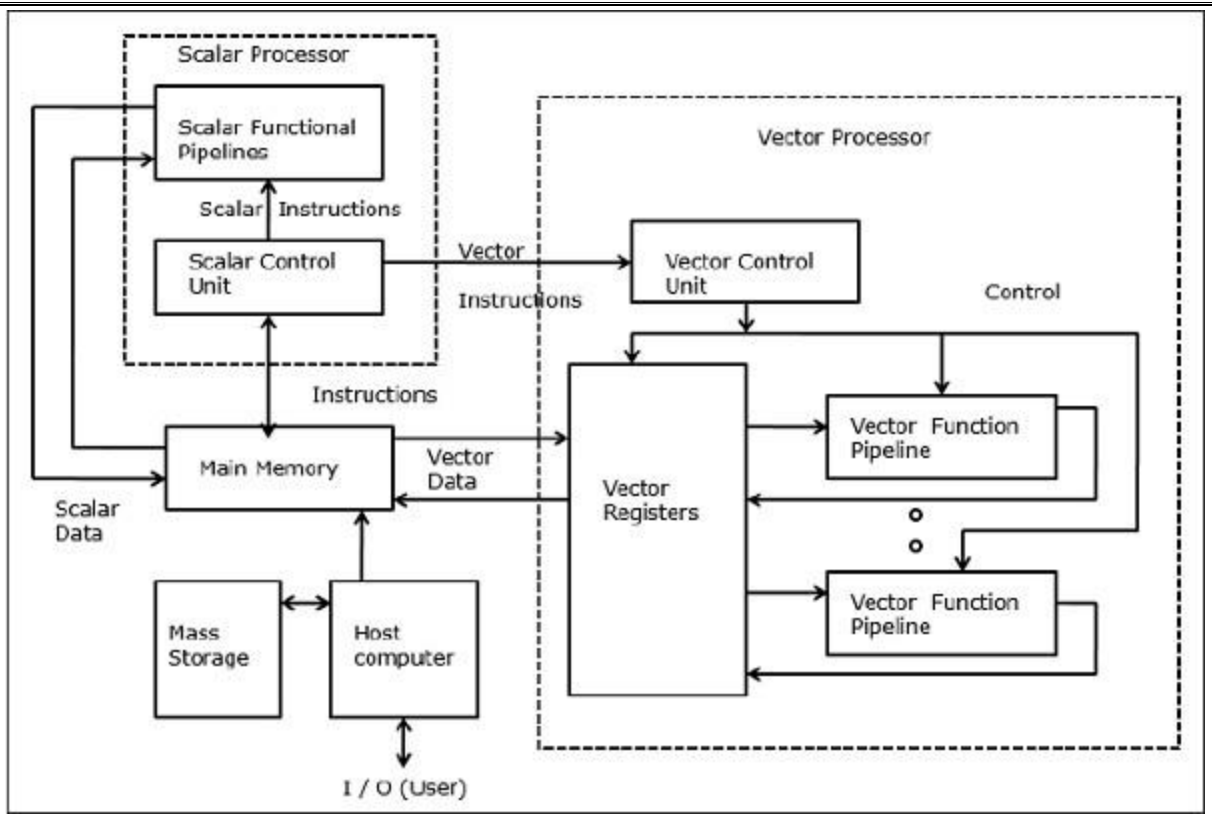
Multivector and SIMD Computers

In this section, we will discuss supercomputers and parallel processors for vector processing and data parallelism.

Vector Supercomputers

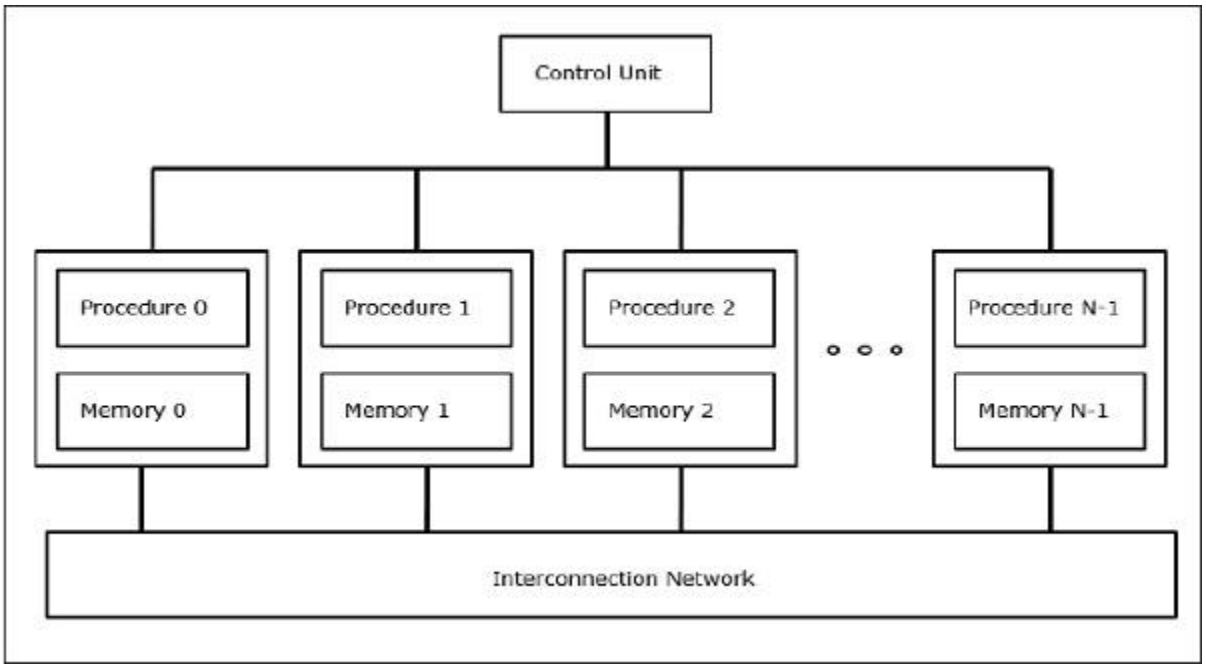
In a vector computer, a vector processor is attached to the scalar processor as an optional feature. The host computer first loads program and data to the main memory. Then the scalar control unit decodes all the instructions. If the decoded instructions are scalar operations or program operations, the scalar processor executes those operations using scalar functional pipelines.

On the other hand, if the decoded instructions are vector operations then the instructions will be sent to vector control unit.



SIMD Supercomputers

In SIMD computers, ‘N’ number of processors are connected to a control unit and all the processors have their individual memory units. All the processors are connected by an interconnection network.



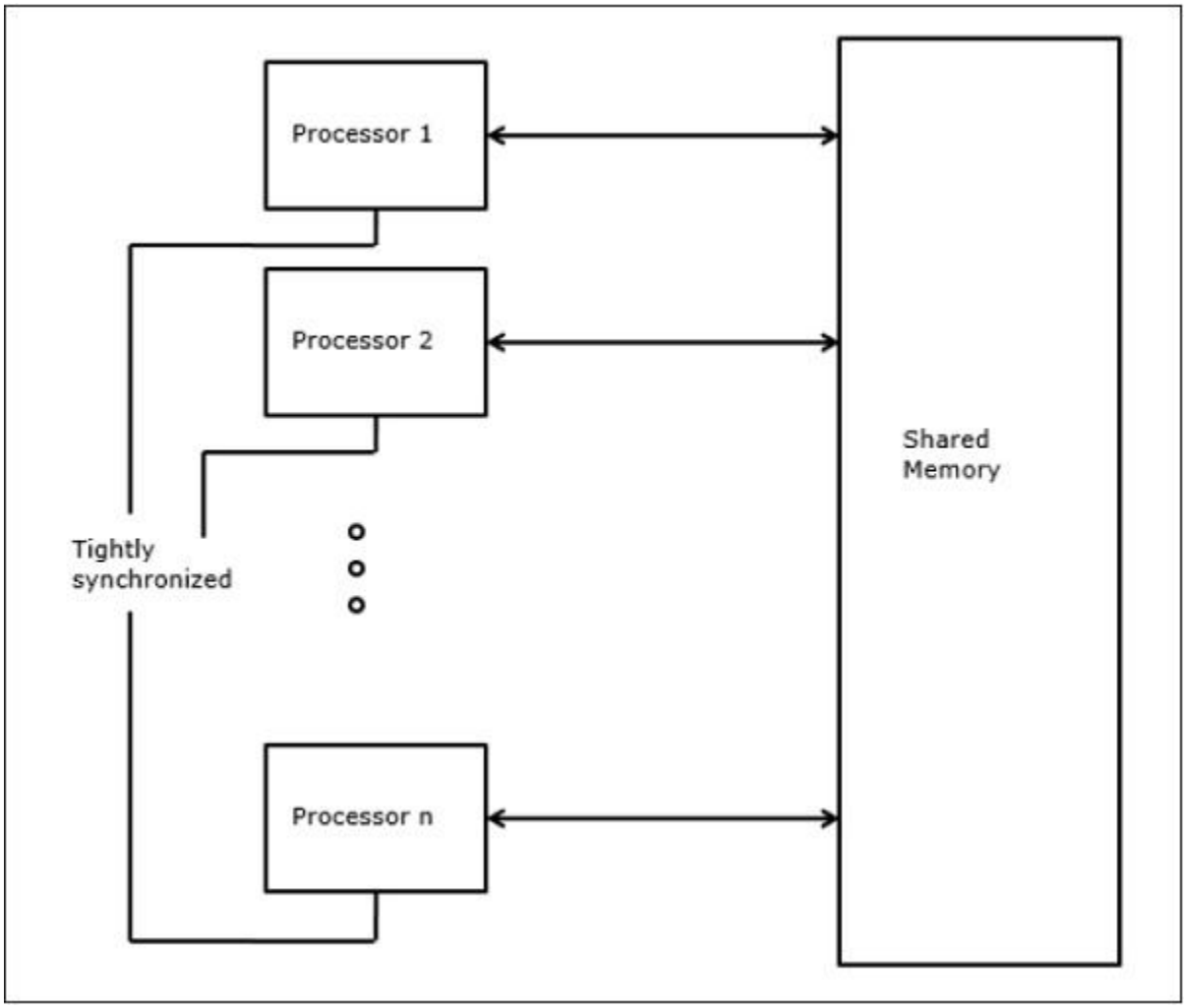
PRAM and VLSI Models

The ideal model gives a suitable framework for developing parallel algorithms without considering the physical constraints or implementation details.

The models can be enforced to obtain theoretical performance bounds on parallel computers or to evaluate VLSI complexity on chip area and operational time before the chip is fabricated.

Parallel Random-Access Machines

Sheperdson and Sturgis (1963) modeled the conventional Uniprocessor computers as random-access-machines (RAM). Fortune and Wyllie (1978) developed a parallel random-access-machine (PRAM) model for modeling an idealized parallel computer with zero memory access overhead and synchronization.



An N-processor PRAM has a shared memory unit. This shared memory can be centralized or distributed among the processors. These processors operate on a synchronized read-memory, write-memory and compute cycle. So, these models specify how concurrent read and write operations are handled.

Following are the possible memory update operations −

- **Exclusive read (ER)** − In this method, in each cycle only one processor is allowed to read from any memory location.
- **Exclusive write (EW)** − In this method, at least one processor is allowed to write into a memory location at a time.

- **Concurrent read (CR)** − It allows multiple processors to read the same information from the same memory location in the same cycle.
- **Concurrent write (CW)** − It allows simultaneous write operations to the same memory location. To avoid write conflict some policies are set up.

Q11 b) What is multicore architecture why there is a need of multicore architecture?

Ans-

12) Write short note on following **any two.**

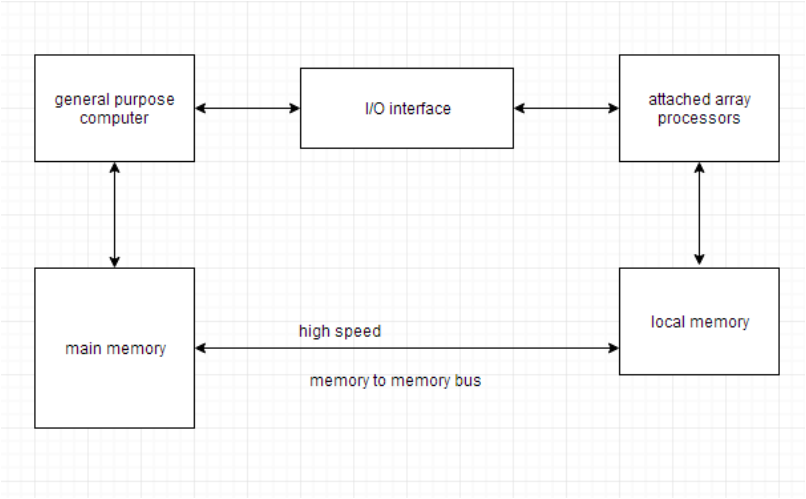
i) Array Processor

ii) Vector processor

iii) Multiprocessor

Array Processor

In computing, a vector **processor** or **array processor** is a central **processing** unit (**CPU**) that implements an instruction set containing instructions that operate on one-dimensional **arrays** of data called vectors, compared to scalar **processors**, whose instructions operate on single data items.

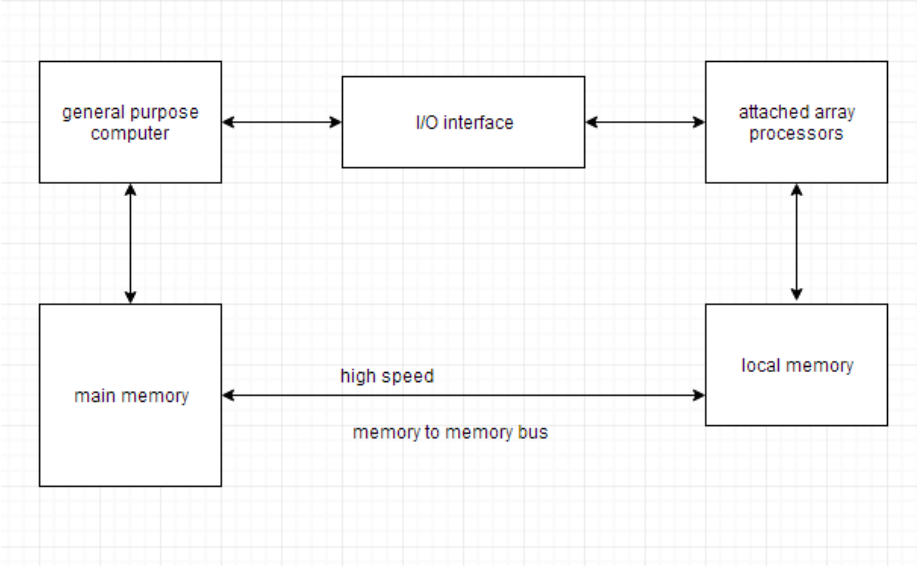


There are basically two types of array processors:

1. Attached Array Processors
2. SIMD Array Processors

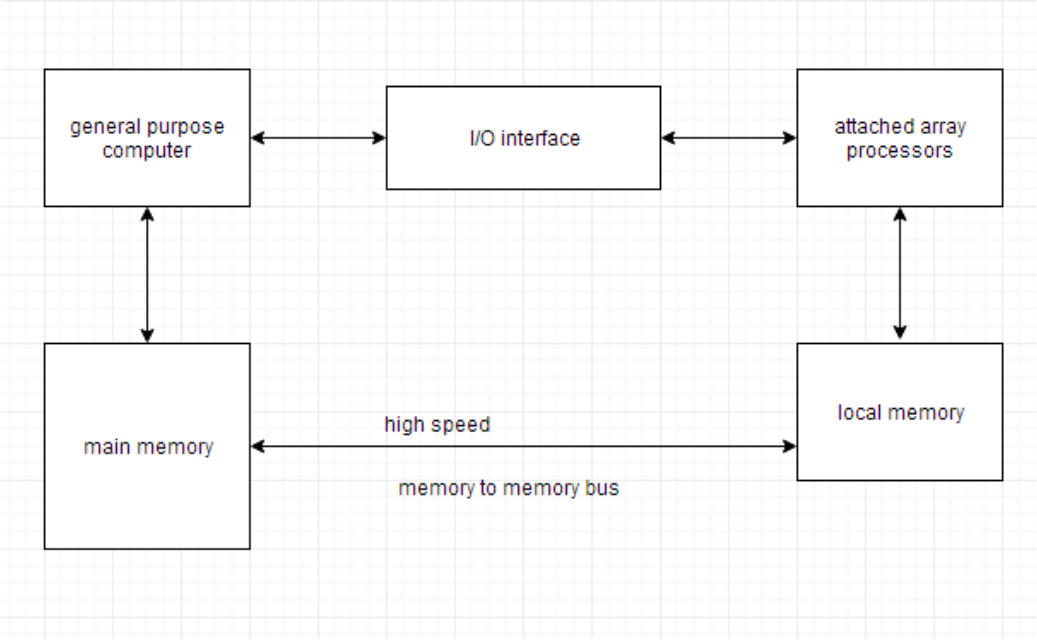
Attached Array Processors

An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks. It achieves high performance by means of parallel processing with multiple functional units.



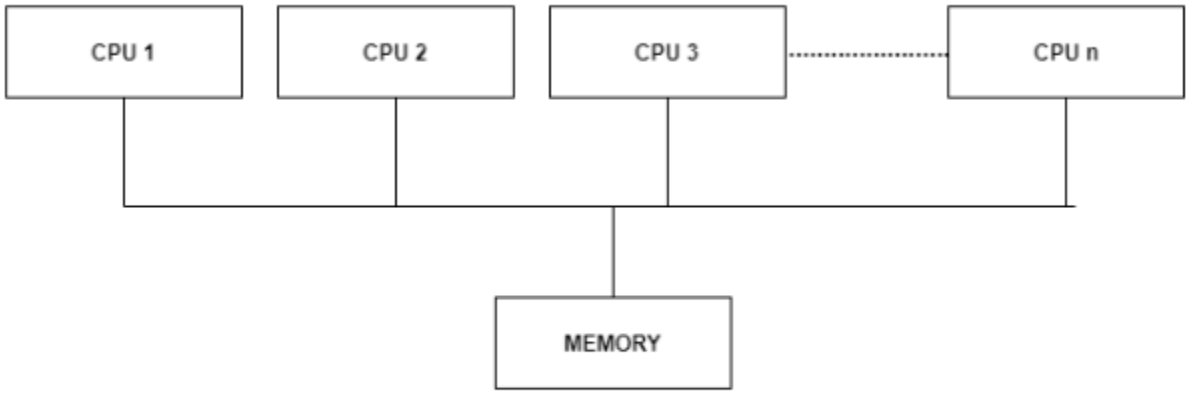
Attached Array Processors

An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks. It achieves high performance by means of parallel processing with multiple functional units.



2) Multiprocessor

Ans- Most computer systems are single processor systems i.e they only have one processor. However, multiprocessor or parallel systems are increasing in importance nowadays. These systems have multiple processors working in parallel that share the computer clock, memory, bus, peripheral devices etc. An image demonstrating the multiprocessor architecture is:



Multiprocessing Architecture

Types of Multiprocessors

There are mainly two types of multiprocessors i.e. symmetric and asymmetric multiprocessors. Details about them are as follows:

Symmetric Multiprocessors

In these types of systems, each processor contains a similar copy of the operating system and they all communicate with each other. All the processors are in a peer to peer relationship i.e. no master - slave relationship exists between them.

An example of the symmetric multiprocessing system is the Encore version of Unix for the Multimax Computer.

Asymmetric Multiprocessors

In asymmetric systems, each processor is given a predefined task. There is a master processor that gives instruction to all the other processors. Asymmetric multiprocessor system contains a master slave relationship.

Asymmetric multiprocessor was the only type of multiprocessor available before symmetric multiprocessors were created. Now also, this is the cheaper option.

Advantages of Multiprocessor Systems

There are multiple advantages to multiprocessor systems. Some of these are:

More reliable Systems

In a multiprocessor system, even if one processor fails, the system will not halt. This ability to continue working despite hardware failure is known as graceful degradation. For example: If there are 5 processors in a multiprocessor system and one of them fails, then also 4 processors are still working. So the system only becomes slower and does not ground to a halt.

Enhanced Throughput

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

More Economic Systems

Multiprocessor systems are cheaper than single processor systems in the long run because they share the data storage, peripheral devices, power supplies etc. If there are multiple processes that share data, it is better to schedule them on multiprocessor systems with shared data than have different computer systems with multiple copies of the data.

Disadvantages of Multiprocessor Systems

There are some disadvantages as well to multiprocessor systems. Some of these are:

Increased Expense

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

Complicated Operating System Required

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