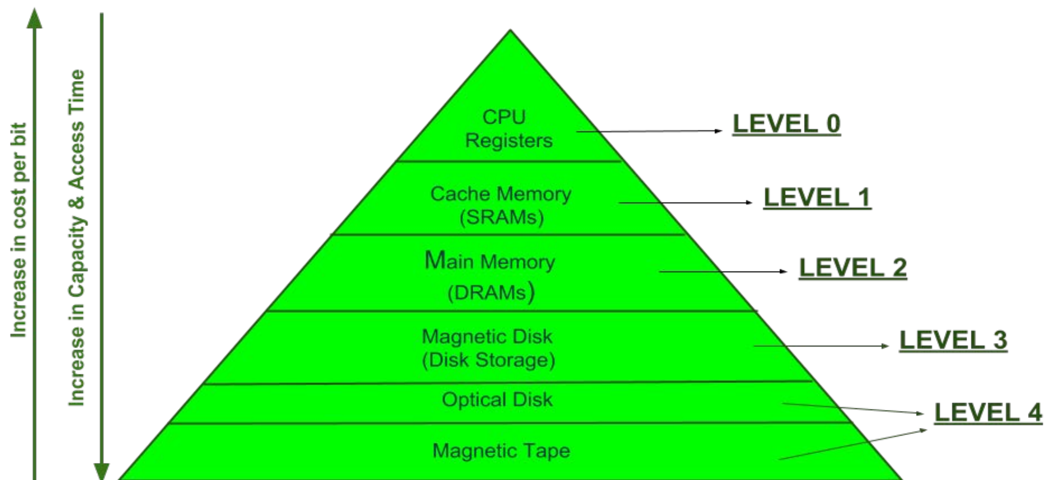
UNIT-IV

Memory Hierarchy Design and its Characteristics

In the Computer System Design, 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 memory hierarchy :



MEMORY HIERARCHY DESIGN

This Memory Hierarchy Design is divided into 2 main types:

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

We can infer the following characteristics of Memory Hierarchy Design from above figure:

1. **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.
2. **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.
3. **Performance:**
   Earlier when the computer system was designed without Memory Hierarchy design, the speed gap increases between the CPU registers and Main Memory due to 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.
4. **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.

RAM and ROM architecture.

1) **Read-only memory, or ROM,** is a form of data storage in computers and other electronic devices that cannot be easily altered or reprogrammed. RAM is referred to as volatile memory and is lost when the power is turned off whereas ROM in non-volatile and the contents are retained even after the power is switched off.

Types of ROM: Semiconductor-Based

Classic mask-programmed ROM chips are integrated circuits that physically encode the data to be stored, and thus it is impossible to change their contents after fabrication. Other types of non-volatile solid-state memory permit some degree of modification:

• **Programmable read-only memory (PROM),** or one-time programmable ROM (OTP), can be written to or programmed via a special device called a PROM programmer. Typically, this device uses high voltages to permanently destroy or create internal links (fuses or antifuses) within the chip. Consequently, a PROM can only be programmed once.

• **Erasable programmable read-only memory (EPROM)** can be erased by exposure to strong ultraviolet light (typically for 10 minutes or longer), then rewritten with a process that again needs higher than usual voltage applied. Repeated exposure to UV light will eventually wear out an EPROM, but the endurance of most EPROM chips exceeds 1000 cycles of erasing and reprogramming. EPROM chip packages can often be identified by the prominent quartz "window" which allows UV light to enter. After programming, the window is typically covered with a label to prevent accidental erasure. Some EPROM chips are factory-erased before they are packaged, and include no window; these are effectively PROM.

• **Electrically erasable programmable read-only memory (EEPROM)** is based on a similar semiconductor structure to EPROM, but allows its entire contents (or selected banks) to be electrically erased, then rewritten electrically, so that they need not be removed from the computer (whether general-purpose or an embedded computer in a camera, MP3 player, etc.). Writing or flashing an EEPROM is much slower (milliseconds per bit) than reading from a ROM or writing to a RAM (nanoseconds in both cases).

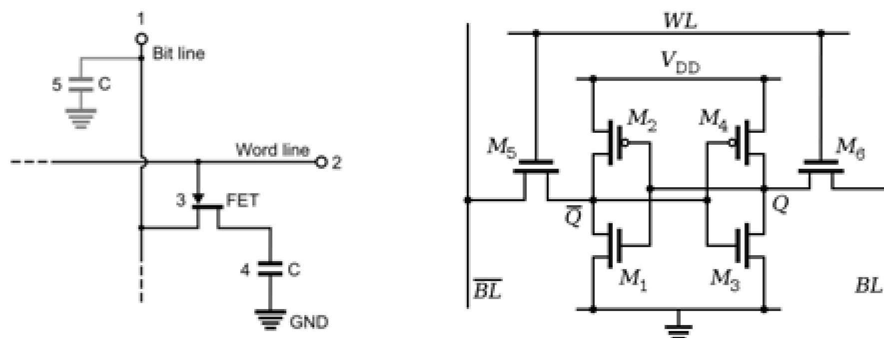
• **Electrically alterable read-only memory (EAROM)** is a type of EEPROM that can be modified one bit at a time. Writing is a very slow process and again needs higher voltage (usually around 12 V) than is used for read access. EAROMs are intended for applications that require infrequent and only partial rewriting. EAROM may be used as non-volatile storage for critical system setup information; in many applications, EAROM has been supplanted by CMOS RAM supplied by mains power and backed-up with a lithium battery.

• **Flash memory (or simply flash)** is a modern type of EEPROM invented in 1984. Flash memory can be erased and rewritten faster than ordinary EEPROM, and newer designs feature very high endurance (exceeding 1,000,000 cycles). Modern NAND flash makes efficient use of silicon chip area, resulting in individual ICs with a capacity as high as 32 GB as of 2007; this feature, along with its endurance and physical durability, has allowed NAND flash to replace magnetic in some applications (such as USB flash drives). Flash memory is sometimes called flash ROM or flash EEPROM when used as a replacement for older ROM types, but not in applications that take advantage of its ability to be modified quickly and frequently.

2) **Random-access memory, or RAM,** is a form of data storage that can be accessed randomly at any time, in any order and from any physical location in contrast to other storage devices, such as hard drives, where the physical location

of the data determines the time taken to retrieve it. RAM is measured in megabytes and the speed is measured in nanoseconds and RAM chips can read data faster than ROM.

Types of RAM: The two widely used forms of modern RAM are **static RAM** (SRAM) and **dynamic RAM** (DRAM). In SRAM, a bit of data is stored using the state of a six transistor memory cell. This form of RAM is more expensive to produce, but is generally faster and requires less dynamic power than DRAM. In modern computers, SRAM is often used as cache memory for the CPU. DRAM stores a bit of data using a transistor and capacitor pair, which together comprise a DRAM cell. The capacitor holds a high or low charge (1 or 0, respectively), and the transistor acts as a switch that lets the control circuitry on the chip read the capacitor's state of charge or change it. As this form of memory is less expensive to produce than static RAM, it is the predominant form of computer memory used in modern computers. The figure below shows DRAM & SRAM resp.



Both static and dynamic RAM are considered volatile, as their state is lost or reset when power is removed from the system. By contrast, read-only memory (ROM) stores data by permanently enabling or disabling selected transistors, such that the memory cannot be altered. Writeable variants of ROM (such as EEPROM and flash memory) share properties of both ROM and RAM, enabling data to persist without power and to be updated without requiring special equipment. These persistent forms of semiconductor ROM include USB flash drives, memory cards for cameras and portable devices, and solid-state drives. ECC memory (which can be either SRAM or DRAM) includes special circuitry to detect and/or correct random faults (memory errors) in the stored data, using parity bits or error correction codes.

In general, the term RAM refers solely to solid-state memory devices (either DRAM or SRAM), and more specifically the main memory in most computers. In optical storage, the term DVD-RAM is somewhat of a misnomer since, unlike CD-RW or DVD-RW it does not need to be erased before reuse. Nevertheless, a DVD-RAM behaves much like a hard disc drive if somewhat slower.

Difference between Static Ram And Dynamic Ram

Static RAM	Dynamic RAM
➢ SRAM uses transistor to store a single bit of data	➢ DRAM uses a separate capacitor to store each bit of data
➢ SRAM does not need periodic refreshment to maintain data	➢ DRAM needs periodic refreshment to maintain the charge in the capacitors for data
➢ SRAM's structure is complex than DRAM	➢ DRAM's structure is simplex than SRAM
➢ SRAM are expensive as compared to DRAM	➢ DRAM's are less expensive as compared to SRAM
➢ SRAM are faster than DRAM	➢ DRAM's are slower than SRAM
➢ SRAM are used in Cache memory	➢ DRAM are used in Main memory

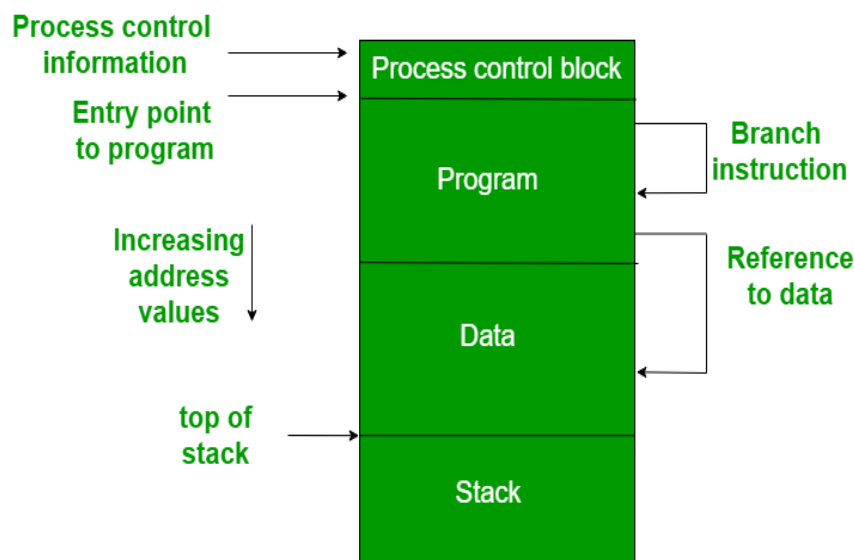
Requirements of Memory Management System

Memory management keeps track of the status of each memory location, whether it is allocated or free. It allocates the memory dynamically to the programs at their request and frees it for reuse when it is no longer needed. Memory management meant to satisfy some requirements that we should keep in mind.

These Requirements of memory management are:

1. **Relocation –** The available memory is generally shared among a number of processes in a multiprogramming system, so it is not possible to know in advance which other programs will be resident in main memory at the time of execution of his program. Swapping the active processes in and out of the main memory enables the operating system to have a larger pool of ready-to-execute process.

   When a program gets swapped out to a disk memory, then it is not always possible that when it is swapped back into main memory then it occupies the previous memory location, since the location may still be occupied by another process. We may need to **relocate** the process to a different area of memory. Thus there is a possibility that program may be moved in main memory due to swapping.

   The figure depicts a process image. The process image is occupying a continuous region of main memory. The operating system will need to know many things including the location of process control information, the execution stack, and the code entry. Within a program, there are memory references in various instructions and these are called logical addresses.

   After loading of the program into main memory, the processor and the operating system must be able to translate logical addresses into physical addresses. Branch instructions contain the address of the next instruction to be executed. Data reference instructions contain the address of byte or word of data referenced.

2. **Protection –** There is always a danger when we have multiple programs at the same time as one program may write to the address space of another program. So every process must be protected against unwanted interference when other process tries to write in a process whether accidental or incidental. Between relocation and protection requirement a trade-off occurs as the satisfaction of

relocation requirement increases the difficulty of satisfying the protection requirement.

Prediction of the location of a program in main memory is not possible, that's why it is impossible to check the absolute address at compile time to assure protection. Most of the programming language allows the dynamic calculation of address at run time. The memory protection requirement must be satisfied by the processor rather than the operating system because the operating system can hardly control a process when it occupies the processor. Thus it is possible to check the validity of memory references.

3. **Sharing –** A protection mechanism must have to allow several processes to access the same portion of main memory. Allowing each processes access to the same copy of the program rather than have their own separate copy has an advantage.

   For example, multiple processes may use the same system file and it is natural to load one copy of the file in main memory and let it shared by those processes. It is the task of Memory management to allow controlled access to the shared areas of memory without compromising the protection. Mechanisms are used to support relocation supported sharing capabilities.

4. **Logical organization –** Main memory is organized as linear or it can be a one-dimensional address space which consists of a sequence of bytes or words. Most of the programs can be organized into modules, some of those are unmodifiable (read-only, execute only) and some of those contain data that can be modified. To effectively deal with a user program, the operating system and computer hardware must support a basic module to provide the required protection and sharing. It has the following advantages:

   - Modules are written and compiled independently and all the references from one module to another module are resolved by `the system at run time.
   - Different modules are provided with different degrees of protection.
   - There are mechanisms by which modules can be shared among processes. Sharing can be provided on a module level that lets the user specify the sharing that is desired.

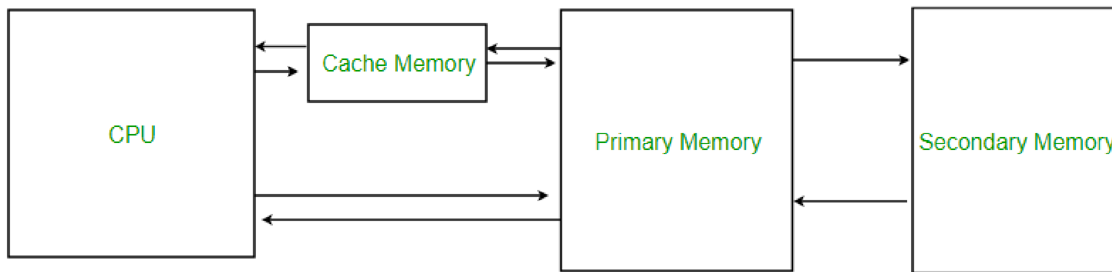
5. **Physical organization –** The structure of computer memory has two levels referred to as main memory and secondary memory. Main memory is relatively very fast and costly as compared to the secondary memory. Main memory is volatile. Thus secondary memory is provided for storage of data on a long-term basis while the main memory holds currently used programs. The major system concern between main memory and secondary memory is the flow of information and it is impractical for programmers to understand this for two reasons:

   - The programmer may engage in a practice known as overlaying when the main memory available for a program and its data may be insufficient. It allows different modules to be assigned to the same region of memory. One disadvantage is that it is time-consuming for the programmer.
   - In a multiprogramming environment, the programmer does not know how much space will be available at the time of coding and where that space will be located inside the memory.

Cache Memory in Computer Organization

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

Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.



Levels of memory:

- **Level 1 or Register –** It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.
- **Level 2 or Cache memory –** It is the fastest memory which has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory –** It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.
- **Level 4 or Secondary Memory –** It is external memory which is not as fast as main memory but data stays permanently in this memory.

Cache Performance:

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache
- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

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

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

We can improve Cache performance using higher cache block size, higher associativity, reduce miss rate, reduce miss penalty, and reduce Reduce the time to hit in the cache.

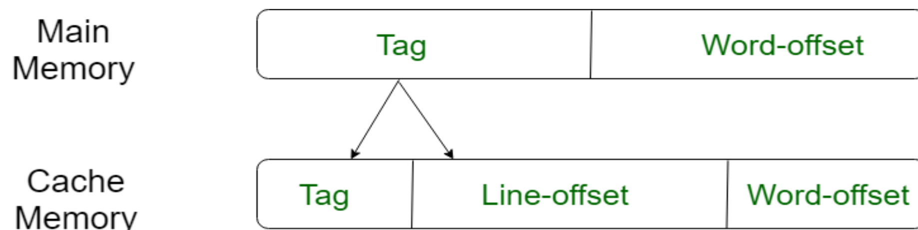
Cache Mapping:

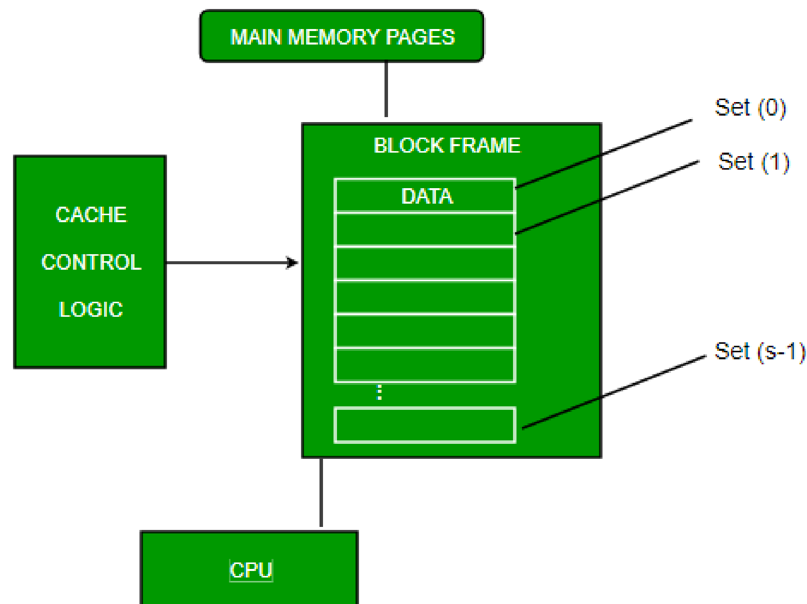
There are three different types of mapping used for the purpose of cache memory which are as follows: Direct mapping, Associative mapping, and Set-Associative mapping. These are explained below.

1. **Direct Mapping –** The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. or In Direct mapping, assigne each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and a tag field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.

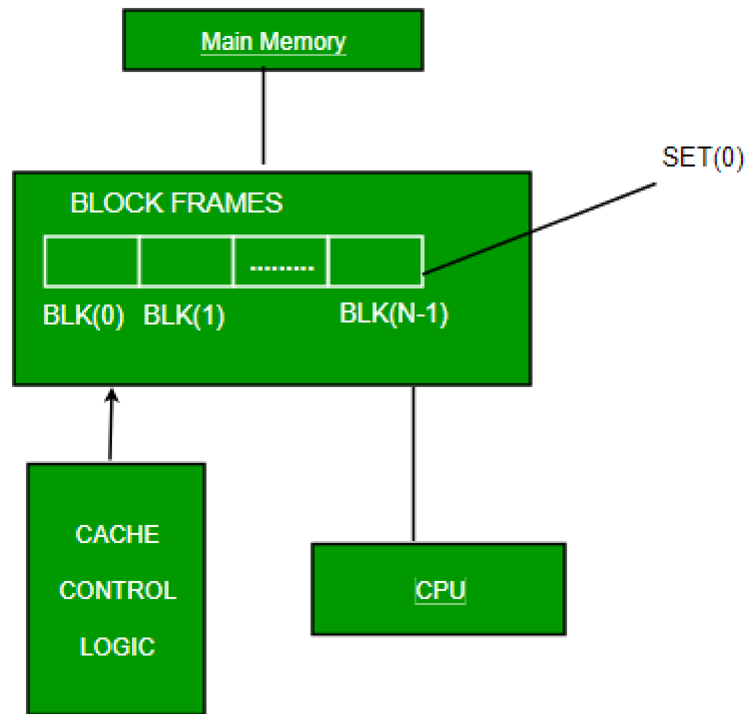
   i = j modulo m
   where
   i=cache line number
   j= main memory block number
   m=number of lines in the cache

   For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant w bits identify a unique word or byte within a block of main memory. In most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2^s blocks of main memory. The cache logic interprets these s bits as a tag of s-r bits (most significant portion) and a line field of r bits. This latter field identifies one of the $m=2^r$ lines of the cache.





2. **Associative Mapping –** In this type of mapping, the associative memory is used to store content and addresses of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits. This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form.



3. **Set-associative Mapping –** This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a ***set***. Then a block in memory can map to any one of the lines of a specific set..Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques.
   In this case, the cache consists of a number of sets, each of which consists of a number of lines. The relationships are

   m = v * k

   i= j mod v

   where

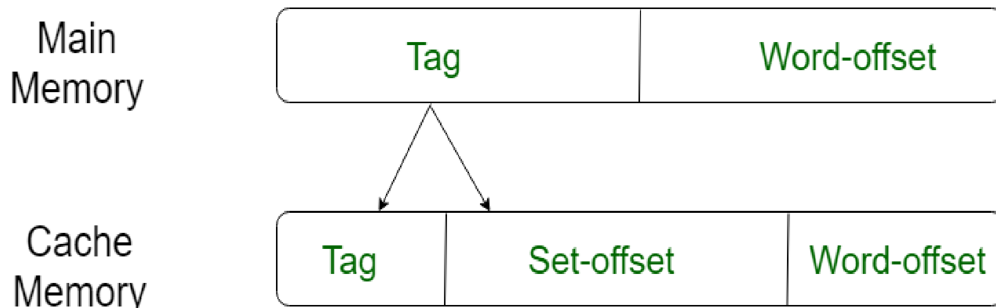
   i=cache set number

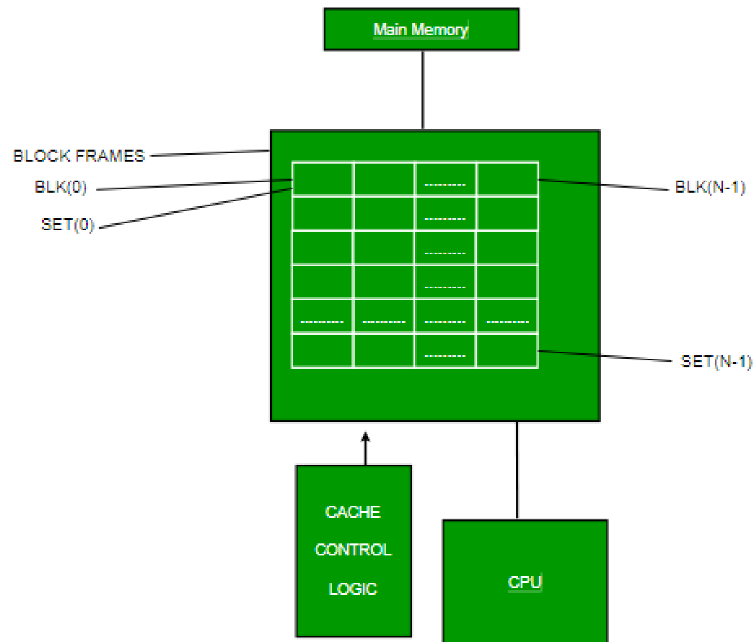
   j=main memory block number

   v=number of sets

   m=number of lines in the cache number of sets

   k=number of lines in each set





Application of Cache Memory –

1. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.
2. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

Types of Cache –

- **Primary Cache –** A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.
- **Secondary Cache –** Secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.

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

Types of Locality of reference

Spatial Locality of reference This says that there is a chance that element will be present in the close proximity to the reference point and next time if again searched then more close proximity to the point of reference.

Temporal Locality of reference In this Least recently used algorithm will be used. Whenever there is page fault occurs within a word will not only load word in main memory but complete page fault will be loaded because spatial locality of reference rule says that if you are referring any word next word will be referred in its register that's why we load complete page table so the complete block will be loaded.

UNIT-V

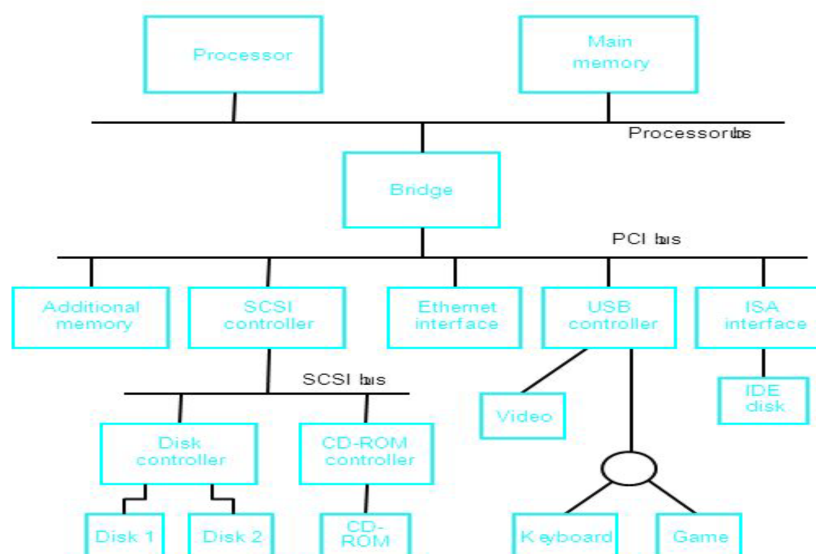
Standard I/O interfaces

The processor bus is the bus defied by the signals on the processor chip itself. Devices that require a very high-speed connection to the processor, such as the main memory, may be connected directly to this bus. For electrical reasons, only a few devices can be connected in this manner. The motherboard usually provides another bus that can support more devices. The two buses are interconnected by a circuit, which we will call a bridge, that translates the signals and protocols of one bus into those of the other. Devices connected to the expansion bus appear to the processor as if they were connected directly to the processor's own bus. The only difference is that the bridge circuit introduces a small delay in data transfers between the processor and those devices.

It is not possible to define a uniform standard for the processor bus. The structure of this bus is closely tied to the architecture of the processor. It is also dependent on the electrical characteristics of the processor chip, such as its clock speed. The expansion bus is not subject to these limitations, and therefore it can use a standardized signaling scheme. A number of standards have been developed. Some have evolved by default, when a particular design became commercially successful. For example, IBM developed a bus they called ISA (Industry Standard Architecture) for their personal computer known at the time as PC AT.

Some standards have been developed through industrial cooperative efforts, even among competing companies driven by their common self-interest in having compatible products. In some cases, organizations such as the IEEE (Institute of Electrical and Electronics Engineers), ANSI (American National Standards Institute), or international bodies such as ISO (International Standards Organization) have blessed these standards and given them an official status.

A given computer may use more than one bus standards. A typical Pentium computer has both a PCI bus and an ISA bus, thus providing the user with a wide range of devices to choose from.



Peripheral Component Interconnect (PCI) Bus:-

The PCI bus is a good example of a system bus that grew out of the need for standardization. It supports the functions found on a processor bus bit in a standardized format that is independent of any particular processor. Devices connected to the PCI bus appear to the processor as if they were connected directly to the processor bus. They are assigned addresses in the memory address space of the processor.

The PCI follows a sequence of bus standards that were used primarily in IBM PCs. Early PCs used the 8-bit XT bus, whose signals closely mimicked those of Intel's 80x86 processors. Later, the 16-bit bus used on the PC At computers became known as the ISA bus. Its extended 32-bit version is known as the EISA bus. Other buses developed in the eighties with similar capabilities are the Microchannel used in IBM PCs and the NuBus used in Macintosh computers.

The PCI was developed as a low-cost bus that is truly processor independent. Its design anticipated a rapidly growing demand for bus bandwidth to support high-speed disks and graphic and video devices, as well as the specialized needs of multiprocessor systems. As a result, the PCI is still popular as an industry standard almost a decade after it was first introduced in 1992.

An important feature that the PCI pioneered is a plug-and-play capability for connecting I/O devices. To connect a new device, the user simply connects the device interface board to the bus. The software takes care of the rest.

Data Transfer:-

In today's computers, most memory transfers involve a burst of data rather than just one word. The reason is that modern processors include a cache memory. Data are transferred between the cache and the main memory in burst of several words each. The words involved in such a transfer are stored at successive memory locations. When the processor (actually the cache controller) specifies an address and requests a read operation from the main memory, the memory responds by sending a sequence of data words starting at that address. Similarly, during a write operation, the processor sends a memory address followed by a sequence of data words, to be written in successive memory locations starting at the address. The PCI is designed primarily to support this mode of operation. A read or write operation involving a single word is simply treated as a burst of length one.

The bus supports three independent address spaces: memory, I/O, and configuration. The first two are self-explanatory. The I/O address space is intended for use with processors, such as Pentium, that have a separate I/O address space. However, as noted , the system designer may choose to use memory-mapped I/O even when a separate I/O address space is available. In fact, this is the approach recommended by the PCI its plug-and-play capability. A 4-bit command that accompanies the address identifies which of the three spaces is being used in a given data transfer operation.

The signaling convention on the PCI bus is similar to the one used, we assumed that the master maintains the address information on the bus until data

transfer is completed. But, this is not necessary. The address is needed only long enough for the slave to be selected. The slave can store the address in its internal buffer. Thus, the address is needed on the bus for one clock cycle only, freeing the address lines to be used for sending data in subsequent clock cycles. The result is a significant cost reduction because the number of wires on a bus is an important cost factor. This approach in used in the PCI bus.

At any given time, one device is the bus master. It has the right to initiate data transfers by issuing read and write commands. A master is called an initiator in PCI terminology. This is either a processor or a DMA controller. The addressed device that responds to read and write commands is called a target.

Device Configuration:-

When an I/O device is connected to a computer, several actions are needed to configure both the device and the software that communicates with it.

The PCI simplifies this process by incorporating in each I/O device interface a small configuration ROM memory that stores information about that device. The configuration ROMs of all devices is accessible in the configuration address space. The PCI initialization software reads these ROMs whenever the system is powered up or reset. In each case, it determines whether the device is a printer, a keyboard, an Ethernet interface, or a disk controller. It can further learn bout various device options and characteristics.

Devices are assigned addresses during the initialization process. This means that during the bus configuration operation, devices cannot be accessed based on their address, as they have not yet been assigned one. Hence, the configuration address space uses a different mechanism. Each device has an input signal called Initialization Device Select, IDSEL#.

The PCI bus has gained great popularity in the PC word. It is also used in many other computers, such as SUNs, to benefit from the wide range of I/O devices for which a PCI interface is available. In the case of some processors, such as the Compaq Alpha, the PCI-processor bridge circuit is built on the processor chip itself, further simplifying system design and packaging.

SCSI Bus:-

The acronym SCSI stands for Small Computer System Interface. It refers to a standard bus defined by the American National Standards Institute (ANSI) under the designation X3.131 . In the original specifications of the standard, devices such as disks are connected to a computer via a 50-wire cable, which can be up to 25 meters in length and can transfer data at rates up to 5 megabytes/s.

The SCSI bus standard has undergone many revisions, and its data transfer capability has increased very rapidly, almost doubling every two years. SCSI-2 and SCSI-3 have been defined, and each has several options. A SCSI bus may have eight data lines, in which case it is called a narrow bus and transfers data one byte at a time. Alternatively, a wide SCSI bus has 16 data lines and transfers data 16 bits at a time. There are also several options for the electrical signaling scheme used.

Devices connected to the SCSI bus are not part of the address space of the processor in the same way as devices connected to the processor bus. The SCSI bus is connected to the processor bus through a SCSI controller. This controller uses DMA to transfer data packets from the main memory to the device, or vice versa. A packet may contain a block of data, commands from the processor to the device, or status information about the device.

To illustrate the operation of the SCSI bus, let us consider how it may be used with a disk drive. Communication with a disk drive differs substantially from communication with the main memory.

A controller connected to a SCSI bus is one of two types – an initiator or a target. An initiator has the ability to select a particular target and to send commands specifying the operations to be performed. Clearly, the controller on the processor side, such as the SCSI controller, must be able to operate as an initiator. The disk controller operates as a target. It carries out the commands it receives from the initiator. The initiator establishes a logical connection with the intended target. Once this connection has been established, it can be suspended and restored as needed to transfer commands and bursts of data. While a particular connection is suspended, other device can use the bus to transfer information. This ability to overlap data transfer requests is one of the key features of the SCSI bus that leads to its high performance.

Data transfers on the SCSI bus are always controlled by the target controller. To send a command to a target, an initiator requests control of the bus and, after winning arbitration, selects the controller it wants to communicate with and hands control of the bus over to it. Then the controller starts a data transfer operation to receive a command from the initiator.

The processor sends a command to the SCSI controller, which causes the following sequence of event to take place:

1. The SCSI controller, acting as an initiator, contends for control of the bus.
2. When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
3. The target starts an output operation (from initiator to target); in response to this, the initiator sends a command specifying the required read operation.
4. The target, realizing that it first needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases the bus.
5. The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation. Then, it reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data to the initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.
6. The target transfers the contents of the data buffer to the initiator and then suspends the connection again. Data are transferred either 8 or 16 bits in parallel, depending on the width of the bus.

7. The target controller sends a command to the disk drive to perform another seek operation. Then, it transfers the contents of the second disk sector to the initiator as before. At the end of this transfer, the logical connection between the two controllers is terminated.
8. As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
9. The SCSI controller sends as interrupt to the processor to inform it that the requested operation has been completed.

This scenario show that the messages exchanged over the SCSI bus are at a higher level than those exchanged over the processor bus. In this context, a "higher level" means that the messages refer to operations that may require several steps to complete, depending on the device. Neither the processor nor the SCSI controller need be aware of the details of operation of the particular device involved in a data transfer. In the preceding example, the processor need not be involved in the disk seek operation.

Direct Memory Access:

The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

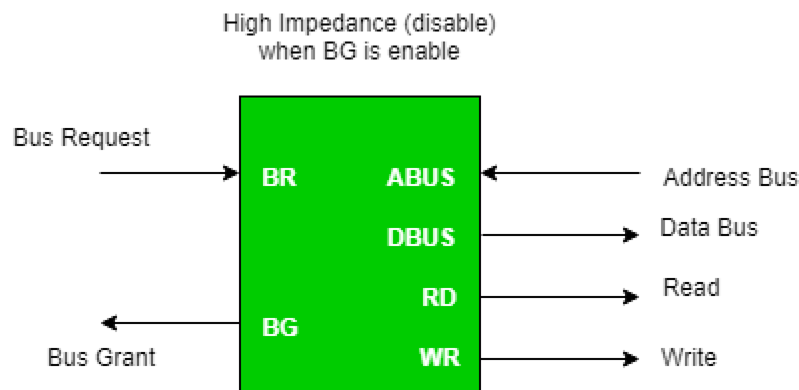


Figure - CPU Bus Signals for DMA Transfer

Bus Request : It is used by the DMA controller to request the CPU to relinquish the control of the buses.

Bus Grant : It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

Types of DMA transfer using DMA controller:

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 So, total time taken to transfer the N bytes = Bus grant request time + (N) * (memory transfer rate) + Bus release control time.

Where,

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

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

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

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

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.

Before moving on transfer next byte of data, device performs step 1 again so that bus isn’t tied up and the transfer won’t depend upon the transfer rate of device. So, for 1 byte of transfer of data, time taken by using cycle stealing mode (T). = time required for bus grant + 1 bus cycle to transfer data + time required to release the bus, it will be N x T

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.

Where,

X μsec =data transfer time or preparation time

(words/block)

Y μsec =memory cycle time or cycle time or transfer

time (words/block)

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

% CPU busy=(X/Y)*100

Interleaved 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.

Computer Organization and Architecture | Pipelining | Set 1 (Execution, Stages and Throughput)

1. To improve the performance of a CPU we have two options: Improve the hardware by introducing faster circuits.
2. Arrange the hardware such that more than one operation can be performed at the same time.

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

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

Let us see a real life example that works on the concept of pipelined operation. Consider a water bottle packaging plant. Let there be 3 stages that a bottle should pass through, Inserting the bottle(**I**), Filling water in the bottle(**F**), and Sealing the bottle(**S**). Let us consider these stages as stage 1, stage 2 and stage 3 respectively. Let each stage take 1 minute to complete its operation. Now, in a non pipelined operation, a bottle is first inserted in the plant, after 1 minute it is moved to stage 2 where water is filled. Now, in stage 1 nothing is happening. Similarly, when the bottle moves to stage 3, both stage 1 and stage 2 are idle. But in pipelined operation, when the bottle is in stage 2, another bottle can be loaded at stage 1. Similarly, when the bottle is in stage 3, there can be one bottle each in stage 1 and stage 2. So, after each minute, we get a new bottle at the end of stage 3. Hence, the average time taken to manufacture 1 bottle is :

Without pipelining = 9/3 minutes = 3m

```
I F S | | | | | |
| | | I F S | | |
| | | | | | I F S (9 minutes)
```

With pipelining = 5/3 minutes = 1.67m

```
I F S | |
| I F S |
| | I F S (5 minutes)
```

Thus, pipelined operation increases the efficiency of a system.

Design of a basic pipeline

- In a pipelined processor, a pipeline has two ends, the input end and the output end. Between these ends, there are multiple stages/segments such that output of one stage is connected to input of next stage and each stage performs a specific operation.
- Interface registers are used to hold the intermediate output between two stages. These interface registers are also called latch or buffer.
- All the stages in the pipeline along with the interface registers are controlled by a common clock.

Execution in a pipelined processor

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

Non overlapped execution:

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

Total time = 8 Cycle

Overlapped execution:

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

Total time = 5 Cycle

Pipeline Stages

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

- **Stage 1 (Instruction Fetch)** In this stage the CPU reads instructions from the address in the memory whose value is present in the program counter.
- **Stage 2 (Instruction Decode)** In this stage, instruction is decoded and the register file is accessed to get the values from the registers used in the instruction.
- **Stage 3 (Instruction Execute)** In this stage, ALU operations are performed.
- **Stage 4 (Memory Access)** In this stage, memory operands are read and written from/to the memory that is present in the instruction.
- **Stage 5 (Write Back)** In this stage, computed/fetched value is written back to the register present in the instructions.

Performance of a pipelined processor Consider a 'k' segment pipeline with clock cycle time as 'Tp'. Let there be 'n' tasks to be completed in the pipelined processor. Now, the first instruction is going to take 'k' cycles to come out of the pipeline but the other 'n – 1' instructions will take only '1' cycle each, i.e, a total of 'n – 1' cycles. So, time taken to execute 'n' instructions in a pipelined processor:

$$ET_{pipeline} = k + n - 1 \text{ cycles}$$
$$= (k + n - 1)\ Tp$$

In the same case, for a non-pipelined processor, execution time of 'n' instructions will be:

$$ET_{non\text{-}pipeline} = n * k * Tp$$

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

S = Performance of pipelined processor /

Performance of Non-pipelined processor

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

$$S = ET_{non\text{-}pipeline} / ET_{pipeline}$$
$$\Rightarrow S = [n * k * Tp] / [(k + n - 1) * Tp]$$
$$S = [n * k] / [k + n - 1]$$

When the number of tasks 'n' are significantly larger than k, that is, n >> k

$$S = n * k / n$$

S = k

where ‘k’ are the number of stages in the pipeline.

Also, **Efficiency** = Given speed up / Max speed up = S / S_{max}
We know that, Smax = k
So, **Efficiency** = S / k

Throughput = Number of instructions / Total time to complete the instructions
So, **Throughput** = n / (k + n – 1) * Tp

Computer Organization and Architecture | Pipelining | Set 2 (Dependencies and Data Hazard)

Dependencies in a pipelined processor
There are mainly three types of dependencies possible in a pipelined processor. These are :

1. Structural Dependency
2. Control Dependency
3. Data Dependency

These dependencies may introduce stalls in the pipeline.

Stall : A stall is a cycle in the pipeline without new input.

Structural dependency

This dependency arises due to the resource conflict in the pipeline. A resource conflict is a situation when more than one instruction tries to access the same resource in the same cycle. A resource can be a register, memory, or ALU.

Example:

INSTRUCTION / CYCLE	1	2	3	4	5
I_1	IF(Mem)	ID	EX	Mem	
I_2		IF(Mem)	ID	EX	
I_3			IF(Mem)	ID	EX
I_4				IF(Mem)	ID

In the above scenario, in cycle 4, instructions I_1 and I_4 are trying to access same resource (Memory) which introduces a resource conflict.
To avoid this problem, we have to keep the instruction on wait until the required resource (memory in our case) becomes available. This wait will introduce stalls in the pipeline as shown below:

CYCLE	1	2	3	4	5	6	7	8
I_1	IF(Mem)	ID	EX	Mem	WB			
I_2		IF(Mem)	ID	EX	Mem	WB		
I_3			IF(Mem)	ID	EX	Mem	WB	
I_4				–	–	–	IF(Mem)	

Solution for structural dependency
To minimize structural dependency stalls in the pipeline, we use a hardware mechanism called Renaming.
Renaming : According to renaming, we divide the memory into two independent modules used to store the instruction and data separately called Code memory(CM) and Data memory(DM) respectively. CM will contain all the instructions and DM will contain all the operands that are required for the instructions.

INSTRUCTION/ CYCLE	1	2	3	4	5	6	7
I_1	IF(CM)	ID	EX	DM	WB		
I_2		IF(CM)	ID	EX	DM	WB	
I_3			IF(CM)	ID	EX	DM	WB
I_4				IF(CM)	ID	EX	DM
I_5					IF(CM)	ID	EX
I_6						IF(CM)	ID
I_7							IF(CM)

Control Dependency (Branch Hazards)

This type of dependency occurs during the transfer of control instructions such as BRANCH, CALL, JMP, etc. On many instruction architectures, the processor will not know the target address of these instructions when it needs to insert the new instruction into the pipeline. Due to this, unwanted instructions are fed to the pipeline.

Consider the following sequence of instructions in the program:
100: I_1
101: I_2 (JMP 250)
102: I_3 250: BI_1
Expected output: I_1 -> I_2 -> BI_1
NOTE: Generally, the target address of the JMP instruction is known after ID stage only.

INSTRUCTION/ CYCLE	1	2	3	4	5	6
I_1	IF	ID	EX	MEM	WB	
I_2		IF	ID (PC:250)	EX	Mem	WB
I_3			IF	ID	EX	Mem
BI_1				IF	ID	EX

Output Sequence: I_1 -> I_2 -> I_3 -> BI_1

So, the output sequence is not equal to the expected output, that means the pipeline is not implemented correctly.

To correct the above problem we need to stop the Instruction fetch until we get target address of branch instruction. This can be implemented by introducing delay slot until we get the target address.

INSTRUCTION/ CYCLE	1	2	3	4	5	6
I_1	IF	ID	EX	MEM	WB	
I_2		IF	ID (PC:250)	EX	Mem	WB
Delay	–	–	–	–	–	–
BI_1				IF	ID	EX

Output Sequence: I_1 -> I_2 -> Delay (Stall) -> BI_1
As the delay slot performs no operation, this output sequence is equal to the expected output sequence. But this slot introduces stall in the pipeline.

Solution for Control dependency Branch Prediction is the method through which stalls due to control dependency can be eliminated. In this at 1st stage prediction is done about which branch will be taken.For branch prediction Branch penalty is zero.

Branch penalty : The number of stalls introduced during the branch operations in the pipelined processor is known as branch penalty.
NOTE : As we see that the target address is available after the ID stage, so the number of stalls introduced in the pipeline is 1. Suppose, the branch target address would have been present after the ALU stage, there would have been 2 stalls. Generally, if the target address is present after the kth stage, then there will be (k – 1) stalls in the pipeline.
Total number of stalls introduced in the pipeline due to branch instructions = **Branch frequency * Branch Penalty**

Data Dependency (Data Hazard)
Let us consider an ADD instruction S, such that
S : ADD R1, R2, R3
Addresses read by S = I(S) = {R2, R3}
Addresses written by S = O(S) = {R1}

Now, we say that instruction S2 depends in instruction S1, when

$$[I(S1) \cap O(S2)] \cup [O(S1) \cap I(S2)] \cup [O(S1) \cap O(S2)] \neq \phi$$

This condition is called Bernstein condition.
Three cases exist:

- Flow (data) dependence: O(S1) ∩ I (S2), S1 → S2 and S1 writes after something read by S2
- Anti-dependence: I(S1) ∩ O(S2), S1 → S2 and S1 reads something before S2 overwrites it
- Output dependence: O(S1) ∩ O(S2), S1 → S2 and both write the same memory location.

Example: Let there be two instructions I1 and I2 such that:
I1 : ADD R1, R2, R3

I2 : SUB R4, R1, R2

When the above instructions are executed in a pipelined processor, then data dependency condition will occur, which means that I_2 tries to read the data before I_1 writes it, therefore, I_2 incorrectly gets the old value from I_1.

INSTRUCTION / CYCLE	1	2	3	4
I_1	IF	ID	EX	DM
I_2		IF	ID(Old value)	EX

To minimize data dependency stalls in the pipeline, **operand forwarding** is used.
Operand Forwarding : In operand forwarding, we use the interface registers present between the stages to hold intermediate output so that dependent instruction can access new value from the interface register directly.
Considering the same example:

I1 : ADD R1, R2, R3

I2 : SUB R4, R1, R2

INSTRUCTION / CYCLE	1	2	3	4
I_1	IF	ID	EX	DM
I_2		IF	ID	EX

Data Hazards
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:

5. RAW (Read after Write) [Flow/True data dependency]
6. WAR (Write after Read) [Anti-Data dependency]
7. WAW (Write after Write) [Output data 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.

Computer Organization and Architecture | Pipelining | Set 3

Types of pipeline

- Uniform delay pipeline
  In this type of pipeline, all the stages will take same time to complete an operation.
  In uniform delay pipeline, **Cycle Time (Tp) = Stage Delay**
  If buffers are included between the stages then, **Cycle Time (Tp) = Stage Delay + Buffer Delay**

- Non-Uniform delay pipeline
  In this type of pipeline, different stages take different time to complete an operation.
  In this type of pipeline, Cycle Time (Tp) = Maximum(Stage Delay)

For example, if there are 4 stages with delays, 1 ns, 2 ns, 3 ns, and 4 ns, then

Tp = Maximum(1 ns, 2 ns, 3 ns, 4 ns) = 4 ns

If buffers are included between the stages,

Tp = Maximum(Stage delay + Buffer delay)

I/O Interface (Interrupt and DMA Mode)

The method that is used to transfer information between internal storage and external I/O devices is known as I/O interface. The CPU is interfaced using special communication links by the peripherals connected to any computer system. These communication links are used to resolve the differences between CPU and peripheral. There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.

Mode 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:** It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.
   Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time

consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.

2. **Interrupt- initiated I/O:** Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

   Note: Both the methods programmed I/O and Interrupt-driven I/O require the active intervention of the processor to transfer data between memory and the I/O module, and any data transfer must transverse a path through the processor. Thus both these forms of I/O suffer from two inherent drawbacks.

- The I/O transfer rate is limited by the speed with which the processor can test and service a device.
- The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.

3. **Direct Memory Access**: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

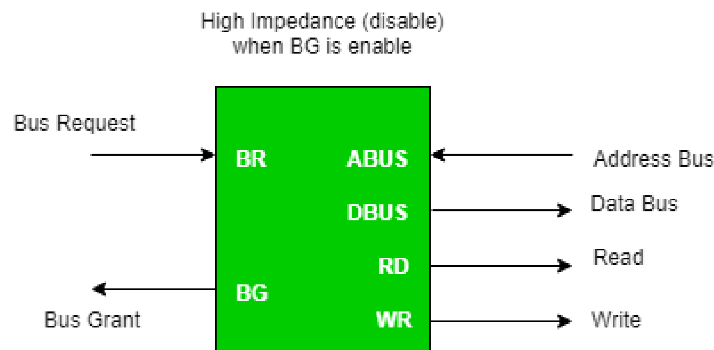


Figure - CPU Bus Signals for DMA Transfer

Bus Request : It is used by the DMA controller to request the CPU to relinquish the control of the buses.
Bus Grant : It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

Introduction to Parallel Computing

Before taking a toll on Parallel Computing, first let's take a look at the background of computations of computer software and why it failed for the modern era.

Computer software were written conventionally for serial computing. This meant that to solve a problem, an algorithm divides the problem into smaller instructions. These discrete instructions are then executed on Central Processing Unit of a computer one by one. Only after one instruction is finished, next one starts.

Real life example of this would be people standing in a queue waiting for movie ticket and there is only cashier.Cashier is giving ticket one by one to the persons. Complexity of this situation increases when there are 2 queues and only one cashier.

So, in short Serial Computing is following:

1. In this, a problem statement is broken into discrete instructions.
2. Then the instructions are executed one by one.
3. Only one instruction is executed at any moment of time.

Look at point 3. This was causing a huge problem in computing industry as only one instruction was getting executed at any moment of time. This was a huge waste of hardware resources as only one part of the hardware will be running for a particular instruction and of time. As problem statements were getting heavier and bulkier, so does the amount of time in execution of those statements. Example of processors are Pentium 3 and Pentium 4.

Now let's come back to our real life problem. We could definitely say that complexity will decrease when there are 2 queues and 2 cashier giving tickets to 2 persons simultaneously. This is an example of Parallel Computing.

Parallel Computing – It is the use of multiple processing elements simultaneously for solving any problem. Problems are broken down into instructions and are solved concurrently as each resource which has been applied to work is working at the same time.

Advantages of Parallel Computing over Serial Computing are as follows:

1. It saves time and money as many resources working together will reduce the time and cut potential costs.
2. It can be impractical to solve larger problems on Serial Computing.
3. It can take advantage of non-local resources when the local resources are finite.
4. Serial Computing 'wastes' the potential computing power, thus Parallel Computing makes better work of hardware.

Types of Parallelism:

1. **Bit-level parallelism:** It is the form of parallel computing which is based on the increasing processor's size. It reduces the number of instructions that the system must execute in order to perform a task on large-sized data. *Example:* Consider a scenario where an 8-bit processor must compute the sum of two 16-bit integers. It must first sum up the 8 lower-order bits, then add the 8 higher-order bits, thus requiring two instructions to perform the operation. A 16-bit processor can perform the operation with just one instruction.
2. **Instruction-level parallelism:** A processor can only address less than one instruction for each clock cycle phase. These instructions can be re-ordered and grouped which are later on executed concurrently without affecting the result of the program. This is called instruction-level parallelism.
3. **Task Parallelism:** Task parallelism employs the decomposition of a task into subtasks and then allocating each of the subtasks for execution. The processors perform execution of sub tasks concurrently.

Why parallel computing?

- The whole real world runs in dynamic nature i.e. many things happen at a certain time but at different places concurrently. This data is extensively huge to manage.
- Real world data needs more dynamic simulation and modeling, and for achieving the same, parallel computing is the key.
- Parallel computing provides concurrency and saves time and money.
- Complex, large datasets, and their management can be organized only and only using parallel computing's approach.
- Ensures the effective utilization of the resources. The hardware is guaranteed to be used effectively whereas in serial computation only some part of hardware was used and the rest rendered idle.
- Also, it is impractical to implement real-time systems using serial computing.

Applications of Parallel Computing:

- Data bases and Data mining.
- Real time simulation of systems.
- Science and Engineering.
- Advanced graphics, augmented reality and virtual reality.

Limitations of Parallel Computing:

- It addresses such as communication and synchronization between multiple sub-tasks and processes which is difficult to achieve.
- The algorithms must be managed in such a way that they can be handled in the parallel mechanism.
- The algorithms or program must have low coupling and high cohesion. But it's difficult to create such programs.
- More technically skilled and expert programmers can code a parallelism based program well.

Future of Parallel Computing: The computational graph has undergone a great transition from serial computing to parallel computing. Tech giant such as Intel has already taken a step towards parallel computing by employing multicore processors. Parallel computation will revolutionize the way computers work in the future, for the better good. With all the world connecting to each other even more than before, Parallel Computing does a better role in helping us stay that way. With faster networks, distributed systems, and multi-processor computers, it becomes even more necessary.

Parallel processing – systolic arrays

Parallel processing approach diverges from traditional Von Neumann architecture. One such approach is the concept of Systolic processing using systolic arrays.
A **systolic array** is a network of processors that rhythmically compute and pass data through the system. They derived their name from drawing an analogy to how blood rhythmically flows through a biological heart as the data flows from memory in a rhythmic fashion passing through many elements before it returns to memory.It is also an example of pipelining along with parallel computing.It was introduced in 1970s and was used by Intel to make CMU's iWarp processor in 1990.
In a systolic array there are a large number of identical simple processors or processing elements (PEs) that are arranged in a well organised structure such as linear or two dimensional array. Each processing element is connected with the other PEs and has a limited private storage.

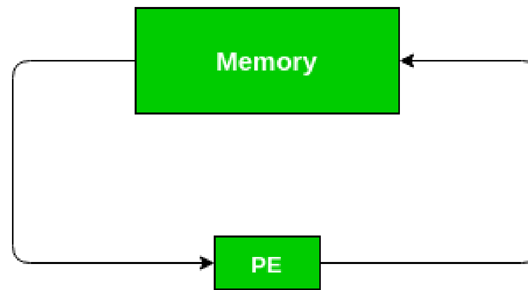


Figure - A Processing Element (PE)

A Host station is often used for communication with the outside world in the network.

Characteristics:

1. **Parallel Computing –** Many processes are carried out simultaneously. As the arrays have a non-centralized structure, parallel computing is implemented.
2. **Pipelinability –** It means that the array can achieve high speed. It shows a linear rate pipelinability.
3. **Synchronous evaluation –** Computation of data is timed by a global clock and then the data is passed through the network.The global clock synchronizes the array and has fixed length clock cycles.
4. **Repetability –** Most of the arrays have the repetition and interconnection of a single type of PE in the entire network.
5. **Spatial Locality –** The cells have a local communication interconnection.
6. **Temporal Locality –** One unit time delay is at least required for the transmission of signals from one cell to another.
7. **Modularity and regularity –** A systolic array consists of processing units that are modular and have homogeneous interconnection and the computer network can be extended indefinitely.

Advantages of Systolic array –

- It employs high degree of parallelism and can sustain a very high throughput.
- These are highly compact, robust and efficient.
- Data and control flow are simple and regular.

Disadvantages of Systolic array –

- They are highly specialized and thus are inflexible regarding the problems they can solve.
- These are difficult to build.
- These are expensive.

List And Briefly Explain Various Ways In Which An Instruction Pipeline Can Deal With Conditional Branch Instructions.

Multiple streams: A brute-force approach is to replicate the initial portions of the pipeline and allow the pipeline to fetch both instructions, making use of two streams.

Prefetch branch target:

When a conditional branch is recognized, the target of the branch is prefetched, in addition to the instruction following the branch. This target is then saved until the branch instruction is executed. If the branch is taken, the target has already been prefetched.

Loop buffer:

A loop buffer is a small, very-high-speed memory maintained by the instruction fetch stage of the pipeline and containing the nmost recently fetched instructions, in sequence. If a branch is to be taken, the hardware first checks whether the branch target is within the buffer. If so, the next instruction is fetched from the buffer.

Branch prediction:

A prediction is made whether a conditional branch will be taken when executed, and subsequent instructions are fetched accordingly.

Delayed branch:

It is possible to improve pipeline performance by automatically rearranging instructions within a program, so that branch instructions occur later than actually desired.

List two sources of interrups and exceptions.

1. Interrupts

• Maskable interrupts: Received on the processor's INTR pin. The processor does not recognize a maskable interrupt unless the interrupt enable flag (IF) is set.

• Nonmaskable interrupts: Received on the processor's NMI pin. Recognition of such interrupts cannot be prevented.

2. Exceptions

• Processor-detected exceptions: Results when the processor encounters an error while attempting to execute an instruction.

• Programmed exceptions: These are instructions that generate an exception (e.g., INTO, INT3, INT, and BOUND).

Difference between instructions and instruction sequencing

Four types of operations

1. Data transfer between memory and processor registers.
2. Arithmetic & logic operations on data
3. Program sequencing & control
4. I/O transfers.

1. Register transfer notations (RTN)

R3<–[R1]+[R2]

- Right hand side of RTN-denotes a value.
- Left hand side of RTN-name of a location.

2. Assembly language notations(ALN)

Add R1, R2, R3

- Adding ontents of R1, R2 & place sum in R3.

3. Basic instruction types-4 types

Three address instructions– Add A,B,C

A, B-source operands

C-destination operands

- Two address instructions-Add A,B

B <–[A] + [B]

- One address instructions –Add A

Add contents of A to accumulator & store sum back to accumulator.

- Zero address instructions

Instruction store operands in a structure called push down stack.

4. Instruction execution & straight line sequencing

- The processor control circuits use information in PC to fetch & execute instructions one at a time in order of increasing address.
- This is called straight line sequencing.
- Executing an instruction-2 phase procedures.
- 1st phase–"instruction fetch"-instruction is fetched from memory location whose address is in PC.
- This instruction is placed in instruction register in processor
- 2nd phase-"instruction execute"-instruction in IR is examined to determine which operation to be performed.

5. Branching

- Branch-type of instruction loads a new value into program counter.
- So processor fetches & executes instruction at this new address called "branch target"
- Conditional branch-causes a branch if a specified condition is satisfied.
- E.g. Branch>0 LOOP –conditional branch instruction .it executes only if it satisfies condition.

6. Condition codes

- Recording required information in individual bits called "condition code flags".
- These flags are grouped together in a special processor register called "condition code register" or "status register"
- Individual condition code flags-1 or 0.
- 4 commonly used flags.

1. N (negative)-set to 1 if result is –ve or else 0.
2. Z (zero)-set to 1 if result is 0, or else 0 .
3. V (overflow)-set to 1if arithmetic overflow occurs or else 0.
4. C(carry)-set to 1 if carry out results from operation or else 0

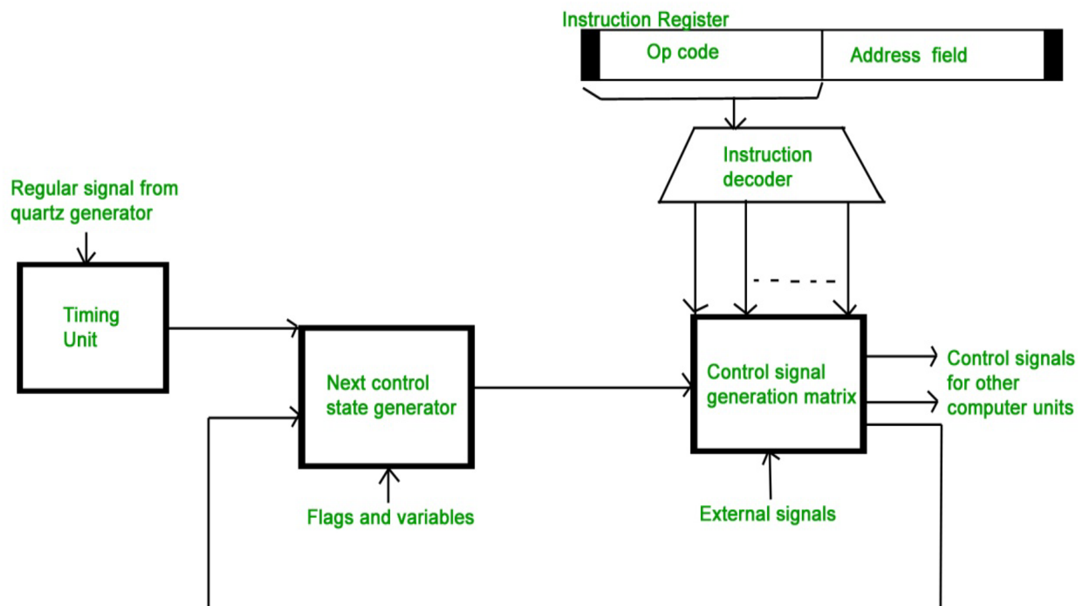
<u>Draw and explain typical Hardwired v/s Micro-programmed Control Unit.</u>

To execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence. There are two approaches used for generating the control signals in proper sequence as Hardwired Control unit and Micro-programmed control unit.

Hardwired Control Unit – The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes and the external inputs. The outputs of the state machine are the control signals. The sequence of the operation carried out by this machine is determined by the wiring of the logic elements and hence named as "hardwired".

- Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.

- Hardwired control is faster than micro-programmed control.
- A controller that uses this approach can operate at high speed.
- RISC architecture is based on hardwired control unit

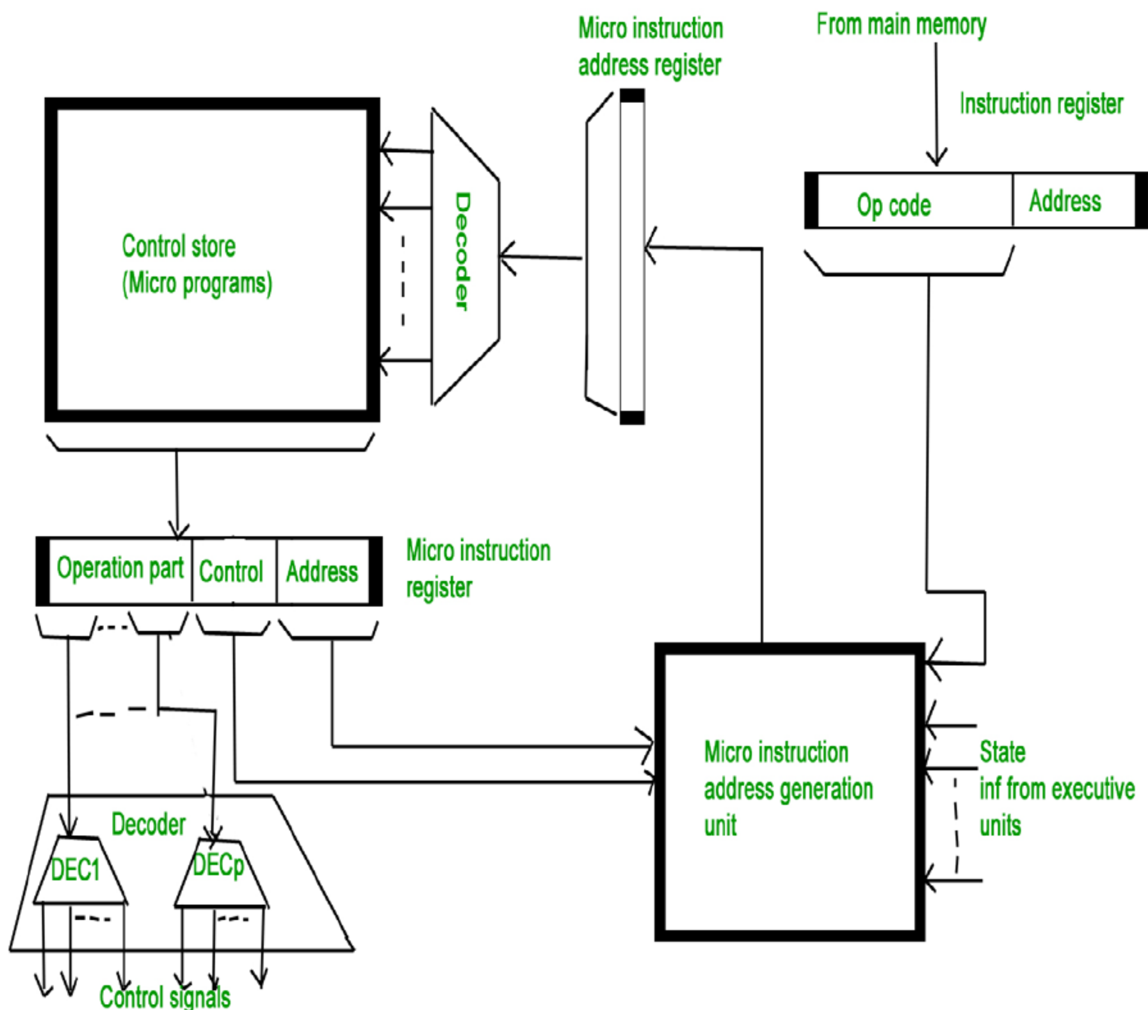


Micro-programmed Control Unit –

- The control signals associated with operations are stored in special memory units inaccessible by the programmer as Control Words.
- Control signals are generated by a program are similar to machine language programs.
- Micro-programmed control unit is slower in speed because of the time it takes to fetch microinstructions from the control memory.

Some Important Terms –

1. **Control Word :** A control word is a word whose individual bits represent various control signals.
2. **Micro-routine :** A sequence of control words corresponding to the control sequence of a machine instruction constitutes the micro-routine for that instruction.
3. **Micro-instruction :** Individual control words in this micro-routine are referred to as microinstructions.
4. **Micro-program :** A sequence of micro-instructions is called a micro-program, which is stored in a ROM or RAM called a Control Memory (CM).
5. **Control Store :** the micro-routines for all instructions in the instruction set of a computer are stored in a special memory called the Control Store.



Types of Micro-programmed Control Unit – Based on the type of Control Word stored in the Control Memory (CM), it is classified into two types :

1. Horizontal Micro-programmed control Unit : The control signals are represented in the decoded binary format that is 1 bit/CS. Example: If 53 Control signals are present in the processor than 53 bits are required. More than 1 control signal can be enabled at a time.

- It supports longer control word.
- It is used in parallel processing applications.
- It allows higher degree of parallelism. If degree is n, n CS are enabled at a time.
- It requires no additional hardware(decoders). It means it is faster than Vertical Microprogrammed.
- It is more flexible than vertical microprogrammed.

2. Vertical Micro-programmed control Unit : The control signals re represented in the encoded binary format. For N control signals- Log2(N) bits are required.

- It supports shorter control words.
- It supports easy implementation of new conrol signals therefore it is more flexible.
- It allows low degree of parallelism i.e., degree of parallelism is either 0 or 1.

- Requires an additional hardware (decoders) to generate control signals, it implies it is slower than horizontal microprogrammed.
- It is less flexible than horizontal but more flexible than that of hardwired control unit