UNIT - 2

Instruction Formats (Zero, One, Two and Three Address Instruction)

Computer perform task on the basis of instruction provided. A instruction in computer comprises of groups called fields. These field contains different information as for computers every thing is in 0 and 1 so each field has different significance on the basis of which a CPU decide what so perform. The most common fields are:

- Operation field which specifies the operation to be performed like addition.
- Address field which contain the location of operand, i.e., register or memory location.
- Mode field which specifies how operand is to be founded.

A instruction is of various length depending upon the number of addresses it contain. Generally CPU organization are of three types on the basis of number of address fields:

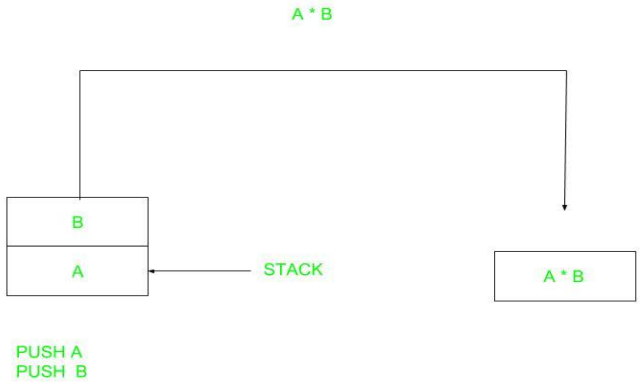
1. Single Accumulator organization
2. General register organization
3. Stack organization

In first organization operation is done involving a special register called accumulator. In second on multiple registers are used for the computation purpose. In third organization the work on stack basis operation due to which it does not contain any address field. It is not necessary that only a single organization is applied a blend of various organization is mostly what we see generally.

On the basis of number of address instruction are classified as:

Note that we will use X = (A+B)*(C+D) expression to showcase the procedure.

1. **Zero Address Instructions –**

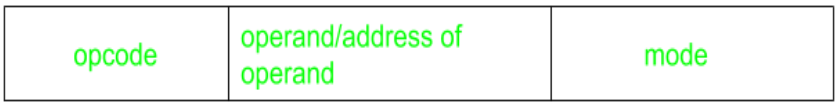


A stack based computer do not use address field in instruction.To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

```
Expression: X = (A+B)*(C+D)
Postfixed : X = AB+CD+*
TOP means top of stack
M[X] is any memory location
```

PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)
POP	X	M[X] = TOP

2. **One Address Instructions –**
   This use a implied ACCUMULATOR register for data manipulation.One operand is in accumulator and other is in register or memory location.Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.




```
Expression: X = (A+B)*(C+D)
AC is accumulator
M[] is any memory location
M[T] is temporary location
```

LOAD	A	AC = M[A]
ADD	B	AC = AC + M[B]
STORE	T	M[T] = AC
LOAD	C	AC = M[C]
ADD	D	AC = AC + M[D]
MUL	T	AC = AC * M[T]
STORE	X	M[X] = AC

3. **Two Address Instructions –**
   This is common in commercial computers.Here two address can be specified in the instruction.Unlike earlier in one address instruction the result was stored in accumulator here result cab be stored at different location rather than just accumulator, but require more number of bit to represent address.

Here destination address can also contain operand.

```
Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location
```

MOV	R1, A	R1 = M[A]
ADD	R1, B	R1 = R1 + M[B]
MOV	R2, C	R2 = C

ADD	R2, D	R2 = R2 + D
MUL	R1, R2	R1 = R1 * R2
MOV	X, R1	M[X] = R1

4. **Three Address Instructions –**
   This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.


```
Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location
```

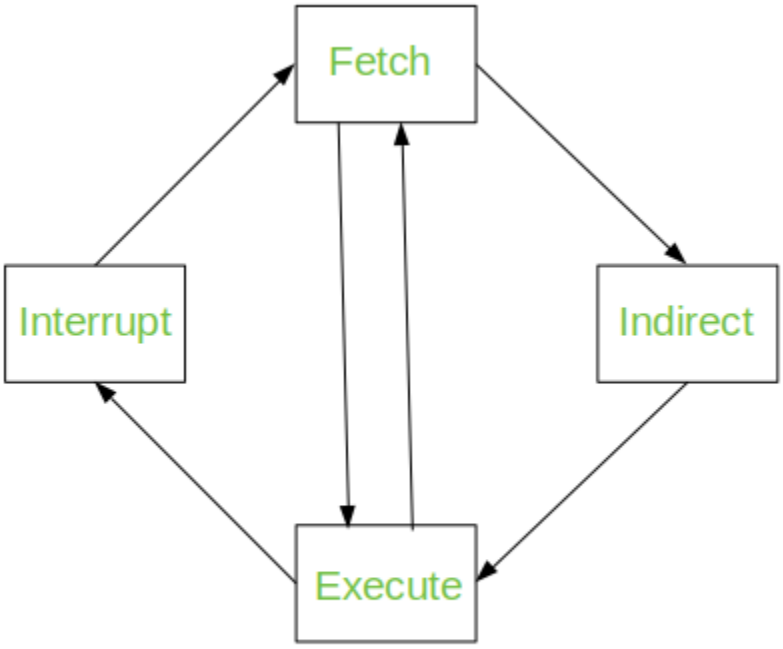
Computer Organization | Different Instruction Cycles

Registers Involved In Each Instruction Cycle:

- **Memory address registers(MAR)** : It is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.
- **Memory Buffer Register(MBR)** : It is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from the memory.
- **Program Counter(PC)** : Holds the address of the next instruction to be fetched.
- **Instruction Register(IR)** : Holds the last instruction fetched.

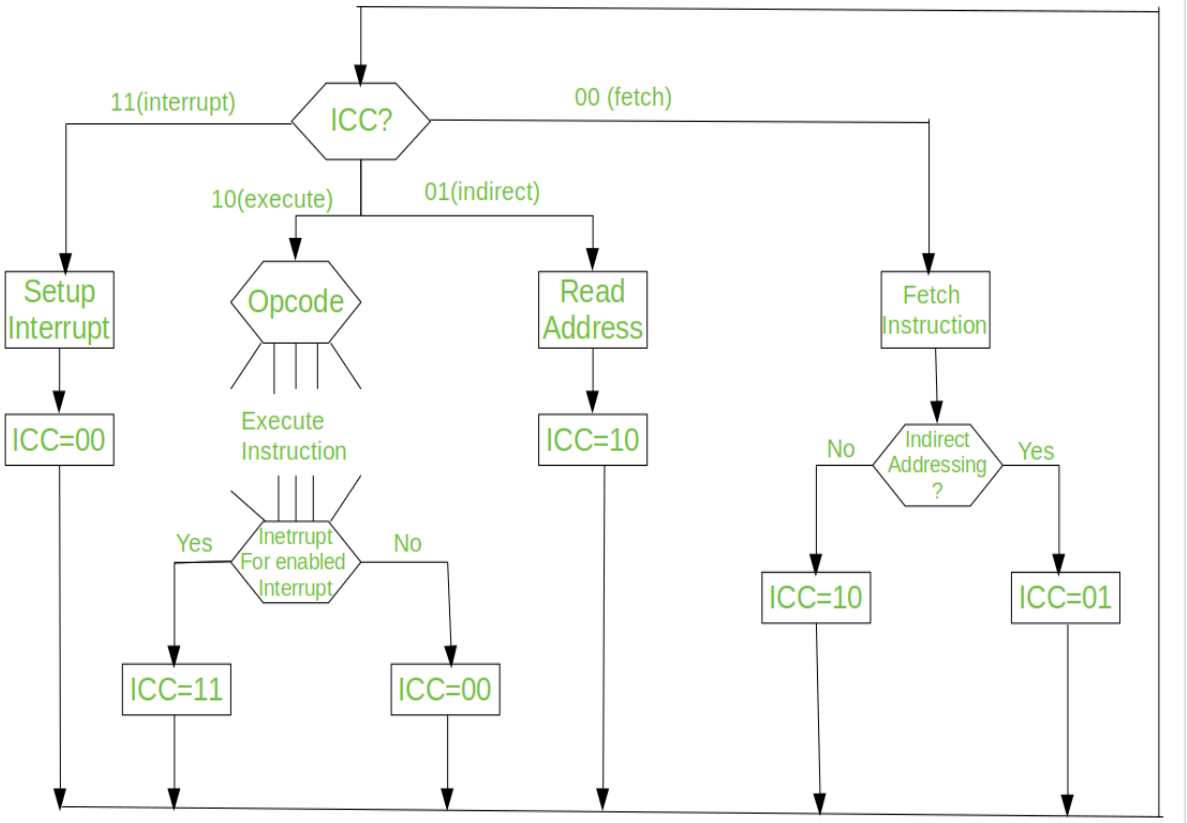
The Instruction Cycle –

Each phase of Instruction Cycle can be decomposed into a sequence of elementary micro-operations. In the above examples, there is one sequence each for the *Fetch, Indirect, Execute and Interrupt Cycles*.



The Instruction Cycle

The *Indirect Cycle* is always followed by the *Execute Cycle*. The *Interrupt Cycle* is always followed by the *Fetch Cycle*. For both fetch and execute cycles, the next cycle depends on the state of the system.

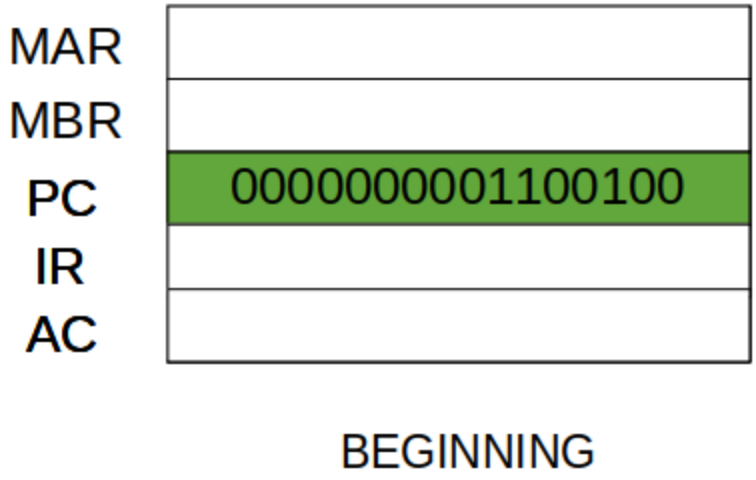


Flowchart for Instruction Cycle

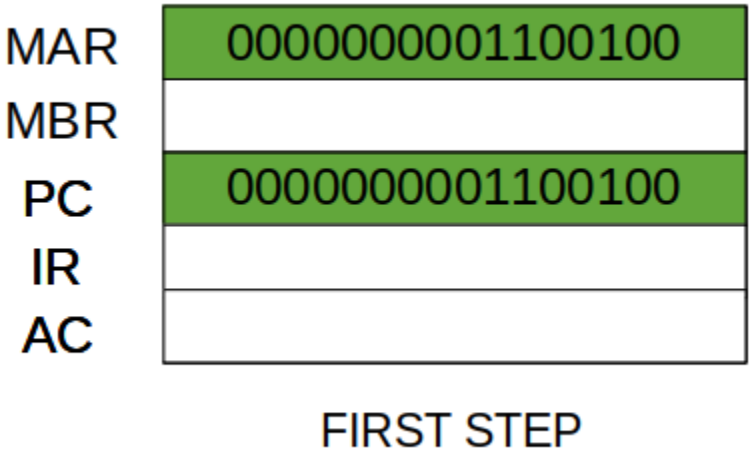
We assumed a new 2-bit register called *Instruction Cycle Code* (ICC). The ICC designates the state of processor in terms of which portion of the cycle it is in:-
00 : Fetch Cycle
01 : Indirect Cycle
10 : Execute Cycle
11 : Interrupt Cycle

At the end of the each cycles, the ICC is set appropriately.The above flowchart of *Instruction Cycle* describes the complete sequence of micro-operations, depending only on the instruction sequence and the interrupt pattern(this is a simplified example). The operation of the processor is described as the performance of a sequence of micro-operation.
Different Instruction Cycles:

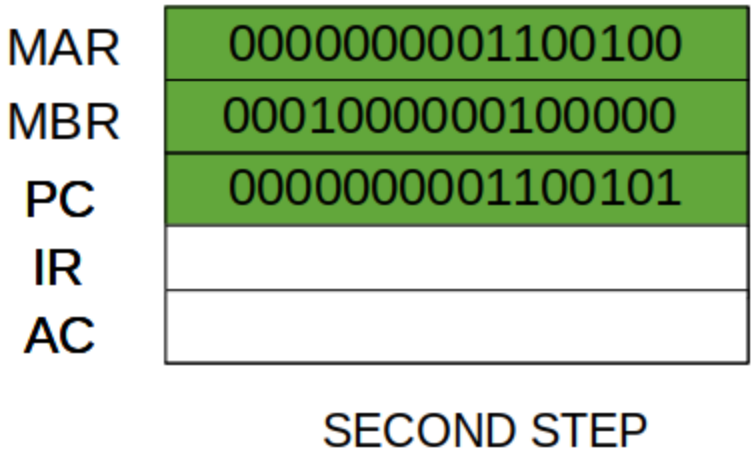
1. **The Fetch Cycle –**
   At the beginning of the fetch cycle, the address of the next instruction to be executed is in the *Program Counter*(PC).



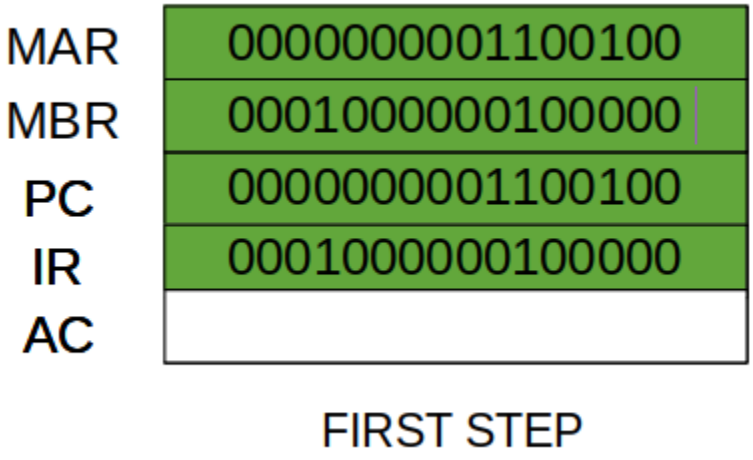
Step 1: The address in the program counter is moved to the memory address register(MAR), as this is the only register which is connected to address lines of the system bus.



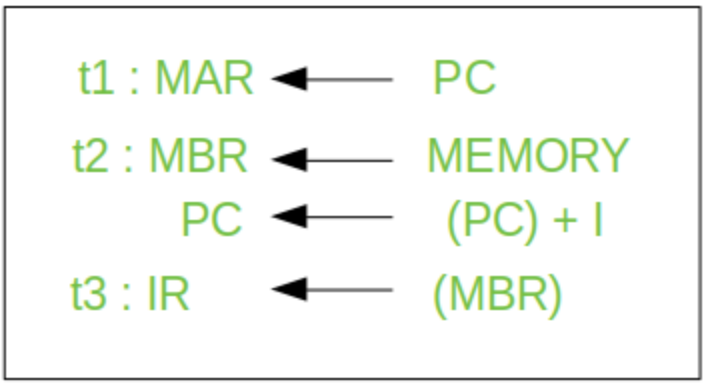
Step 2: The address in MAR is placed on the address bus, now the control unit issues a READ command on the control bus, and the result appears on the data bus and is then copied into the memory buffer register(MBR). Program counter is incremented by one, to get ready for the next instruction.(These two action can be performed simultaneously to save time)



Step 3: The content of the MBR is moved to the instruction register(IR).

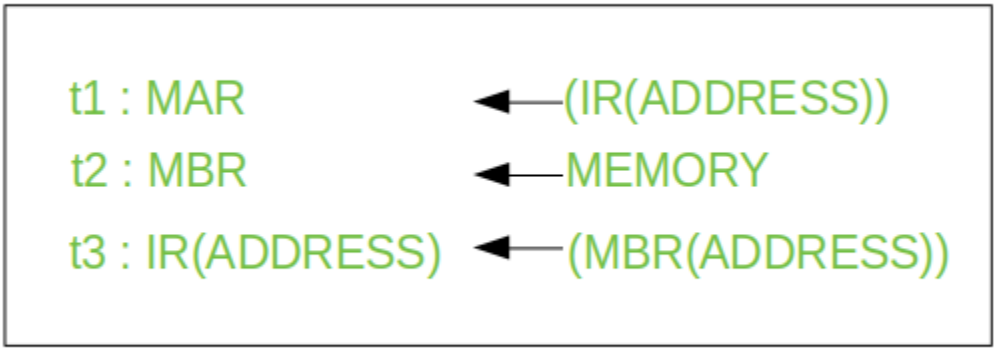


Thus, a simple *Fetch Cycle* consist of three steps and four micro-operation. Symbolically, we can write these sequence of events as follows:-



Here 'I' is the instruction length. The notations (t1, t2, t3) represents successive time units. We assume that a clock is available for timing purposes and it emits regularly spaced clock pulses. Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro-operation can be performed within the time of a single time unit.
First time unit: Move the contents of the PC to MAR.
Second time unit: Move contents of memory location specified by MAR to MBR. Increment content of PC by I.
Third time unit: Move contents of MBR to IR.
Note: Second and third micro-operations both take place during the second time unit.

2. **The Indirect Cycles –**
   Once an instruction is fetched, the next step is to fetch source operands. *Source Operand* is being fetched by indirect addressing(it can be fetched by any addressing mode, here its done by indirect addressing). Register-based operands need not be fetched. Once the opcode is executed, a similar process may be needed to store the result in main memory. Following *micro-operations* takes place:-



Step 1: The address field of the instruction is transferred to the MAR. This is used to fetch the address of the operand.
Step 2: The address field of the IR is updated from the MBR.(So that it now contains a direct addressing rather than indirect addressing)
Step 3: The IR is now in the state, as if indirect addressing has not been occurred.

Note: Now IR is ready for the execute cycle, but it skips that cycle for a moment to consider the *Interrupt Cycle* .

3. **The Execute Cycle**

The other three cycles(*Fetch, Indirect and Interrupt*) are simple and predictable. Each of them requires simple, small and fixed sequence of micro-operation. In each case same micro-operation are repeated each time around.
Execute Cycle is different from them. Like, for a machine with N different opcodes there are N different sequence of micro-operations that can occur.
Lets take an hypothetical example :-
consider an add instruction:

```
ADD   R , X
```

Here, this instruction adds the content of location X to register R. Corresponding micro-operation will be:-

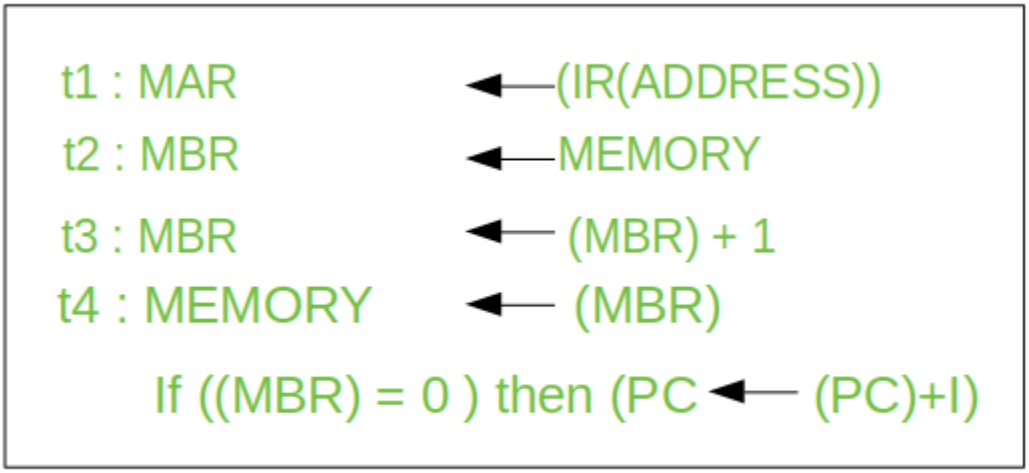
```
t1 : MAR   ←(IR(ADDRESS))
t2 : MBR   ←MEMORY
t3 : R     ←(R) + (MBR)
```

We begin with the IR containing the ADD instruction.
Step 1: The address portion of IR is loaded into the MAR.
Step 2: The address field of the IR is updated from the MBR, so the reference

memory location is read.
Step 3: Now, the contents of R and MBR are added by the ALU.

Lets take a complex example :-

Here, the content of location X is incremented by 1. If the result is 0, the next instruction will be skipped. Corresponding sequence of micro-operation will be :-



Here, the PC is incremented if (MBR) = 0. This test (is MBR equal to zero or not) and action (PC is incremented by 1) can be implemented as one micro-operation.
Note : This test and action micro-operation can be performed during the same time unit during which the updated value MBR is stored back to memory.

4. **The Interrupt Cycle**:
   At the completion of the Execute Cycle, a test is made to determine whether any enabled interrupt has occurred or not. If an enabled interrupt has occurred then Interrupt Cycle occurs. The natare of this cycle varies greatly from one machine to another.
   Lets take a sequence of micro-operation:-

Step 1: Contents of the PC is transferred to the MBR, so that they can be saved for return.
Step 2: MAR is loaded with the address at which the contents of the PC are to be saved.
PC is loaded with the address of the start of the interrupt-processing routine.
Step 3: MBR, containing the old value of PC, is stored in memory.

Note: In step 2, two actions are implemented as one micro-operation. However, most processor provide multiple types of interrupts, it may take one or more micro-operation to obtain the save_address and the routine_address before they are transferred to the MAR and PC respectively.

Difference between Horizontal and Vertical micro-programmed Control Unit

Basically, **control unit (CU)** is the engine that runs the entire functions of a computer with the help of control signals in the proper sequence. In the **micro-programmed** control unit approach, the control signals that are associated with the operations are stored in special memory units. It is convenient to think of sets of control signals that cause specific micro-operations to occur as being “microinstructions”. The sequences of microinstructions could be stored in an internal “*control*” memory. Micro-programmed control unit can be classified into two types based on the type of Control Word stored in the Control Memory, viz., Horizontal micro-programmed control unit and Vertical micro-programmed control unit.

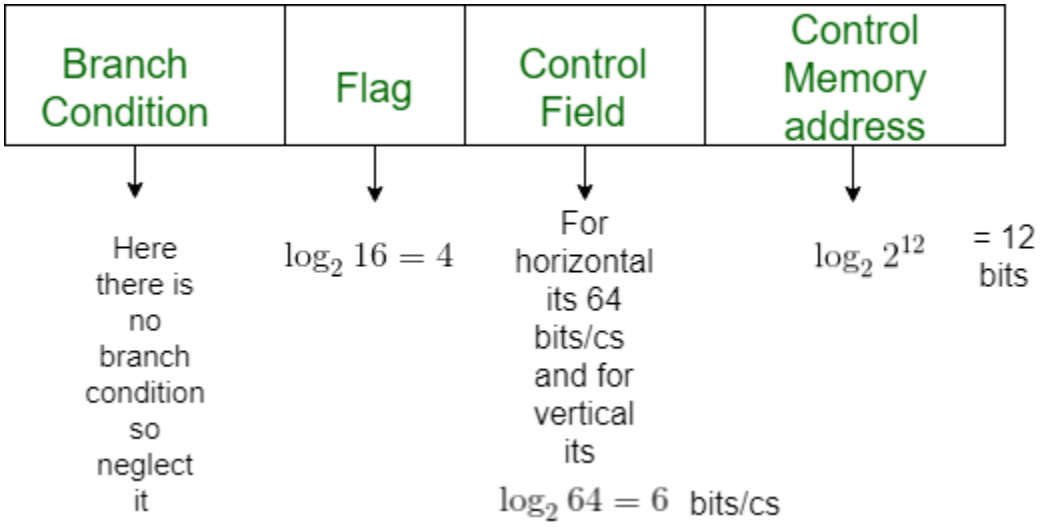
- In *Horizontal micro-programmed* control unit, the control signals are represented in the decoded binary format, i.e., 1 bit/CS. Here ‘n’ control signals require n bit encoding. On the other hand.
- In *Vertical micro-programmed* control unit, the control signals are represented in the encoded binary format. Here ‘n’ control signals require $\log_2 n$ bit encoding.

Comparison between Horizontal micro-programmed control unit and Vertical micro-programmed control unit:

HORIZONTAL M-PROGRAMMED CU	VERTICAL M-PROGRAMMED CU
It supports longer control word.	It supports shorter control word.
It allows higher degree of parallelism. If degree is n, then n Control Signals are enabled at a time.	It allows low degree of parallelism i.e., degree of parallelism is either 0 or 1.
No additional hardware is required.	Additional hardware in the form of decoders are required to generate control signals.
It is faster than Vertical micro-programmed control unit.	it is slower than Horizontal micro-programmed control unit.
It is less flexible than Vertical micro-programmed control unit.	It is more flexible than Horizontal micro-programmed control unit.
Horizontal micro-programmed control unit uses horizontal microinstruction, where every bit in the control field attaches to a control line.	Vertical micro-programmed control unit uses vertical microinstruction, where a code is used for each action to be performedand thedecoder translates this code into individual control

HORIZONTAL M-PROGRAMMED CU	VERTICAL M-PROGRAMMED CU
	signals.
Horizontal micro-programmed control unit makes less use of ROM encoding than vertical micro-programmed control unit.	Vertical micro-programmed control unit makes more use of ROM encoding to reduce the length of the control word.

Example: Consider a hypothetical Control Unit which supports 4 k words. The Hardware contains 64 control signals and 16 Flags. What is the size of control word used in bits and control memory in byte using:
a) Horizontal Programming
b) Vertical programming
Solution:



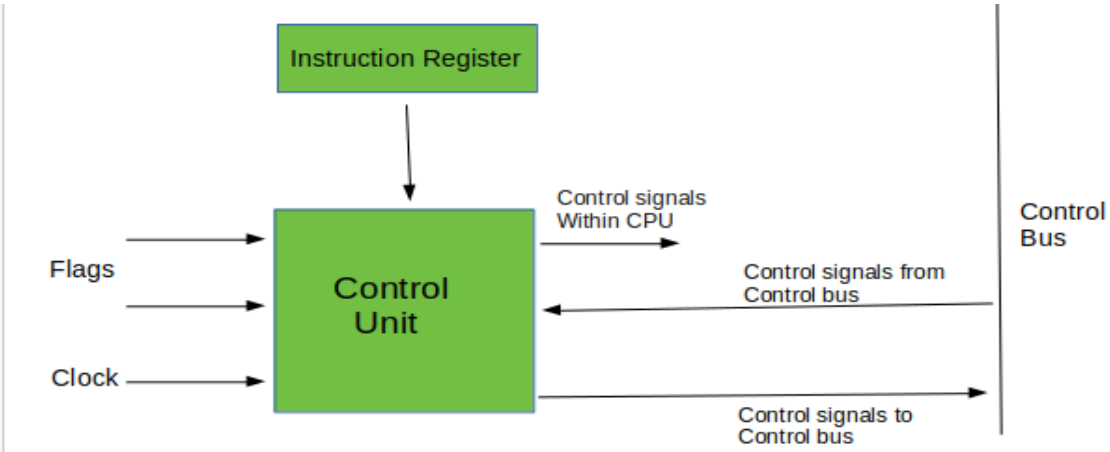

```
a)For Horizontal


64 bits for 64 signals
Control Word Size = 4 + 64 + 12 = 80 bits


Control Memory = 4 kW = ( (4* 80) / 8 ) = 40 kByte
```

```
a)For Vertical

6 bits for 64 signals i.e log2 64

Control Word Size = 4 + 6 + 12 = 22 bits

Control Memory = 4 kW = ( (4* 22) / 8 ) = 11 kByte
```

Introduction of Control Unit and its Design

Control Unit is the part of the computer's central processing unit (CPU), which directs the operation of the processor. It was included as part of the Von Neumann Architecture by John von Neumann. It is the responsibility of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been sent to the processor. It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions.
A control unit works by receiving input information to which it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to perform. The functions that a control unit performs are dependent on the type of CPU because the architecture of CPU varies from manufacturer to manufacturer. Examples of devices that require a CU are:

- Control Processing Units(CPUs)
- Graphics Processing Units(GPUs)



Block Diagram of the Control Unit

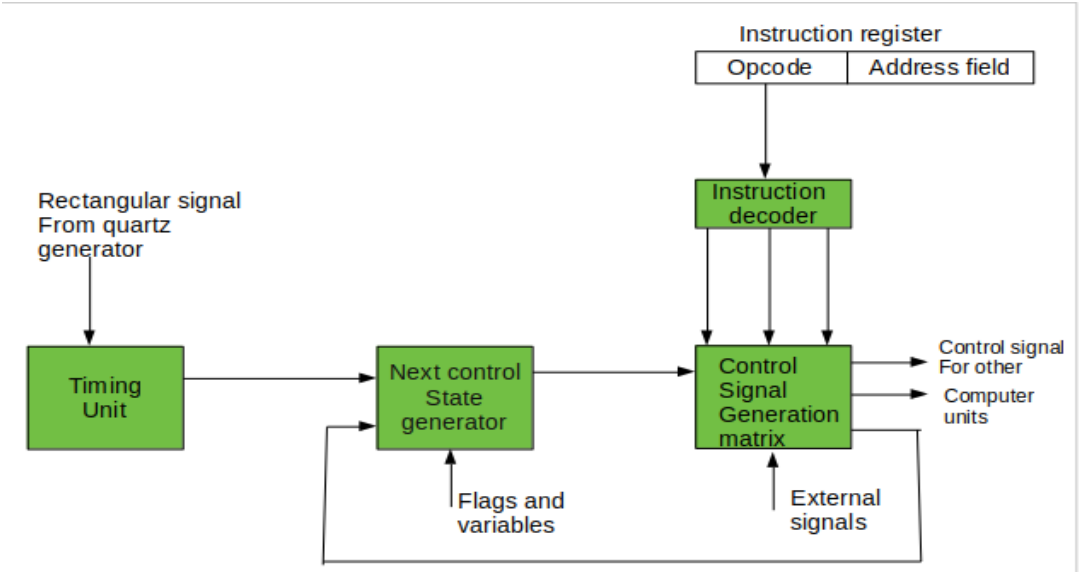
Functions of the Control Unit –

1. It coordinates the sequence of data movements into, out of, and between a processor's many sub-units.
2. It interprets instructions.
3. It controls data flow inside the processor.
4. It receives external instructions or commands to which it converts to sequence of control signals.
5. It controls many execution units(i.e. ALU, data buffers and registers) contained within a CPU.
6. It also handles multiple tasks, such as fetching, decoding, execution handling and storing results.

Types of Control Unit –

There are two types of control units: Hardwired control unit and Microprogrammable control unit.

1. **Hardwired Control Unit –**
   In the Hardwired control unit, the control signals that are important for instruction execution control are generated by specially designed hardware logical circuits, in which we can not modify the signal generation method without physical change of the circuit structure. The operation code of an instruction contains the basic data for control signal generation. In the instruction decoder, the operation code is decoded. The instruction decoder constitutes a set of many decoders that decode different fields of the instruction opcode. As a result, few output lines going out from the instruction decoder obtains active signal values. These output lines are connected to the inputs of the matrix that generates control signals for executive units of the computer. This matrix implements logical combinations of the decoded signals from the instruction opcode with the outputs from the matrix that generates signals representing consecutive control unit states and with signals coming from the outside of the processor, e.g. interrupt signals. The matrices are built in a similar way as a programmable logic arrays.



Block diagram of a hardwired control unit of a computer

Control signals for an instruction execution have to be generated not in a single time point but during the entire time interval that corresponds to the instruction execution cycle. Following the structure of this cycle, the suitable sequence of internal states is organized in the control unit.

A number of signals generated by the control signal generator matrix are sent back to inputs of the next control state generator matrix. This matrix combines these signals with the timing signals, which are generated by the timing unit based on the rectangular patterns usually supplied by the quartz generator. When a new instruction arrives at the control unit, the control units is in the initial state of new instruction fetching. Instruction decoding allows the control unit enters the first state relating execution of the new instruction, which lasts as long as the timing signals and other input signals as flags and state information of the computer remain unaltered. A change of any of the earlier mentioned signals stimulates the change of the control unit state.

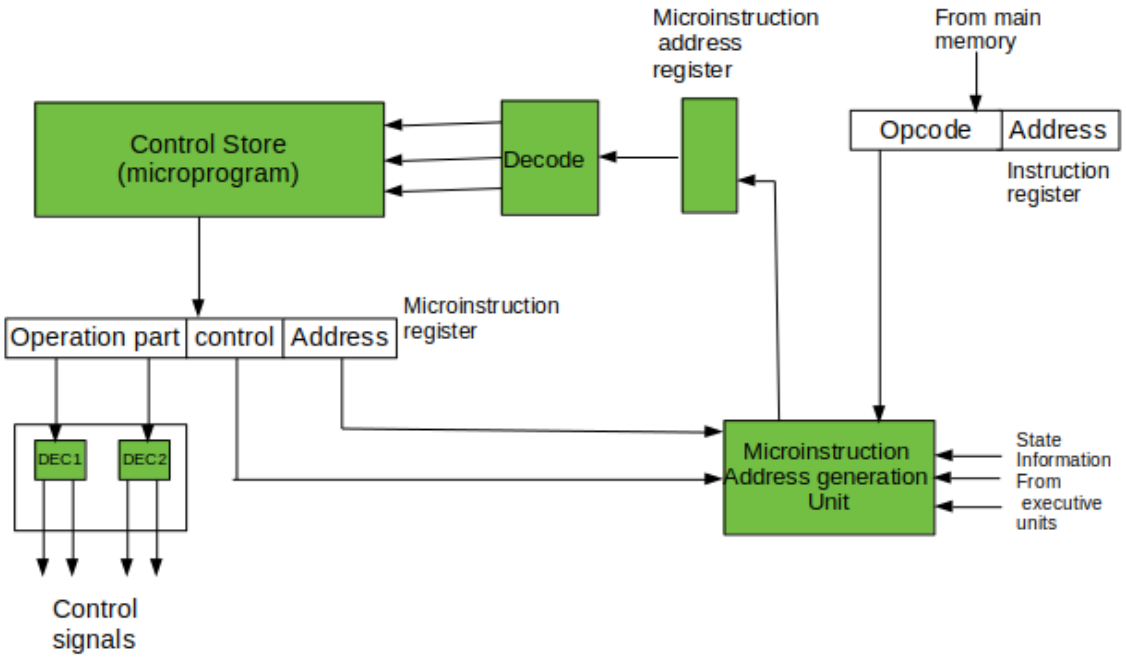
This causes that a new respective input is generated for the control signal generator matrix. When an external signal appears, (e.g. an interrupt) the control unit takes entry into a next control state that is the state concerned with the reaction to this external signal (e.g. interrupt processing). The values of flags and state variables of the computer are used to select suitable states for the instruction execution cycle.

The last states in the cycle are control states that commence fetching the next instruction of the program: sending the program counter content to the main memory address buffer register and next, reading the instruction word to the instruction register of computer. When the ongoing instruction is the stop instruction that ends program execution, the control unit enters an operating system state, in which it waits for a next user directive.

2. **Microprogrammable control unit –**
The fundamental difference between these unit structures and the structure of the hardwired control unit is the existence of the control store that is used for storing words containing encoded control signals mandatory for instruction execution.

In microprogrammed control units, subsequent instruction words are fetched into the instruction register in a normal way. However, the operation code of each instruction is not directly decoded to enable immediate control signal generation but it comprises the initial address of a microprogram contained in the control store.

- **With a single-level control store:**
In this, the instruction opcode from the instruction register is sent to the control store address register. Based on this address, the first microinstruction of a microprogram that interprets execution of this instruction is read to the microinstruction register. This microinstruction contains in its operation part encoded control signals, normally as few bit fields. In a set microinstruction field decoders, the fields are decoded. The microinstruction also contains the address of the next microinstruction of the given instruction microprogram and a control field used to control activities of the microinstruction address generator.

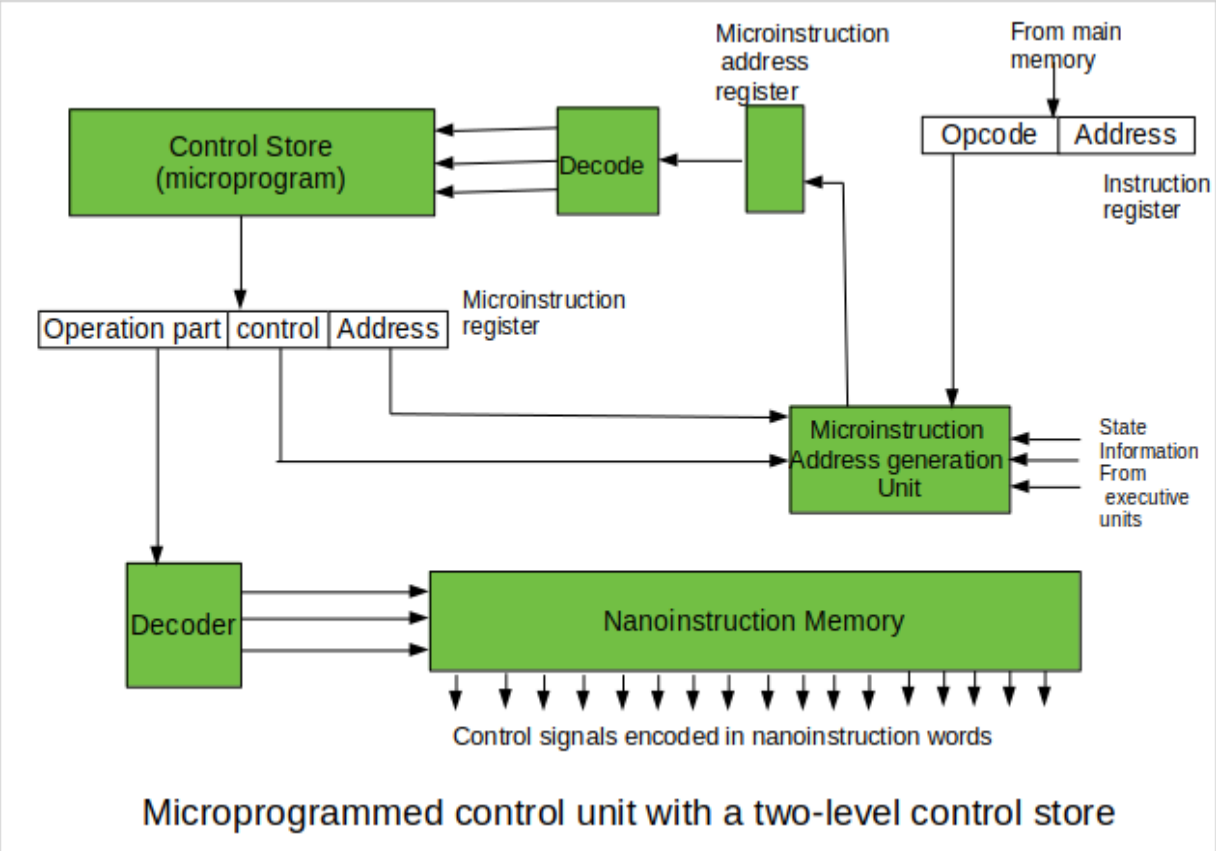


Microprogrammed control unit with a single level control store

The last mentioned field decides the addressing mode (addressing operation) to be applied to the address embedded in the ongoing microinstruction. In microinstructions along with conditional addressing mode, this address is refined by using the processor condition flags that represent the status of computations in the current program. The last microinstruction in the

instruction of the given microprogram is the microinstruction that fetches the next instruction from the main memory to the instruction register.

- **With a two-level control store:**
  In this, in a control unit with a two-level control store, besides the control memory for microinstructions, a nano-instruction memory is included. In such a control unit, microinstructions do not contain encoded control signals. The operation part of microinstructions contains the address of the word in the nano-instruction memory, which contains encoded control signals. The nano-instruction memory contains all combinations of control signals that appear in microprograms that interpret the complete instruction set of a given computer, written once in the form of nano-instructions.

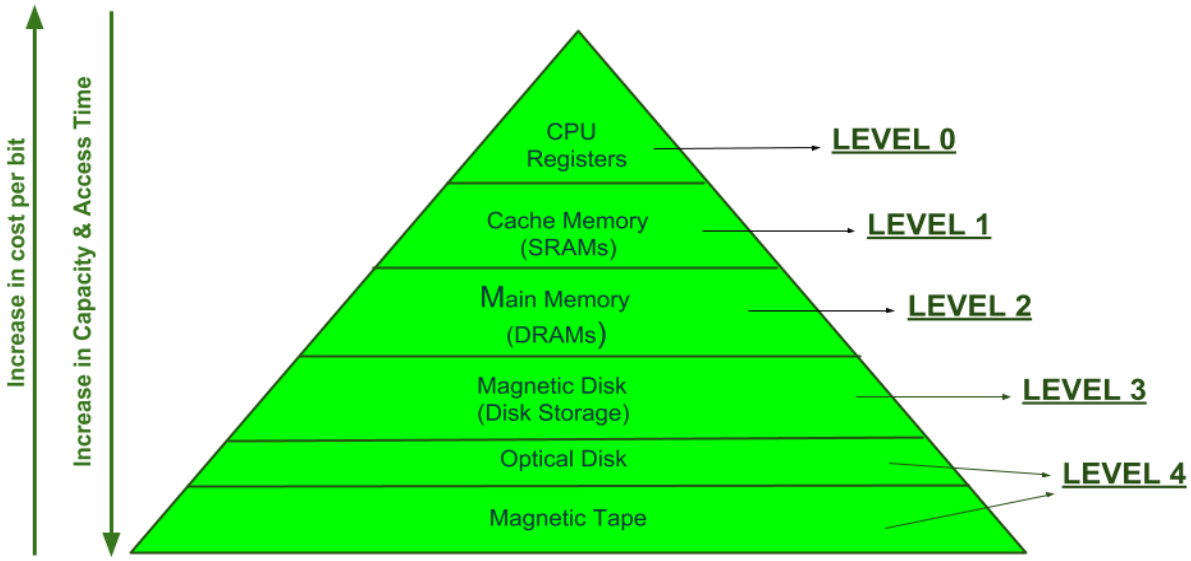


Microprogrammed control unit with a two-level control store

In this way, unnecessary storing of the same operation parts of microinstructions is avoided. In this case, microinstruction word can be much shorter than with the single level control store. It gives a much smaller size in bits of the microinstruction memory and, as a result, a much smaller size of the entire control memory. The microinstruction memory contains the control for selection of consecutive microinstructions, while those control signals are generated at the basis of nano-instructions. In nano-instructions, control signals are frequently encoded using 1 bit/ 1 signal method that eliminates decoding.

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.

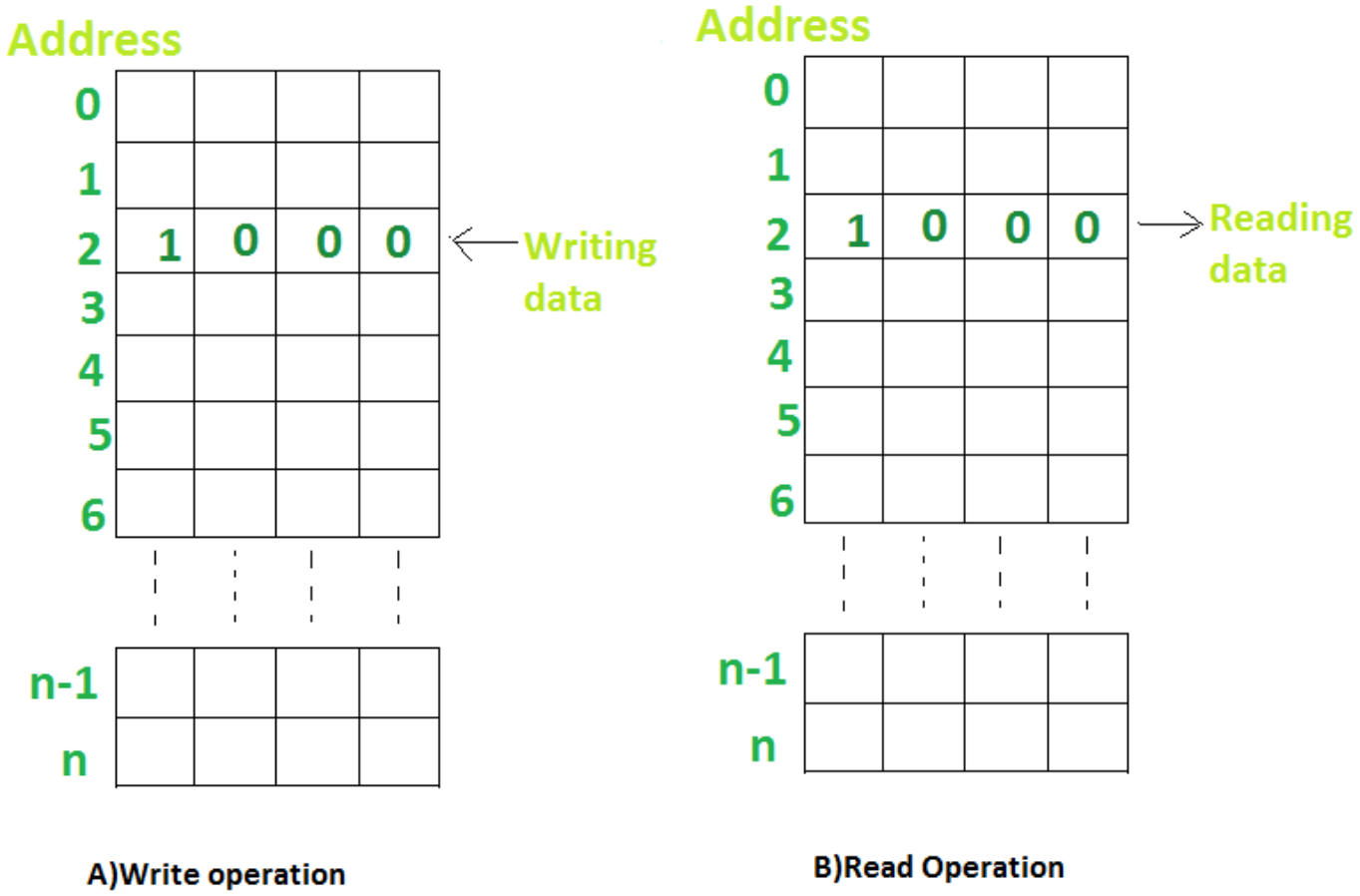
We can infer the following characteristics of Memory Hierarchy Design from above figure:

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.

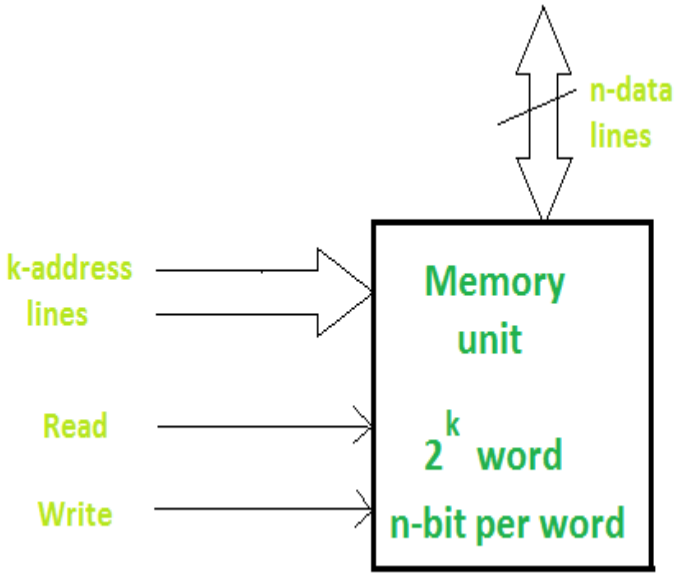
3. **Performance:**
   Earlier when the computer system was designed without Memory Hierarchy design, the speed gap increases between the CPU registers and Main Memory due to large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement 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.

Introduction to memory and memory units

Memories are made up of registers. Each register in the memory is one storage location. Storage location is also called as memory location. Memory locations are identified using **Address**. The total number of bit a memory can store is its **capacity**. A storage element is called a **Cell**. Each register is made up of storage element in which one bit of data is stored. The data in a memory are stored and retrieved by the process called **writing** and **reading** respectively.



A **word** is a group of bits where a memory unit stores binary information. A word with group of 8 bits is called a **byte**.
A memory unit consists of data lines, address selection lines, and control lines that specify the direction of transfer. The block diagram of a memory unit is shown below:

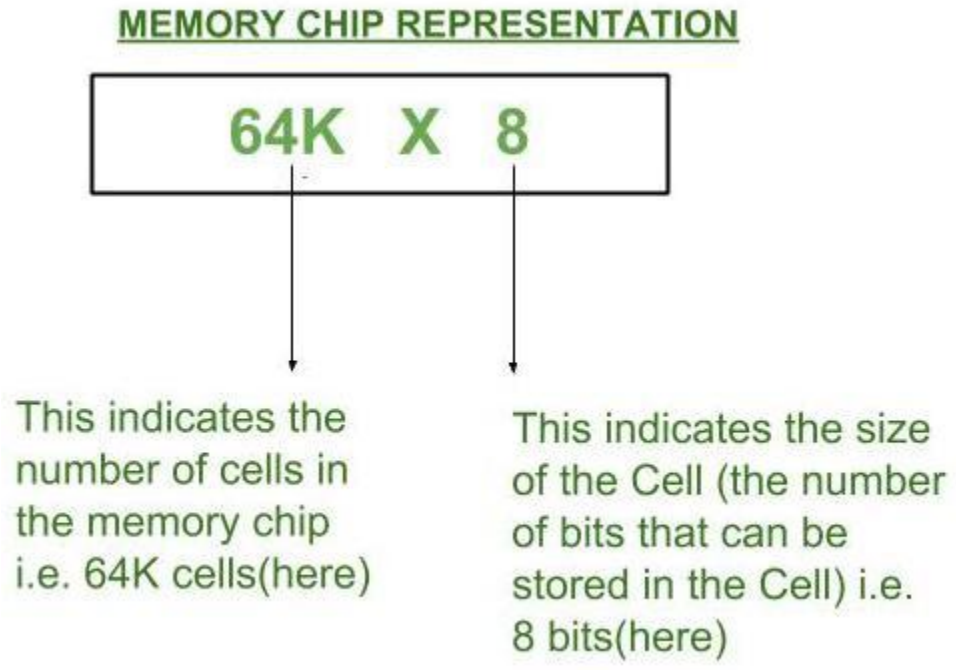


Data lines provide the information to be stored in memory. The control inputs specify the direction transfer. The k-address lines specify the word chosen.

When there are k address lines, 2^k memory word can be accessed.
Refer for RAM and ROM, different types of RAM, cache memory, and secondary memory.

Difference between Byte Addressable Memory and Word Addressable Memory

Memory is a storage component in the Computer used to store application programs. The Memory Chip is divided into equal parts called as *"CELLS"*. Each Cell is uniquely identified by a binary number called as *"ADDRESS"*. For example, the Memory Chip configuration is represented as **'64 K x 8'** as shown in the figure below.



The following information can be obtained from the memory chip representation shown above:

1. Data Space in the Chip = 64K X 8
2. Data Space in the Cell = 8 bits

3. Address Space in the Chip = =16 bits
Now we can clearly state the difference between Byte Addressable Memory & Word Addressable Memory.

BYTE ADDRESSABLE MEMORY	WORD ADDRESSABLE MEMORY
When the ***data space in the cell = 8 bits*** then the corresponding ***address space*** is called as **Byte Address**.	When the ***data space in the cell = word length of CPU*** then the corresponding ***address space*** is called as **Word Address**.
Based on this data storage i.e. *Bytewise storage*, the memory chip configuration is named as **Byte Addressable Memory**.	Based on this data storage i.e. *Wordwise storage*, the memory chip configuration is named as **Word Addressable Memory**.
For eg. : **64K X 8** chip has 16 bit Address and **cell size = 8 bits (1 Byte)** which means that in this chip, data is stored **byte by byte**.	For eg. : For a 16-bit CPU, **64K X 16** chip has 16 bit Address & **cell size = 16 bits (Word Length of CPU)** which means that in this chip, data is stored **word by word**.

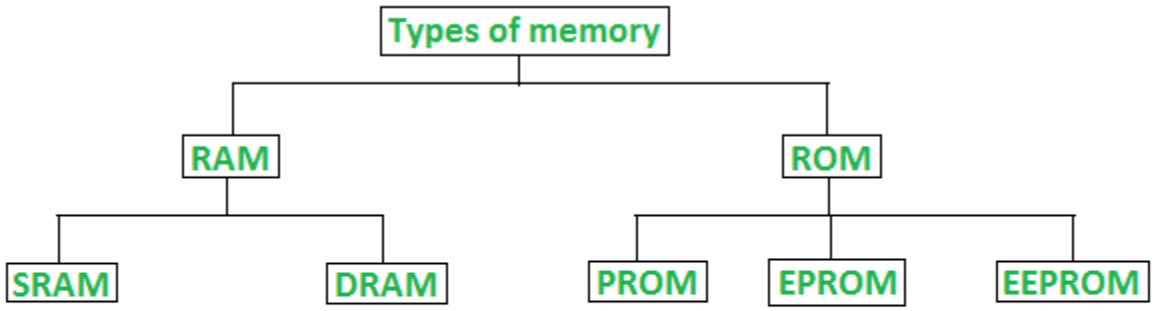
NOTE :

i) The most important point to be noted is that in case of either of Byte Address or Word Address, *the address size* can be any number of bits (depends on the number of cells in the chip) but the *cell size* differs in each case.

ii)The default memory configuration in the Computer design is Byte Addressable .

Random Access Memory (RAM) and Read Only Memory (ROM)

Memory is the most essential element of a computing system because without it computer can't perform simple tasks. Computer memory is of two basic type – Primary memory(RAM and ROM) and Secondary memory(hard drive,CD,etc.). Random Access Memory (RAM) is primary-volatile memory and Read Only Memory (ROM) is primary-non-volatile memory.



Classification of computer memory

1. Random Access Memory (RAM) –

- It is also called as *read write memory* or the *main memory* or the *primary memory*.
- The programs and data that the CPU requires during execution of a program are stored in this memory.
- It is a volatile memory as the data loses when the power is turned off.
- RAM is further classified into two types- *SRAM (Static Random Access Memory)* and *DRAM (Dynamic Random Access Memory)*.

DRAM	SRAM
1. Constructed of tiny capacitors that leak electricity.	1.Constructed of circuits similar to D flip-flops.
2.Requires a recharge every few milliseconds to maintain its data.	2.Holds its contents as long as power is available.
3.Inexpensive.	3.Expensive.
4. Slower than SRAM.	4. Faster than DRAM.
5. Can store many bits per chip.	5. Can not store many bits per chip.
6. Uses less power.	6.Uses more power.
7.Generates less heat.	7.Generates more heat.
8. Used for main memory.	8. Used for cache.

Difference between SRAM and DRAM

2. Read Only Memory (ROM) –

- Stores crucial information essential to operate the system, like the program essential to boot the computer.
- It is not volatile.
- Always retains its data.
- Used in embedded systems or where the programming needs no change.
- Used in calculators and peripheral devices.
- ROM is further classified into 4 types- *ROM*, *PROM*, *EPROM*, and *EEPROM*.

Types of Read Only Memory (ROM) –

1. **PROM (Programmable read-only memory)** – It can be programmed by user. Once programmed, the data and instructions in it cannot be changed.
2. **EPROM (Erasable Programmable read only memory)** – It can be reprogrammed. To erase data from it, expose it to ultra violet light. To reprogram it, erase all the previous data.
3. **EEPROM (Electrically erasable programmable read only memory)** – The data can be erased by applying electric field, no need of ultra violet light. We can erase only portions of the chip.

RAM	ROM
1. Temporary Storage.	1. Permanent storage.
2. Store data in MBs.	2. Store data in GBs.
3. Volatile.	3. Non-volatile.
4.Used in normal operations.	4. Used for startup process of computer.
5. Writing data is faster.	5. Writing data is slower.

Difference between RAM and ROM

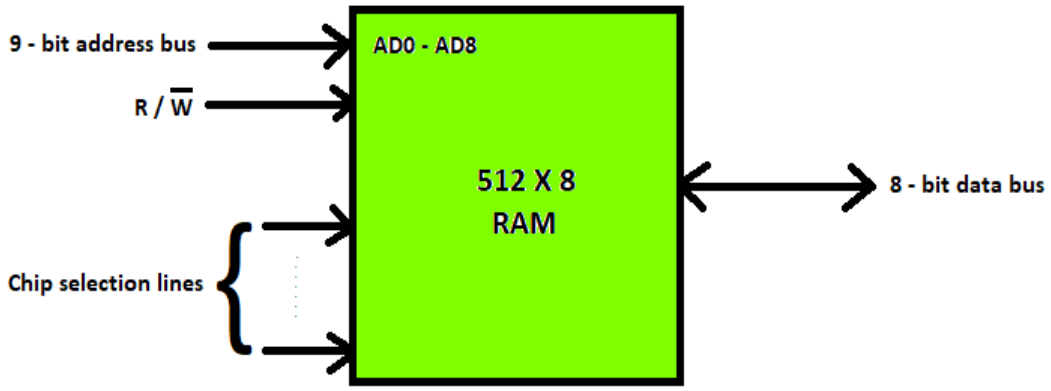
Different Types of RAM (Random Access Memory)

RAM(Random Access Memory) is a part of computer's Main Memory which is directly accessible by CPU. RAM is used to Read and Write data into it which is accessed by CPU randomly. RAM is volatile in nature, it means if the power goes off, the stored information is lost. RAM is used to store the data that is currently processed by the CPU. Most of the programs and data that are modifiable are stored in RAM.

Integrated RAM chips are available in two form:

1. SRAM(Static RAM)
2. DRAM(Dynamic RAM)

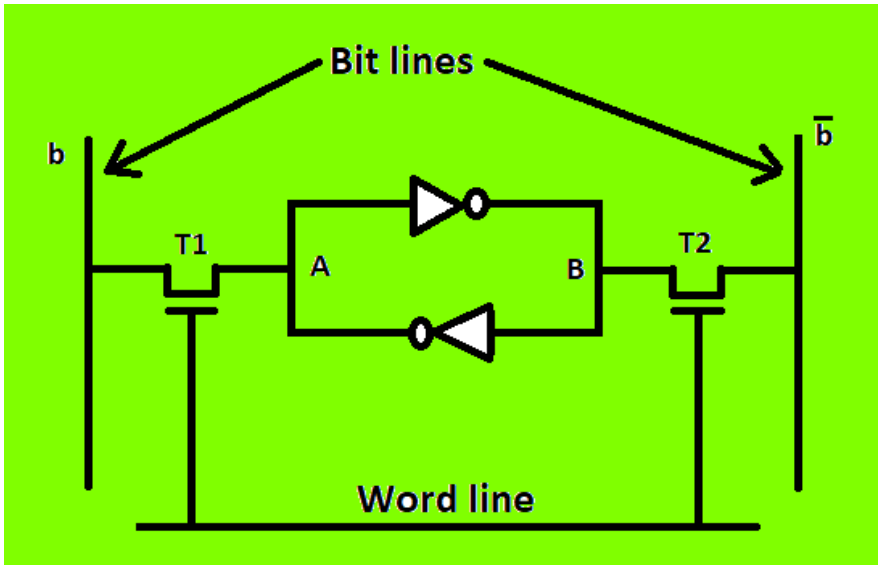
The block diagram of RAM chip is given below.



SRAM

The SRAM memories consist of circuits capable of retaining the stored information as long as the power is applied. That means this type of memory requires constant power. SRAM memories are used to build Cache Memory.

SRAM Memory Cell: Static memories(SRAM) are memories that consist of circuits capable of retaining their state as long as power is on. Thus this type of memories is called volatile memories. The below figure shows a cell diagram of SRAM. A latch is formed by two inverters connected as shown in the figure. Two transistors T1 and T2 are used for connecting the latch with two bit lines. The purpose of these transistors is to act as switches that can be opened or closed under the control of the word line, which is controlled by the address decoder. When the word line is at 0-level, the transistors are turned off and the latch remains its information. For example, the cell is at state 1 if the logic value at point A is 1 and at point B is 0. This state is retained as long as the word line is not activated.



For **Read operation**, the word line is activated by the address input to the address decoder. The activated word line closes both the transistors (switches) T1 and T2. Then the bit values at points A and B can transmit to their respective bit lines. The sense/write circuit at the end of the bit lines sends the output to the processor.

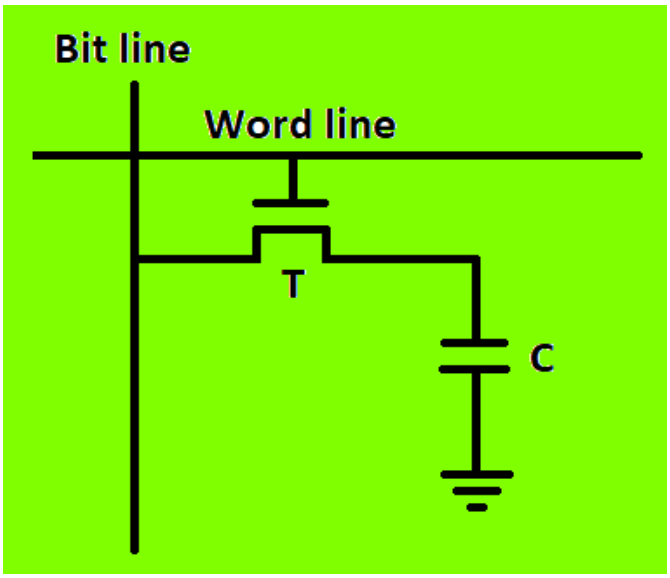
For **Write operation**, the address provided to the decoder activates the word line to close both the switches. Then the bit value that to be written into the cell is provided through the sense/write circuit and the signals in bit lines are then stored in the cell.

DRAM

DRAM stores the binary information in the form of electric charges that applied to capacitors. The stored information on the capacitors tend to lose over a period of time and thus the capacitors must be periodically recharged to retain their usage. The main memory is generally made up of DRAM chips.

DRAM Memory Cell: Though SRAM is very fast, but it is expensive because of its every cell requires several transistors. Relatively less expensive RAM is DRAM, due to the use of one transistor and one capacitor in each cell, as shown in the below figure.,

where C is the capacitor and T is the transistor. Information is stored in a DRAM cell in the form of a charge on a capacitor and this charge needs to be periodically recharged. For storing information in this cell, transistor T is turned on and an appropriate voltage is applied to the bit line. This causes a known amount of charge to be stored in the capacitor. After the transistor is turned off, due to the property of the capacitor, it starts to discharge. Hence, the information stored in the cell can be read correctly only if it is read before the charge on the capacitors drops below some threshold value.



Types of DRAM

There are mainly 5 types of DRAM:

1. **Asynchronous DRAM (ADRAM):** The DRAM described above is the asynchronous type DRAM. The timing of the memory device is controlled asynchronously. A specialized memory controller circuit generates the necessary control signals to control the timing. The CPU must take into account the delay in the response of the memory.
2. **Synchronous DRAM (SDRAM):** These RAM chips' access speed is directly synchronized with the CPU's clock. For this, the memory chips remain ready for operation when the CPU expects them to be ready. These memories operate at the CPU-memory bus without imposing wait states. SDRAM is commercially available as modules incorporating multiple SDRAM chips and forming the required capacity for the modules.
3. **Double-Data-Rate SDRAM (DDR SDRAM):** This faster version of SDRAM performs its operations on both edges of the clock signal; whereas a standard SDRAM performs its operations on the rising edge of the clock signal. Since they transfer data on both edges of the clock, the data transfer rate is doubled. To access the data at high rate, the memory cells are organized into two groups. Each group is accessed separately.
4. **Rambus DRAM (RDRAM):** The RDRAM provides a very high data transfer rate over a narrow CPU-memory bus. It uses various speedup mechanisms, like

synchronous memory interface, caching inside the DRAM chips and very fast signal timing. The Rambus data bus width is 8 or 9 bits.

5. **Cache DRAM (CDRAM):** This memory is a special type DRAM memory with an on-chip cache memory (SRAM) that acts as a high-speed buffer for the main DRAM.

Difference between SRAM and DRAM

Below table lists some of the differences between SRAM and DRAM:

SRAM	DRAM
1. SRAM has lower access time, so it is faster compared to DRAM.	1. DRAM has higher access time, so it is slower than SRAM.
2. SRAM is costlier than DRAM.	2. DRAM costs less compared to SRAM.
3. SRAM requires constant power supply, which means this type of memory consumes more power.	3. DRAM offers reduced power consumption, due to the fact that the information is stored in the capacitor.
4. Due to complex internal circuitry, less storage capacity is available compared to the same physical size of DRAM memory chip.	4. Due to the small internal circuitry in the one-bit memory cell of DRAM, the large storage capacity is available.
5. SRAM has low packaging density.	5. DRAM has high packaging density.

2D and 2.5D Memory organization

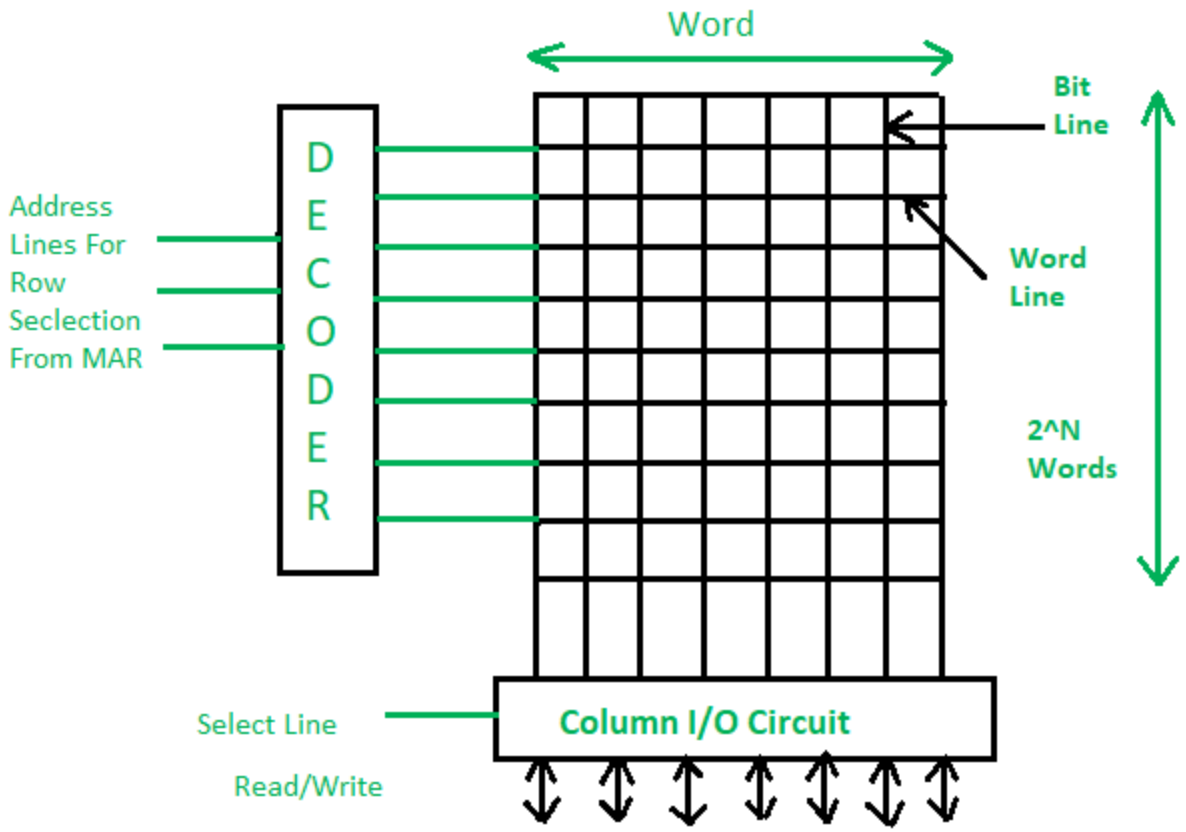
Internal structure of Memory either RAM or ROM is made of memory cells which contains a memory bit. A group of 8 bits makes a word.The memory is formed in multidimensional array of rows and columns. In which each cell stores a bit and a complete row contains a word. A memory simply can be divided in this below form.

$2^n = N$

where,n is the no. of address lines and N is the total memory in bytes.
There will be 2^n words.

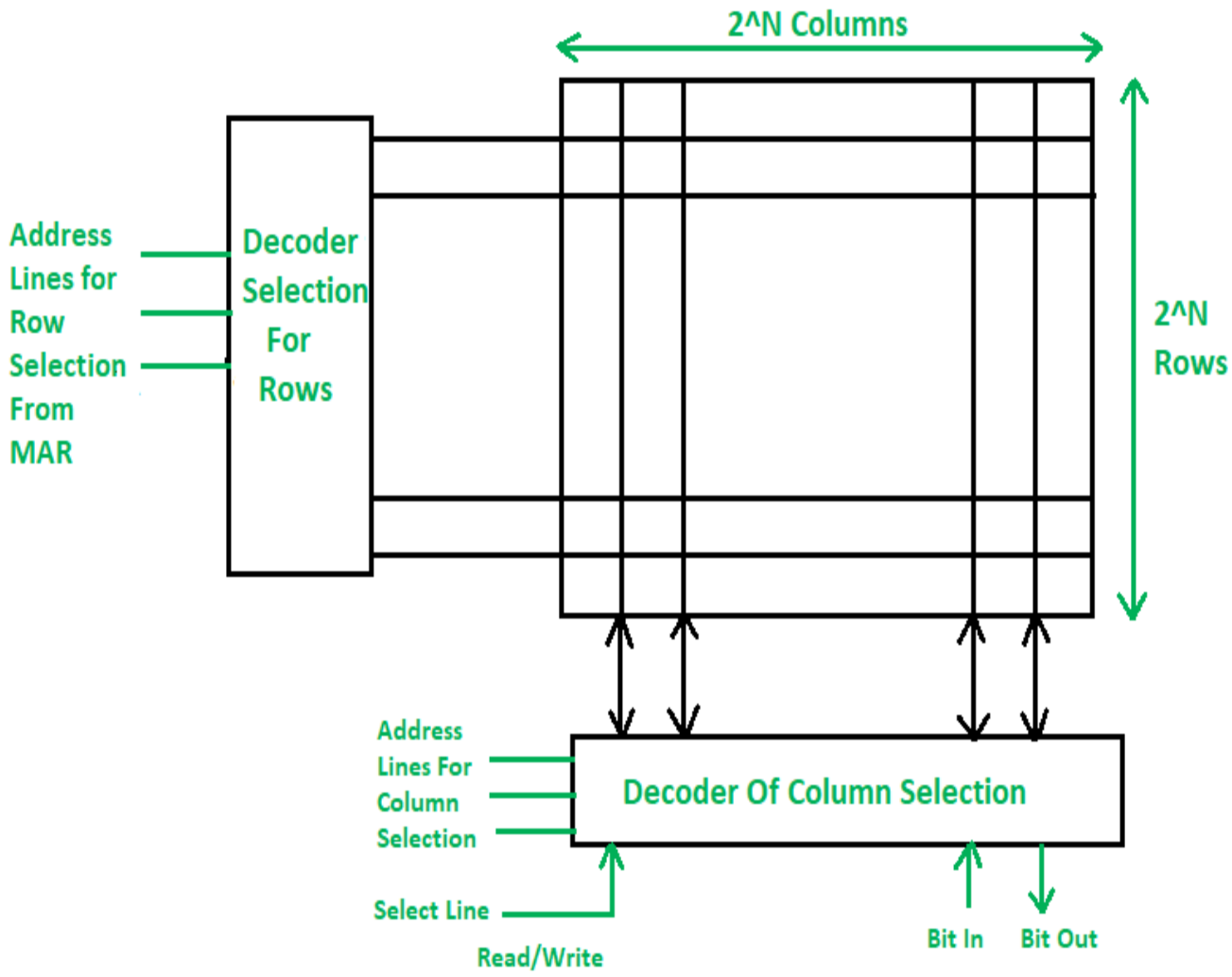
2D Memory organization –

In 2D organization memory is divided in the form of rows and columns(Matrix) . Each row contains a word, now in this memory organization there is a decoder. A decoder is a combinational circuit which contains n input lines and 2^n output lines. One of the output line will select the row which address is contained in the MAR and the word which is represented by that row that will get selected and either read or write through the data lines.



2D Memory Organization

2.5D Memory organization –
In 2.5D Organization the scenario is the same but we have two different decoders one is column decoder and another is row decoder. Column decoder used to select the column and row decoder is used to select the row. Address from the MAR will go in decoders' input. Decoders will select the respective cell through the bit outline, the data from that location will be read or through the bit in line data will be written at that memory location.



2.5D Memory Organization

Read and Write Operations –

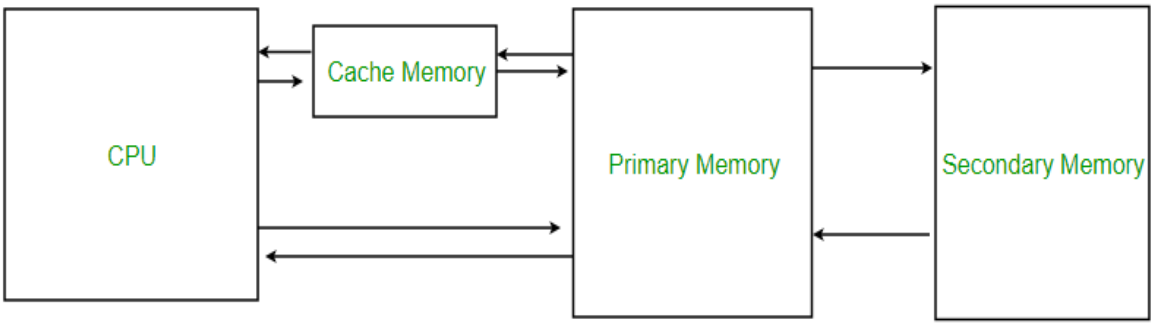
1. If the select line is in Read mode then the Word/bit which is represented by the MAR that will be coming out to the data lines and get read.
2. If the select line is in write mode then the data from memory data register (MDR) will go to the respective cell which is addressed by the memory address register (MAR).
3. With the help of the select line the data will get selected where the read and write operations will take place.

Comparison between 2D & 2.5D Organizations –

1. In 2D organization hardware is fixed but in 2.5D hardware changes.
2. 2D Organization requires more no. of Gates while 2.5D requires less no. of Gates.
3. 2D is more complex in comparison to the 2.5D Organization.
4. Error correction is not possible in the 2D organization but In 2.5D error correction is easy.
5. 2D is more difficult to fabricate in comparison to the 2.5D organization.

Cache Memory

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:

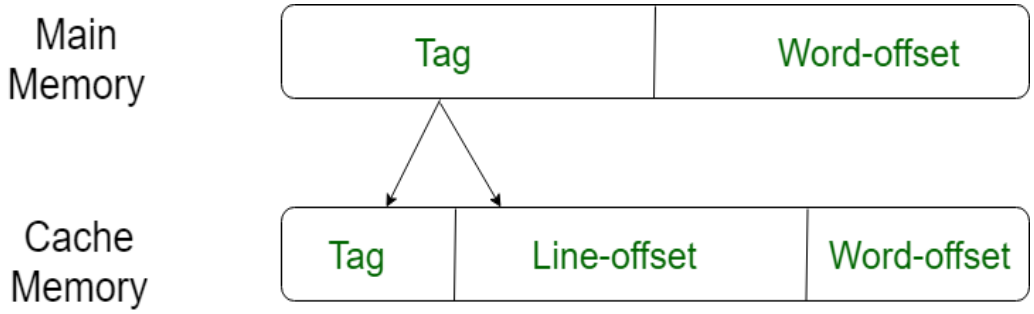
There are three different types of mapping used for the purpose of cache memory which are as follows: Direct mapping, Associative mapping, and Set-Associative mapping. These are explained below.

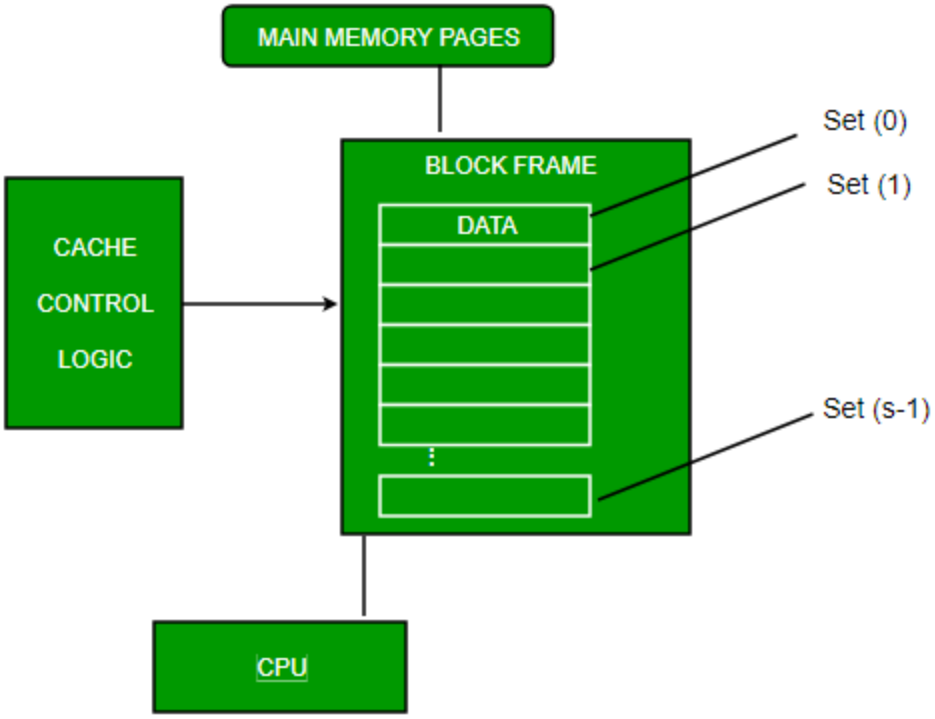
1. **Direct Mapping –**
   The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. or
   In Direct mapping, assigne each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to 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.

```
2. i = j modulo m

3. where

4. i=cache line number

5. j= main memory block number
```

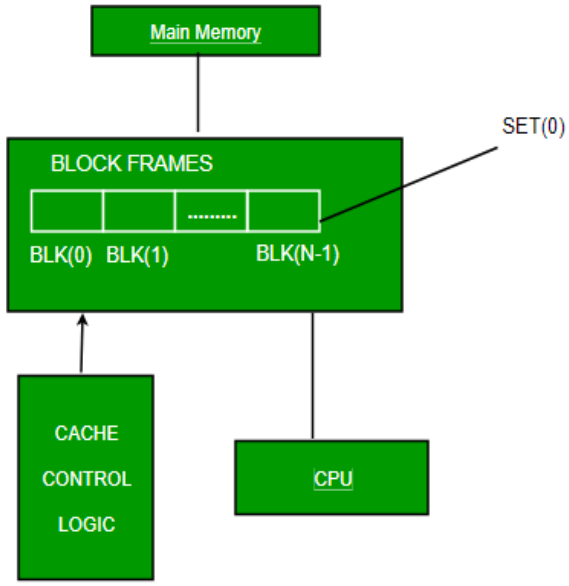
```
m=number of lines in the cache
```

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.



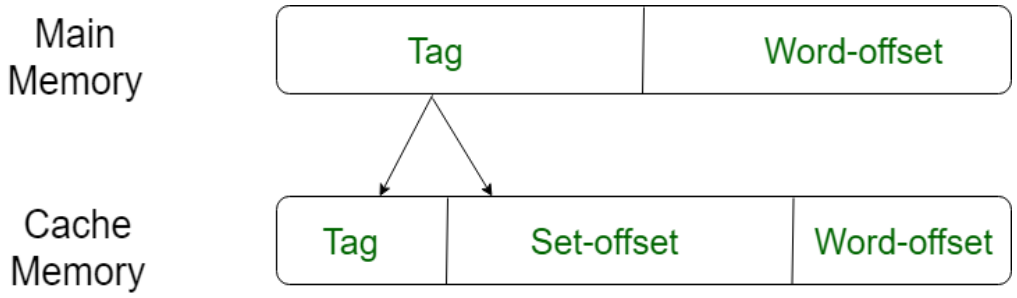


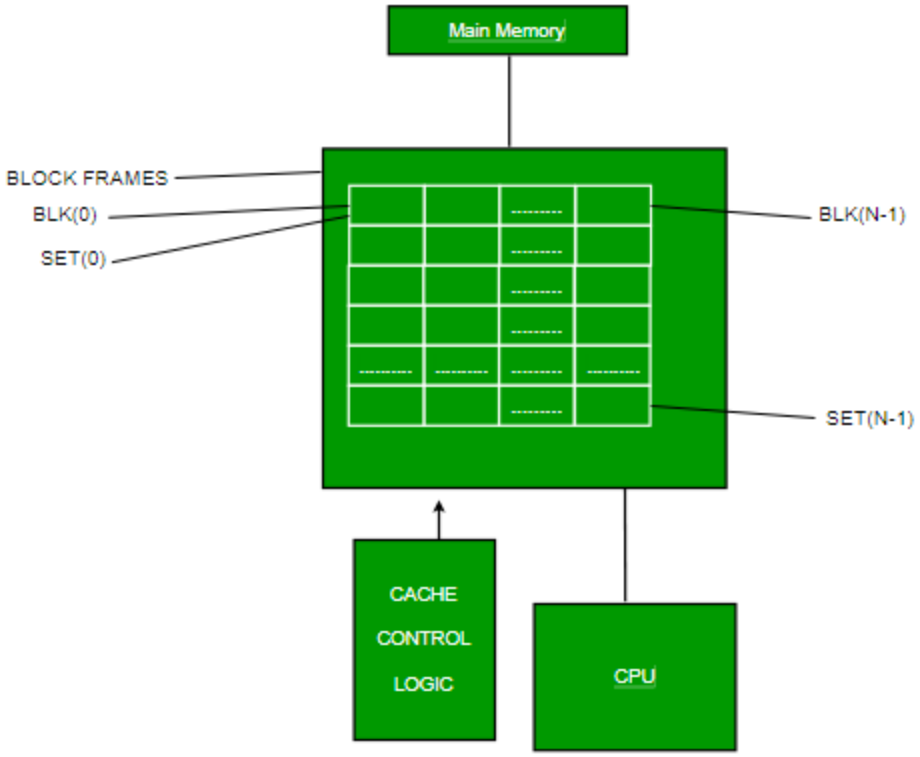
6. **Associative Mapping –**
   In this type of mapping, the associative memory is used to store content and addresses of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the 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.



7. **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

5. **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.

6. **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.

Practice Questions –

Que-1: A computer has a 256 KByte, 4-way set associative, write back data cache with the block size of 32 Bytes. The processor sends 32-bit addresses to the cache controller. Each cache tag directory entry contains, in addition, to address tag, 2 valid bits, 1 modified bit and 1 replacement bit. The number of bits in the tag field of an address is

```
(A) 11

(B) 14

(C) 16

(D) 27
```

Answer: (C)

Que-2: Consider the data given in previous question. The size of the cache tag directory is

```
(A) 160 Kbits

(B) 136 bits

(C) 40 Kbits

(D) 32 bits
```

Answer: (A)

Que-3: An 8KB direct-mapped write-back cache is organized as multiple blocks, each of size 32-bytes. The processor generates 32-bit addresses. The cache controller maintains the tag information for each cache block comprising of the following.

```
1 Valid bit

1 Modified bit
```

As many bits as the minimum needed to identify the memory block mapped in the cache. What is the total size of memory needed at the cache controller to store meta-data (tags) for the cache?

```
(A) 4864 bits

(B) 6144 bits
```

```
(C) 6656 bits
(D) 5376 bits
```

Answer: (D)

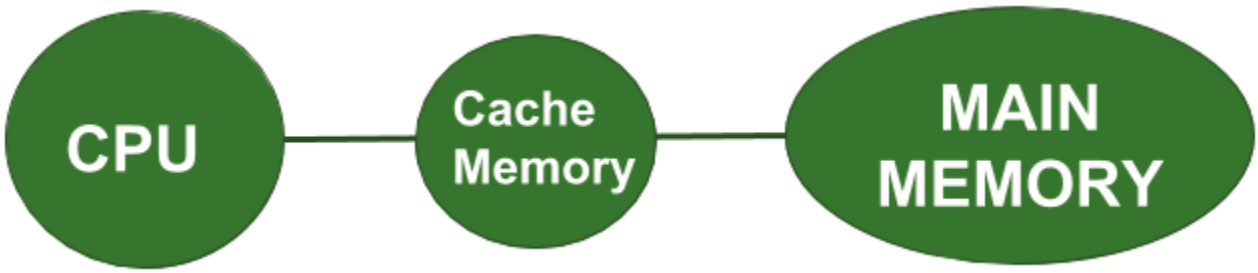
Multilevel Cache Organization

Cache is a random access memory used by the CPU to reduce the average time taken to access memory.
Multilevel Caches is one of the techniques to improve Cache Performance by reducing the *"MISS PENALTY"*. Miss Penalty refers to the extra time required to bring the data into cache from the Main memory whenever there is a ***"miss"*** in cache .
For clear understanding let us consider an example where CPU requires 10 Memory References for accessing the desired information and consider this scenario in the following 3 cases of System design :

Case 1 : System Design without Cache Memory

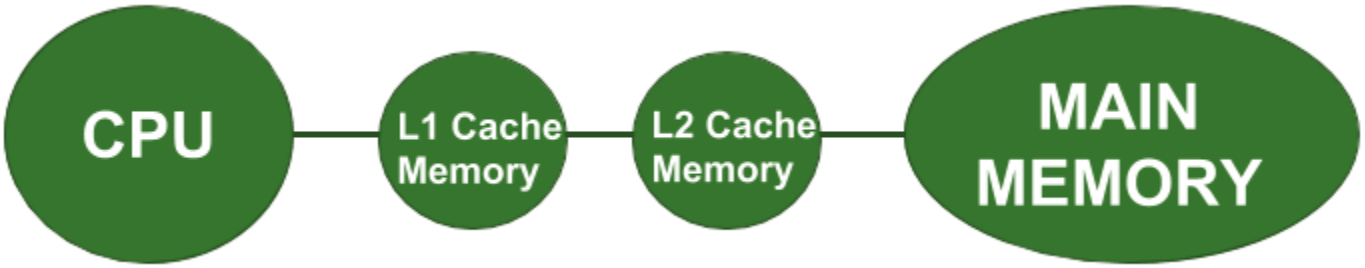
Here the CPU directly communicates with the main memory and no caches are involved.
In this case, the CPU needs to access the main memory 10 times to access the desired information.

Case 2 : System Design with Cache Memory



Here the CPU at first checks whether the desired data is present in the Cache Memory or not i.e. whether there is a ***"hit"*** in cache or ***"miss"*** in cache. Suppose there are *3 miss* in Cache Memory then the Main Memory will be accessed only 3 times. We can see that here the miss penalty is reduced because the Main Memory is accessed a lesser number of times than that in the previous case.

Case 3 : System Design with Multilevel Cache Memory



Here the Cache performance is optimized further by introducing multilevel Caches. As shown in the above figure, we are considering 2 level Cache Design. Suppose there are *3 miss* in the L1 Cache Memory and out of these 3 misses there are *2 miss* in the L2 Cache Memory then the Main Memory will be accessed only 2 times. It is clear that here the Miss Penalty is reduced considerably than that in the previous case thereby improving the Performance of Cache Memory.

NOTE :

We can observe from the above 3 cases that we are trying to decrease the number of Main Memory References and thus decreasing the Miss Penalty in order to improve the overall System Performance. Also, it is important to note that in the Multilevel Cache Design, L1 Cache is attached to the CPU and it is small in size but fast. Although, L2 Cache is attached to the Primary Cache i.e. L1 Cache and it is larger in size and slower but still faster than the Main Memory.

```
Effective Access Time = Hit rate * Cache access time
                      + Miss rate * Lower level access time
```

Average access Time For Multilevel Cache:(T_{avg})

$T_{avg} = H_1 * C_1 + (1 – H_1) * (H_2 * C_2 + (1 – H_2) * M)$

where
H1 is the Hit rate in the L1 caches.
H2 is the Hit rate in the L2 cache.
C1 is the Time to access information in the L1 caches.
C2 is the Miss penalty to transfer information from the L2 cache to an L1 cache.
M is the Miss penalty to transfer information from the main memory to the L2 cache.

Example:
Find the Average memory access time for a processor with a 2 ns clock cycle time, a miss rate of 0.04 misses per instruction, a miss penalty of 25 clock cycles, and a cache access time (including hit detection) of 1 clock cycle. Also, assume that the read and write miss penalties are the same and ignore other write stalls.
Solution:
Average Memory access time(AMAT)= Hit Time + Miss Rate * Miss Penalty.

Hit Time = 1 clock cycle (Hit time = Hit rate * access time) but here Hit time is directly given so,

Miss rate = 0.04

Miss Penalty= 25 clock cycle (this is time taken by the above level of memory after the hit)

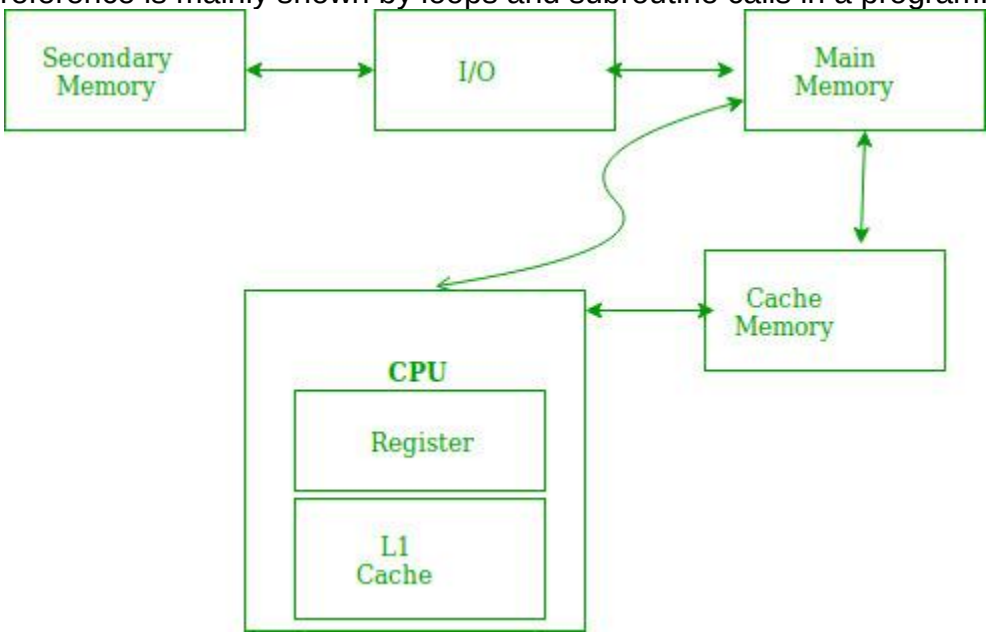
so, AMAT= 1 + 0.04 * 25
AMAT= 2 clock cycle

according to question 1 clock cycle = 2 ns

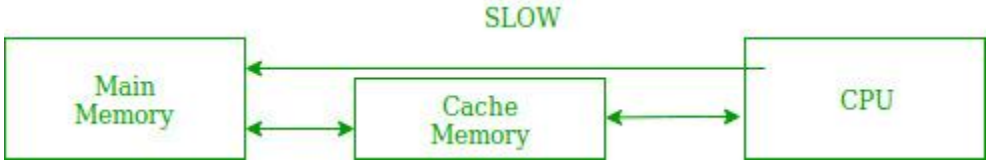
AMAT = 4ns

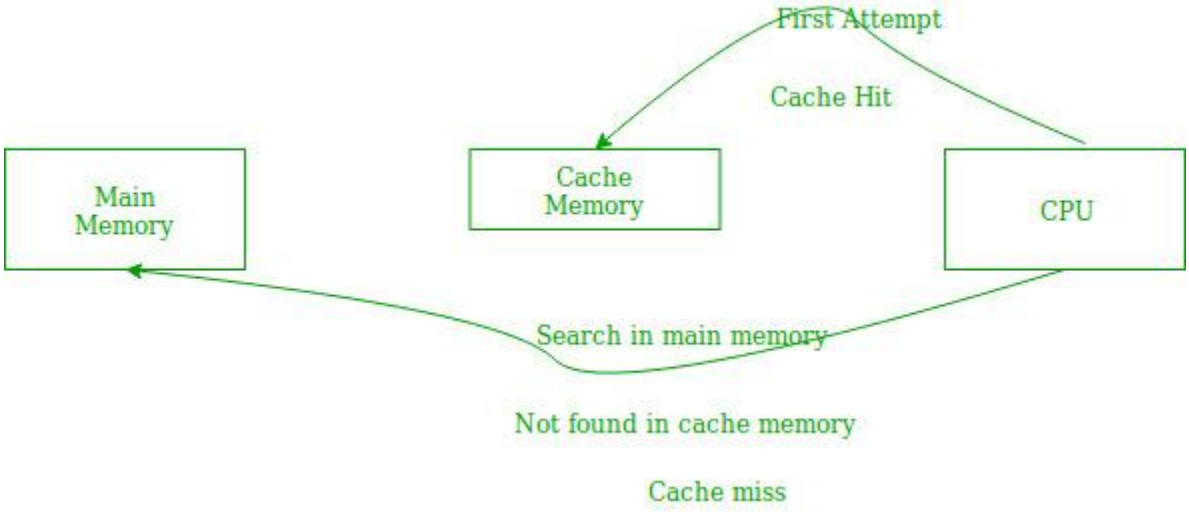
Locality of Reference and Cache Operation in Cache Memory

Locality of reference refers to a phenomenon in which a computer program tends to access same set of memory locations for a particular time period. In other words, **Locality of Reference** refers to the tendency of the computer program to access instructions whose addresses are near one another. The property of locality of reference is mainly shown by loops and subroutine calls in a program.



1. In case of loops in program control processing unit repeatedly refers to the set of instructions that constitute the loop.
2. In case of subroutine calls, every time the set of instructions are fetched from memory.
3. References to data items also get localized that means same data item is referenced again and again.





In the above figure, you can see that the CPU wants to read or fetch the data or instruction. First, it will access the cache memory as it is near to it and provides very fast access. If the required data or instruction is found, it will be fetched. This situation is known as a cache hit. But if the required data or instruction is not found in the cache memory then this situation is known as a cache miss. Now the main memory will be searched for the required data or instruction that was being searched and if found will go through one of the two ways:

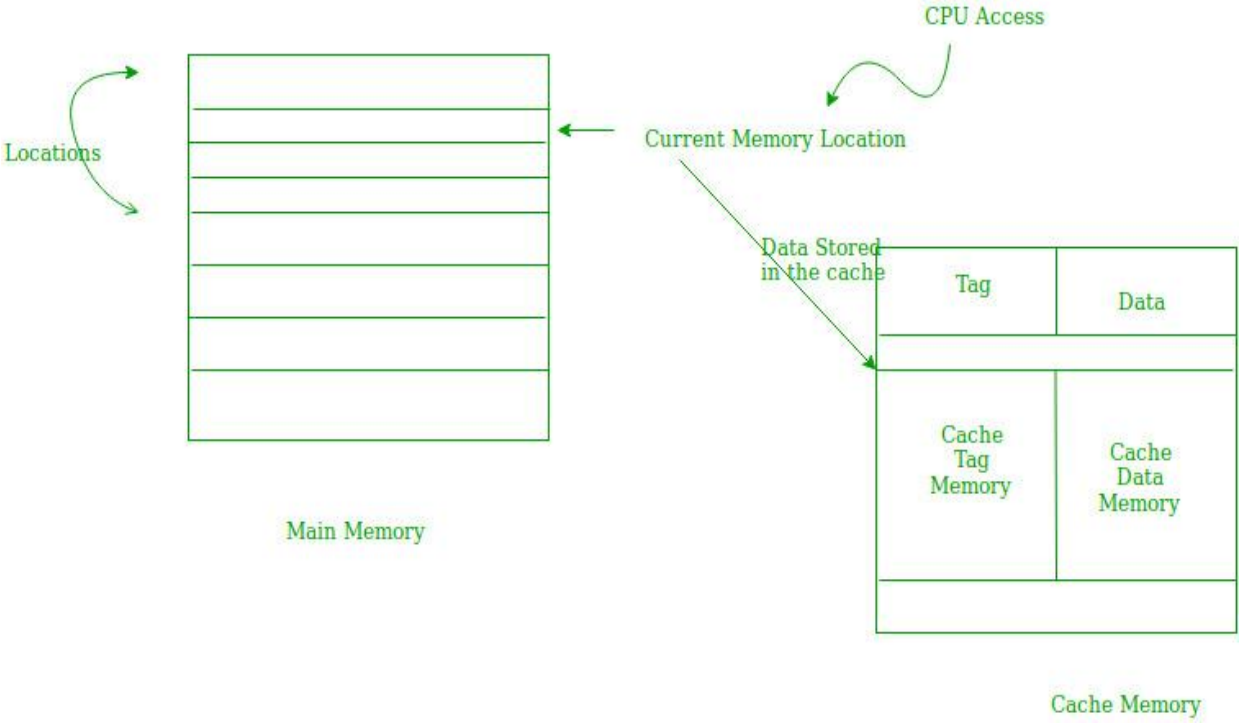
1. First way is that the CPU should fetch the required data or instruction and use it and that's it but what, when the same data or instruction is required again.CPU again has to access the same main memory location for it and we already know that main memory is the slowest to access.
2. The second way is to store the data or instruction in the cache memory so that if it is needed soon again in the near future it could be fetched in a much faster way.

Cache Operation:

It is based on the principle of locality of reference. There are two ways with which data or instruction is fetched from main memory and get stored in cache memory. These two ways are the following:

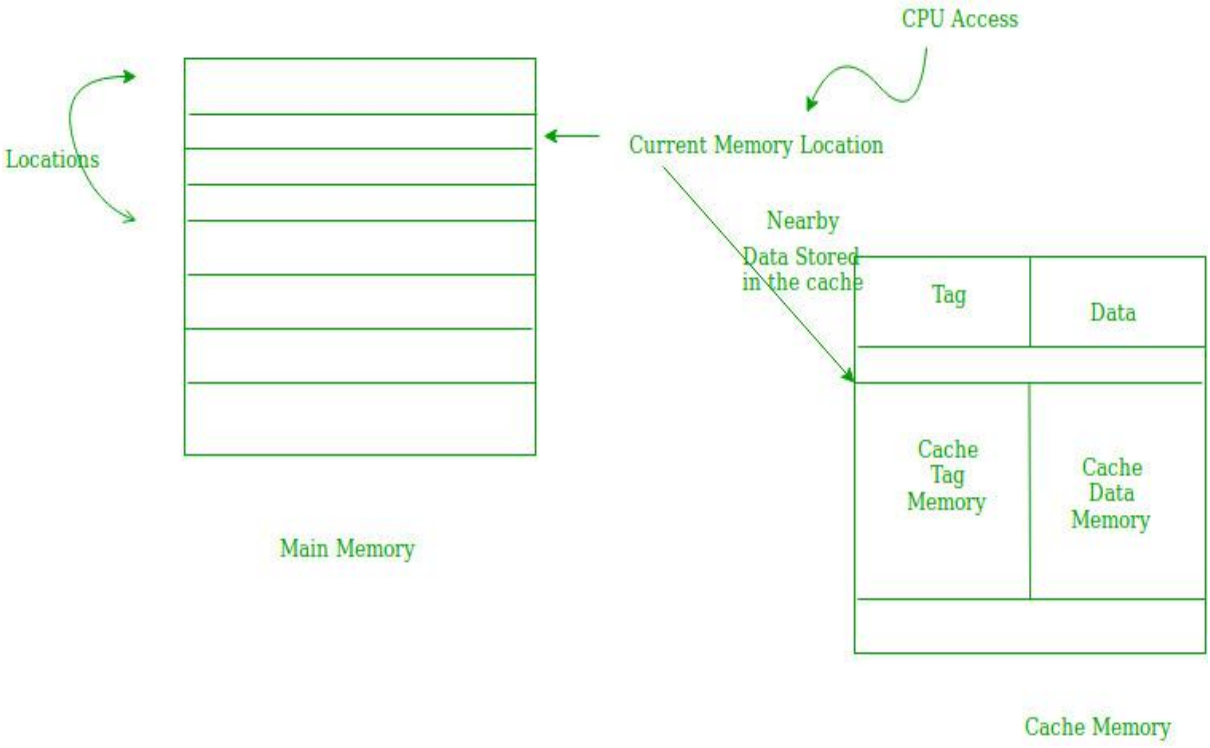
1. **Temporal Locality –**
   Temporal locality means current data or instruction that is being fetched may be needed

soon. So we should store that data or instruction in the cache memory so that we can avoid again searching in main memory for the same data.



When CPU accesses the current main memory location for reading required data or instruction, it also gets stored in the cache memory which is based on the fact that same data or instruction may be needed in near future. This is known as temporal locality. If some data is referenced, then there is a high probability that it will be referenced again in the near future.

2. **Spatial Locality –**
   Spatial locality means instruction or data near to the current memory location that is being fetched, may be needed soon in the near future. This is slightly different from the temporal locality. Here we are talking about nearly located memory locations while in temporal locality we were talking about the actual memory location that was being fetched.



Cache Performance:
The performance of the cache is measured in terms of hit ratio. When CPU refers to memory and find the data or instruction within the Cache Memory, it is known as cache hit. If the desired data or instruction is not found in the cache memory and CPU refers to the main memory to find that data or instruction, it is known as a cache miss.

```
Hit + Miss  = Total CPU Reference

Hit Ratio(h) = Hit / (Hit+Miss)
```

Average access time of any memory system consists of two levels: Cache and Main Memory. If *Tc* is time to access cache memory and *Tm* is the time to access main memory then we can write:

```
Tavg = Average time to access memory

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