UNIT-5

Memory Hierarchy

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 the memory hierarchy.

Why Memory Hierarchy is Required in the System?

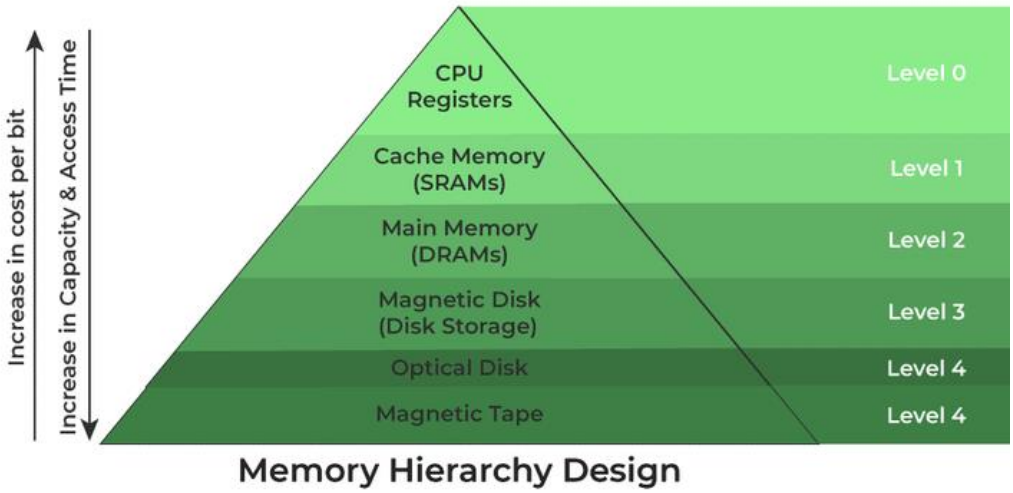
Memory Hierarchy is one of the most required things in Computer Memory as it helps in optimizing the memory available in the computer. There are multiple levels present in the memory, each one having a different size, different cost, etc. Some types of memory like cache, and main memory are faster as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

Types of Memory Hierarchy

This Memory Hierarchy Design is divided into 2 main types:

External Memory or Secondary Memory:Comprising of Magnetic Disk, Optical Disk, and Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via an I/O Module.

Internal Memory or Primary Memory:Comprising of Main Memory, Cache Memory & **CPU registers**. This is directly accessible by the processor.



Memory Hierarchy Design

Memory Hierarchy Design

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

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

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

Types of Main Memory

Static RAM:Static RAM stores the binary information in flip flops and information remains valid until power is supplied. It has a faster access time and is used in implementing cache memory.

Dynamic RAM:It stores the binary information as a charge on the capacitor. It requires refreshing circuitry to maintain the charge on the capacitors after a few milliseconds. It contains more memory cells per unit area as compared to SRAM.

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

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

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

Typically, a memory unit can be classified into two categories:

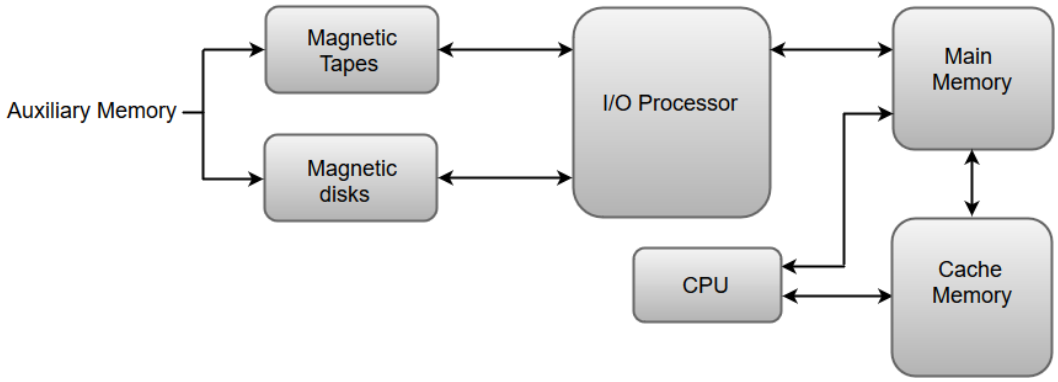
1.The memory unit that establishes direct communication with the CPU is called **Main Memory**. The main memory is often referred to as RAM (Random Access Memory).

2.The memory units that provide backup storage are called **Auxiliary Memory**. For instance, magnetic disks and magnetic tapes are the most commonly used auxiliary memories.

Apart from the basic classifications of a memory unit, the memory hierarchy consists all of the storage devices available in a computer system ranging from the slow but high-capacity auxiliary memory to relatively faster main memory.

The following image illustrates the components in a typical memory hierarchy.

Memory Hierarchy in a Computer System:



Auxiliary Memory

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

A magnetic disk is a digital computer memory that uses a magnetization process to write, rewrite and access data. For example, hard drives, zip disks, and floppy disks.

Magnetic tape is a storage medium that allows for data archiving, collection, and backup for different kinds of data.

Main Memory

The main memory in a computer system is often referred to as **Random Access Memory (RAM)**. This memory unit communicates directly with the CPU and with auxiliary memory devices through an I/O processor.

The programs that are not currently required in the main memory are transferred into auxiliary memory to provide space for currently used programs and data.

I/O Processor

The primary function of an I/O Processor is to manage the data transfers between auxiliary memories and the main memory.

Cache Memory

The data or contents of the main memory that are used frequently by CPU are stored in the cache memory so that the processor can easily access that data in a shorter time. Whenever the CPU requires accessing memory, it first checks the required data into the cache memory. If

the data is found in the cache memory, it is read from the fast memory. Otherwise, the CPU moves onto the main memory for the required data.

Characteristics of Memory Hierarchy

Capacity:It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.

Access Time:It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.

Performance:Earlier when the computer system was designed without a Memory Hierarchy design, the speed gap increased between the CPU registers and Main Memory due to a large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.

Cost Per Bit:As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

Advantages of Memory Hierarchy

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

System-Supported Memory Standards

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

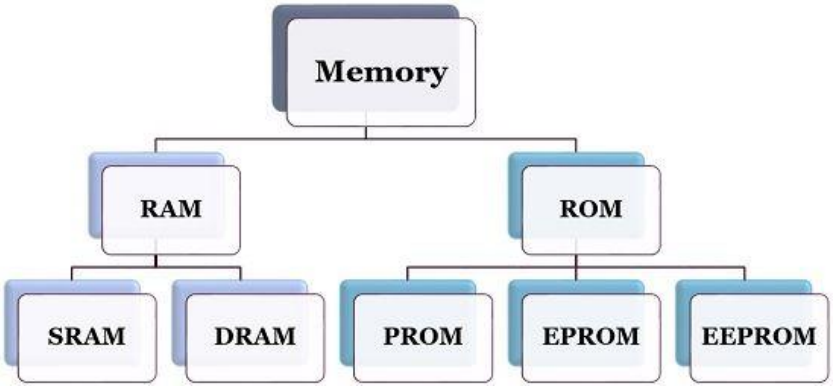
Level	**1**	**2**	**3**	**4**
Name	Register	Cache	Main Memory	Secondary Memory
Size	<1 KB	less than 16 MB	<16GB	>100 GB
Implementation	Multi-ports	On-chip/SRAM	DRAM (capacitor	Magnetic

			memory)	
Access Time	0.25ns to 0.5ns	0.5 to 25ns	80ns to 250ns	50 lakh ns
Bandwidth	20000 to 1 lakh MB	5000 to 15000	1000 to 5000	20 to 150
Managed by	Compiler	Hardware	Operating System	Operating System
Backing Mechanism	From cache	from Main Memory	from Secondary Memory	from ie

RAM and ROM

Primary Memory

Primary Memory is a type of Computer Memory which is directly accessed by the Preprocessor. It is basically used to store data on which computer is currently working. It has lesser storage than Secondary Memory. Primary Memory is classified into two types: RAM and ROM.



Random Access Memory

Random Access Memory (RAM) is used to store the programs and data being used by the CPU in real time. The data on the random access memory can be read, written, and erased any number of times. RAM is a hardware element where the data currently used is stored. It is a volatile memory. It is also called as Main Memory or Primary Memory. This is users memory. The software(program) as well as data files are stored on the hard dish when the software or those files are opened. They get expanded into RAM. It id a users space where the temporary data are automatically stored fill the user saves it into the secondary storage devices.

Types of RAM

Static RAM: Static RAM or SRAM stores a bit of data using the state of a six-transistor memory cell.

Dynamic RAM: Dynamic RAM or DRAM stores a bit of data using a pair of transistors and capacitors which constitute a DRAM memory cell.

Read Only Memory

Read Only Memory (ROM) is a type of memory where the data has been prerecorded. Data stored in ROM is retained even after the computer is turned off ie, non-volatile. It is generally used in Embedded Parts, where the programming requires almost no changes. It is also called as Secondary Memory. It is a permanent CNO4 erasable memory gets initiated when the power is supplied to the computer ROM is a memory chip fixed on the motherboard at the time of manufacturing. It stores a program called BIOS(Basic Input Output Setup). This program checks the status of all the devices attached to the computer.

Types of ROM

Programmable ROM: It is a type of ROM where the data is written after the memory chip has been created. It is non-volatile.

Erasable Programmable ROM: It is a type of ROM where the data on this non-volatile memory chip can be erased by exposing it to high-intensity UV light.

Electrically Erasable Programmable ROM: It is a type of ROM where the data on this non-volatile memory chip can be electrically erased using field electron emission.

Mask ROM: It is a type of ROM in which the data is written during the manufacturing of the memory chip.

Difference between RAM and ROM

Difference	Random Access Memory (RAM)	Read Only Memory (ROM)
Data-Retention	RAM is a volatile memory that could store the data as long as the power is supplied.	ROM is a non-volatile memory that the could retain the data even when the power is turned off.
Read/Write	Read and write operations are supported.	Only read operations are supported.
Use	Used to store the data that has to be currently processed by CPU temporarily.	It is typically used to store firmware or microcode, which is used to initialize and control hardware components of the computer.
Speed	It is a high-speed memory.	It is much slower than the RAM.
CPU Interaction	CPU can easily access data stored in RAM.	CPU cannot easily access data stored in ROM.
Size and Capacity	Large size with higher capacity, concerning ROM.	Small size with less capacity, concerning RAM.
Used as/in	**CPU Cache**,Primary memory.	Firmware, Micro-controllers.
Accessibility	The data stored is easily accessible.	The data stored is not as easily accessible as in the concerning RAM.
Cost	RAM is more costlier than ROM.	ROM is cheaper than RAM.
Chip Size	A RAM chip can store only a few gigabytes (GB) of data.	A ROM chip can store multiple megabytes (MB) of data.
Function	Used for the temporary storage of data currently being processed by the CPU.	Used to store firmware, BIOS, and other data that needs to be retained.

Associative Memory

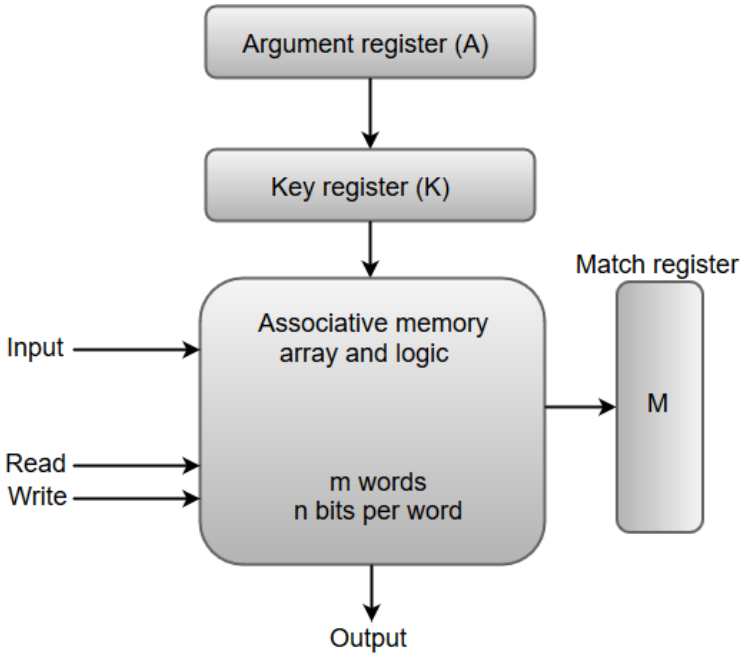
An associative memory can be considered as a memory unit whose stored data can be identified for access by the content of the data itself rather than by an address or memory location.

Associative memory is often referred to as **Content Addressable Memory (CAM)**.

When a write operation is performed on associative memory, no address or memory location is given to the word. The memory itself is capable of finding an empty unused location to store the word.

On the other hand, when the word is to be read from an associative memory, the content of the word, or part of the word, is specified. The words which match the specified content are located by the memory and are marked for reading.

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



From the block diagram, we can say that an associative memory consists of a memory array and logic for 'm' words with 'n' bits per word.

The functional registers like the argument register **A** and key register **K** each have **n** bits, one for each bit of a word. The match register **M** consists of **m** bits, one for each memory word.

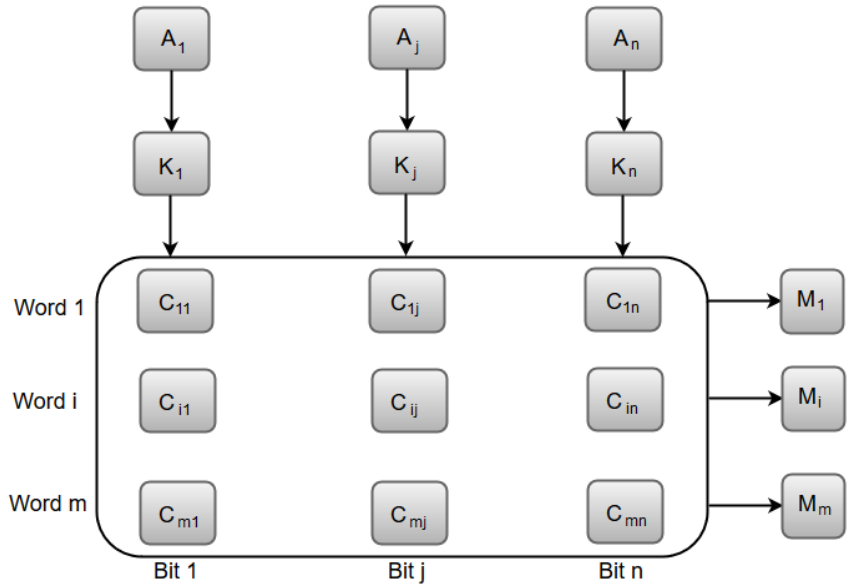
The words which are kept in the memory are compared in parallel with the content of the argument register.

The key register (K) provides a mask for choosing a particular field or key in the argument word. If the key register contains a binary value of all 1's, then the entire argument is compared with each memory word. Otherwise, only those bits in the argument that have 1's in their

corresponding position of the key register are compared. Thus, the key provides a mask for identifying a piece of information which specifies how the reference to memory is made.

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

Associative memory of m word, n cells per word:



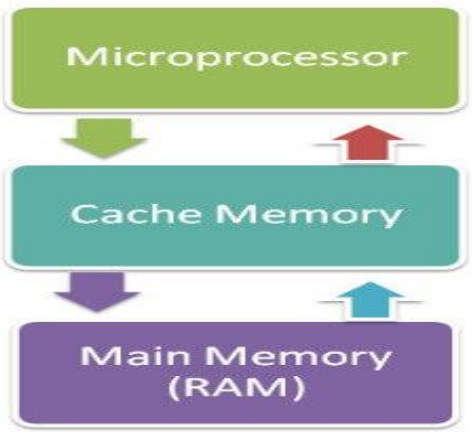
The cells present inside the memory array are marked by the letter C with two subscripts. The first subscript gives the word number and the second specifies the bit position in the word. For instance, the cell C_{ij} is the cell for bit **j** in word **i**.

A bit A_j in the argument register is compared with all the bits in column **j** of the array provided that $K_j = 1$. This process is done for all columns **j** = 1, 2, 3....., n.

If a match occurs between all the unmasked bits of the argument and the bits in word **i**, the corresponding bit M_i in the match register is set to 1. If one or more unmasked bits of the argument and the word do not match, M_i is cleared to 0.

Cache Memory

Cache memory is a high-speed memory, which is small in size but faster than the main memory (RAM). The CPU can access it more quickly than the primary memory. So, it is used to synchronize with high-speed CPU and to improve its performance.



Cache memory can only be accessed by CPU. It can be a reserved part of the main memory or a storage device outside the CPU. It holds the data and programs which are frequently used by the CPU. So, it makes sure that the data is instantly available for CPU whenever the CPU needs this data. In other words, if the CPU finds the required data or instructions in the cache memory, it doesn't need to access the primary memory (RAM). Thus, by acting as a buffer between RAM and CPU, it speeds up the system performance.

Types of Cache Memory:

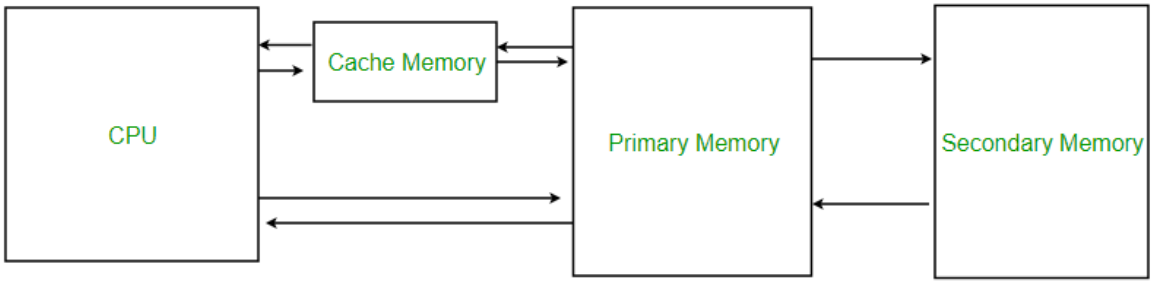
L1: It is the first level of cache memory, which is called Level 1 cache or L1 cache. In this type of cache memory, a small amount of memory is present inside the CPU itself. If a CPU has four cores (quad core cpu), then each core will have its own level 1 cache. As this memory is present in the CPU, it can work at the same speed as of the CPU. The size of this memory ranges from 2KB to 64 KB. The L1 cache further has two types of caches: Instruction cache, which stores instructions required by the CPU, and the data cache that stores the data required by the CPU.

L2: This cache is known as Level 2 cache or L2 cache. This level 2 cache may be inside the CPU or outside the CPU. All the cores of a CPU can have their own separate level 2 cache, or they can share one L2 cache among themselves. In case it is outside the CPU, it is connected with the CPU with a very high-speed bus. The memory size of this cache is in the range of 256 KB to the 512 KB. In terms of speed, they are slower than the L1 cache.

L3: It is known as Level 3 cache or L3 cache. This cache is not present in all the processors; some high-end processors may have this type of cache. This cache is used to enhance the performance of Level 1 and Level 2 cache. It is located outside the CPU and is shared by all the cores of a CPU. Its memory size ranges from 1 MB to 8 MB. Although it is slower than L1 and L2 cache, it is faster than Random Access Memory (RAM).

Characteristics of Cache Memory

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



Cache Performance

When the processor needs to read or write a location in the 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 the 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 the 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(H) = hit / (hit + miss) = no. of hits/total accesses

Miss Ratio = miss / (hit + miss) = no. of miss/total accesses = 1 - hit ratio(H)

We can improve Cache performance using higher cache block size, and higher associativity, reduce miss rate, reduce miss penalty, and 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 is as follows:

- Direct Mapping
- Associative Mapping
- Set-Associative Mapping

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, assign each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and a tag

field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.

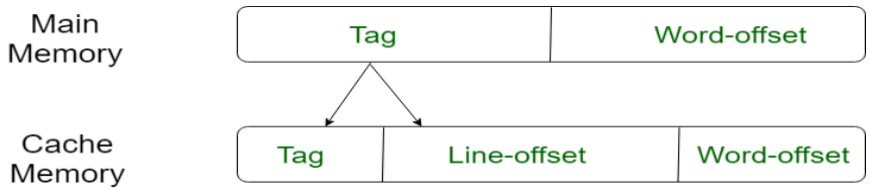
i = j modulo m

where

i = cache line number

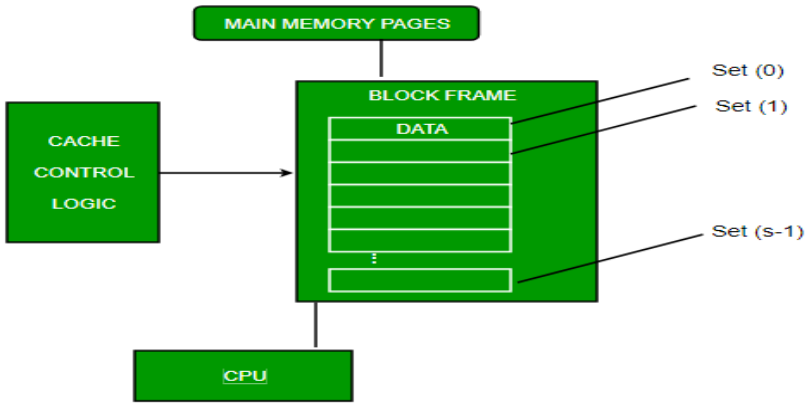
j = main memory block number

m = number of lines in the cache



Direct Mapping

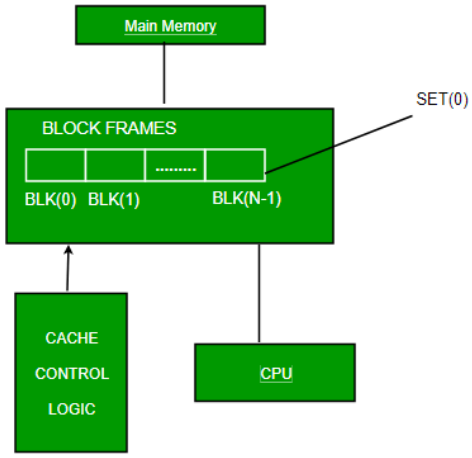
For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant w bits identify a unique word or byte within a block of main memory. In most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2^s blocks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



2. Associative Mapping

In this type of mapping, associative memory is used to store the 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

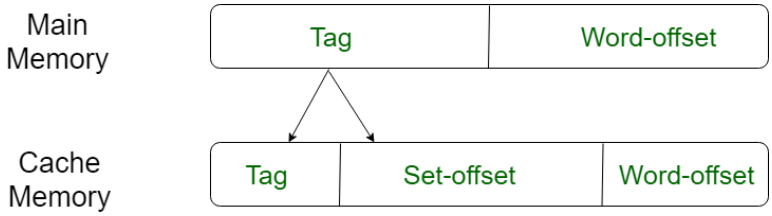
considered to be the fastest and most flexible mapping form. In associative mapping, the index bits are zero.



Associative Mapping – Structure

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 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 set associative mapping the index bits are given by the set offset bits. In this case, the cache consists of a number of sets, each of which consists of a number of lines.



Set-Associative Mapping

Relationships in the Set-Associative Mapping can be defined as:

m = v * k

i= j mod v

where

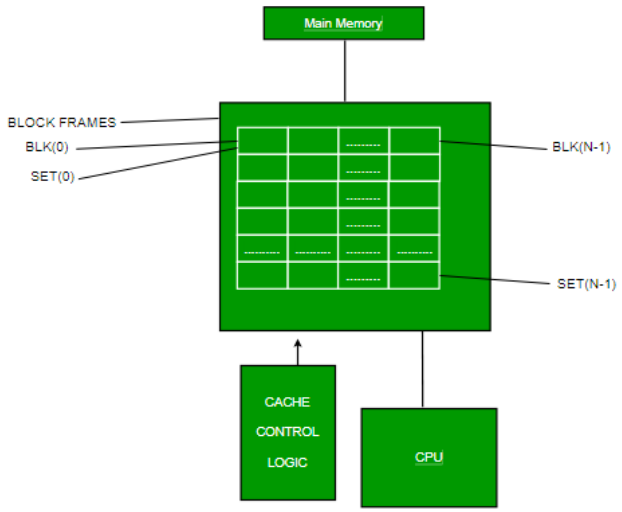
i = cache set number

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



Set-Associative Mapping – Structure

Application of Cache Memory

Here are some of the applications of Cache Memory.

1. **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.
2. **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.
3. **Spatial Locality of Reference:** Spatial Locality of Reference says that there is a chance that the element will be present in close proximity to the reference point and next time if again searched then more close proximity to the point of reference.
4. **Temporal Locality of Reference:** Temporal Locality of Reference uses the Least recently used algorithm will be used. Whenever there is page fault occurs within a word will not only load the word in the main memory but the complete page fault will be loaded because the spatial locality of reference rule says that if you are referring to any word next word will be referred to in its register that's why we load complete page table so the complete block will be loaded.

Advantages of Cache Memory

- Cache Memory is faster in comparison to main memory and secondary memory.
- Programs stored by Cache Memory can be executed in less time.
- The data access time of Cache Memory is less than that of the main memory.

- Cache Memory stored data and instructions that are regularly used by the CPU, therefore it increases the performance of the CPU.

Disadvantages of Cache Memory

- Cache Memory is costlier than primary memory and secondary memory.
- Data is stored on a temporary basis in Cache Memory.
- Whenever the system is turned off, data and instructions stored in cache memory get destroyed.
- The high cost of cache memory increases the price of the Computer System.

Parallel Processing

Parallel processing can be described as a class of techniques which enables the system to achieve simultaneous data-processing tasks to increase the computational speed of a computer system.

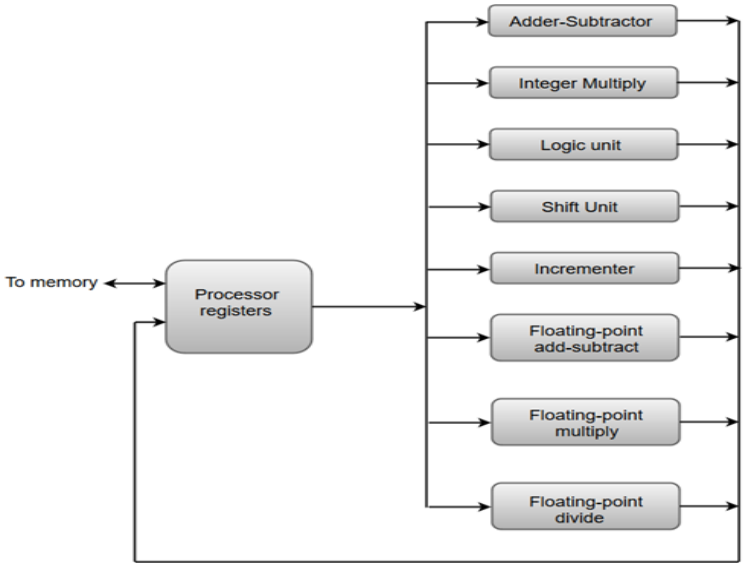
A parallel processing system can carry out simultaneous data-processing to achieve faster execution time. For instance, while an instruction is being processed in the ALU component of the CPU, the next instruction can be read from memory.

The primary purpose of parallel processing is to enhance the computer processing capability and increase its throughput, i.e. the amount of processing that can be accomplished during a given interval of time.

A parallel processing system can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously. The data can be distributed among various multiple functional units.

The following diagram shows one possible way of separating the execution unit into eight functional units operating in parallel.

The operation performed in each functional unit is indicated in each block if the diagram:



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

Pipelining

The term Pipelining refers to a technique of decomposing a sequential process into sub-operations, with each sub-operation being executed in a dedicated segment that operates concurrently with all other segments.

The most important characteristic of a pipeline technique is that several computations can be in progress in distinct segments at the same time. The overlapping of computation is made possible by associating a register with each segment in the pipeline. The registers provide isolation between each segment so that each can operate on distinct data simultaneously.

The structure of a pipeline organization can be represented simply by including an input register for each segment followed by a combinational circuit.

Let us consider an example of combined multiplication and addition operation to get a better understanding of the pipeline organization.

The combined multiplication and addition operation is done with a stream of numbers such as:

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

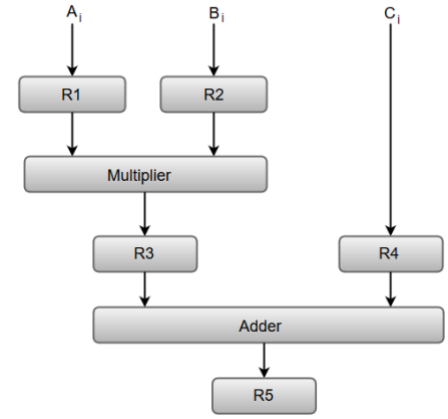
The operation to be performed on the numbers is decomposed into sub-operations with each sub-operation to be implemented in a segment within a pipeline.

The sub-operations performed in each segment of the pipeline are defined as:

$R1 \leftarrow A_i$, $R2 \leftarrow B_i$	Input A_i, and B_i
$R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$	Multiply, and input C_i
$R5 \leftarrow R3 + R4$	Add C_i to product

The following block diagram represents the combined as well as the sub-operations performed in each segment of the pipeline.

Pipeline Processing:



Registers R1, R2, R3, and R4 hold the data and the combinational circuits operate in a particular segment.

The output generated by the combinational circuit in a given segment is applied as an input register of the next segment. For instance, from the block diagram, we can see that the register R3 is used as one of the input registers for the combinational adder circuit.

In general, the pipeline organization is applicable for two areas of computer design which includes:

1. Arithmetic Pipeline
2. Instruction Pipeline

We will discuss both of them in our later sections.

Arithmetic Pipeline

Arithmetic Pipelines are mostly used in high-speed computers. They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems.

To understand the concepts of arithmetic pipeline in a more convenient way, let us consider an example of a pipeline unit for floating-point addition and subtraction.

The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers defined as:

$X = A * 2^a = 0.9504 * 10^3$
$Y = B * 2^b = 0.8200 * 10^2$

Where **A** and **B** are two fractions that represent the mantissa and **a** and **b** are the exponents.

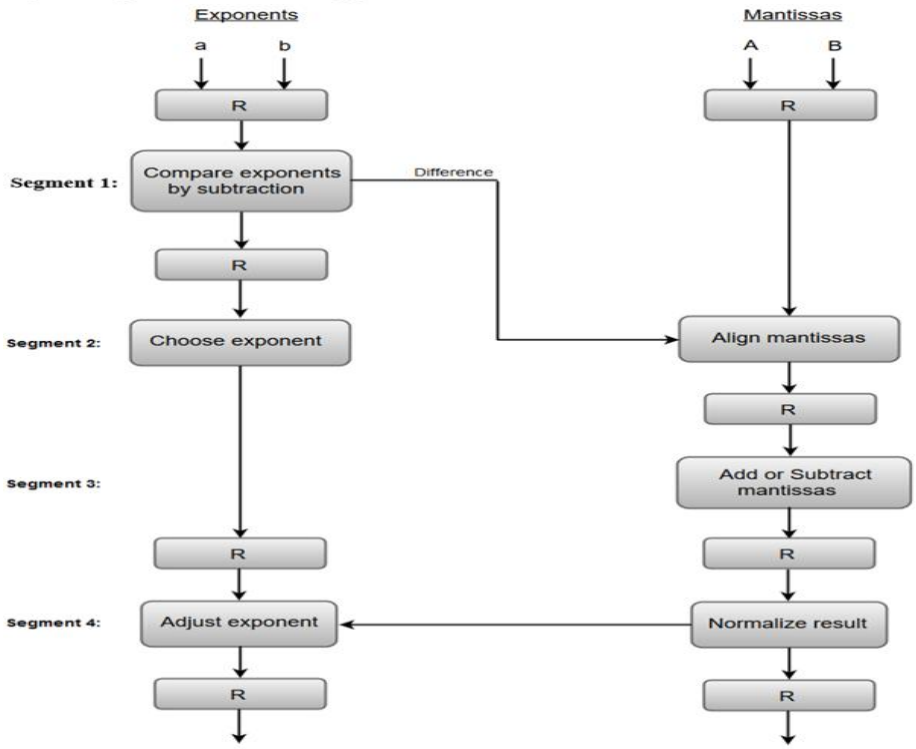
The combined operation of floating-point addition and subtraction is divided into four segments. Each segment contains the corresponding suboperation to be performed in the given pipeline. The suboperations that are shown in the four segments are:

1. Compare the exponents by subtraction.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalize the result.

We will discuss each suboperation in a more detailed manner later in this section.

The following block diagram represents the suboperations performed in each segment of the pipeline.

Pipeline organization for floating point addition and subtraction:



1. Compare exponents by subtraction:

The exponents are compared by subtracting them to determine their difference. The larger exponent is chosen as the exponent of the result.

The difference of the exponents, i.e., **3 - 2** = **1** determines how many times the mantissa associated with the smaller exponent must be shifted to the right.

2. Align the mantissas:

The mantissa associated with the smaller exponent is shifted according to the difference of exponents determined in segment one.

$X = 0.9504 * 10^3$
$Y = 0.08200 * 10^3$

3. Add mantissas:

The two mantissas are added in segment three.

$Z = X + Y = 1.0324 * 10^3$

4. Normalize the result:

After normalization, the result is written as:

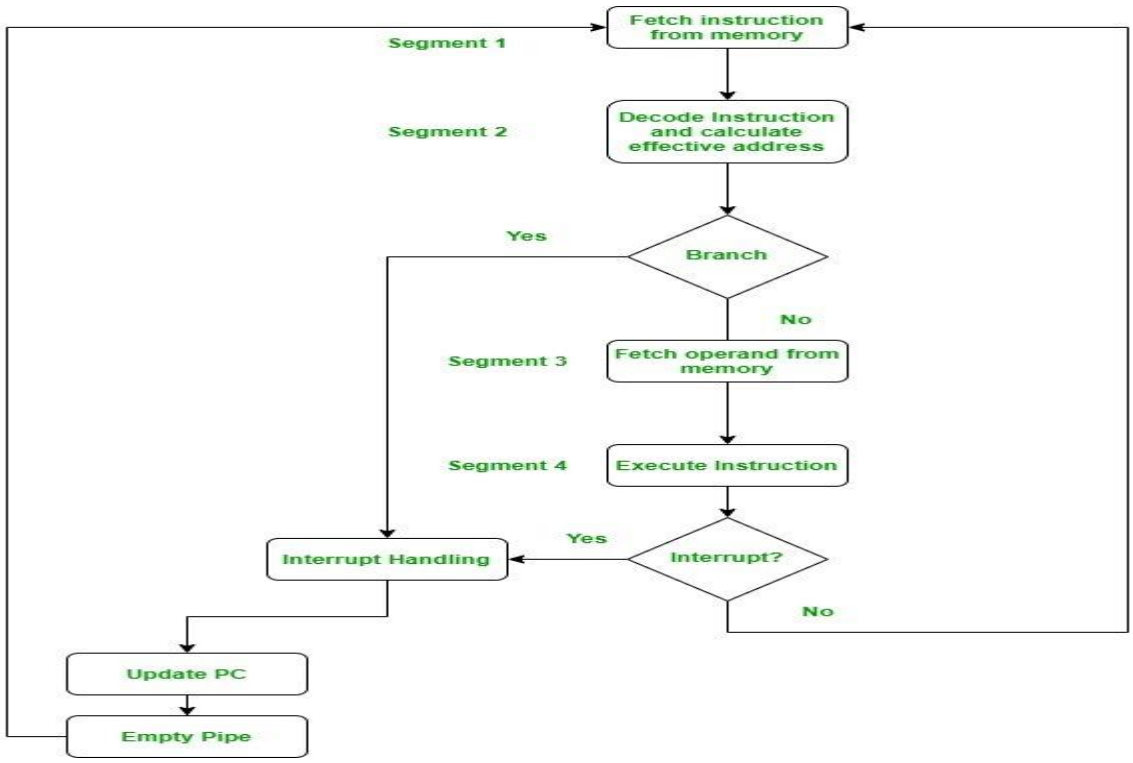
$Z = 0.1324 * 10^4$

Instruction Pipeline:

In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system. An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.
In the most general case computer needs to process each instruction in following sequence of steps:

1. Fetch the instruction from memory (FI)
2. Decode the instruction (DA)
3. Calculate the effective address
4. Fetch the operands from memory (FO)
5. Execute the instruction (EX)
6. Store the result in the proper place

The flowchart for instruction pipeline is shown below.



Let us see an example of instruction pipeline.

Example:

	Stage	1	2	3	4	5	6	7	8	9	10	11	12	13
	1	FI	DA	FO	EX									
Instruction	2		FI	DA	FO	EX								
Branch	3			FI	DA	FO	EX							
	4				FI	---	---	FI	DA	FO	EX			
	5								FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Here the instruction is fetched on first clock cycle in segment 1.
Now it is decoded in next clock cycle, then operands are fetched and finally the instruction is executed. We can see that here the fetch and decode phase overlap due to pipelining. By the time the first instruction is being decoded, next instruction is fetched by the pipeline.

In case of third instruction we see that it is a branched instruction. Here when it is being decoded 4th instruction is fetched simultaneously. But as it is a branched instruction it may point to some other instruction when it is decoded. Thus fourth instruction is kept on hold until the branched instruction is executed. When it gets executed then the fourth instruction is copied back and the other phases continue as usual.