Direct Memory Access (DMA)

In the Direct Memory Access (DMA) the interface transfers the data into and out of the memory unit through the memory bus. The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access (DMA).

During the DMA transfer, the CPU is idle and has no control of the memory buses. A DMA Controller takes over the buses to manage the transfer directly between the I/O device and memory. The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessor is to disable the buses through special control signals such as:

1. Bus Request (BR)
2. Bus Grant (BG).

These two control signals in the CPU facilitates the DMA transfer. The Bus Request (BR) input is used by the DMA controller to request the CPU. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, data bus and read write lines into a high Impedance state. High Impedance state means that the output is disconnected.

The CPU activates the Bus Grant (BG) output to inform the external DMA that the Bus Request (BR) can now take control of the buses to conduct memory transfer without processor. When the DMA terminates the transfer, it disables the Bus Request (BR) line. The CPU disables the Bus Grant (BG), takes control of the buses and return to its normal operation. The transfer can be made in several ways that are:

1. DMA Burst
2. Cycle Stealing

DMA Burst: In DMA Burst transfer, a block sequence consisting of a number of memory words is transferred in continuous burst while the DMA controller is master of the memory buses.

Cycle Stealing: Cycle stealing allows the DMA controller to transfer one data word at a time, after which it must returns control of the buses to the CPU.

DMA Controller:

The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. The DMA controller has three registers:

1. Address Register
2. Word Count Register

3. Control Register

Address Register: Address Register contains an address to specify the desired location in memory.

Word Count Register: WC holds the number of words to be transferred. The register is increase/decrease by one after each word transfer and internally tested for zero.

Control Register: Control Register specifies the mode of transfer

The unit communicates with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (Register select) inputs. The RD (read) and WR (write) inputs are bidirectional. When the BG (Bus Grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When BG =1, the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.

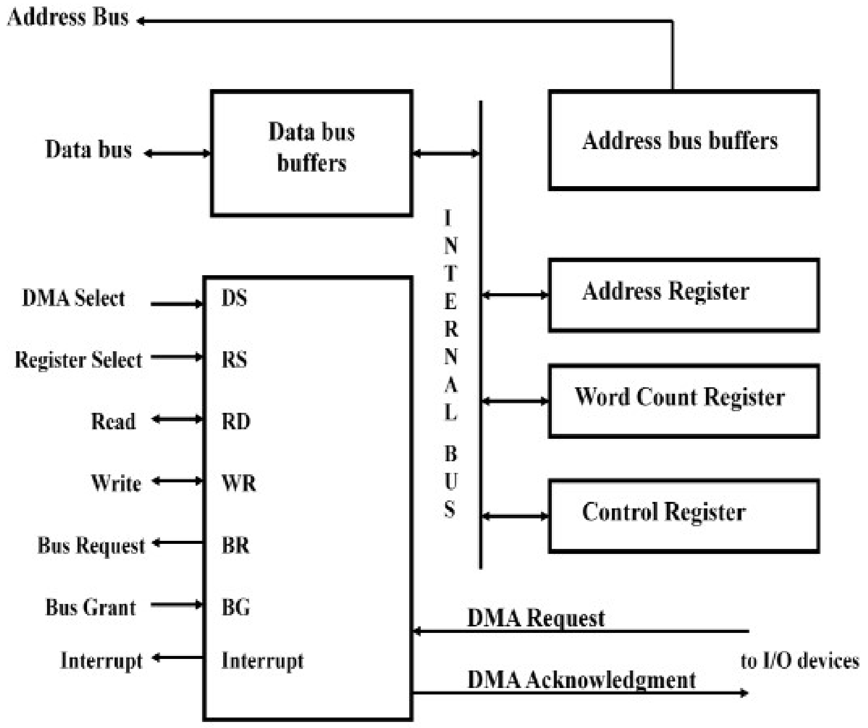


Fig. Block Diagram of DMA Controller

DMA Transfer:

The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initializes the

DMA through the data bus. Once the DMA receives the start control command, it can transfer between the peripheral and the memory. When BG = 0 the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers. When BG=1, the RD and WR are output lines from the DMA controller to the random access memory to specify the read or write operation of data.

BUS Arbitration

In a computer system, multiple devices, such as the CPU, memory, and I/O controllers, are connected to a common communication pathway, known as a bus. In order to transfer data between these devices, they need to have access to the bus. Bus arbitration is the process of resolving conflicts that arise when multiple devices attempt to access the bus at the same time.

When multiple devices try to use the bus simultaneously, it can lead to data corruption and system instability. To prevent this, a bus arbitration mechanism is used to ensure that only one device has access to the bus at any given time.

There are several types of bus arbitration methods, including centralized, decentralized, and distributed arbitration. In centralized arbitration, a single device, known as the bus controller, is responsible for managing access to the bus. In decentralized arbitration, each device has its own priority level, and the device with the highest priority is given access to the bus. In distributed arbitration, devices compete for access to the bus by sending a request signal and waiting for a grant signal.

Bus Arbitration refers to the process by which the current bus master accesses and then leaves the control of the bus and passes it to another bus requesting processor unit. The controller that has access to a bus at an instance is known as a **Bus master**.

A conflict may arise if the number of DMA controllers or other controllers or processors try to access the common bus at the same time, but access can be given to only one of those. Only one processor or controller can be Bus master at the same point in time. To resolve these conflicts, the Bus Arbitration procedure is implemented to coordinate the activities of all devices requesting memory transfers. The selection of the bus master must take into account the needs of various devices by establishing a priority system for gaining access to the bus. The **Bus Arbiter** decides who would become the current bus master.

Applications of bus arbitration in computer organization:

Shared Memory Systems: In shared memory systems, multiple devices need to access the memory to read or write data. Bus arbitration allows multiple devices to access the memory without interfering with each other.

Multi-Processor Systems: In multi-processor systems, multiple processors need to communicate with each other to share data and coordinate processing. Bus arbitration allows multiple processors to share access to the bus to communicate with each other and with shared memory.

Input/Output Devices: Input/Output devices such as keyboards, mice, and printers need to communicate with the processor to exchange data. Bus arbitration allows multiple input/output devices to share access to the bus to communicate with the processor and memory.

Real-time Systems: In real-time systems, data needs to be transferred between devices and memory within a specific time frame to ensure timely processing. Bus arbitration can help to ensure that data transfer occurs within a specific time frame by managing access to the bus.

Embedded Systems: In embedded systems, multiple devices such as sensors, actuators, and controllers need to communicate with the processor to control and monitor the system. Bus arbitration allows multiple devices to share access to the bus to communicate with the processor and memory.

There are two approaches to bus arbitration:

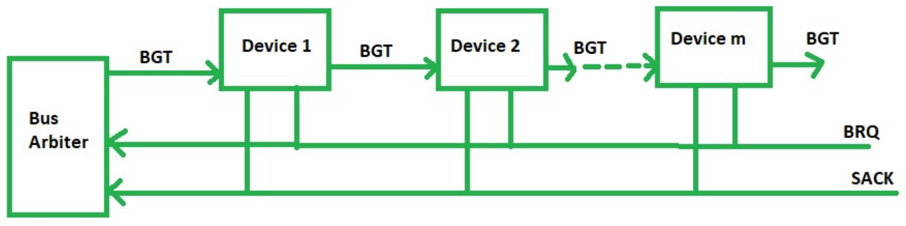
1. **Centralized bus arbitration:** A single bus arbiter performs the required arbitration.

2. **Distributed bus arbitration:** All devices participating in the selection of the next bus master.

Methods of Centralized BUS Arbitration:

There are three bus arbitration methods:

(i) **Daisy Chaining method:** It is a simple and cheaper method where all the bus masters use the same line for making bus requests. The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This master blocks the propagation of the bus grant signal, therefore any other requesting module will not receive the grant signal and hence cannot access the bus. During any bus cycle, the bus master may be any device – the processor or any DMA controller unit, connected to the bus.



Daisy chained bus arbitration

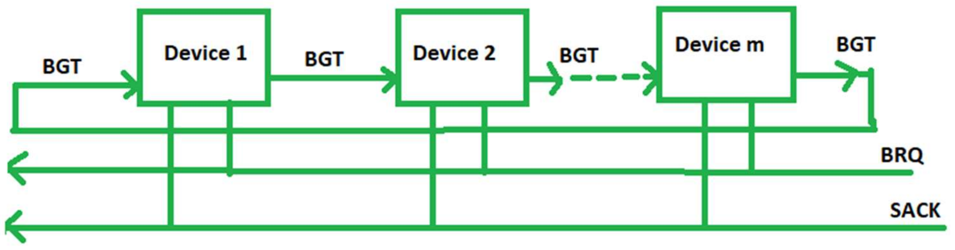
Advantages:

- Simplicity and Scalability.
- The user can add more devices anywhere along the chain, up to a certain maximum value.

Disadvantages:

- The value of priority assigned to a device depends on the position of the master bus.
- Propagation delay arises in this method.
- If one device fails then the entire system will stop working.

(ii) Polling or Rotating Priority method: In this, the controller is used to generate the address for the master (unique priority), the number of address lines required depends on the number of masters connected in the system. The controller generates a sequence of master addresses. When the requesting master recognizes its address, it activates the busy line and begins to use the bus.



Rotating priority bus arbitration

Advantages –

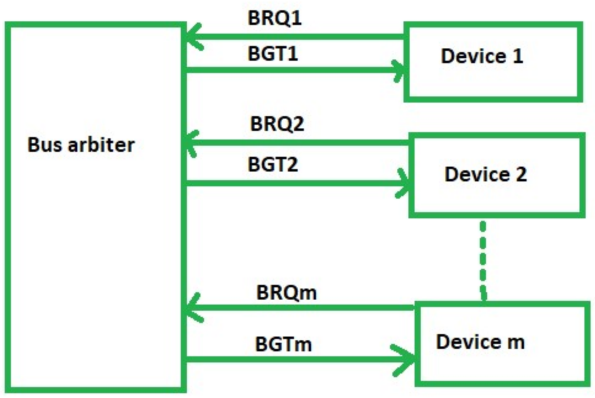
- This method does not favor any particular device and processor.
- The method is also quite simple.
- If one device fails then the entire system will not stop working.

Disadvantages –

- Adding bus masters is difficult as increases the number of address lines of the circuit.

(iii) Fixed priority or Independent Request method: In this, each master has a separate pair of bus request and bus grant lines and each pair has a priority assigned to it.

The built-in priority decoder within the controller selects the highest priority request and asserts the corresponding bus grant signal.



Fixed priority bus arbitration method

Advantages –

- This method generates a fast response.

Disadvantages –

- Hardware cost is high as a large no. of control lines is required.

Distributed BUS Arbitration: In this, all devices participate in the selection of the next bus master. Each device on the bus is assigned a 4bit identification number. The priority of the device will be determined by the generated ID.

Uses of BUS Arbitration in Computer Organization:

Bus arbitration is a critical process in computer organization that has several uses and benefits, including:

1. Efficient use of system resources: By regulating access to the bus, bus arbitration ensures that each device has fair access to system resources, preventing any single device from monopolizing the bus and causing system slowdowns or crashes.

2. Minimizing data corruption: Bus arbitration helps prevent data corruption by ensuring that only one device has access to the bus at a time, which minimizes the risk of multiple devices writing to the same location in memory simultaneously.

3. Support for multiple devices: Bus arbitration enables multiple devices to share a common communication pathway, which is essential for modern computer systems with multiple peripherals, such as printers, scanners, and external storage devices.

4. Real-time system support: In real-time systems, bus arbitration is essential to ensure that high-priority tasks are executed quickly and efficiently. By prioritizing access to the bus, bus arbitration can ensure that critical tasks are given the resources they need to execute in a timely manner.

5. Improved system stability: By preventing conflicts between devices, bus arbitration helps to improve system stability and reliability. This is especially important in mission-critical systems where downtime or data corruption could have severe consequences.

Issues of BUS Arbitration in Computer Organization:

Bus arbitration is a critical process in computer organization that has several uses and benefits, including:

1. Efficient use of system resources: By regulating access to the bus, bus arbitration ensures that each device has fair access to system resources, preventing any single device from monopolizing the bus and causing system slowdowns or crashes.

2. Minimizing data corruption: Bus arbitration helps prevent data corruption by ensuring that only one device has access to the bus at a time, which minimizes the risk of multiple devices writing to the same location in memory simultaneously.

3. Support for multiple devices: Bus arbitration enables multiple devices to share a common communication pathway, which is essential for modern computer systems with multiple peripherals, such as printers, scanners, and external storage devices.

4. Real-time system support: In real-time systems, bus arbitration is essential to ensure that high-priority tasks are executed quickly and efficiently. By prioritizing access to the bus, bus arbitration can ensure that critical tasks are given the resources they need to execute in a timely manner.

5. Improved system stability: By preventing conflicts between devices, bus arbitration helps to improve system stability and reliability. This is especially important in mission-critical systems where downtime or data corruption could have severe consequences.

Cache Memory

Cache Memory is a special very high-speed memory. The cache is a smaller and faster memory that 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. The most important use of cache memory is that it is used to reduce the average time to access data from the main memory.

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.

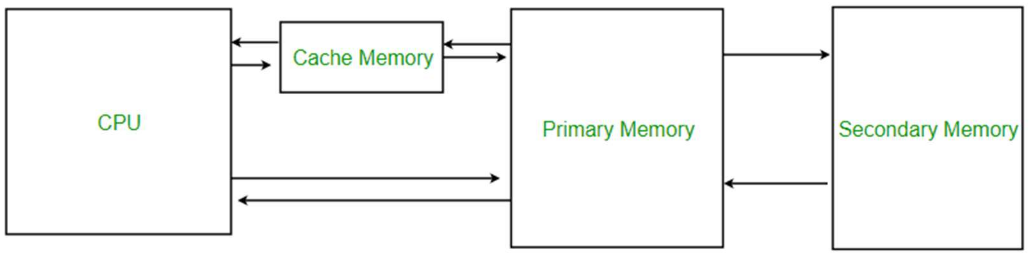


Fig. Cache Memory

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 the CPU. The most commonly used register is Accumulator, Program counter, Address Register, etc.
- **Level 2 or Cache memory:** It is the fastest memory that has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory:** It is the memory on which the 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 that is not as fast as the main memory but data stays permanently in this memory.

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 and m = number of lines in the cache

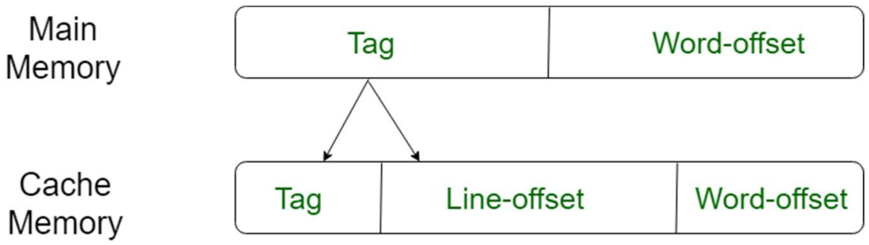


Fig. Direct Mapping

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.

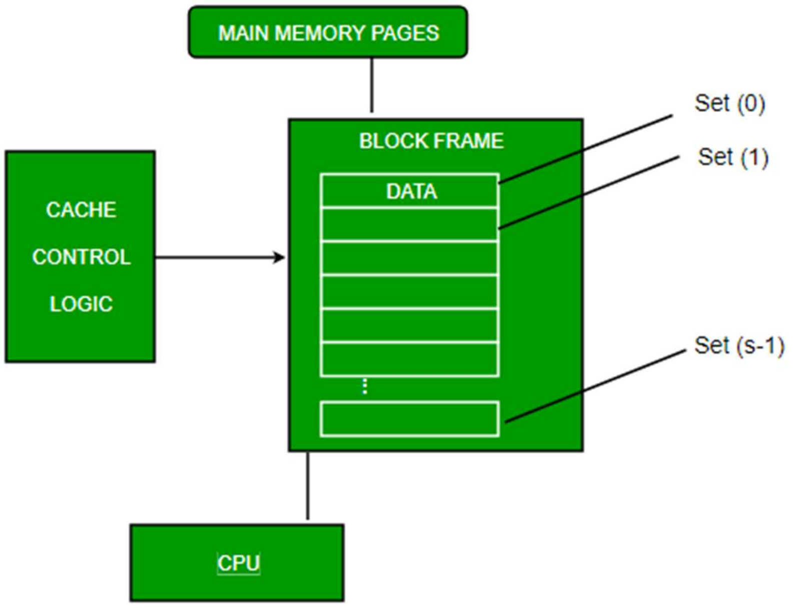


Fig. Direct Mapping – Structure

3. **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.

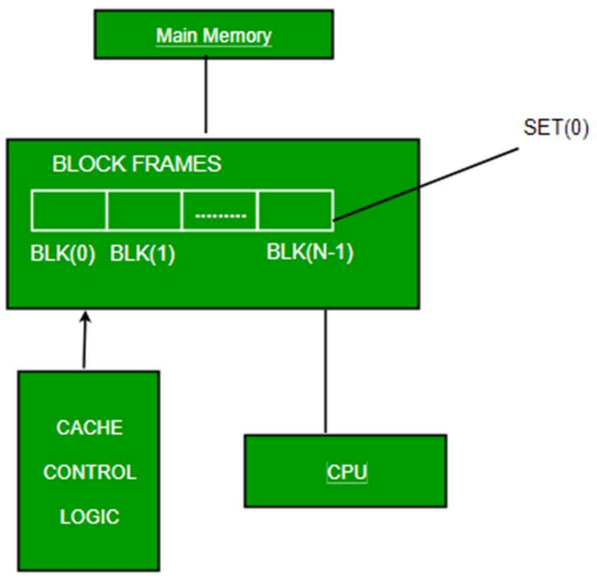


Fig. Associative Mapping – Structure

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.

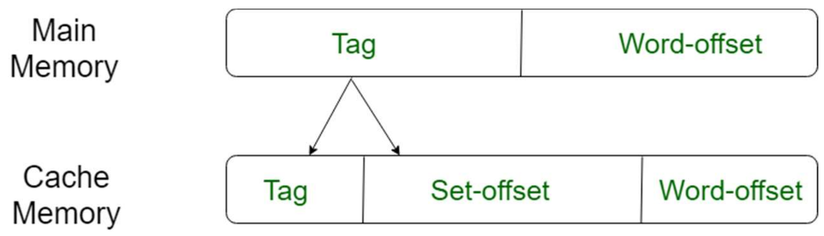


Fig. Set-Associative Mapping

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

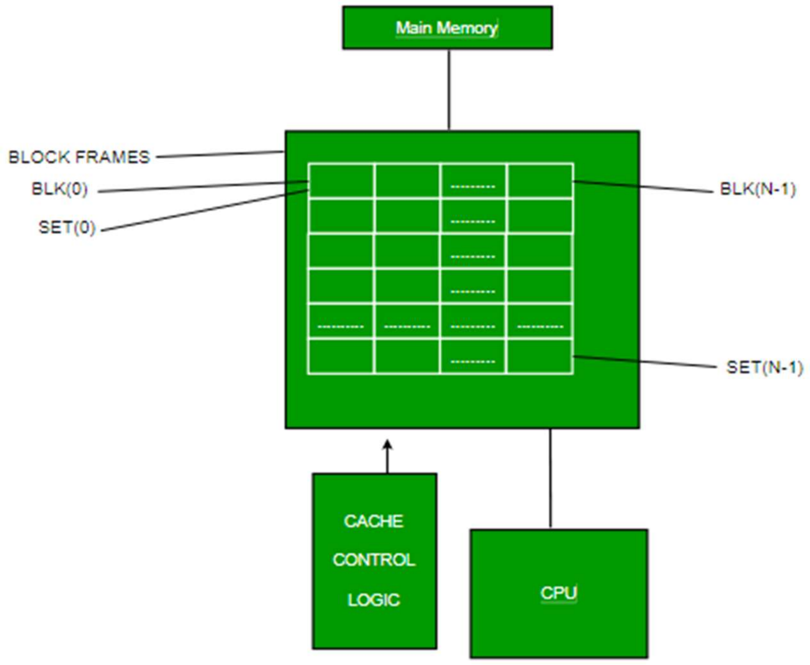


Fig. Set-Associative Mapping – Structure

Application of Cache Memory: Here are some of the applications of Cache Memory.

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.

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.

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:

- **(i)** Cache Memory is faster in comparison to main memory and secondary memory.
- **(ii)** Programs stored by Cache Memory can be executed in less time.
- **(iii)** The data access time of Cache Memory is less than that of the main memory.
- **(iv)** 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:

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