CHAPTER:7

MEMORY ORGANIZATION

BY: ER. PRALHAD CHAPAGAIN

(Contact: pralhad.chapagain@gmail.com)

Syllebus:

- **Hierarchical Memory System**
- **Cache Memory: Associative Memory**
- **Cache mapping with Associative, Direct and Set- Associative Mapping**
- **Replacing Data in cache, writing Data to the cache, Cache performance Basics**
- **Virtual memory: Paging, Segmentation and Memory Protection**

Microcomputer Memory:

- A memory unit is the collection of storage units or devices together
- The memory unit stores the binary information in the form of bits
- Generally, memory/storage is classified into 2 categories
    - Volatile Memory:
        - This loses its data, when power is switched off.
    - Non-Volatile Memory:
        - This is a permanent storage and does not lose any data when power is switched off.
- The memory unit that communicates directly with the CPU is called main memory.
- Devices that provide backup storage are called auxiliary memory or secondary memory.

Characteristics of memory Devices:

- **Location**
  - CPU
  - Internal or Main
  - External or Secondary
- **Capacity**
- **Unit of transfer**
- **Access Method**
  - Random Access
  - Serial Access
  - Semi random Access
- **Performance**
- **Physical types**
- **Physical characteristics**
  - Volatile/Non-Volatile
- **Organization**
  - Erasable/ Non-Erasable

Memory Hierarchy(1):

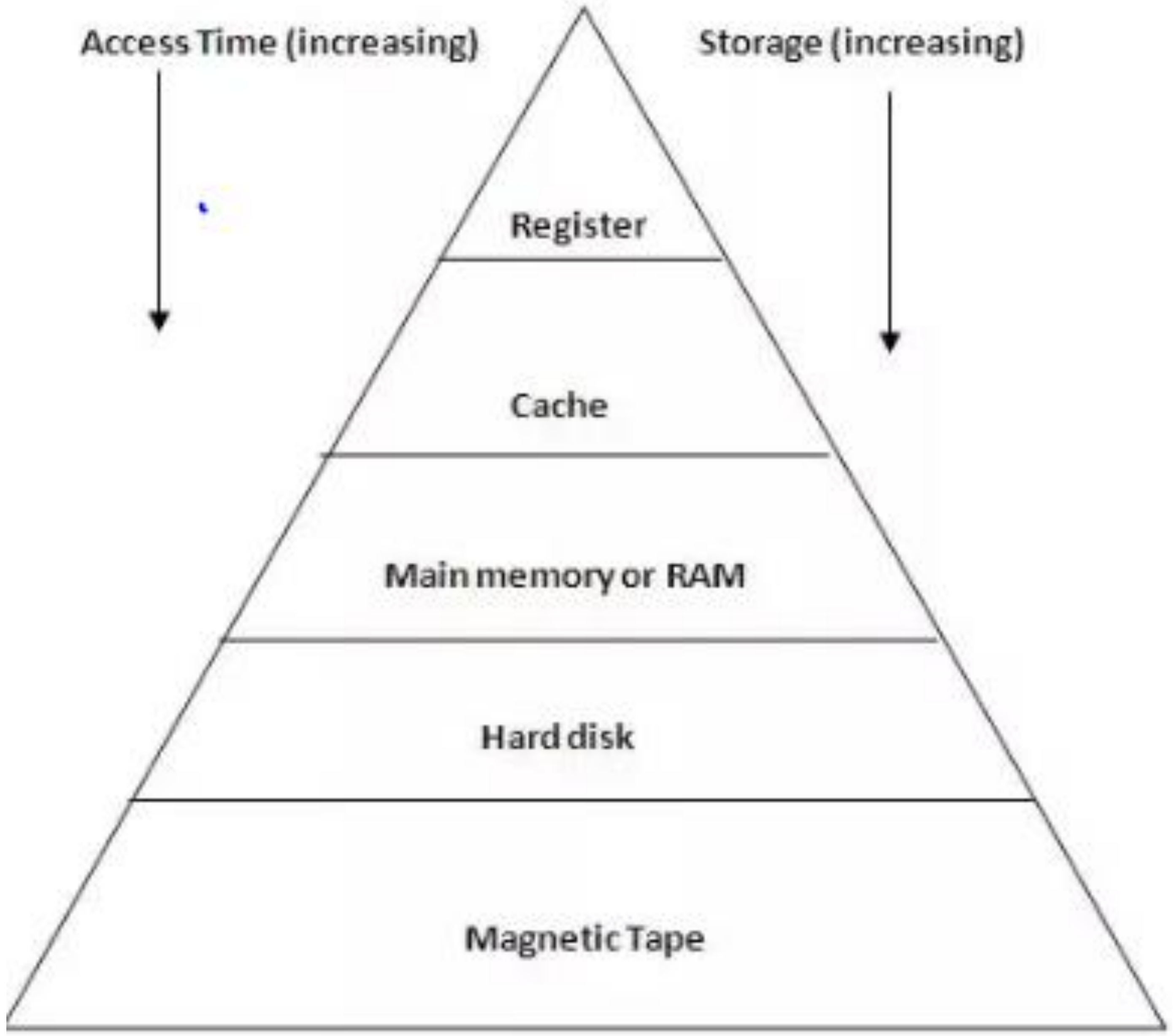
- Capacity, cost and speed of different types of memory play a vital role while designing a memory system for computers.

- There is a tradeoff between three characteristics cost, capacity and access time. One cannot achieve all these quantities in same memory module because
  - If capacity increases, access time increases (slower) and due to which cost per bit decreases.
  - If access time decreases (faster), capacity decreases and due to which cost per bit increases.

- The designer tries to increase capacity because cost per bit decreases and the more application program can be accommodated. But at the same time, access time increases and hence decreases the performance. So, the best idea will be to use memory hierarchy.

- Memory Hierarchy is to obtain the highest possible access speed while minimizing the total cost of the memory system.

Memory Hierarchy(2):



Memory Hierarchy(3):

- As we go down in the hierarchy
  - Cost per bit decreases
  - Capacity of memory increases
  - Access time increases
  - Frequency of access of memory by processor also decreases.
- **Register**:
  - CPU registers are at the top most level of this hierarchy, they hold the most frequently used data. They are very limited in number and are the fastest.
  - They are often used by the CPU and the ALU for performing arithmetic and logical operations, for temporary storage of data.
- **Cache**:
  - It is used by CPU for memory which is being accessed over and over again. Instead of pulling it every time from the main memory, it is put in cache for fast access.
  - It is also a smaller memory, however, larger than internal register.
  - Cache is further classified to L1, L2, L3:

Memory Hierarchy(4):

  - L1 cache: It is accessed without any delay
  - L2 cache: It takes more clock cycles to access than L1 cache.
  - L3 cache: It takes more clock cycles to access than L2 cache.
- **Main memory or RAM:**
  - It is a type of computer memory and is a hardware component. It can be increased provided the operating system can handle it.
- **Hard Disk:**
  - Hard Disk is a hardware component in a computer. Data is kept permanently kept in this memory.
  - Memory from hard disk is not directly accessed by the CPU, hence it is slower. As compared with RAM, hard disk is cheaper per bit.
- **Magnetic tape:**
  - It is usually used for backing up large data.
  - When a system needs to access a tape, it is first mounted to access the data. When the data is accessed, it is then un-mounted.
  - The memory access time is slower in magnetic tape and it usually takes few minutes to access a tape.

Associative Memory (Content Addressable Memory) (1):

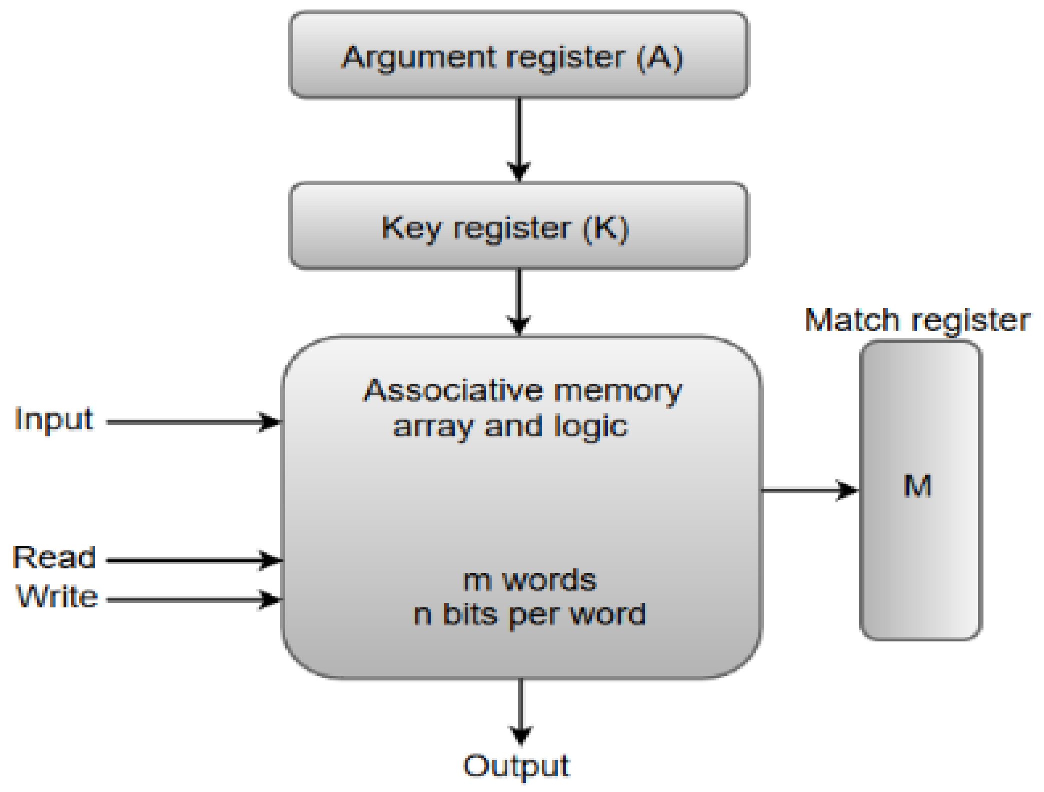
- An associative memory can be considered as a memory unit whose stored data can be identified for access by the content of the data itself rather than by an address or memory location.

- 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.

- 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.

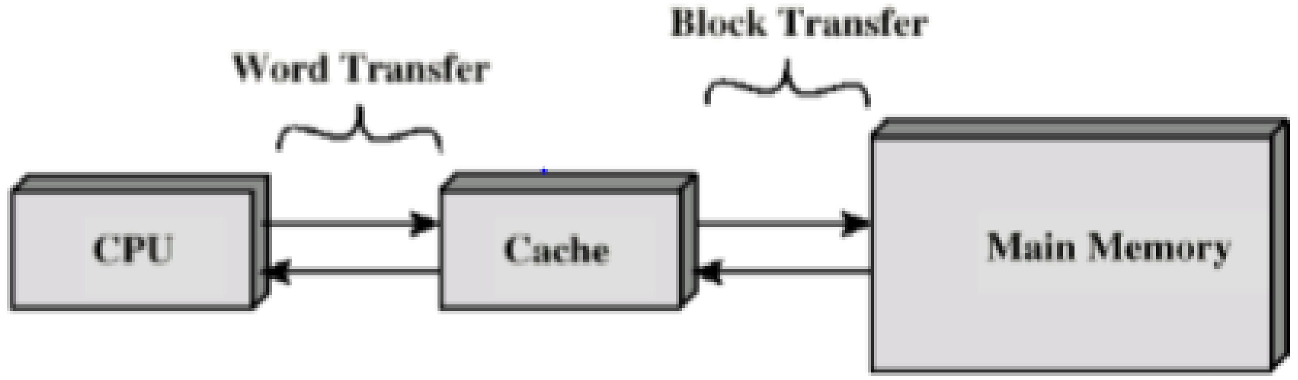
Associative Memory (Content Addressable Memory) (2):



Associative Memory (Content Addressable Memory) (3):

- 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.

Cache memory (1):



- Main memory is slow. Cache is a small high speed memory that creates the illusion of a fast Main Memory. Stores data from some frequently used addresses (of main memory).

- When CPU attempts to read a word from main memory, check is made to determine if the word is in cache. If so, then word is delivered from cache.

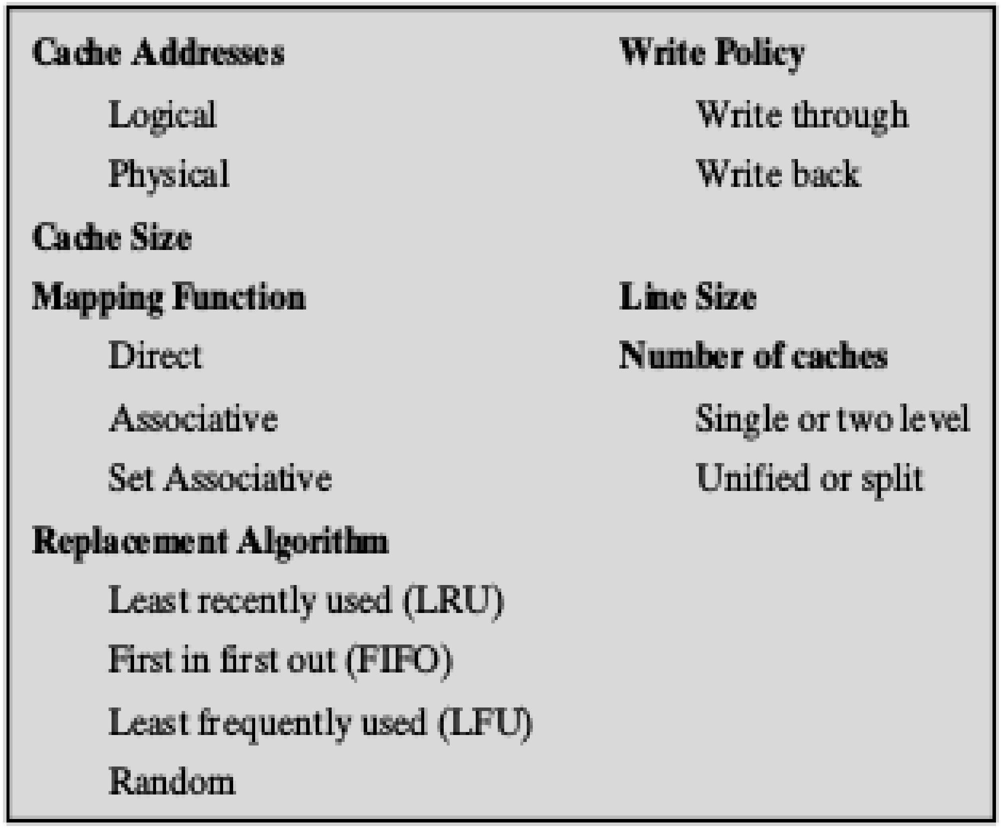
- If word is not there in cache then a block of main memory consisting some word along with that word, is read into cache and the required word is delivered to CPU. This is called “Principle of Locality of Reference”

Cache memory (2):

- **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.
- Cache hit: Data found in cache. Results in data transfer at maximum speed.
- Cache miss: Data not found in cache. Processor loads data from memory and copies into cache. This results in extra delay, called miss penalty.
- Hit ration= Percentage of memory accesses satisfied by the cache
- Miss ration= 1- hit ratio

Cache memory (3):

- The key elements of cache design are:



Cache Addresses
- Logical
- Physical

Cache Size

Mapping Function
- Direct
- Associative
- Set Associative

Replacement Algorithm
- Least recently used (LRU)
- First in first out (FIFO)
- Least frequently used (LFU)
- Random

Write Policy
- Write through
- Write back

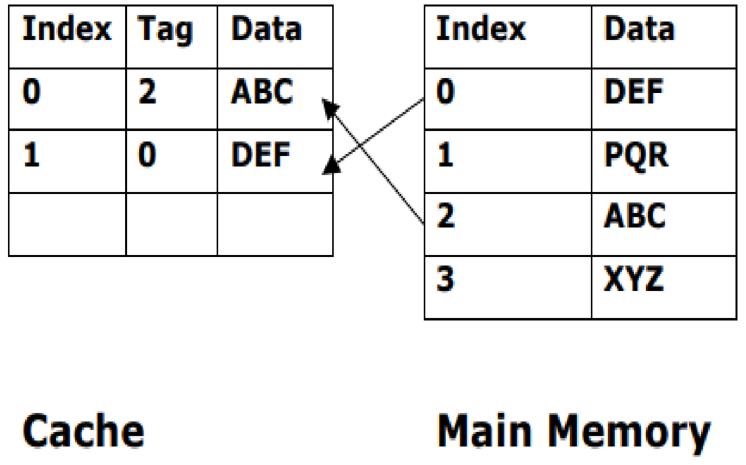
Line Size

Number of caches
- Single or two level
- Unified or split

Cache memory (4):

Cache line:

- Cache is partitioned into lines (also called blocks). Each line has 4-64 bytes in it. During data transfer, a whole line is read or written.
- Each line has a tag that indicates the address in memory from which the line has been copied
- Cache hit is detected through an associative search of all the tags. Associative search provides a fast response to the query: “Does this key match any of the tags?”
- Data is read only if a match is found.



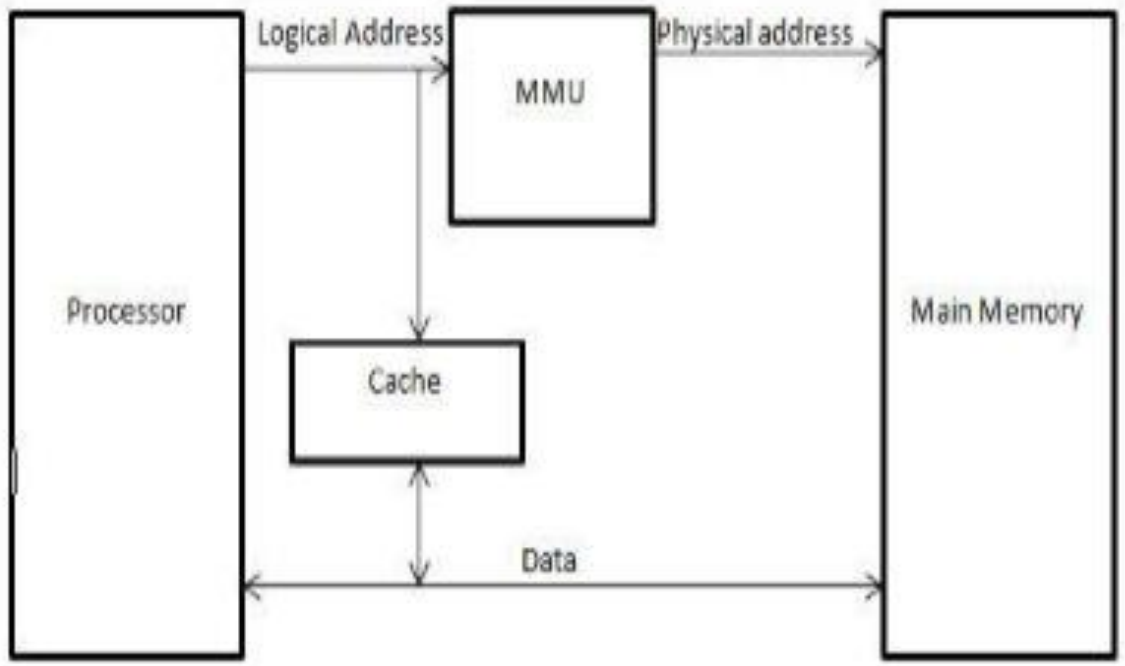
Index	Tag	Data
0	2	ABC
1	0	DEF

Index	Data
0	DEF
1	PQR
2	ABC
3	XYZ

Cache memory (5):

Cache Address:

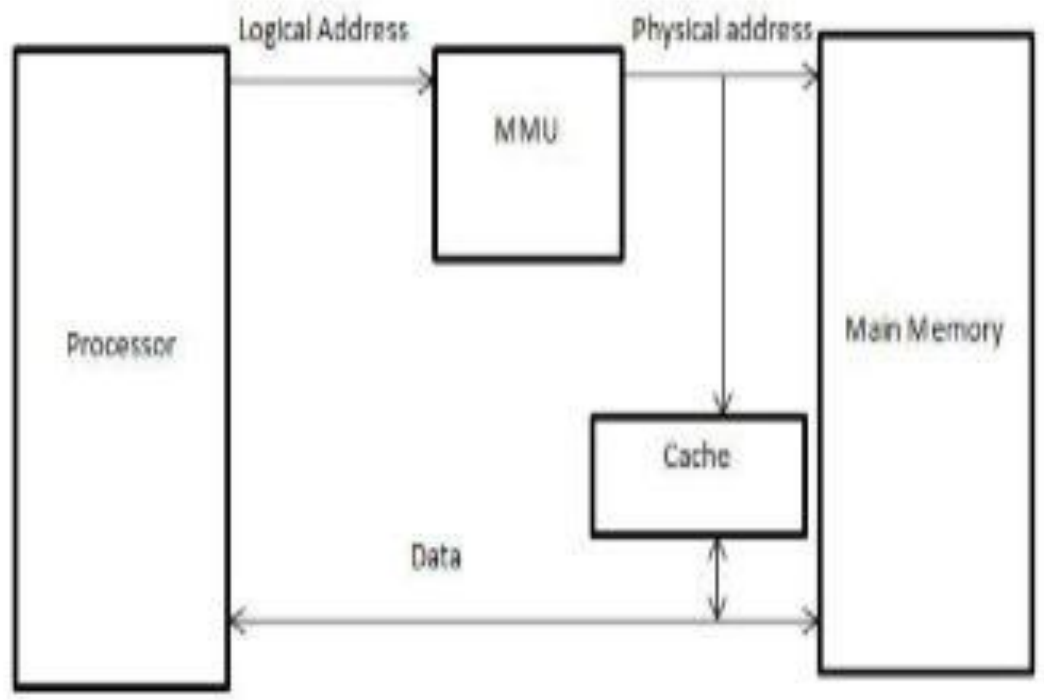
- When **virtual addresses** are used, the cache can be placed between the processor and the MMU. A logical cache, also known as a virtual cache, stores data using virtual addresses. The processor accesses the cache directly, without going through the MMU.



Cache memory (6):

Cache Address:

- A **physical** cache stores data using main memory physical addresses. One advantage of the logical cache is that cache access speed is faster than for a physical cache, because the cache can respond before the MMU performs an address translation



Cache memory (8):

Cache Mapping Technique:

- Cache mapping is the method by which the contents of main memory are brought into the cache and referenced by the CPU
- Cache mapping used to assign main memory address to cache address and determine hit or miss.

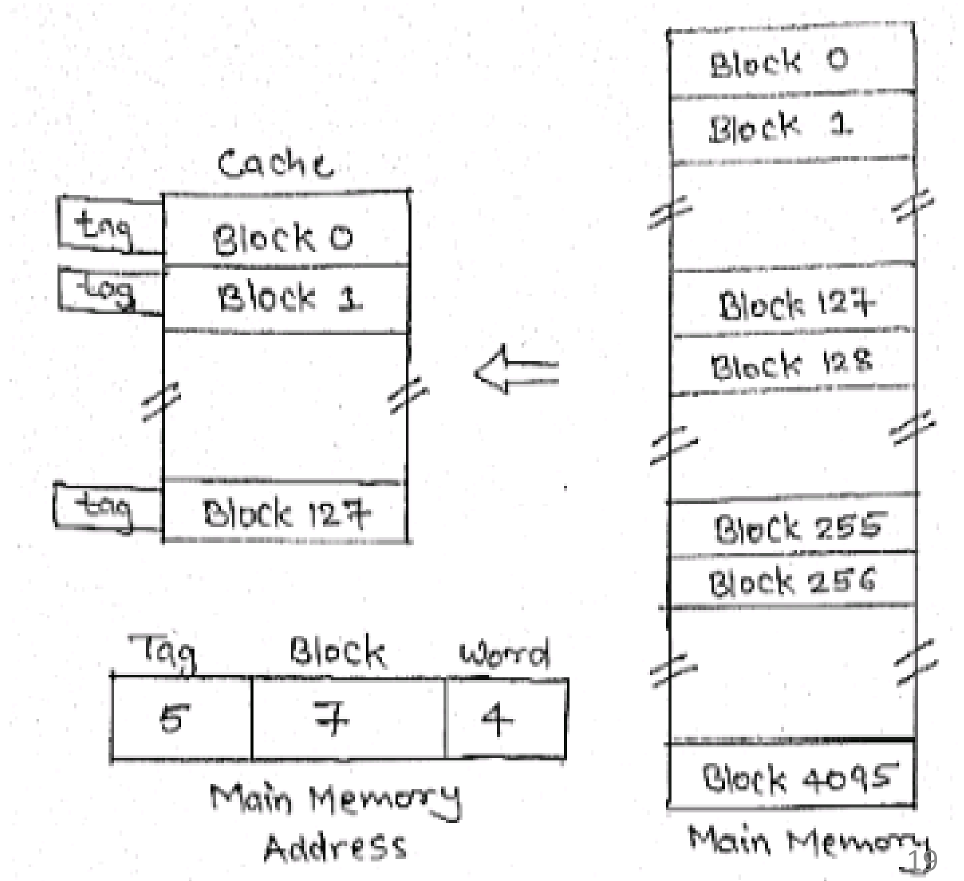
- Three basic techniques:
    - **Direct mapping**
    - **Associative mapping**
    - **Set-associative mapping**

- Consider a cache consisting of 128 blocks of 16 words each, for total of 2048(2K) words and assume that the main memory is addressable by 16 bit address. Main memory is 64K which will be viewed as 4K blocks of 16 words each.

Cache memory (9):

Direct mapping

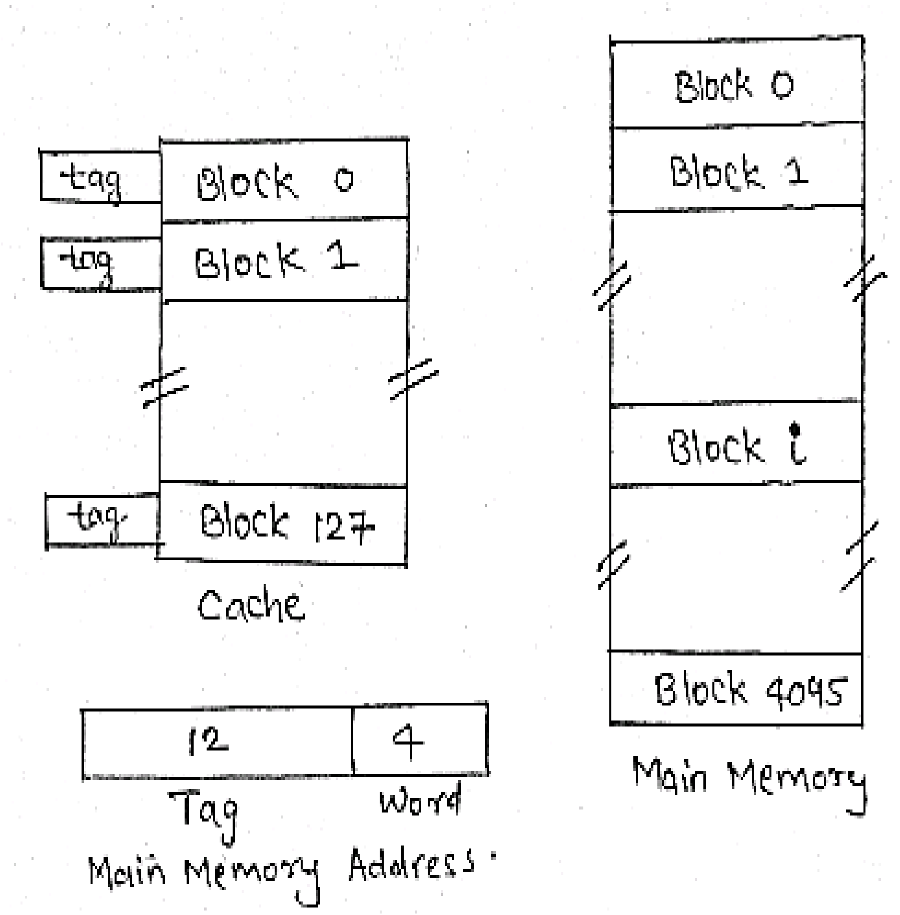
- The simplest way to determine cache locations in which Memory blocks is stored is direct Mapping technique.
- In this, block J of the main memory maps on to block J modulo 128 of the cache.
- Placement of a block in the cache is determined from memory address.
- Memory address is divided into 3 fields, the lower 4-bits selects one of the 16 words in a block.
- When new block enters the cache, the 7-bit cache block field determines the cache positions in which this block must be stored.
- The higher order 5-bits of the memory address of the block are stored in 5 tag bits associated with its location in cache. They identify which of the 32 blocks that are mapped into this cache position are currently resident in the cache.
- It is easy to implement, but not Flexible.



Cache memory (10):

Associative mapping

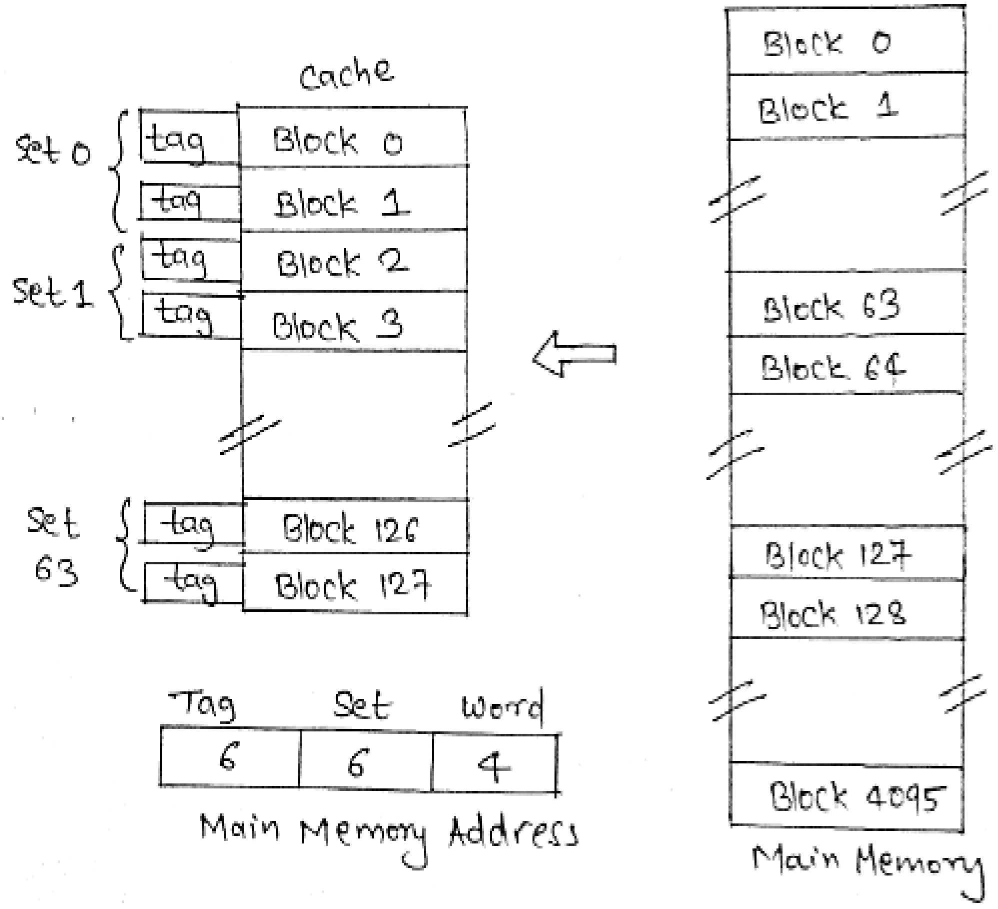
- This is more flexible mapping method, in which main memory block can be placed into any cache block position.
- In this, 12 tag bits are required to identify a memory block when it is resident in the cache.
- The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to see, if the desired block is present. This is known as Associative Mapping technique.
- Cost of an associated mapped cache is higher than the cost of direct-mapped because of the need to search all 128 tag patterns to determine whether a block is in cache. This is known as associative search.



Cache memory (11):

Set-Associative mapping

- It is the combination of direct and associative mapping technique.
- Cache blocks are grouped into sets and mapping allow block of main memory reside into any block of a specific set. Hence contention problem of direct mapping is eased, at the same time, hardware cost is reduced by decreasing the size of associative search.
- For a cache with two blocks per set. In this case, memory block 0, 64, 128….. 4032 map into cache set 0 and they can occupy any two block within this set.
- Having 64 sets means that the 6-bit set field of the address determines which set of the cache might contain the desired block. The tag bits of address must be associatively compared to the tags of the two blocks of the set to check if desired block is present. This is two-way associative search.



Cache memory (12):

Q. Consider a Cache with 256 blocks of 16 words each. The memory is addressed with 16 bits. How the address is divided or what is the tag size in

- Direct Mapping
- Associative Mapping
- Set Associative Mapping (2 way)

Solution:

Given:

- memory address is 16 bits (2^4 bits)
- Total Blocks 256 (2^8 blocks)
- Total Sets 128 (2^7)

For Direct Mapping:

Tag	**Block**	**Words(offset)**
4	8	4

For Associative Mapping:

Tag	**Words(offset)**
12	4

For Set Associative Mapping:

Tag	**Set**	**Words(offset)**
5	7	4

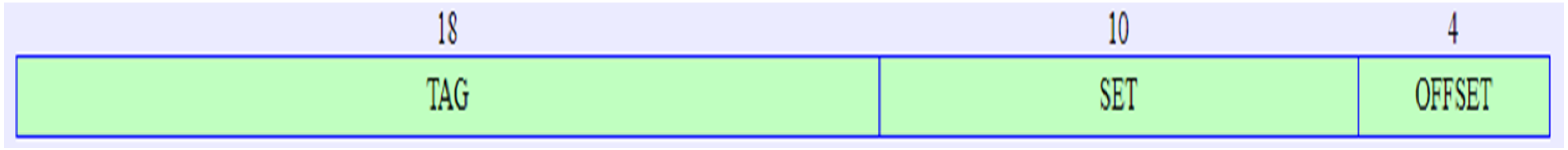
Cache memory (13):

Q. A computer uses 32-bit addressing. The computer uses a 2 way associative cache with a capacity of 32KB. Each cache block contains 16 bytes. Calculate the number of bits in a tag, Set and offset fields of main memory.

Solution:

Given:

- Since there are 16 bytes in a cache block, the offset field must contain 4 bits (2^4=16).
- Each set contains 2 cache blocks(2-way Associative). So a set contains 32 bytes.
- There are 32KB bytes in the entire cache, so there are 32KB/32B= 1K sets. Thus the set field contains 10 bits (2^{10}=1k)



Cache memory (14):

Q. A computer system uses 16-bit memory addresses. It has a 2K-byte cache organized in a direct-mapped manner with 64 bytes per cache block. Assume that the size of each memory word is 1 byte. (a) Calculate the number of bits in each of the Tag, Block, and Word fields of the memory address.

Solution:

- Block size = 64 bytes = 2^6 bytes = 2^6 words (since 1 word = 1 byte)
- Therefore, Number of bits in the Word field = 6
- Cache size = 2K-byte = 2^{11} bytes
- Number of cache blocks = Cache size / Block size = $2^{11}/2^6 = 2^5$
- Therefore, Number of bits in the Block field = 5
- Total number of address bits = 16
- Therefore, Number of bits in the Tag field = 16 - 6 - 5 = 5
- For a given 16-bit address, the 5 most significant bits, represent the Tag, the next 5 bits represent the Block, and the 6 least significant bits represent the Word.

:

Cache memory (15):

Replacing data in the cache

- For direct mapping where there is only one possible line for a block of memory, no replacement algorithm is needed.
- For associative and set associative mapping, however, an algorithm is needed.
- For maximum speed, this algorithm is implemented in the hardware. Four of the most common algorithms are:

1. **Least recently used** this replaces the candidate line in cache memory that has been there the longest with no reference to it.

2. **First in first out** this replaces the candidate line in the cache that has been there the longest.

3. **Least frequently used** this replaces the candidate line in the cache that has had the fewest references.

4. **Random** replacement this algorithm randomly chooses a line to be replaced from among the candidate lines. Studies have shown that this yields only slightly inferior performance than other algorithms.

Cache memory (16):

Write Policy

- This is important because if changes were made to a line in cache memory, the appropriate changes should be made to the block in main memory before removing the line from the cache.
- The problems to contend with are more than one device may have access to main memory (I/O modules). If more than one processor on the same bus with its own cache is involved, the problem becomes more complex. Any change in either cache or main memory could invalidate the others.
- The simplest technique is called "**write through**". In this technique, both main memory and cache are written to when a write operation is performed, ensuring that main memory is always valid. The main disadvantage of this technique is that it may generate substantial main memory traffic, causing a bottle neck and decreasing performance.
- An alternative technique, known as "**write back**" minimizes main memory writes. Updates are made only in the cache. An update bit associated with the line is set. Main memory is updated when the line in cache gets replaces only if the update bit has been set. The problem with this technique is that all changes to main memory have to be made through the cache in order not to invalidate parts of main memory, which potentially may cause a bottle neck.

Cache memory (18):

Number of Caches

- When caches were originally introduced, the typical system had a single cache. More recently, the use of multiple caches has become an important aspect. There are two design issues surrounding number of caches.
- **MULTILEVEL CACHES:** Most contemporary designs include both on-chip and external caches. The simplest such organization is known as a two-level cache, with the internal cache designated as level 1 (L1) and the external cache designated as level 2 (L2). There can also be 3 or more levels of cache. This helps in reducing main memory accesses.
- **UNIFIED VERSUS SPLIT CACHES:** Earlier on-chip cache designs consisted of a single cache used to store references to both data and instructions. This is the unified approach. More recently, it has become common to split the cache into two: one dedicated to instructions and one dedicated to data. These two caches both exist at the same level. This is the split cache. Using a unified cache or a split cache is another design issue.

Cache memory (19):

Cache performances

- Every time the CPU accesses memory, it checks the cache. If the requested data is in the cache, it accesses data from cache rather than physical memory, this is a **cache hit.**
- If the data is not in the cache, the CPU accesses the data from main memory (and usually writes the data into the cache as well). This is a **cache miss.**
- **Hit ratio** is the ratio of total number of hits to the total number of CPU references.
- **Hit latency** is the time taken to find whether required element is in cache or not.
- The **average memory access time,** $T_M = hT_c + (1-h)T_p$

  where, h = hit ratio

  T_c = cache access time

  T_p = physical access time

Cache memory (20):

Q. In a certain system the main memory access time is 100 ns. The cache is 10 time faster than the main memory and uses the write though protocol. If the hit ratio for read request is 0.92 and 85% of the memory requests generated by the CPU are for read, the remaining being for write; then find the average time consideration both read and write requests.

Solution:

The effective access time for reading:

$T_{m\_r} = H_r * T_c + (1\text{-}H_r)T_n$

$= 0.92*10 + (1 - 0.92)100$

$= 9.2 + 8$

$= 17.2$

The effective access time for writing,

$T_{m\_w} = H_w * T_c + (1\text{-}H_w)T_n$

$=0.85*17.2 + 0.15*100$

$= 14.62 + 15$

$= 29.62$

Cache memory (21):

Q. Consider a system with 2 level caches. Access times of Level 1 cache, Level 2 cache and main memory are 1 ns, 10ns, and 500 ns, respectively. The hit rates of Level 1 and Level 2 caches are 0.8 and 0.9, respectively. What is the average access time of the system ignoring the search time within the cache?

Solution:

Average access time = $[H_1*T_1]+[(1-H_1)*H_2*T_2]+[(1-H_1)(1-H_2)*H_m*T_m]$

where,

H_1 = Hit rate of level 1 cache = 0.8

T_1 = Access time for level 1 cache = 1 ns

H_2 = Hit rate of level 2 cache = 0.9

T_2 = Access time for level 2 cache = 10 ns

H_m = Hit rate of Main Memory = 1

T_m = Access time for Main Memory = 500 ns

So, Average Access Time = $(0.8 * 1) + (0.2 * 0.9 * 10) + (0.2 * 0.1 * 1 * 500)$

$= 0.8 + 1.8 + 10$

$= 12.6$ ns

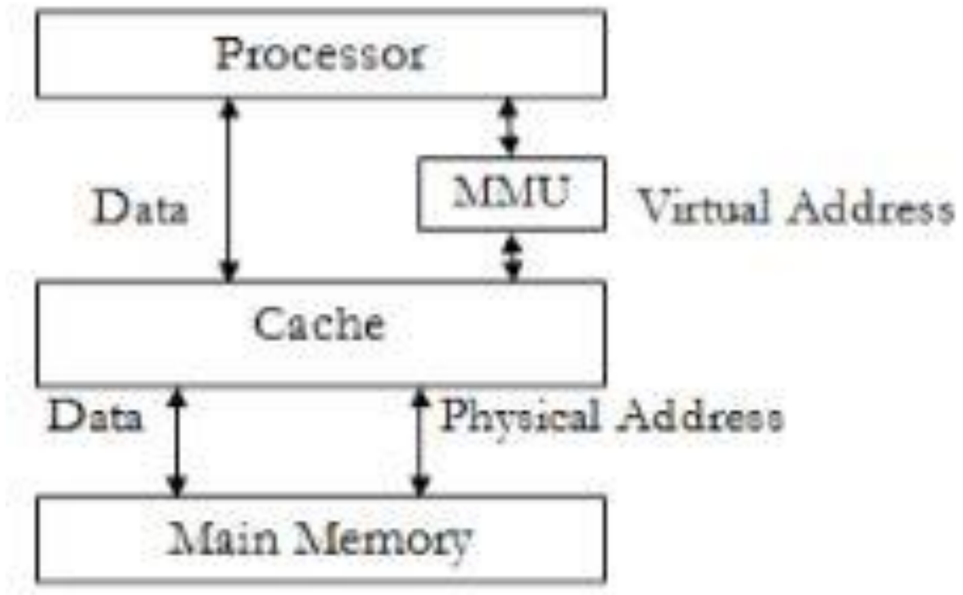
Cache memory (22):

Cache Design Issues

- *Block placement:* where can a block be placed in the cache?
- *Block replacement:* which block should be replaced on a miss?
- *Write strategy:* what happens on a write?
- *Cache parameters:* e.g. block size, cache size, associativity

Virtual Memory (1):

- When a program does not fit into the main memory, the parts of the program that are currently not being executed are stored on the hard disk and are transferred to main memory when required.
- Virtual memory is just the extension of the main memory in the disk where parts of very big programs that are currently being executed but cannot be stored completely in the main memory itself are stored.
- This shows that the CPU no longer generates the physical address directly. This is translated to the physical address by the MMU which is then used to address the main memory



Virtual Memory (2):

- Main advantage of this scheme is that program can be larger than physical memory.

Example:

  - error handling routines are used only when error occurs in the data or computation.
  - certain feature of program is used rarely.
  - small portion of table is loaded instead of whole table.

Advantages:

  - less number of I/O would be required to load or swap program into memory.
  - program would no longer be constrained by the amount of physical memory available.
  - takes less physical memory, so many program can be run at the same time.

- **Address Translation** is done by two techniques:
  - **Paging**
  - **Segmentation**

Paging(1):

- Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages**
- The size of the process is measured in the number of pages.
- Main memory is divided into small fixed-sized blocks of (physical) memory called **frames**
- The size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

Address translation

- Page address is called **logical address** and represented by **page number** and the **offset**.

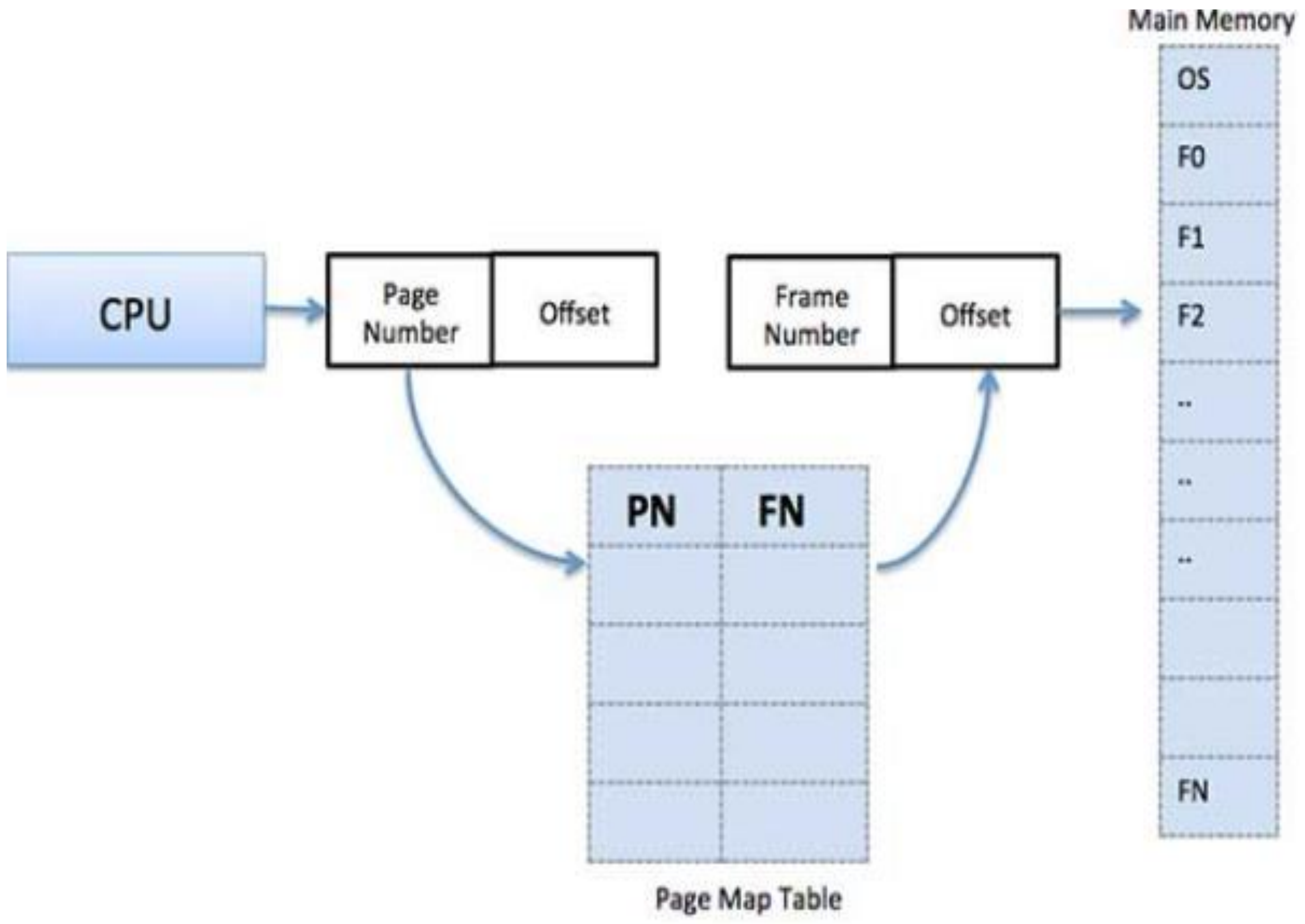
Logical address = page number + page offset

- Frame address is called **physical address** and represented by a **frame number** and the **offset**.

Physical address = frame number + page offset

Paging(2):

- A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.



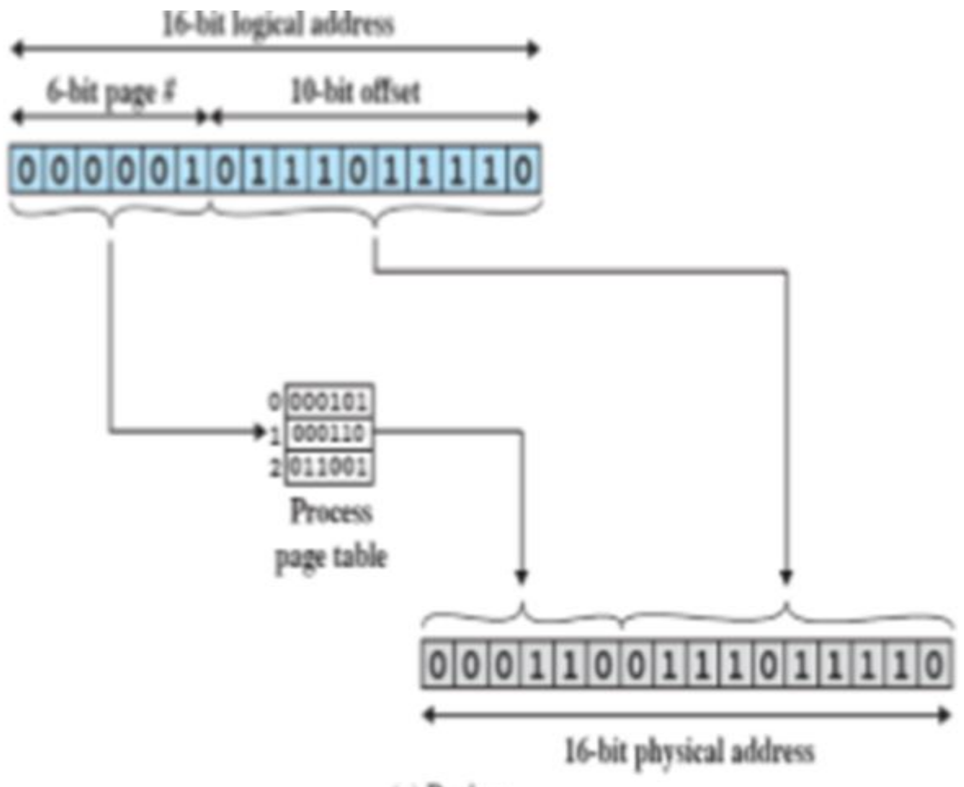
Paging(3):

Example:

In this example, we have the logical address 0000010111011110, which is page number 1, offset 478.

Suppose that this page is residing in main memory frame 6 = binary 000110.

Then the physical address is frame number 6, offset 478 = 0001100111011110



(a) Paging

Paging(4):

- When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.
- When a process is to be executed, its corresponding pages are loaded into any available memory frames.

Advantages and Disadvantages of Paging:

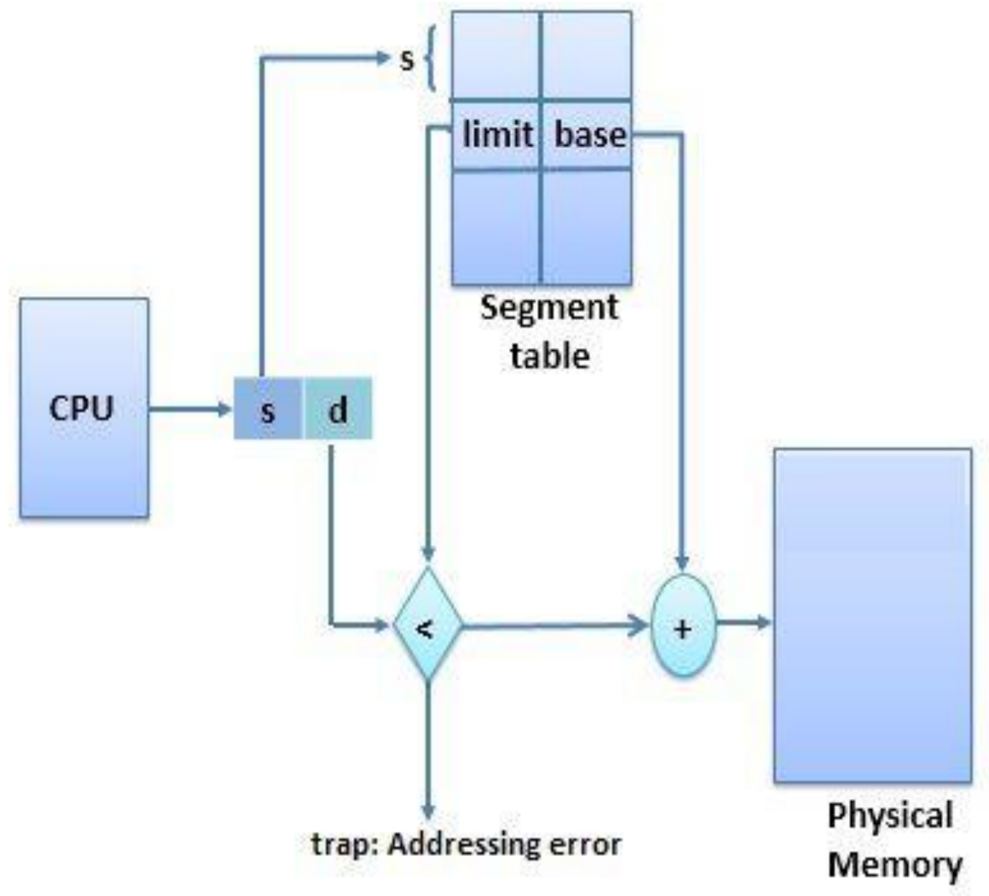
- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

Segmentation (1):

- Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions.
- Each segment is actually a different logical address space of the program.
- When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.
- Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.
- The operating system maintains a segment map table for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory
- For each segment, the table stores the starting address of the segment and the length of the segment
- The logical address space is the collection of variable size segments. Each segment has its name and length. For the execution, the segments from logical memory space are loaded to the physical memory space

Segmentation (2):

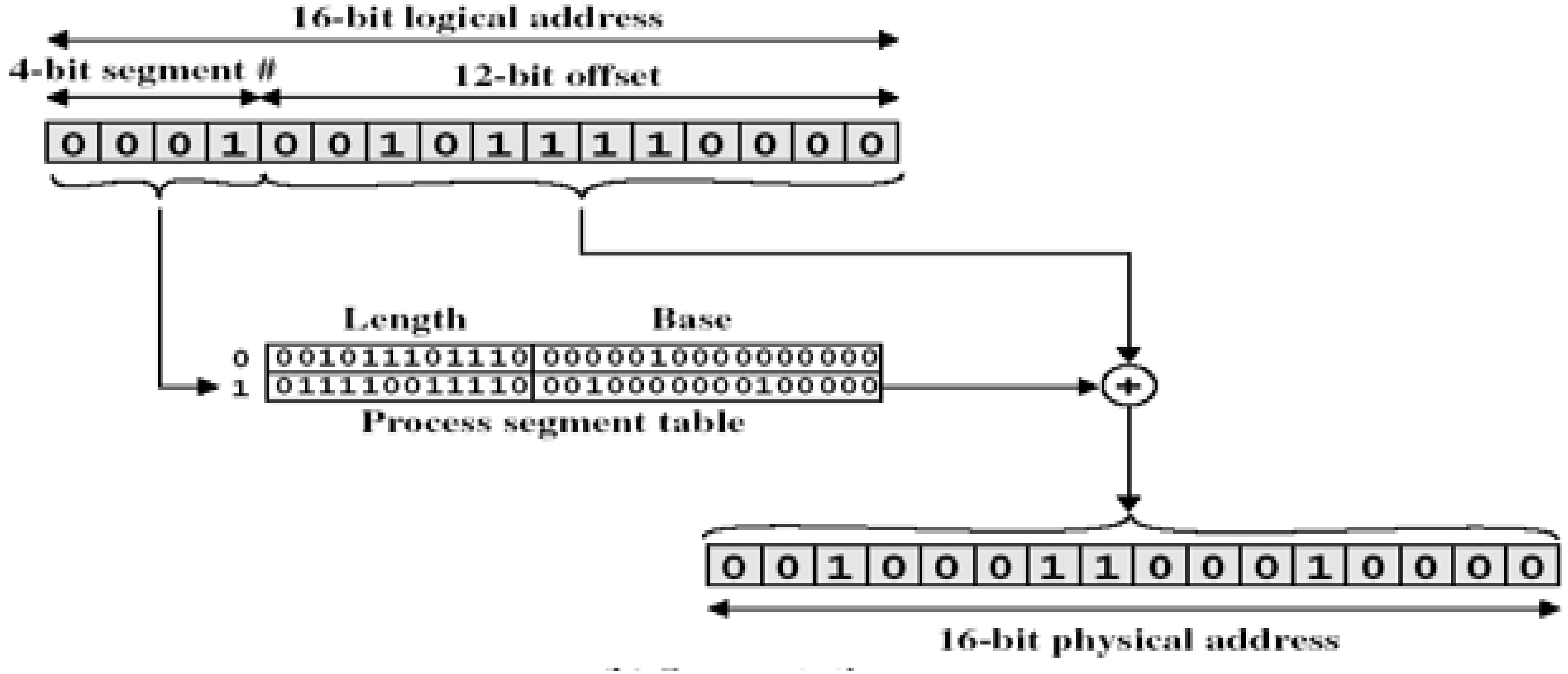
- The address specified by the user contain two quantities the segment name and the Offset.
- The segments are numbered and referred by the segment number instead of segment name.
- This segment number is used as an index in the segment table, and offset value decides the length or limit of the segment.
- The segment number and the offset together generates the address of the segment in the physical memory space.



Segmentation

Segmentation (3):

Example:

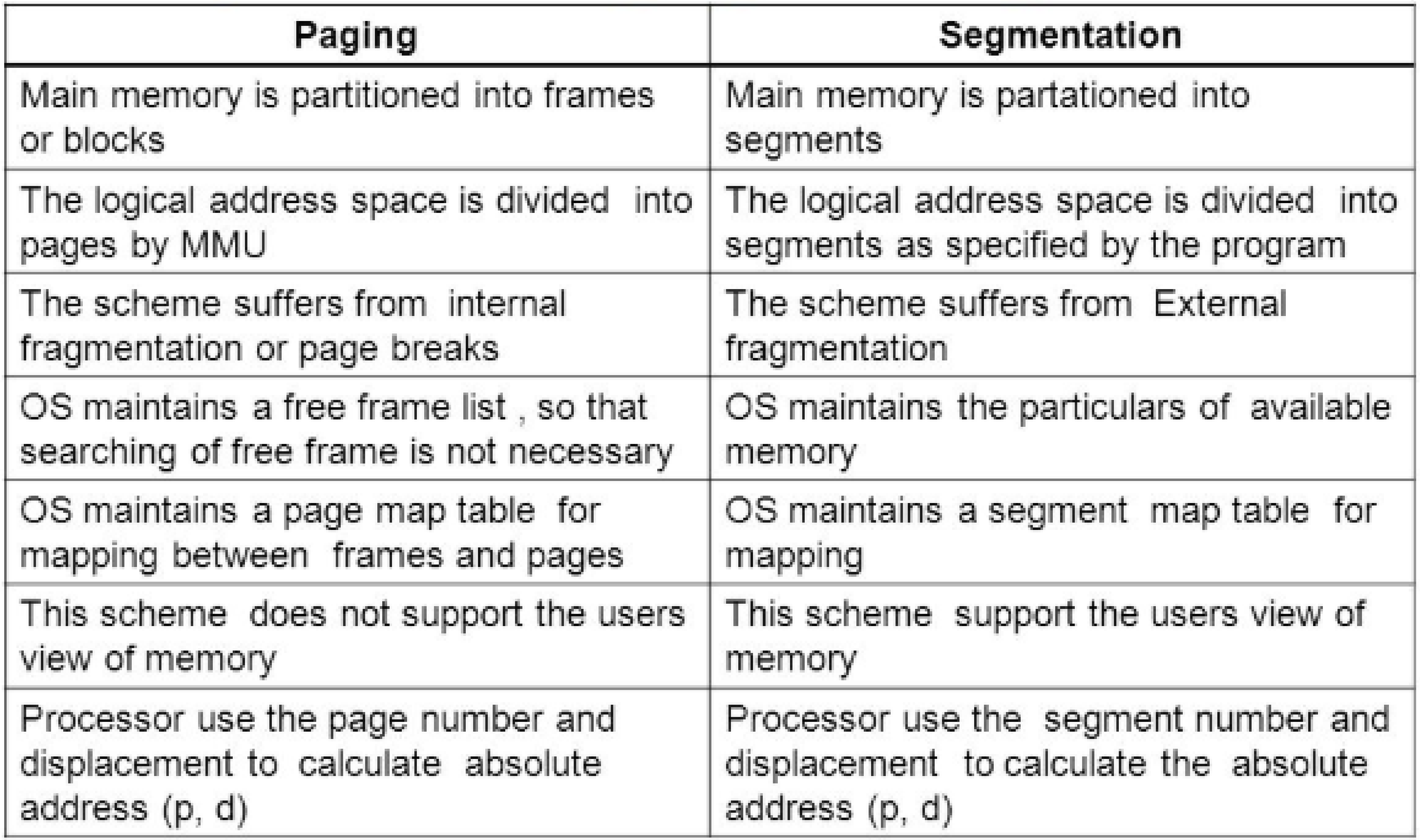


Segmentation (4):

Advantages :
- No internal fragmentation
- May save memory if segments are very small and should not be combined into one page.
- Segment tables: only one entry per actual segment as opposed to one per page in VM
- Average segment size >> average page size
- Less overhead.

Disadvantages :
- External fragmentation.
- Costly memory management algorithms.
- Segmentation: find free memory area big enough.
- Paging: keep list of free pages, any page is ok.
- Segments of unequal size not suited as well for swapping.

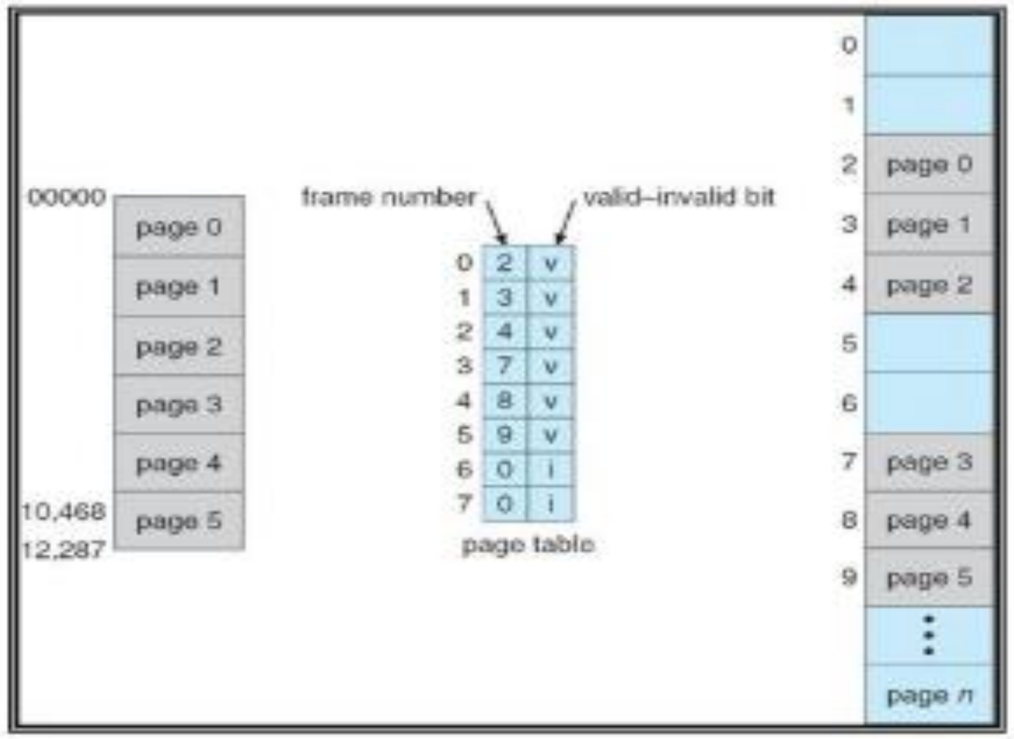
Paging Vs. Segmentation:



Paging	Segmentation
Main memory is partitioned into frames or blocks	Main memory is partationed into segments
The logical address space is divided into pages by MMU	The logical address space is divided into segments as specified by the program
The scheme suffers from internal fragmentation or page breaks	The scheme suffers from External fragmentation
OS maintains a free frame list , so that searching of free frame is not necessary	OS maintains the particulars of available memory
OS maintains a page map table for mapping between frames and pages	OS maintains a segment map table for mapping
This scheme does not support the users view of memory	This scheme support the users view of memory
Processor use the page number and displacement to calculate absolute address (p, d)	Processor use the segment number and displacement to calculate the absolute address (p, d)

Memory Protection:

- Main memory must accommodate both the operating system and user processes.
- Main memory is usually divided into two partitions:
    - Resident operating system, usually held in low memory
    - User processes then held in high memory
- In this case memory protection must be done.
- Memory protection means protecting the operating system from user processes and protecting user processes from one another
- Memory protection implemented by associating protection bit with each frame
- **Valid – Invalid** bit attached to each entry in the page table.
- **Valid –** indicates that the associated page is in the processors logical address space and is thus a legal page
- **Invalid-** indicates that the page is not in the processors logical address space.



Page Replacement Algorithm (1):

- When there are more pages than the memory frames, then the page from the frame must be loaded out so as to load the unloaded page into the memory.
- Replacing the pages in the memory follows certain algorithm.

1) **First In First Out Page Replacement Algorithm**

- The oldest page in the physical memory is the one selected for replacement
- Very simple to implement
- Keep a list on a page fault, the page at the head is removed and the new page added to the tail of the list.
- This algorithm associates with each page the time that page was brought into memory
- When a page must be replaced, the oldest page is chosen.

2) **Optimal Page Replacement**

- It is simply replace the page that will not be used for the longest period of time

Page Replacement Algorithm (2):

- It has the lowest page- fault rate.

3) Least Recently Used Algorithm

- The FIFO uses the time when a page was brought into memory; the OPT uses the time when a page is to be used.
- In LRU, replace the page that has not been used for the longest period of time.
- LRU replacement associates with each page the time of that pages last use. When a page must be replaced, LRU choses that page that has not been used for the longest period of time.
- Counter implementation
    - Every page entry has a counter, every time page is referenced through this entry, copy the clock into the counter
    - When a page needs to be changed, look at the counters to determine which are to change.

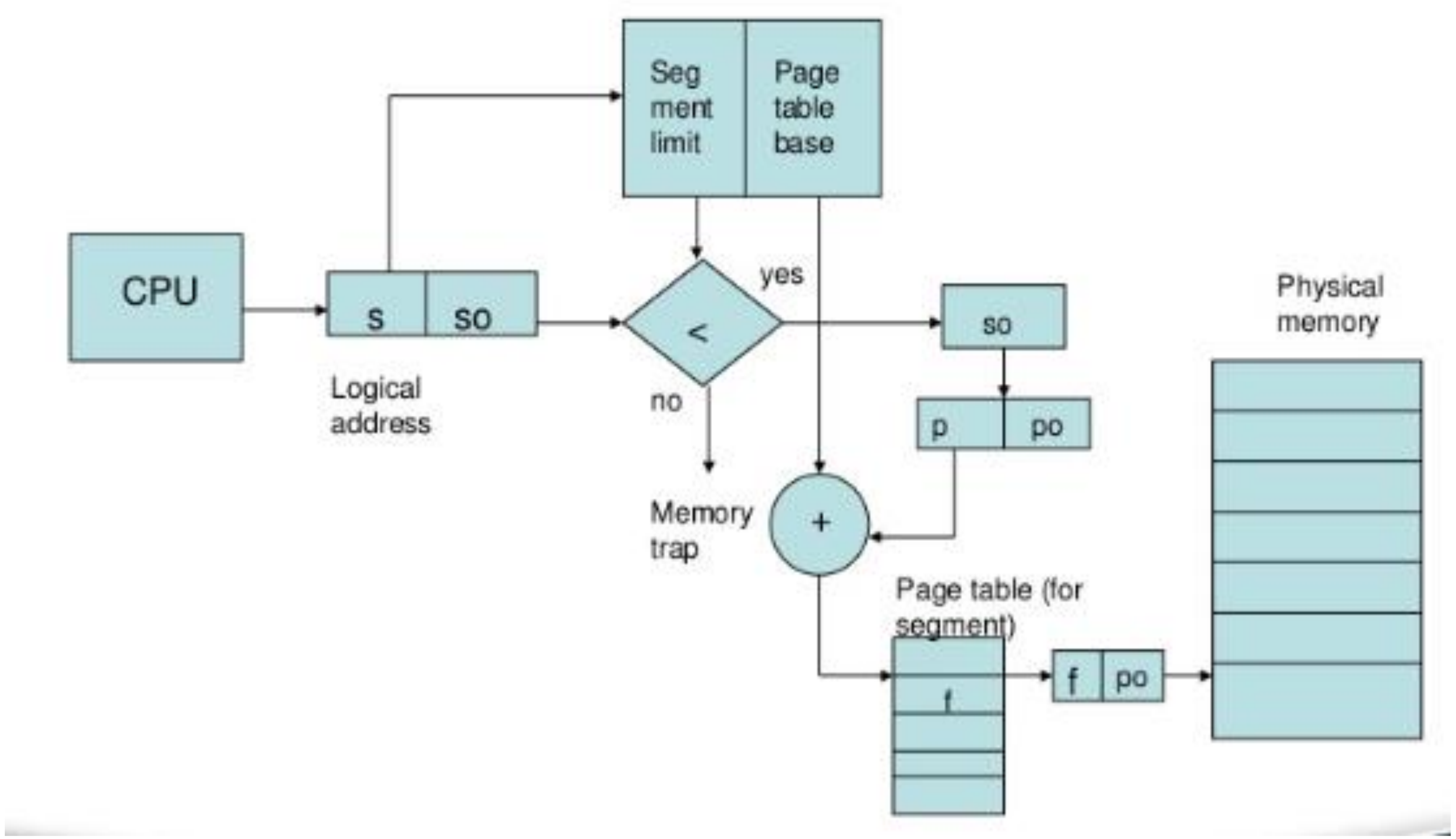
ASSIGNMENT QUESTIONS: (from old question set):

1. *What is virtual memory? Differentiate between paging and segmentation.*
2. *What is paging? What is page table? How is page table used to convert logical address to physical address? Illustrate.*
3. *What is memory hierarchy? Describe the significance of cache and virtual memory.*
4. *Differentiate between different types of cache mapping.*
5. *What is cache performance? Describe in brief the strategies used to write data to cache memory.*
6. *Why do we need virtual memory? "Paging as one dimensional virtual memory management". Justify.*
7. *What is cache? Do you agree that system works without cache? Give suitable reasons to support your answer.*
8. *Differentiate cache memory and virtual memory. Show the conversion of logical address to physical address using segmentation with paging.*
9. *Differentiate between segmentation and paging? Describe the four most common replacement algorithms related to design issues of cache memory?*
10. *What are the significance of cache memory? Write different types of mapping technique.*
11. *Show the layout of the cache for CPU that can address 1M x 16 of memory; the cache hold 8K x 16 of data and has the following mapping strategies. Given the number of bits per location and total number of locations as well.*
    1. *Fully associative*
    2. *Direct mapped*
    3. *Two-way associative*
12. *Describe how a 1KB cache is mapped directly with 1 MB main memory. Use suitable diagram to show the configuration and determine tag and index.*
13. *Explain different design issues of cache memory.*

14. *Write short Notes:*
    1. *Paging*
    2. *Memory and its types*

<u>ASSIGNMENT QUESTIONS: (from old question set):</u>

Hints Q8



<u>CHAPTER COMPLETED</u>