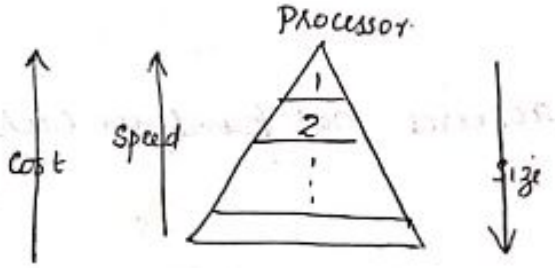
Memory Hierarchy

Memory is used for storing data.

		Speed	Cost	Size
CPU				
SRAM		fastest (0.5ns - 5ns)	highest	Smallest
DRAM		50ns - 70ns		
Magnetic disk		(5lack ns - 20 crores ns) Slowest	Lowest	largest



The memory near to the processor is ~~cache~~ main memory. As the distance from processor increases, the size of the memory increases whereas the speed decreases. The smallest memory has the highest cost. The memory near to the processor is main memory. When the processor asks for a data, it will be first searched in main memory. If it is not present there, we look for the secondary memory. The gap between main memory and secondary memory can be

Reduced by using cache memory. The cache is placed in between processor and main memory. The commonly used datas are stored in cache which in turn decreases the access time between the processor and main memory.

Cache hit: If the required memory access is found in cache, then it is called cache hit.

Cache miss: If a required memory access is not found in cache, it is called cache miss.

Hit rate: The no: of memory accesses found inside the cache for a particular amount of time is called hit rate

Miss rate: No: of memory accesses not found in cache is called miss rate

Miss penalty: Total amount of time taken to make memory access from main memory in case of a cache miss, and to deliver the amount of data access from main memory to the processor and to make a copy of the same in the cache memory.

Locality

1) Temporal Locality
2) Spacial Locality

Temporal locality ~~is~~ If an item is referenced once and keep within for further reference.

Spatial locality - When searched for a particular item, and if it is not found, reference another one for the same time.

Block / Line in cache.

Unit of information that may or may not present is a 2 level hierarchy.

Cache data transfer.

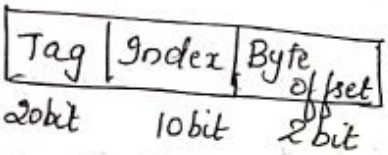
Direct mapping - (Block address) (modullo no: of blocks in cache)

When the processor requests for a data, the data is first checked in cache ~~main~~ memory. If it is found there, it is a cache hit. If it is not found in cache, access the same from main memory and deliver it to the processor, and at the same time make a copy of the same to the cache.

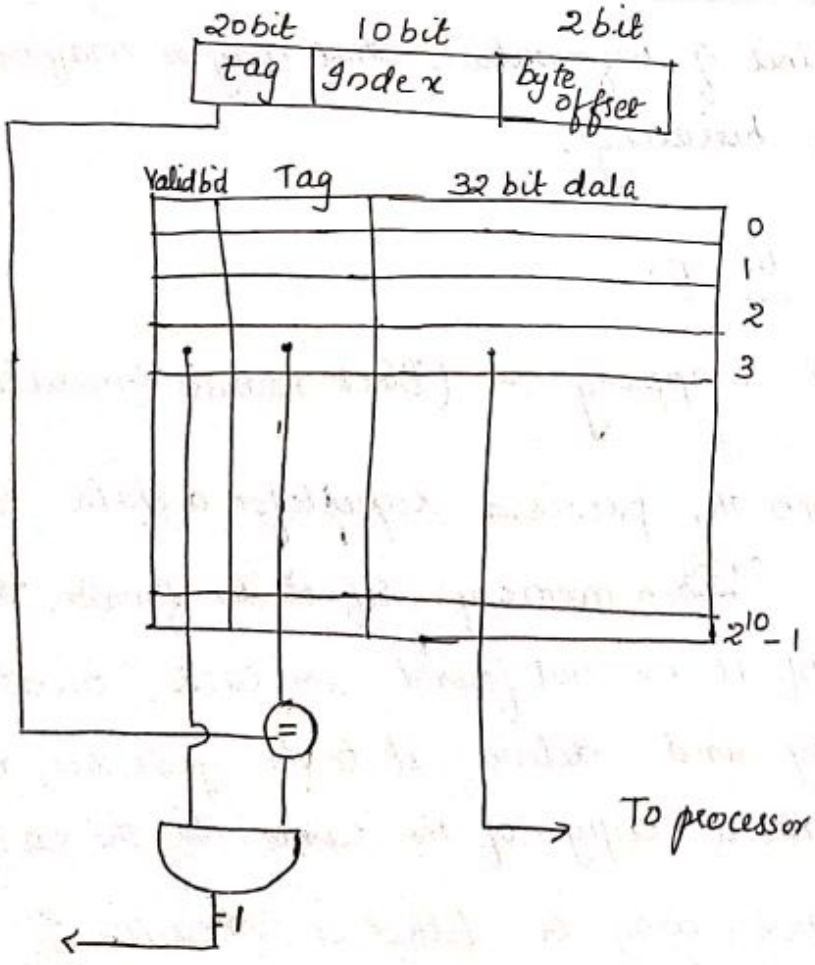
The cache locations may be filled or vacant. If the cache

is vacant, the newly taken datas from main memory can be entered into cache. Else replace the less frequently used datas with the new ones.

The requested data is checked within cache using the given format



Here tag is the most important field. Index field is of 10 bits → 2^{10} blocks can be represented.



Valid bit is either 1 or 0. If a data is present in a cache location valid bit is 1 else 0.

Handling cache miss.

When the processor request for a data and found that it is a cache miss, it has to go through a number of steps. The cache miss creates a stall which means freezing the contents of temporary programmable registers while we wait for the requested data. If it is a cache data miss, the requested data will be taken from the main memory and then deliver the same to the processor and also make a copy of same in cache. If an instruction access results in a miss, then the content of instruction register is invalid. Since the PC value is incremented just after 1st clock cycle, the instruction that generates an instruction cache miss is PC-4. The steps to be followed are

1) Send the original PC value (current value of PC-4) to memory.

2) Instruct main memory to perform a read or wait for memory to complete its access.

3) Write the cache by entering the data from memory and upper bits of address into tag field and turn the valid bit on

4) Now restart the instruction execution which will refetch the instruction and results in a cache hit.

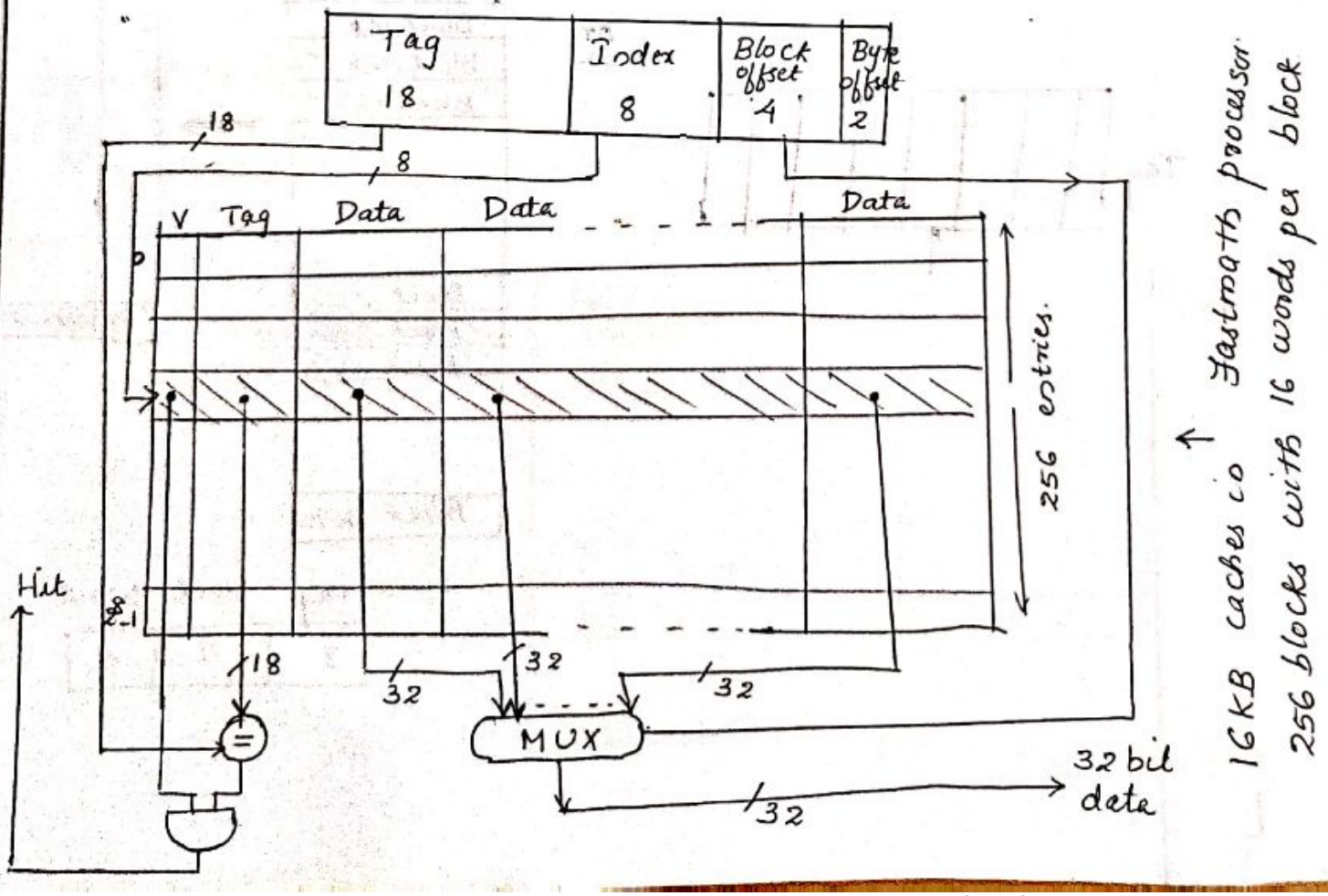
Handling cache writes.

Suppose while executing store instruction, we write data only on the data cache and not in main memory. Thus the value inside the cache and main memory are different. So that it is said to be inconsistent. The simplest way to keep the main memory and cache consistent by writing data into both memory & cache. This scheme is called write through. Although this scheme is very simple, it would not provide very good performance. The write into cache and main memory will take long time and thus the processor slow down considerably.

The solution to this problem is to use a write buffer. A write buffer stores the data while it is waiting to be written into memory. After writing the data into the cache and write buffer, the processor can continue execution. When the write to main memory is completed, the entry in the write buffer is free.

If the write buffer is full when the processor reaches write, the processor must stall until there is an empty portion in the write buffer. The time required for write from write buffer is less compared to the write of the processor.

The alternative scheme to write through is called write back scheme. In write back, when a write occurs, the new value is written only to the block in the cache. The block modified is written to the lower level of the hierarchy when it is replaced. In write back scheme, if a newly write is to be added to cache block and it is done by replacing the least recently used data. That replaced value is written back to the lower level of memory (Main memory)

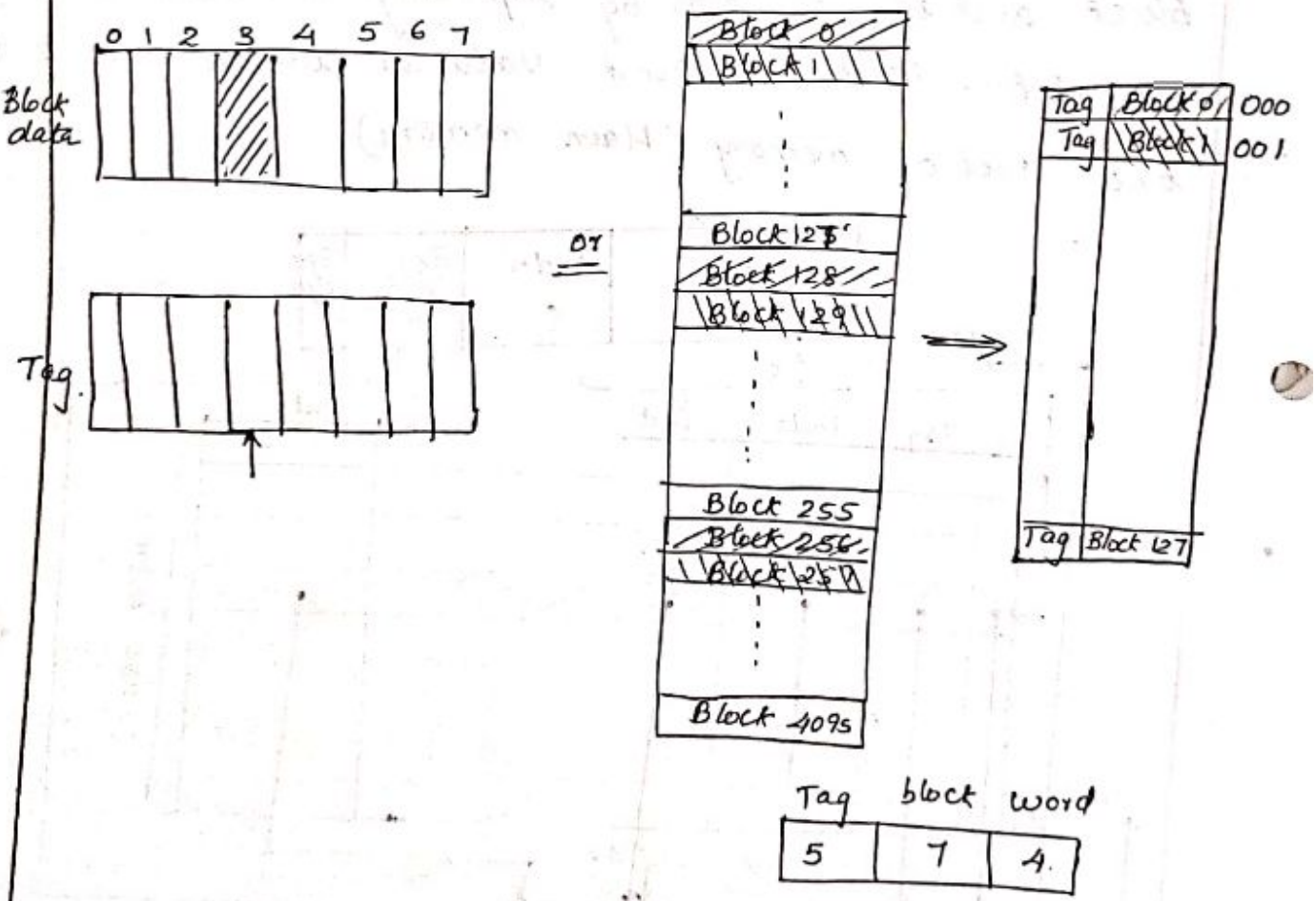


Cache Mapping Techniques.

Direct mapping Technique

In this technique, any block address in the memory ~~to~~ a is mapped to a single location of upper level of hierarchy. The block of data can be placed in exactly one cache location.

Memory block = (Block address) modulo (no. of blocks in cache)

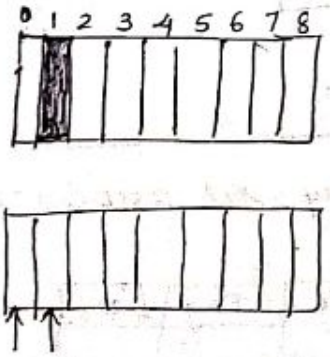


Tag	block	word
5	7	4

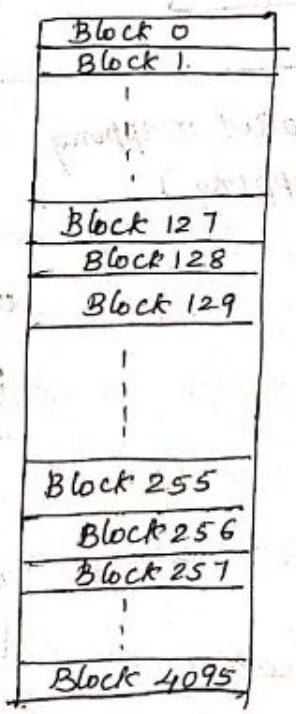
Set associative mapping.

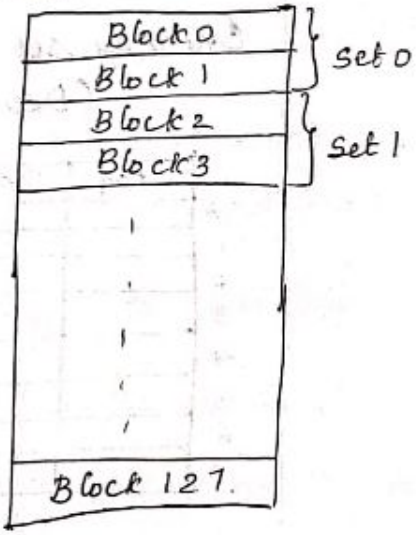
In this mapping there are fixed number of locations (at least 2) where each block can be placed. A set associative cache with n locations for a block called n-way set associative cache. It consists of a number of sets each of which contains n blocks, each block in the memory maps to a unique set in the cache given by index field and a block can be placed in any element of that set. Thus a set associative placement combines direct mapping and fully associative mapping.

Memory block = (block address) modulo (no. of sets in cache)



OR

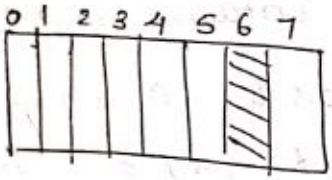


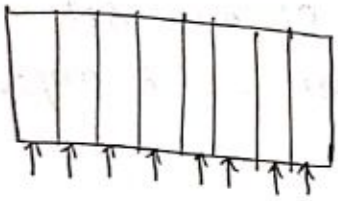


Tag (6)	Block 6	Word 4

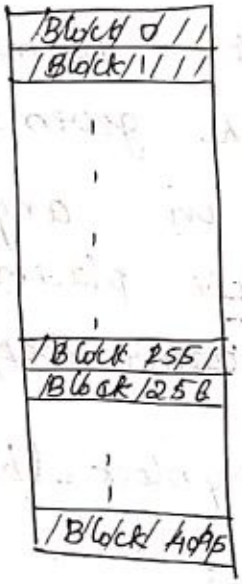
Fully associative mapping

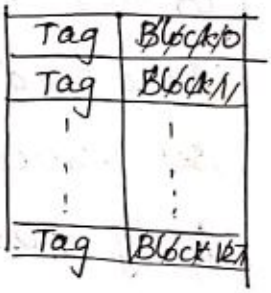
In this case the block in the main memory is associated with any block in the cache. To find a given block in a fully associative cache all entries in the cache must be searched.

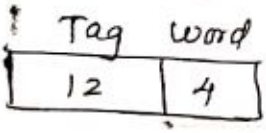




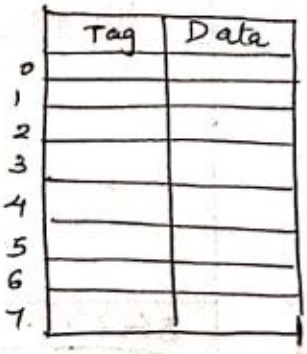
or







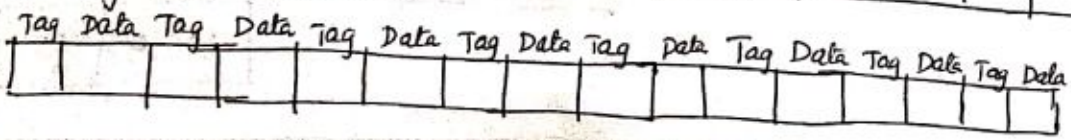
One way set associative mapping (Direct mapping)



2-way set associative

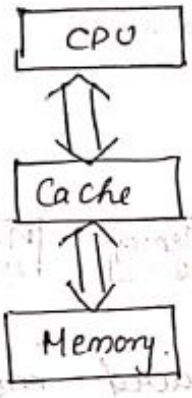
4-way set associative

8 way set associative



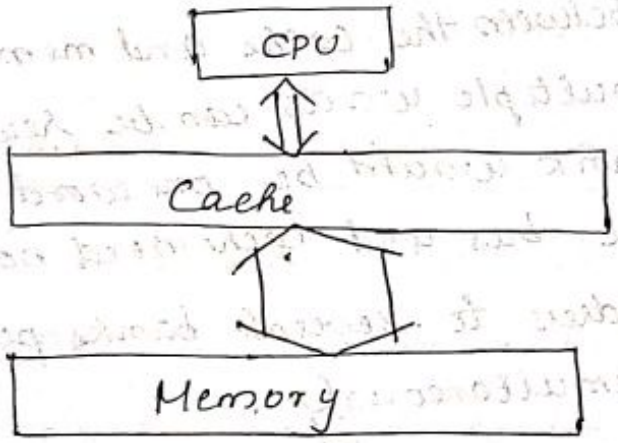
Cache Memory Organization

1) One word wide memory organization



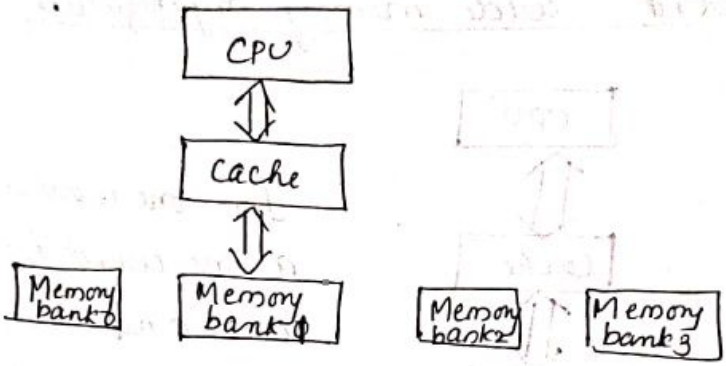
In one word wide organization a one word wide memory is used and all accesses are made sequentially.

2) Wide memory organization



In wide memory organization the bandwidth is increased and the buses between the processor and memory is widened which allows parallel access to all the words of the block. It make use of multiplexers and control logic between the processor and cache which makes them costlier than other schemes

3) Interleaved memory organization :



In Interleaved memory organization the bandwidth is increased by widening the memory but not the interconnection bus. ie: instead of making the entire path between the cache and memory wider. In this case multiple words can be read in one access time. Each bank would be one word wide so that the width of the bus and cache need not change, but sending an address to several banks permits them all to read simultaneously.

Measuring and improving cache performance.

We know that,

CPU execution time = CPU clock cycles x clock cycle time.

= (CPU clock cycle + Memory access cycles) x clock cycle time

During a cache miss we consider a delay to the processor ie: a stall included in memory access clock cycles

Memory access clock cycles = Write miss stall cycles + Read miss stall cycles

Read miss stall cycles = no: of reads in a program x no: of misses x miss penalty.

= no: of reads/program x miss rate x miss penalty

Write miss stall cycles = (no: of writes/pgm x Write miss rate x Write miss penalty) + Write buffer stall.

∴ Memory access stall cycles = Memory access/pgm x Memory access miss rate x miss penalty

Cache replacement algorithm (LRU algorithm)

In LRU scheme, the block replaced is one that has been unused for the longest time. LRU algorithm is implemented by keeping track of when each element in a set was used relative to other elements in a set. For a two way set associative cache, a bit can be kept in each set indicating when that element is referenced. Each time an element in that particular set is referenced that bit can be set.

Multilevel caches

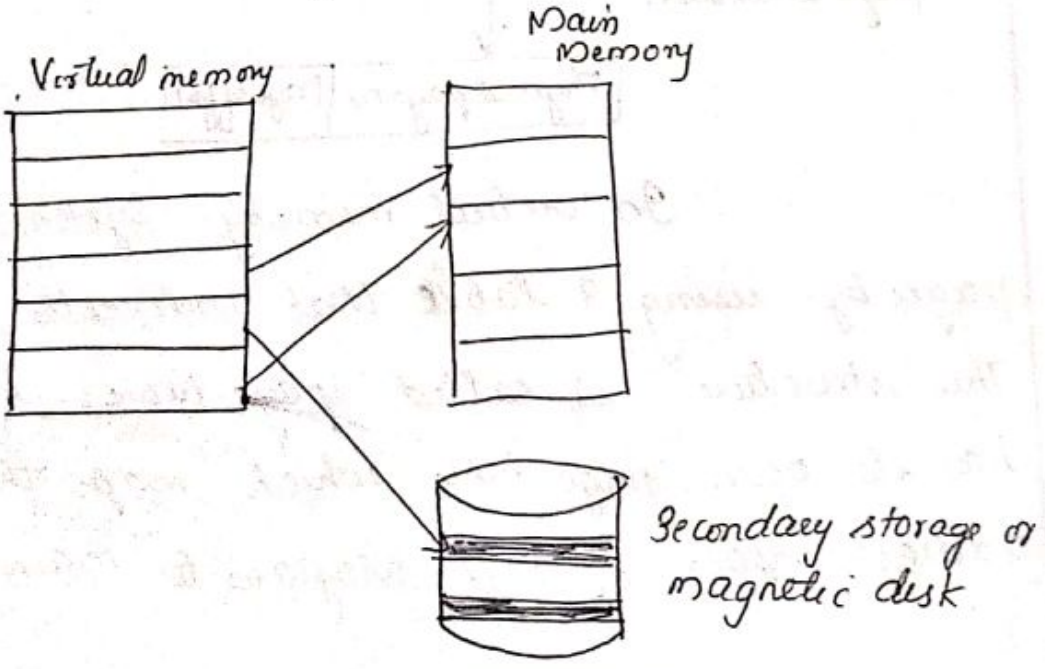
The total time requried for accessing data from main memory can be reduced by the use of cache placed b/w processor & main memory. Multiple caches can be used for this. Initially the processor looks for the requried data in 1st level of cache. If it is not found, it is a first level cache miss. The datas which are frequently used are stored in first level of cache and as the distance from processor increases the size of the multilevels of cache increases.

Virtual Memory.

The main memory can act as a cache for the secondary storage, usually implemented with magnetic disks. This technique is called virtual memory. Here we have two motivations for using virtual memory.

→ to allow efficient and safe sharing of memory among multiple programs and to remove the programming burdens of a small, limited amount of main memory.

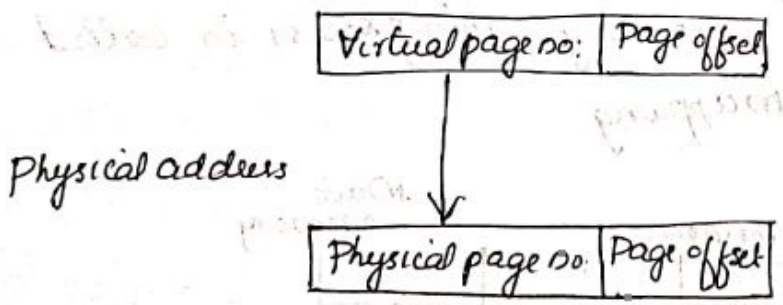
Virtual memory implements the translation of a program's address space to physical address. A virtual memory block is called a page and a virtual memory miss is called a page fault. The processor produces a virtual address which is translated by means of software and hardware to physical address which in turn can be used to access main memory. This process is called address translation or mapping



Virtual memory also simplifies loading the program for execution by providing relocation. Relocation maps the virtual address used by a program to different physical addresses before the addresses are used to access memory. This relocation allow us to load a program anywhere in main memory.

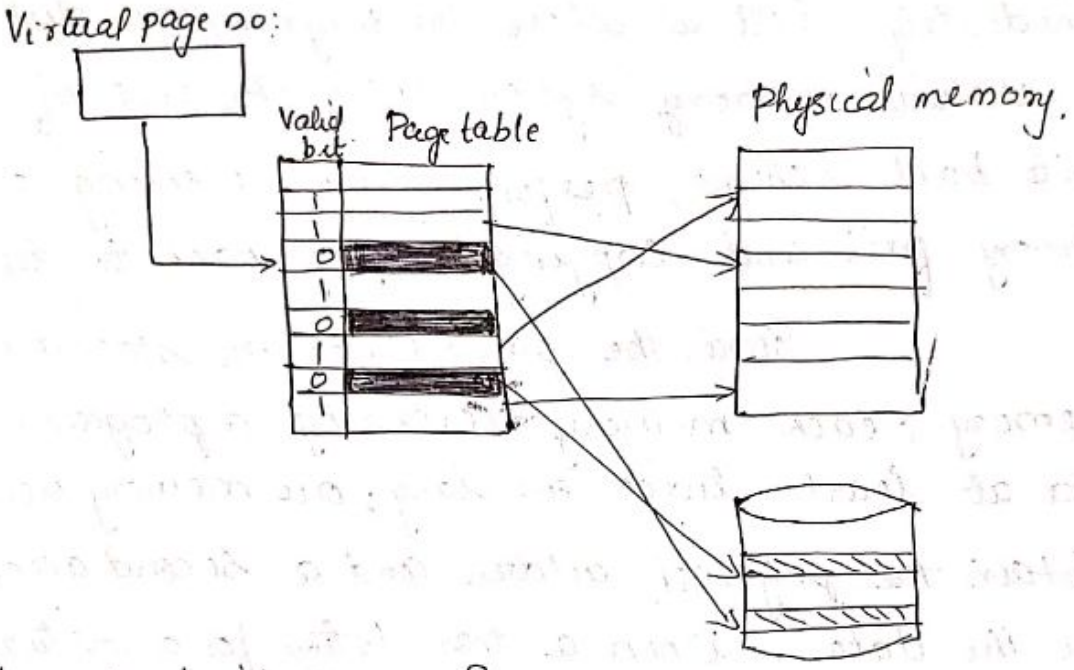
In virtual memory the address is broken into a virtual page number and a page offset. If we allow a virtual page to be mapped to any physical page, the operating system can then choose to replace any page it wants when a page fault occurs.

Virtual address

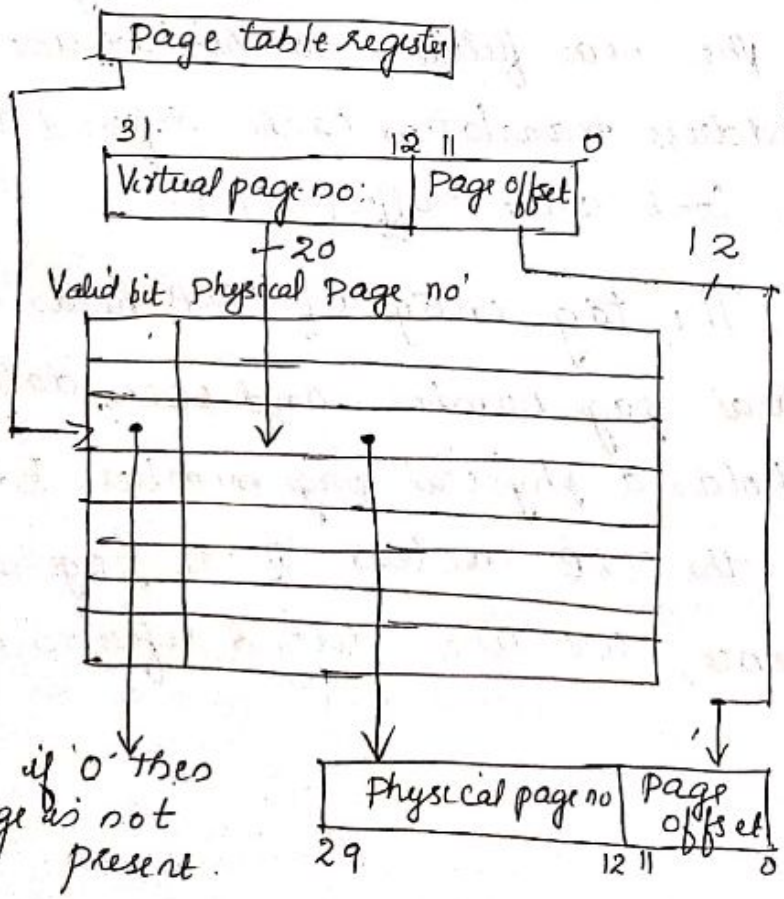


In virtual memory system, we locate pages by using a table that indexes the memory. This structure is called page table. Each program has its own page table which maps the virtual address space of that program to main memory.

To indicate the location of the start of the page table we use a page table register.



How to find a page?



Page fault occues when a valid bit of virtual page is off. The virtual address alone doesn't immediately tell us where the page is on disk. In virtual memory systems we make use of write back scheme, performing the writing on memory first and copying the page back to disk.

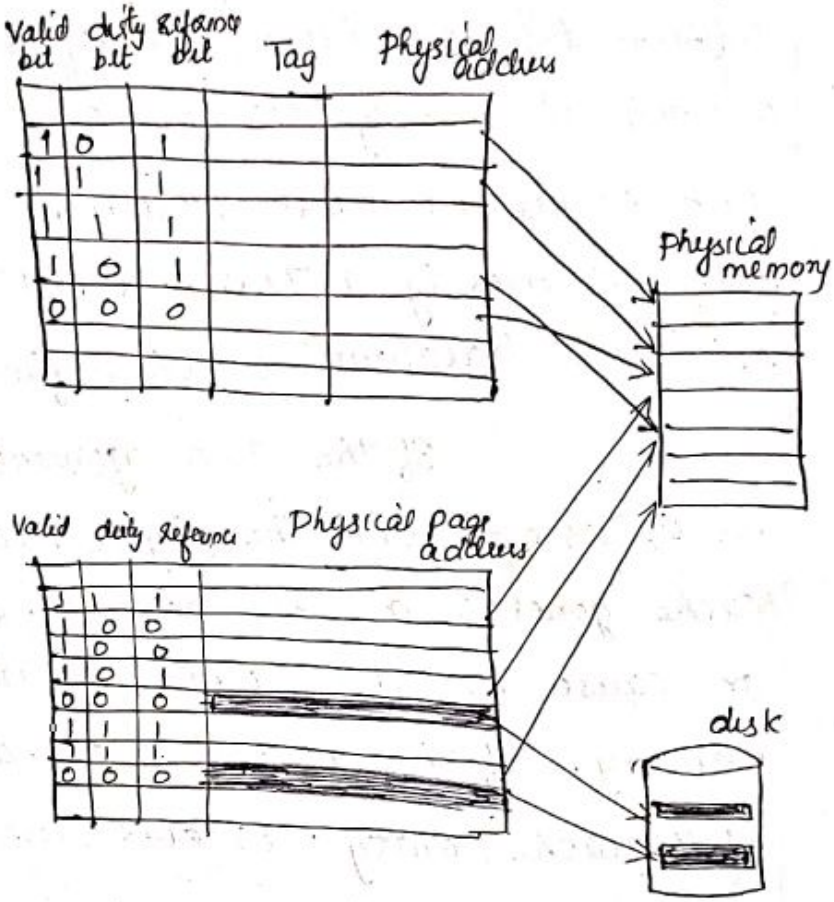
Since the page tables are stored in main memory, each memory access by a program can take at least twice as long; one memory access to obtain the physical address and a second access to get the data. When a translation for a virtual page number is used, it will probably be needed again in the near future. To this we use a special address translation cache referred to as translation look aside Buffer (TLB)

The tag entry of TLB holds a portion of the virtual page number and each data entry of TLB holds a physical page number. Because we access the TLB instead of the page table on every reference, we use dirty & reference bits in TLB.

If we get a TLB hit, the physical page no: is used to form the address and the corresponding reference bit is turned on. If the processor is performing a write the duty bit is also turned on. If a miss in TLB occurs we must determine whether it is a page fault or merely a TLB miss. TLB miss can be handled either in hardware or in software.

If the TLB generates a hit, the cache can be accessed with the physical address. For a read, the cache generates a hit or miss and supplies the data or causes a stall while the data is brought from memory. If the operation is a write, a portion of the cache entry is ~~over~~ written for a hit and the data is sent to the write buffer if we assume write through. A write miss is just like a read miss except that the block is modified after it is read from memory. A TLB hit and a cache hit are independent events and a cache hit can occur only after a TLB hit occurs, which means that the data must be present in memory.

Virtual page no



Flow chart