4.2 MEMORY CHIP ORGANISATION

A memory consists of cells in the form of an array. The basic element of the semiconductor memory is the **cell**. Each cell is capable of storing one bit of information. Each row of the cells constitutes a memory words and all cells of a row are connected to a common line referred to as a **word line**. AW×b memory has w words, each word having b number of bits. The bellow Fig 1 shows the 16x8 memory.

The basic memory element called cell can be in two states (0 or 1). The data can be written into the cell and can be read from it.

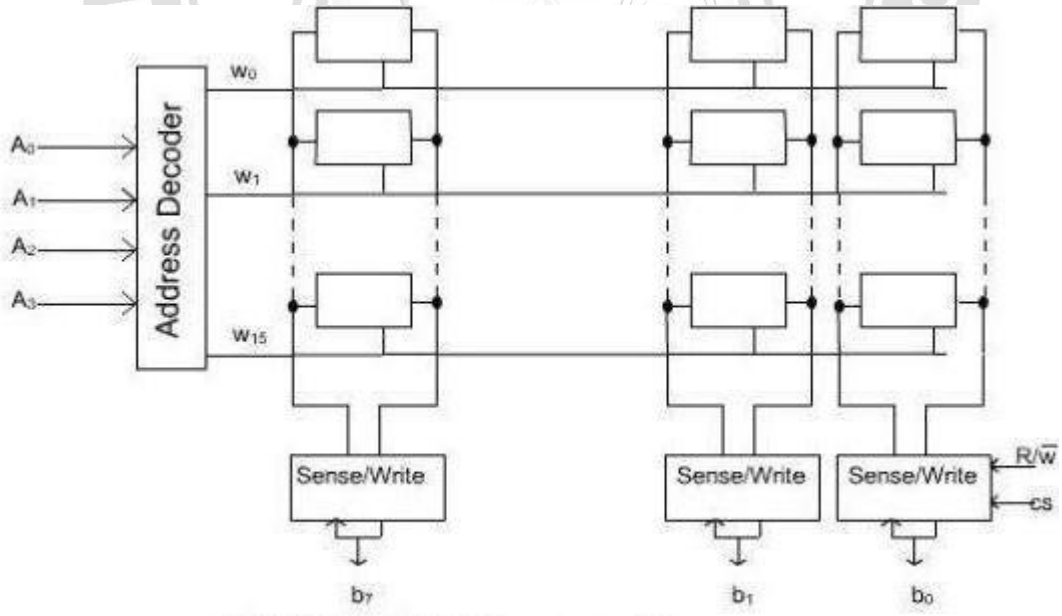


Fig 1: Organization of 16 x 8 memory

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach

- In the above diagram there are 16 memory locations named as w_0, w_1, $w_3 \ldots w_{15}$. Each location can store at most 8 bits of data (b_0, b_1, $b_3 \ldots$ b_7). Each location (w_n)

is the word line. The word line of Fig 4.4 is 8.

- Each row of the cell is a memory word. The memory words are connected to a common line termed as word line. The word line is activated based on the address it receives from the address bus.
- An address decoder is used to activate a word line.
- The cells in the memory are connected by two **bit lines** (column wise). These are connected to data input and data output lines through sense/ write circuitry.
- **Read Operation:** During read operation the sense/ write circuit reads the information by selecting the cell through word line and bit lines. The data from this cell is transferred through the output data line.
- **Write Operation:** During write operation, the sense/ write circuitry gets the data and writes into the selected cell.
- The data input and output line of sense / write circuit is connected to a bidirectional data line.
- It is essential to have n bus lines to read 2^n words.

Organization of 1M x 1 memory chip:

The organization of 1024 x 1 memory chips, has 1024 memory words of size 1 bit only. The size of data bus is 1 bit and the size of address bus is 10 bits. A particular memory location is identified by the contents of memory address bus. A decoder is used to decode the memory address.

Organization of memory word as a row:

The whole memory address bus is used together to decode the address of the specified location.

The bellow Fig 2 represents Organization of memory word as row

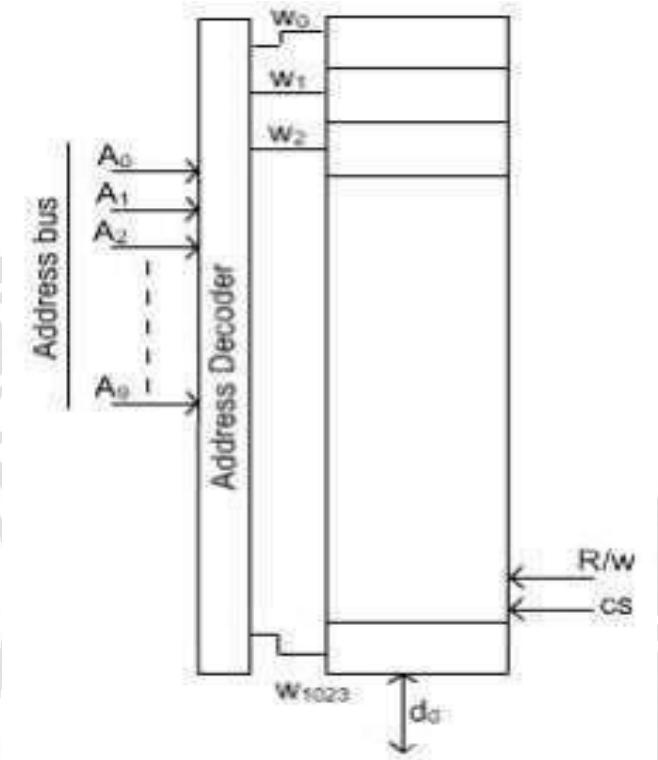


Fig 2: Organization of memory word as row

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach

Organization of several memory words in row:

The bellow Fig represents Organization of several memory word as row.

- One group is used to form the row address and the second group is used to form the column address.
- The 10-bit address is divided into two groups of 5 bits each to form the row and column address of the cell array.
- A row address selects a row of 32 cells, all of which could be accessed in parallel. Regarding the column address, only one of these cells is connected to the external data line via the input output multiplexers

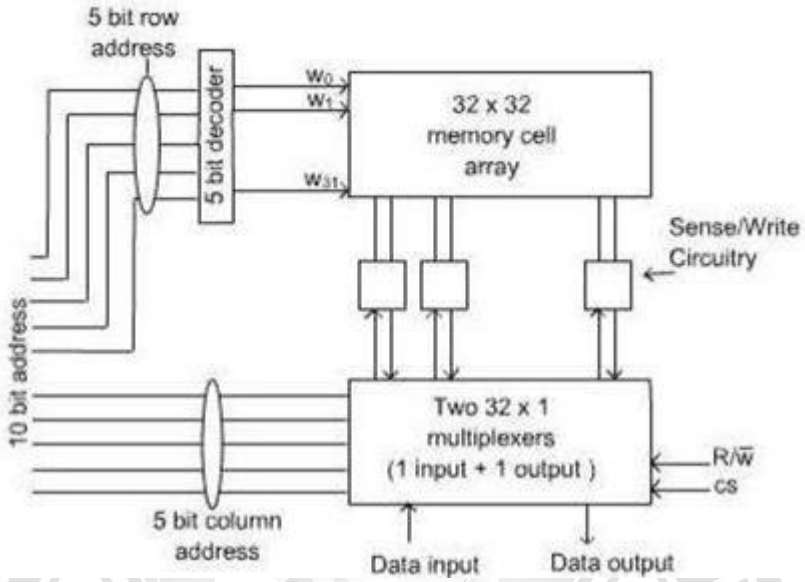


Fig 3: Organization of several memory words in row

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach

Signals used in memory chip:

- A memory unit of 1MB size is organized as 1M x 8 memory cells.
- It has got220 memory location and each memory location contains 8 bits of information.
- The size of address bus is20 and the size of data bus is 8.
- The number of pins of a memory chip depends on the data bus and address bus of the memory module.
- To reduce the number of pins required for the chip, the cells are organized in the form of a square array.
- The address bus is divided into two groups, one for column address and other one is for row address.
- In this case, high- and low-order 10 bits of 20-bitaddress constitute of row and column address of a given cell, respectively.
- In order to reduce the number of pin needed for external connections, the row and

column addresses are multiplexed on tenpins.

- During a Read or a Write operation, the row address is applied first. In response to a signal pulse on the Row Address Strobe (RAS) input of the chip, this part of the address is loaded into the row address latch.
- All cell of this particular row is selected. Shortly after the row address is latched, the column address is applied to the address pins. The bellow Fig 4 shows signals in accessing the memory.
- It is loaded into the column address latch with the help of Column Address Strobe (CAS) signal, similar to RAS.
- The information in this latch is decoded and the appropriate Sense/Write circuit is selected.

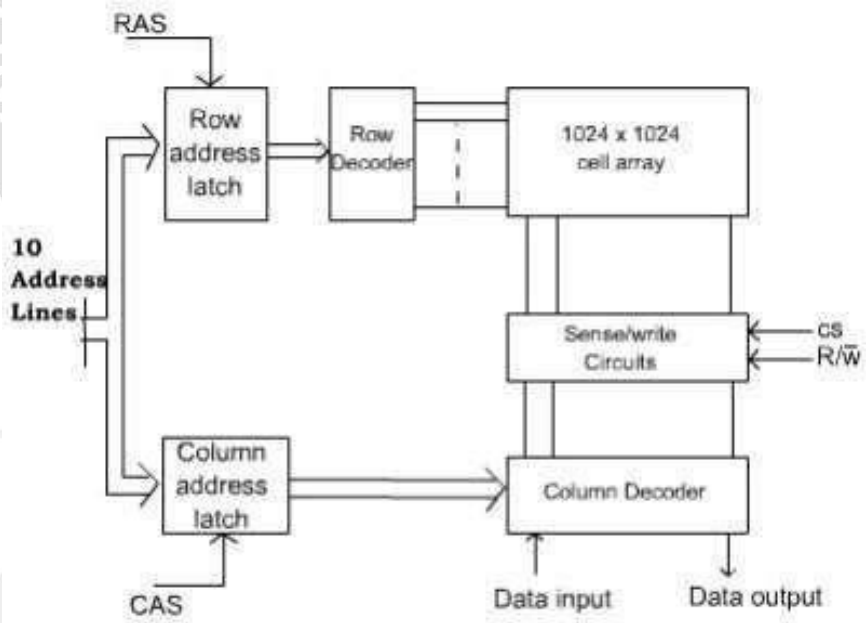


Fig 4: Signals in accessing the memory

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach

- Each chip has a control input line called Chip Select (CS). A chip can be enabled to accept data input or to place the data on the output bus by setting its

Chip Select input to 1.

- The address bus for the 64K memory is 16 bits wide.
- The high order two bits of the address are decoded to obtain the four chip select control signals.
- The remaining 14 address bits are connected to the address lines of all the chips.
- They are used to access a specific location inside each chip of the selected row.
- The R/ W inputs of all chips are tied together to provide a common read / write control.

4.2.1 CACHE MEMORY

The cache memory exploits the locality of reference to enhance the speed of the processor.

Cache memory or CPU memory, is high-speed SRAM that a processor can access more quickly than a regular RAM. This memory is integrated directly into the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU.

The cache memory stores instructions and data that are more frequently used or data that is likely to be used next. The processor looks first in the cache memory for the data. If it finds the instructions or data then it does perform a more time-consuming reading of data from larger main memory or other data storage devices.

The processor do not need to know the exact location of the cache. It can simply issue read and write instructions. The cache control circuitry determines whether the requested data resides in the cache.

- **Cache and temporal reference:** When data is requested by the processor, the data should be loaded in the cache and should be retained till it is needed again.
- **Cache and spatial reference:** Instead of fetching single data, a contiguous block of data is loaded into the cache.

Terminologies in Cache

- **Split cache:** It has separate data cache and a separate instruction cache. The two caches work in parallel, one transferring data and the other transferring

instructions.

- **A dual or unified cache:** The data and the instructions are stored in the same cache. A combined cache with a total size equal to the sum of the two split caches will usually have a better hit rate.
- **Mapping Function:** The correspondence between the main memory blocks and those in the cache is specified by a mapping function.
- **Cache Replacement:** When the cache is full and a memory word that is not in the cache is referenced, the cache control hardware must decide which block should be removed to

  create space for the new block that contains the referenced word. The collection of rules for making this decision is the replacement algorithm.

Cache performance:

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache. If the processor finds that the memory location is in the cache, a **cache hit** has said to be occurred. If the processor does not find the

Hit ratio = hit / (hit + miss) = Number of hits / Total accesses to the cache

memory location in the cache, a **cache miss** has occurred. When a cache miss occurs, the cache replacement is made by allocating a new entry and copies in data from main memory. The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

Miss penalty or cache penalty is the sum of time to place a bock in the cache and time to deliver the block to CPU.

Miss Penalty = time for block replacement + time to deliver the block to CPU

Cache performance can be enhanced by using higher cache block size, higher associativity, reducing miss rate, reducing miss penalty, and reducing the time to hit in the cache. CPU execution Time of a given task is defined as the time spent by the system executing that task, including the time spent executing run-time or system services.

CPU execution time=(CPU clock cycles + memory stall cycles (if any)) x Clock cycle time

The **memory stall cycles** are a measure of count of the memory cycles during which the CPU is waiting for memory accesses. This is dependent on caches misses and cost per miss (cache penalty).

Memory stall cycles = number of cache misses x miss penalty

- Instruction Count x (misses/ instruction) x miss penalty
- Instruction Count (IC) x (memory access/ instruction) x miss penalty
- IC x Reads per instruction x Read miss rate X Read miss penalty + IC x Write per instruction x Write miss rate X Write miss penalty

Misses / instruction = (miss rate x memory access)/ instruction

Issues in Cache memory:

- **Cache placement:** where to place a block in the cache?
- **Cache identification:** how to identify that the requested information is available in the cache or not?
- **Cache replacement:** which block will be replaced in the cache, making way for an incoming block?

Cache Mapping Policies:

The given Fig 5 represents the Catch Mapping. These policies determine the way of loading the main memory to the cache block. Main memory is divided into equal size partitions called as **blocks or frames.** The **c**ache memory is divided into fixed size partitions called as **lines.** During cache mapping, block of main memory is copied to the cache and further access is made from the cache not from the main memory.

Cache mapping is a technique by which the contents of main memory are brought into the cache memory.

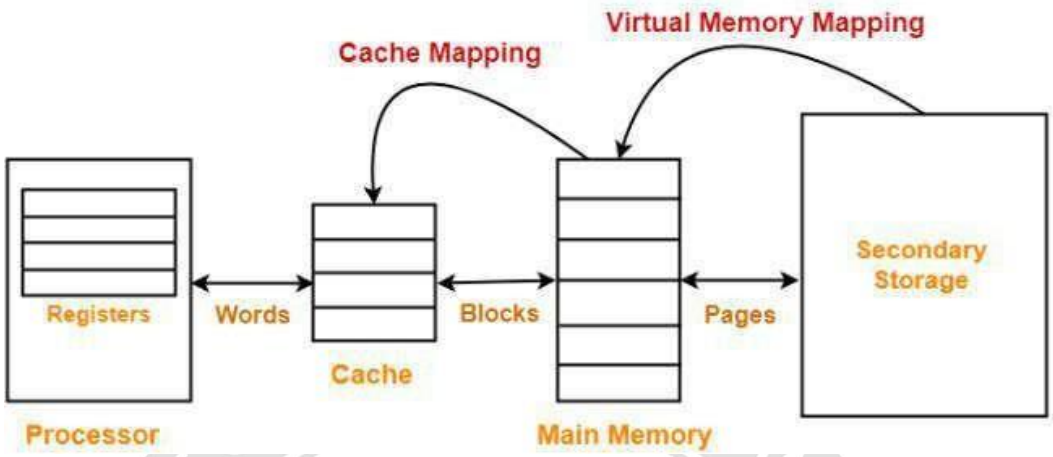


Fig 5: Cache mapping

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach

There are three different cache mapping policies or mapping functions:

- Direct mapping
- Fully Associative mapping
- Set Associative mapping

Direct Mapping

- The simplest technique is direct mapping that maps each block of main memory into only one possible cache line. The following Fig 6 shows the Direct Memory Mapping.
- Here, each memory block is assigned to a specific line in the cache.
- If a line is previously taken up by a memory block and when a new block needs to be loaded, then the old block is replaced.

The direct mapping concept is if the i^{th} block of main memory has to be placed at the j^{th} block of cache memory $j = i \% (number\ of\ blocks\ in\ cache\ memory)$

- Direct mapping' s performance is directly proportional to the It's ratio.
- Consider a 128 block cache memory. Whenever the main memory blocks 0,

128, 256 are loaded in the cache, they will be allotted cache block 0, since j= (0 or 128 or 256) % 128 is zero).

- Contention or collision is resolved by replacing the older contents with latest contents.
- The placement of the block from main memory to the cache is determined from the 16 bit memory address.
- The lower order four bits are used to select one of the 16 words in the block.
- The 7 bit block field indicates the cache position where the block has to be stored.
- The 5 bit tag field represents which block of main memory resides inside the cache. This method is easy to implement but is not flexible.
- **Drawback:** The problem was that every block of main memory was directly mapped to the cache memory. This resulted in high rate of conflict miss. Cache memory has to be very frequently replaced even when other blocks in the cache memory were present as empty.

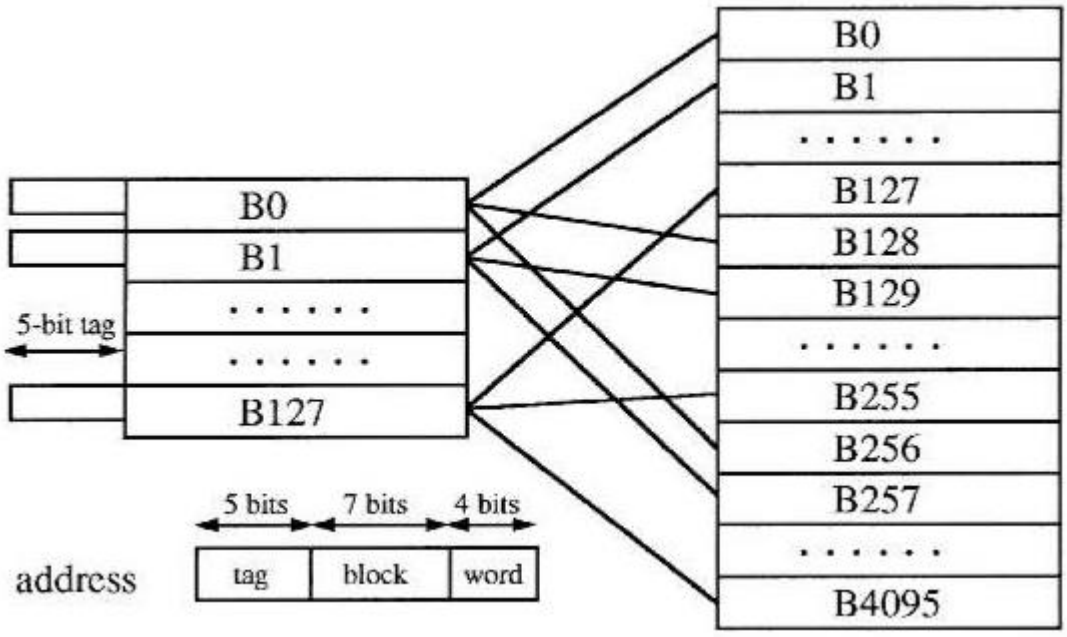


Fig 6: Direct memory mapping

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach

Associative Mapping:

- The associative memory is used to store content and addresses of the memory word. The bellow Fig 7 shows the Associative Memory Mapping.
- Any block can go into any line of the cache. The 4 word id bits are used to identify which word in the block is needed and the remaining 12 bits represents the tag bit that identifies the main memory block inside the cache.
- This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form.
- The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to check, if the desired block is present. Hence it 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.

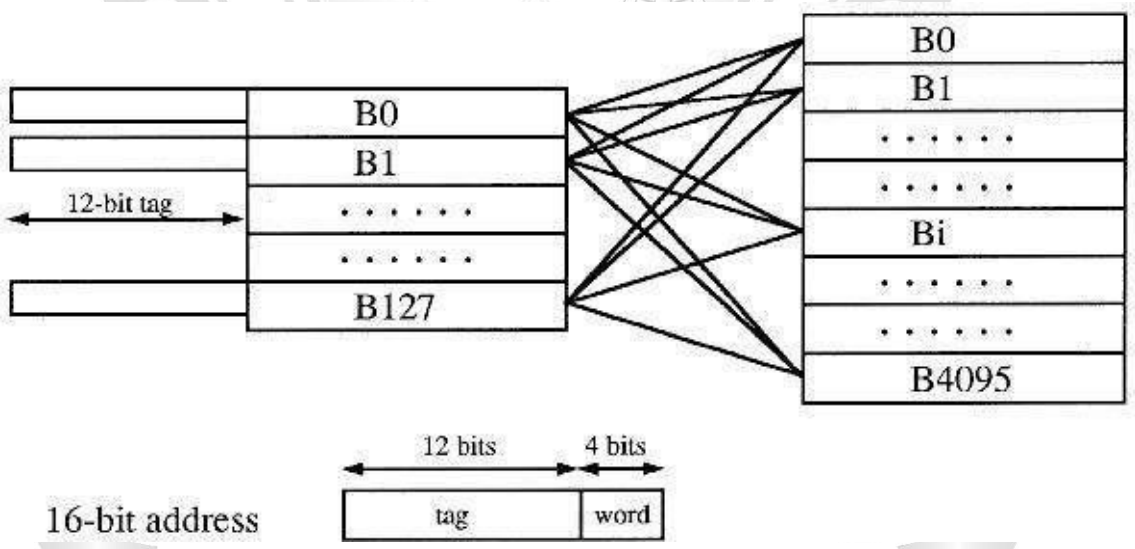


Fig 7: Associative Mapping

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach

Set associative mapping:

- It is the combination of direct and associative mapping technique. The bellow Fig 8 shows Set associative mapping. Cache blocks are grouped into sets and mapping allow block of main memory to reside into any block of a specific set.
- This reduces contention problem (issue in direct mapping) with low hardware cost (issue in associative mapping). 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. 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.

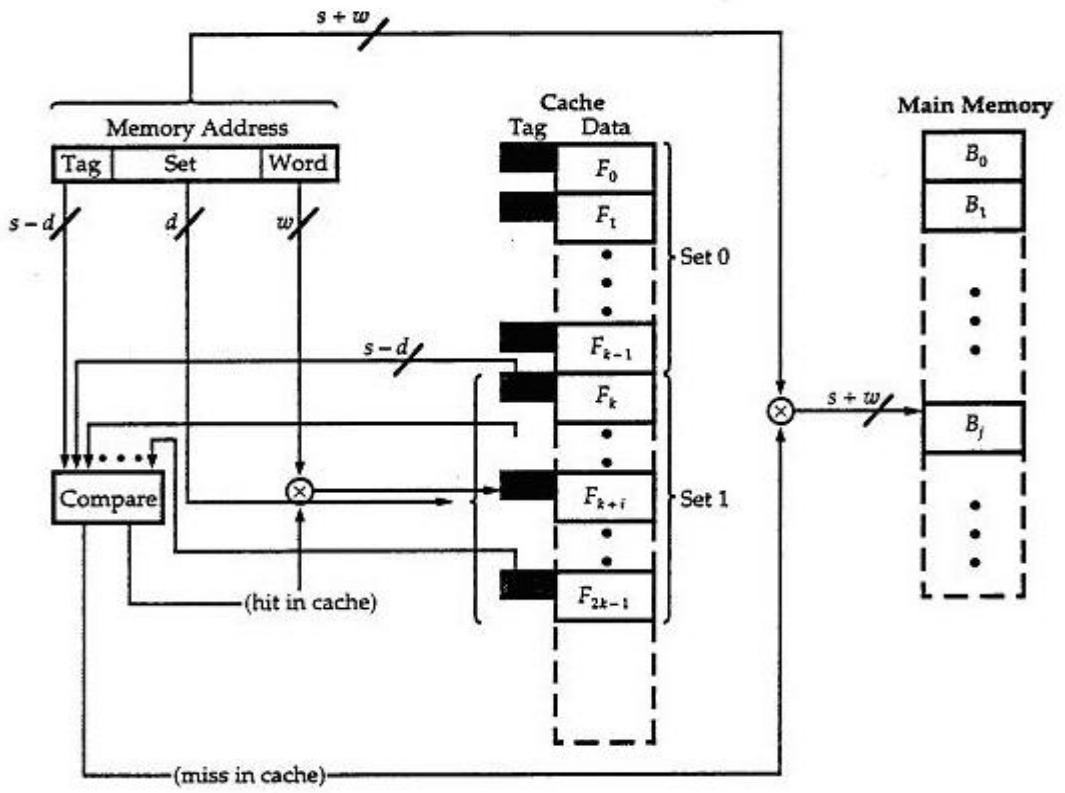


Fig 8: Set associative mapping

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach

Handling Cache misses:

When a program accesses a memory location that is not in the cache, it is called a cache miss. The performance impact of a cache miss depends on the latency of fetching the data from the next cache level or main memory. The cache miss handling is done with the processor control unit and with a separate controller that initiates the memory access and refills the cache. The following are the steps taken when a cache miss occurs:

- Send the original PC value (PC - 4) to the memory.
- Instruct main memory to perform a read and wait for the memory to complete its access.
- Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.
- Restart the instruction execution at the first step, which will refetch the instruction, this time finding it in the cache.

Writing to a cache:

- Suppose on a store instruction, the data is written into only the data cache (without changing main memory); then, after the write into the cache, memory would have a different value from that in the cache. This leads to inconsistency. Fig 9 shows the Cache Memory.
- The simplest way to keep the main memory and the cache consistent is to always write the data into both the memory and the cache. This scheme is called write-through.

Write through is a scheme in which writes always update both the cache and the memory, ensuring that data is always consistent between the two.

- With a write-through scheme, every write causes the data to be written to main memory.

  These writes will take a long time.
- A potential solution to this problem is deploying write buffer.

- A write buffer stores the data while it is waiting to be written to memory.
- After writing the data into the cache and into the write buffer, the processor can continue execution.
- When a write to main memory completes, the entry in the write buffer is freed.
- If the write buffer is full when the processor reaches a write, the processor must stall until there is an empty position in the write buffer.
- If the rate at which the memory can complete writes is less than the rate at which the processor is generating writes, no amount of buffering can help because writes are being generated faster than the memory system can accept them.

Write buffer is a queue that holds data while the data are waiting to be written to memory.

i) The rate at which writes are generated may also be less than the rate at which the memory can accept them, and yet stalls may still occur. To reduce the occurrence of such stalls, processors usually increase the depth of the write buffer beyond a single entry.

ii) Another alternative to a write-through scheme is a scheme called write-back. When a write occurs, the new value is written only to the block in the cache.

iii) The modified block is written to the lower level of the hierarchy when it is replaced.

iv) Write-back schemes can improve performance, especially when processors can generate writes as fast or faster than the writes can be handled by main memory; a write-back scheme is, however, more complex to implement than write-through.

Write-back is a scheme that handles writes by updating values only to the block in the cache, then writing the modified block to the lower level of the hierarchy when the block is replaced.

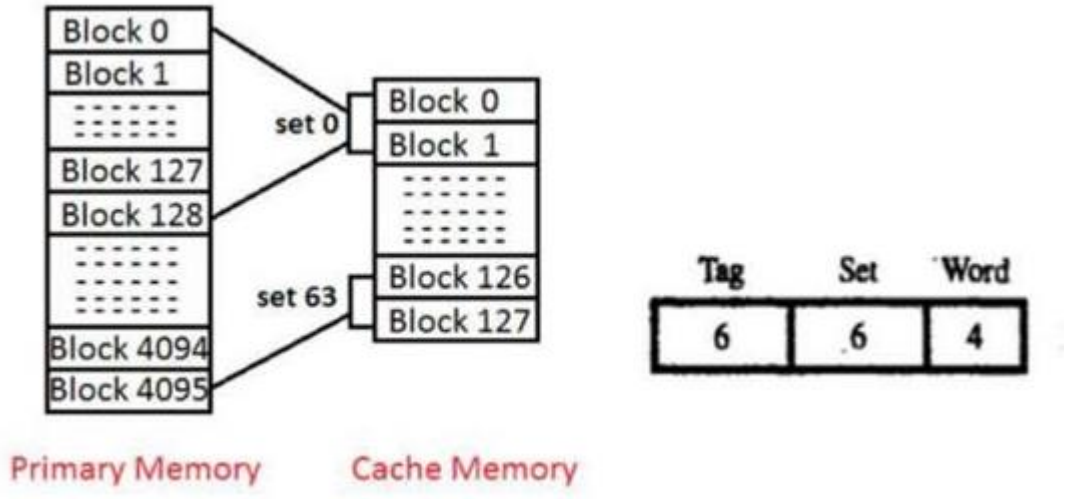


Fig 9 Cache Memory

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach

Cache Replacement Algorithms

When a main memory block needs to be brought into the cache while all the blocks are occupied, then one of them has to be replaced. This selection of the block to be replaced is using cache replacement algorithms. Replacement algorithms are only needed for associative and set associative techniques. The following are the common replacement techniques:

- **Least Recently Used (LRU):** This replaces the cache line that has been in the cache the longest with no references to it.
- **First-in First-out (FIFO):** This replaces the cache line that has been in the cache the longest.
- **Least Frequently Used (LFU):** This replaces the cache line that has experienced the fewest references.

- **Random:** This picks a line at random from the candidate lines.

Example 4.1: Program P runs on computer A in 10 seconds. Designer says clock rate can be increased significantly, but total cycle count will also increase by 20%. What clock rate do we need on computer B for P to run in 6 seconds? (Clock rate on A is 100 MHz). The new machine is B. We want CPU Time_B = 6 seconds.

We know that Cycles count_B = 1.2 Cycles count_A. Calculate Cycles count_A. CPU Time_A = 10 sec. = ; Cycles count_A = 1000 x 106 cycles Calculate Clock rate_B:

CPU Time_B = 6 sec. = ; Clock rate_B = = 200 MHz

Machine B must run at twice the clock rate of A to achieve the target execution time.

Example 4.2: We have two machines with different implementations of the same ISA. Machine A has a clock cycle time of 10 ns and a CPI of 2.0 for program P; machine B has a clock cycle time of 20 ns and a CPI of 1.2 for the same program. Which machine is faster? Let IC be the number of instructions to be executed. Then Cycles count_A = 2.0 IC

Cycles count_B = 1.2 IC

calculate CPU Time for each machine: CPU Time_A = 2.0 IC x 10 ns = 20.0 IC ns

CPU Time_B = 1.2 IC x 20 ns = 24.0 IC ns

» Machine A is 20%faster.

Example 4.3: Consider an implementation of MIPS ISA with 500 MHz clock and

– each ALU instruction takes 3 clock cycles,

– each branch/jump instruction takes 2 clock cycles,

– each sw instruction takes 4 clock cycles,

– eachlw instruction takes 5 clock cycles.

Also, consider a program that during its execution executes:

– x=200 million ALU instructions

– y=55 million branch/jump instructions

– z=25 million sw instructions

– w=20 million lw instructions

Find CPU time. Assume sequentially executing CPU. Clock cycles for a program = (3x + 2y + 4z + 5w)

= 910 x 10^6 clock cycles CPU_time = Clock cycles for a program /

Clock rate = 910 x 10^6 / 500 x 10^6 = 1.82 sec

Example 4.4: Consider another implementation of MIPS ISA with 1 GHz clock and each ALU instruction takes 4 clock cycles,

– each branch/jump instruction takes 3 clock cycles,

– each sw instruction takes 5 clock cycles,

– eachlw instruction takes 6 clock cycles.

Also, consider the same program as in Example 1.

Find CPI and CPU time. Assume sequentially executing CPU. CPI = (4x + 3y + 5z + 6w) / (x + y + z + w)

= 4.03 clock cycles/ instruction

CPU time = Instruction count x CPI / Clock rate

= (x+y+z+w) x 4.03 / 1000 x 10^6

= 300 x10^6 x 4.03 /1000 x 10^6

= 1.21 sec

4.2.2 VIRTUAL MEMORY

Virtual memory is a memory management capability of an operating system that uses hardware and software to allow a computer to compensate for physical memory shortages by temporarily transferring data from RAM to disk storage.

The concept of virtual memory in computer organization is allocating memory from the hard disk and making that part of the hard disk as a temporary RAM. In other

words, it is a technique that uses main memory as a cache for secondary storage. The motivations for virtual memory are:

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

Virtual memory provides an illusion to the users that the PC has enough primary memory left to run the programs. Sometimes the size of programs to be executed may sometimes very bigger than the size of primary memory left, the user never feels that the system needs a bigger primary storage to run that program. When the RAM is full, the operating system occupies a portion of the hard disk and uses it as a RAM. In that part of the secondary storage, the part of the program which not currently being executed is stored and all the parts of the program that are executed are first brought into the main memory. This is the theory behind virtual memory.

Terminologies:

- **Physical address** is an address in main memory.
- **Protection** is a set of mechanisms for ensuring that multiple processes sharing the processor, memory, or I/O devices cannot interfere, with one another by reading or writing each other's data.
- Virtual memory breaks programs into fixed-size blocks called **pages.**
- **Page fault** is an event that occurs when an accessed page is not present in main memory.
- **Virtual address** is an address that corresponds to a location in virtual space and is translated by address mapping to a physical address when memory is accessed.
- **Address translation or address mapping** is the process by which a virtual address is mapped to an address used to access memory.

Working mechanism

- In virtual memory, blocks of memory are mapped from one set of addresses

(virtual addresses) to another set (physical addresses).

- The processor generates virtual addresses while the memory is accessed using physical addresses.
- Both the virtual memory and the physical memory are broken into pages, so that a virtual page is really mapped to a physical page.
- It is also possible for a virtual page to be absent from main memory and not be mapped to a physical address, residing instead on disk.
- Physical pages can be shared by having two virtual addresses point to the same physical address. This capability is used to allow two different programs to share data or code. Virtual memory also simplifies loading the program for execution by providing relocation. The following Fig 10 represents Mapping of virtual and physical memory.
- **Relocation** maps the virtual addresses used by a program to different physical addresses before the addresses are used to access memory. This relocation allows us to load the program anywhere in main memory.

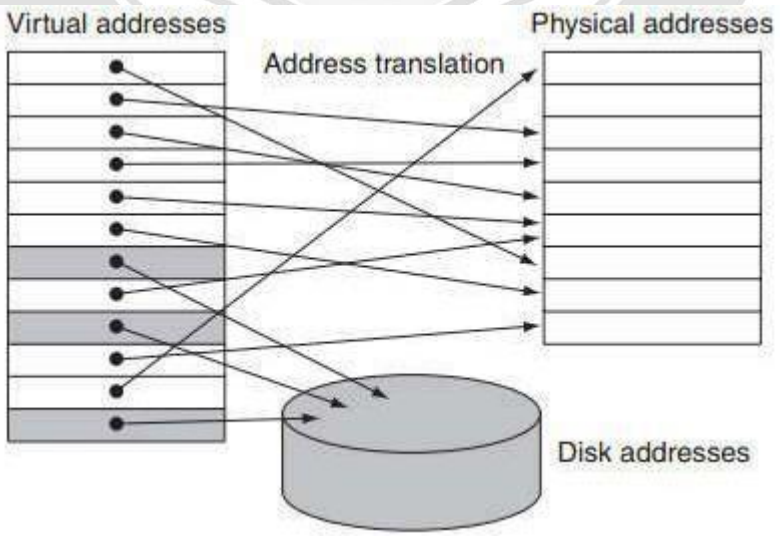


Fig 10: Mapping of virtual and physical memory

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach

Addressing in virtual memory

- A virtual address is considered as a pair (p,d) where lower order bits give an offset d within the page and high-order bits specify the page p.
- The job of the Memory Management Unit (MMU) is to translate the page number p to a frame number f. The physical address is then (f,d), and this is what goes on the memory bus. For every process, there is a page and page-number p is used as an index into this array for the translation. The following Fig 11 shows Conversion of logical address to physical address
- The following are the entries in page tables:
  1. Validity bit: Set to 0 if the corresponding page is not in memory
  2. Frame number: Number of bits required depends on size of physical memory
  3. Protection bits: Read, write, execute accesses
  4. Referenced bit is set to 1 by hardware when the page is accessed: used by page replacement policy
  5. Modified bit (dirty bit) set to 1 by hardware on write-access: used to avoid writing when swapped out.

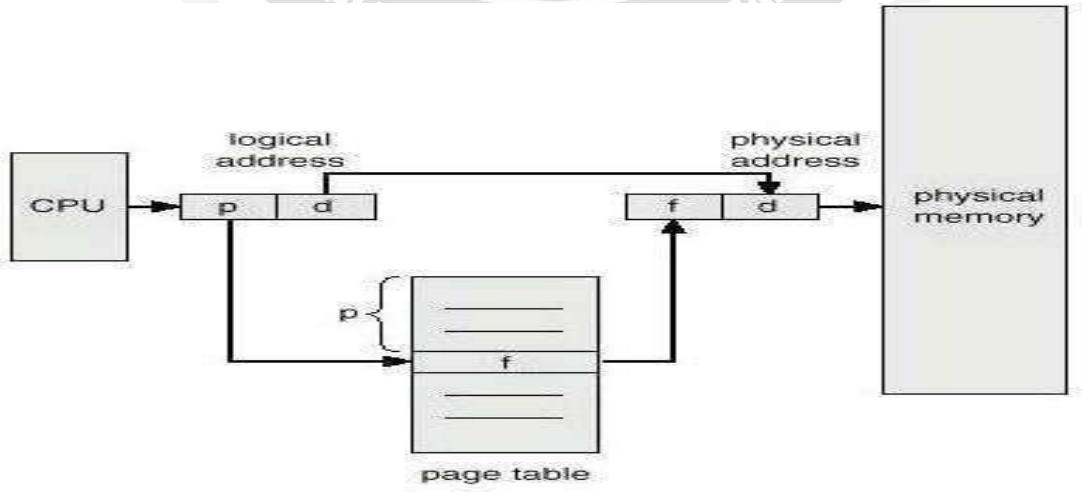


Fig 11: Conversion of logical address to physical address

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach

Role of control bit in page table

The control bit (v) indicates whether the page is loaded in the main memory. It also indicates whether the page has been modified during its residency in the main memory.

This information is crucial to determine whether to write back the page to the disk before it is removed from the main memory during next page replacement. The bellow Fig 12 shows Page table Page faults and page replacement algorithms.

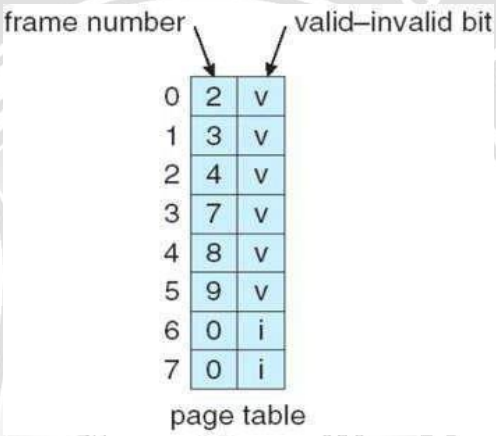


Fig 12: Page table Page faults and page replacement algorithms

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization

Page replacement is a process of swapping out an existing page from the frame of a main memory and replacing it with the required page.

An Integrated approach

A page fault occurs when a page referenced by the CPU is not found in the main memory. The required page has to be brought from the secondary memory into the main memory. A page that is currently residing in the main memory, has to be replaced if all the frames of main memory are already occupied.

Page replacement is done when all the frames of main memory are already occupied and a page has to be replaced to create a space for the newly referenced page.

A good replacement algorithm will have least number of page faults. The bellow Fig 13 shows the Occurrence of page fault.

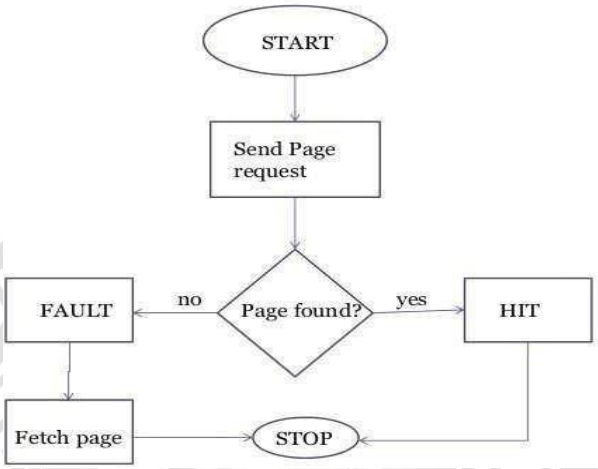


Fig 13: Occurrence of page fault

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach

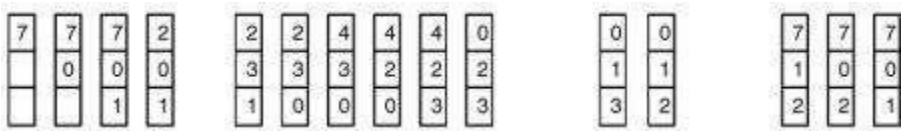
The following are the page replacement algorithms:

6. FIFO Page Replacement Algorithm
7. LIFO Page Replacement Algorithm
8. LRU Page Replacement Algorithm
9. Optimal Page Replacement Algorithm
10. Random Page Replacement Algorithm

1. **First In First Out (FIFO) page replacement algorithm**

It replaces the oldest page that has been present in the main memory for the longest time. It is implemented by keeping track of all the pages in a queue.

Example 4.5. Find the page faults when the following pages are requested to be



loaded in a page frame of size 3: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Page faults= 15

2. **Last In First Out (LIFO) page replacement algorithm**

It replaces the newest page that arrived at last in the main memory. It is implemented by keeping track of all the pages in a stack.

3. **Least Recently Used (LRU) page replacement algorithm** The new page will be replaced with least recently used page.

Example 4.6: Consider the following reference string. Calculate the number of page faults when the page frame size is 3 using LRU policy.7, 0, 1, 2, 0, 3, 4, 2, 3, 0, 3, 2,1,2,0,1,7,0,1

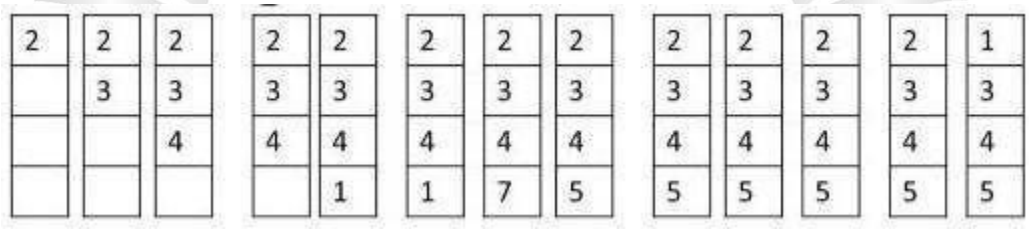
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
F	F	F	F		F		F	F	F	F			F		F		F		

Page faults= 12 (F bit indicates the occurrence of page faults)

4. **Optimal page replacement algorithm**

In this method, pages are replaced which would not be used for the longest duration of time in the future.

Example 4.7: Find the number of misses and hits while using optimal page replacement algorithm on the following reference string with page frame size as 4: 2, 3, 4, 2, 1, 3, 7, 5, 4,
3, 2, 3, 1.



Page fault=13Number of page hit= 6Number of page misses=7

5. **Random page replacement algorithms**

Random replacement algorithm replaces a random page in memory. This eliminates the overhead cost of tracking page references.

Translation Look aside Buffer (TLB)

The page tables are stored in main memory and every memory access by a program to the page table takes longer time. This is because it does one memory access to obtain the physical address and a second access to get the data. The virtual to physical memory address translation occurs twice. But a TLB will exploit the locality of reference and can reduce the memory access time.

TLB hit is a condition where the desired entry is found in translation look aside buffer. If this happens then the CPU simply access the actual location in the main memory.

If the entry is not found in TLB (TLB miss) then CPU has to access page table in the main memory and then access the actual frame in the main memory. Therefore, in the case of TLB hit, the effective access time will be lesser as compare to the case of TLB miss.

If the probability of TLB hit is P% (TLB hit rate) then the probability of TLB miss (TLB miss rate) will be (1-P) %. The effective access time can be defined as

$$Effective\ access\ time = P(t + m) + (1 - p)(t + k.m + m)$$

Where, p is the TLB hit rate, t is the time taken to access TLB, m is the time taken to access main memory. K indicates the single level paging has been implemented. The following Fig 14 shows Cache Access Levels.

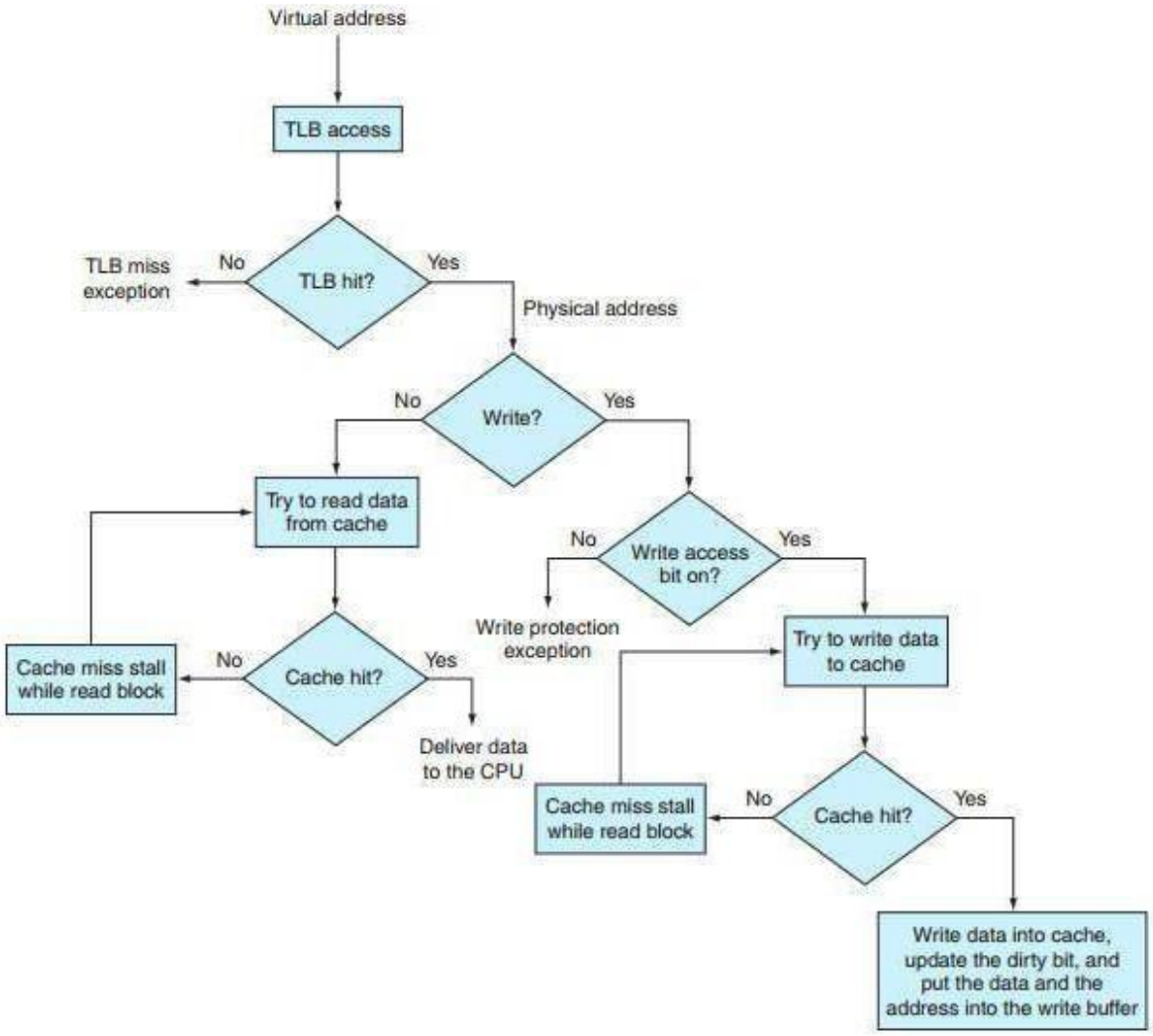


Fig 14: Cache access levels

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach

4.3.5 Protection in Virtual memory

- Virtual memory allows sharing of main memory by multiple processes. So protection mechanisms, while providing memory protection.
- The protection mechanism must ensure one process cannot write into the address space of another user process or into the operating system.
- Memory protection can be done at two levels: hardware and software levels.

Hardware Level:

Memory protection at hardware level is done in three methods:

- The machine should support two modes: supervisor mode and user mode. This indicates whether the current running process is a user or supervisory process. The processes running in supervisor or kernel mode is an operating system process.
- Include user / supervisor bit in TLB to indicate whether the process is in user or supervisor mode. This is an access control mechanism imposed on the user process only to read from the TLB and not write to it.
- The processors can switch between user and supervisor mode. The switching from user to system mode is done through system calls that transfers control to a dedicated location in supervisor code space.

System call is a special instruction that transfers control from user mode to a dedicated location in supervisor code space, invoking the exception mechanism in the process.