MEMORY MANAGEMENT HARDWARE

The use of multiprogramming create the use of memory management hardware.

→ A memory management system is a collection of hardw -are & software procedures for managing the various programs residing in memory.

→ Memory management software is part of an operating system.

→ Hardware unit associated with the memory manageme -nt system are,

a) A facility for dynamic storage relocation that maps logical memory references into physical memory address

b) A provision for sharing common programs stored in memory by different users.

c) Protection of information against unauthorized access between users & preventing users from changing operating system functions.

→ Memory management hardware is required for the multiprogramming for the following reasons –

1. Sharing of common programs is an integral part of a multiprogramming system.

2. Multiprogramming need protection of one program from unwanted interaction or access from other.

e.g. One user's unauthorized copying of another user's program.

Dynamic storage Relocation →

Dynamic storage relocation is similar to thepaging system. But the major drawback here is,

→ The fixed page size used in virtual memory system causes certain problems in logical structure & size of program. Hence these problems is avoided by dividing programs & data into logical parts called segments.

→ A segment is a set of logically related instructions & data elements associated with a given name. A segment may be generated by programmer or by operating system.

e.g. of subroutine are subroutines or arrays or user programs.

SEGMENTS →

→ The address generated segmented program is called as a logical address.

→ The logical address is similar to virtual address except that logical addresses space are associated with variable length segment where virtual address space is related to fixed size page.

→ A segment is related to logical address but a page is related to virtual address.

→ By using relocation information each segment has protection information associated with it.

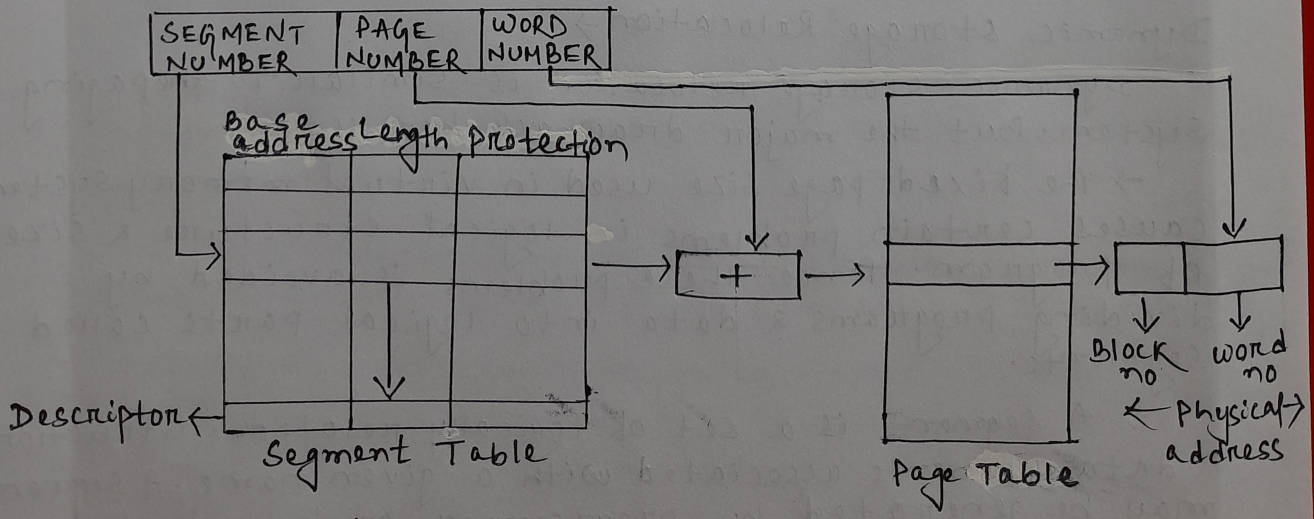
→ Shared programs are placed in a unique segment in each user's logical address space so that a single physical copy can be shared.

→ Memory management performs the mapping of logical address to physical address which is similar to virtual memory mapping concept.

SEGMENTED PAGE MAPPING →

→ Segments are of variable length & the length of each segment grows & contracts according to need of the program being executed.

→ The length of each segment is specified by associating it with a no. of equal size pages.



(Block Diagram of segmented page mapping)

→ Mapping of logical address into physical address is done by means of 2 tables. They are - segment table & page

1. Entry in segment table is a pointer address table. for a page table base.

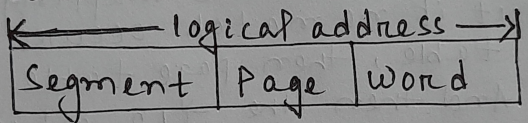
2. The page table base address is added with page number given in logical address.

3. The sum of page number & page table base address produces a pointer address to an entry in the page table.

4. Now, the value of entry found in page table provides the block number in physical memory.

5. The logical address is partitioned into three fields, those are,

(a) Segment field
(b) Page field
(c) word field



a) Segment field specifies a segment number.
b) Page field specifies a page within the identified segment.
c) word field specifies the word within the page.

→ If page field = K bits
Then there are 2^K no. of pages that can be specified by using 'K' no. of bits.

→ A segment number may be associated with one page or 2^K no. of pages as many as. Hence the length of a segment would vary according to no. of pages that are assigned to it.

6. The concatenation of the block field with word field produces final physical mapped address.

7. The two different mapping tables (segment table & page table) may be stored in 2 separate small memories or in main memory.

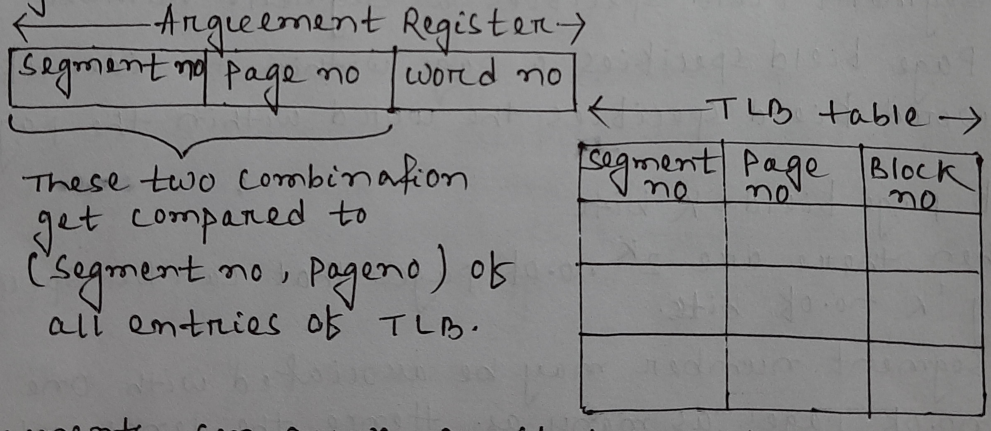
To retrieve a word from memory, it requires 3 different memory access i.e.,

(1) First access for segment table.
(2) Second access for page table.
(3) Third access for main memory.

As it requires 3 access to read a word from main memory so the speed of the system goes slow. To avoid slow speed, a fast associative memory is used to hold the most recently referenced page table entries. This type of memory is sometimes called as translation lookaside buffer (TLB).

8. When first time a block is referenced, its value together with corresponding segment & page no are entered into the associative memory or TLB.

Therefore the mapping process is first attempted by associative search with the given segment & page no. If it is a success one then the mapping delay is only for the TLB. But if no match occurs then the processor go for segment table, then page table & then to M/M. main memory. And then the TLB get updated for the new entry.



9. Segments can grows or shrinks without affecting others. Pages can move to different block depending on requirement. so in page table or in TLB, only block no field values has changed with time.

Sharing of programs or codes →

If different users requires to share same program then different segments can use the same blocks of memory.

MEMORY PROTECTION →

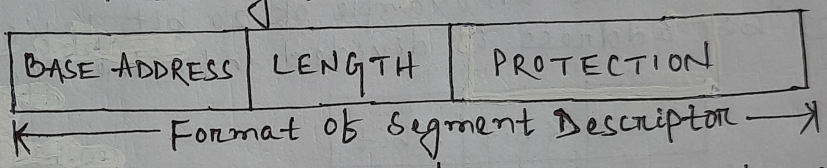
→ Memory protection can be assigned to the physical address (blocks) or logical address (segment).

→ The protection of memory through physical address can be done by assigning some protection bits to each block or segment that indicates the type of access allowed to its corresponding blocks or segments.

→ Every time a page is moved out from a block, it is necessary to update the block protection bits.

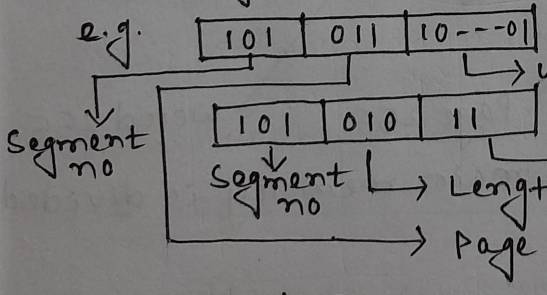
→ The protection bits of memory through logical address can be done by including protection information within segment table or segment registers of memory management hardware.

→ The content of each entry of segment table or segment register is called a descriptor. A descriptor would contain the following fields -



(a) Base address - The base address field gives the base address value of 1st page table of the identified segment in segmented-page orientation.

(b) Length - Length field specifies the segment size by identifying or calculating the no. of pages assigned to the segment. Length field is compared with page no. in logical address. If page no. > Length value then there is a size violation or the page number fall outside segment boundary.



Here segment 5 can contain maximum of 2 pages as per descriptor information. But in logical address the search is for page no. 3. Hence page 3 is not present in segment 5.

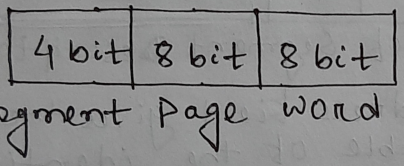
⇒ Page no > Length

⇒ Size violation.

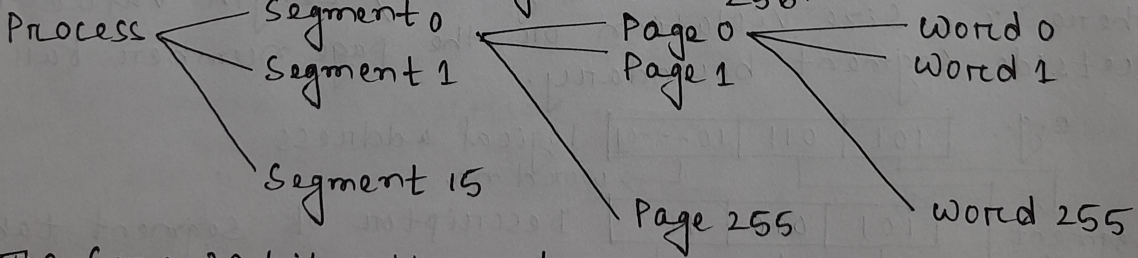
(c) Protection - Protection field in a segment descriptor specifies the access rights to that particular segment. It is all set by master control program of operating system. Some access rights are -

1. Full read & write privileges - It is given to a program when it is executing it's own instruction.
2. Read only or write protection - It is given to sharing system programs i.e. only reading is allowed not writting
3. Execute only or program protection - It protects programs from being copied. It allows the users to execute program instructions but prevent reading the instructions.
4. System only or operating system protection - It prevents unauthorized users or occasional users from accessing the operating system segments.

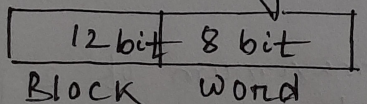
e.g. Consider an example of 20 bit logical address. The 20 bit address is divided into the following ways -



The above partition shows that no. of segments in the Process or application program is $2^4 = 16$.
No. of pages in each segment varies from 0 to 255 because total no. of pages can be $= 2^8 = 256$.
No. of words in each page $= 2^8 = 256$.



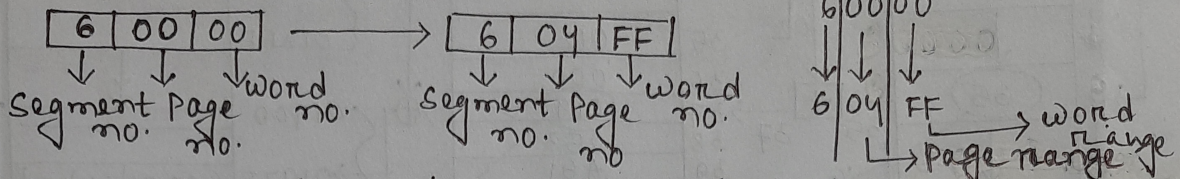
The same 20 bit address for main memory is divided into the following,



The main memory consist of no. of blocks & each block contain no. of words.

Therefore no. of blocks in main memory $= 2^{12} = 4096$
no. of words in each block of main memory $= 2^8 = 256$

→ consider that the processor wants 5 pages of segment 6 then the logical address range for segment 6 varies from, 60000 to 604FF.

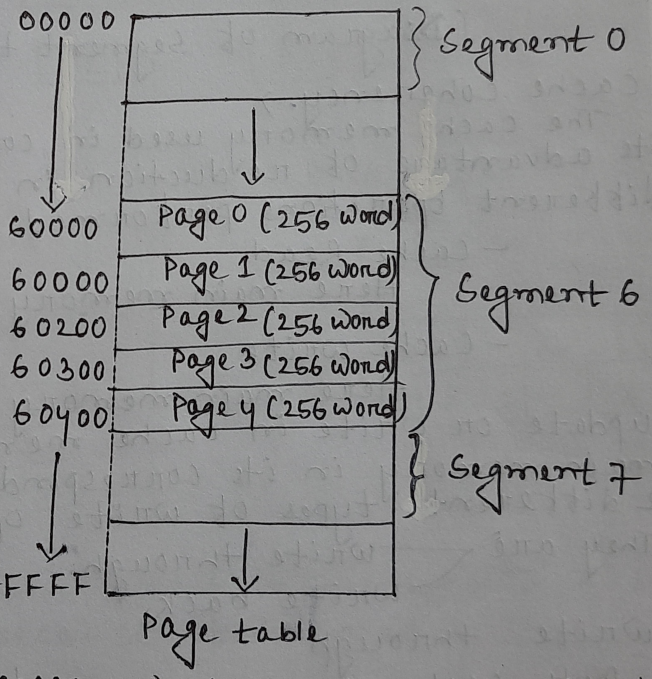


This means segment 6 contain 5 pages i.e. page 0 to page 4 stored at Location 60000 to 604FF respectively or each address or location within the range points to or reference to page's word within the range page 0 to page 4.

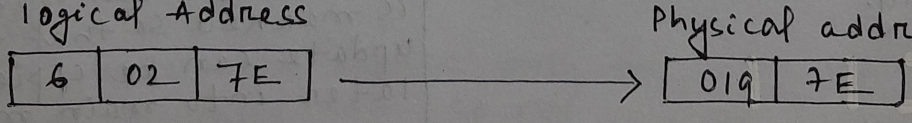
Segmented-page TLB

Segment	Page	BLOCK
6	00	000
6	01	012
6	02	019
6	03	053
6	04	061

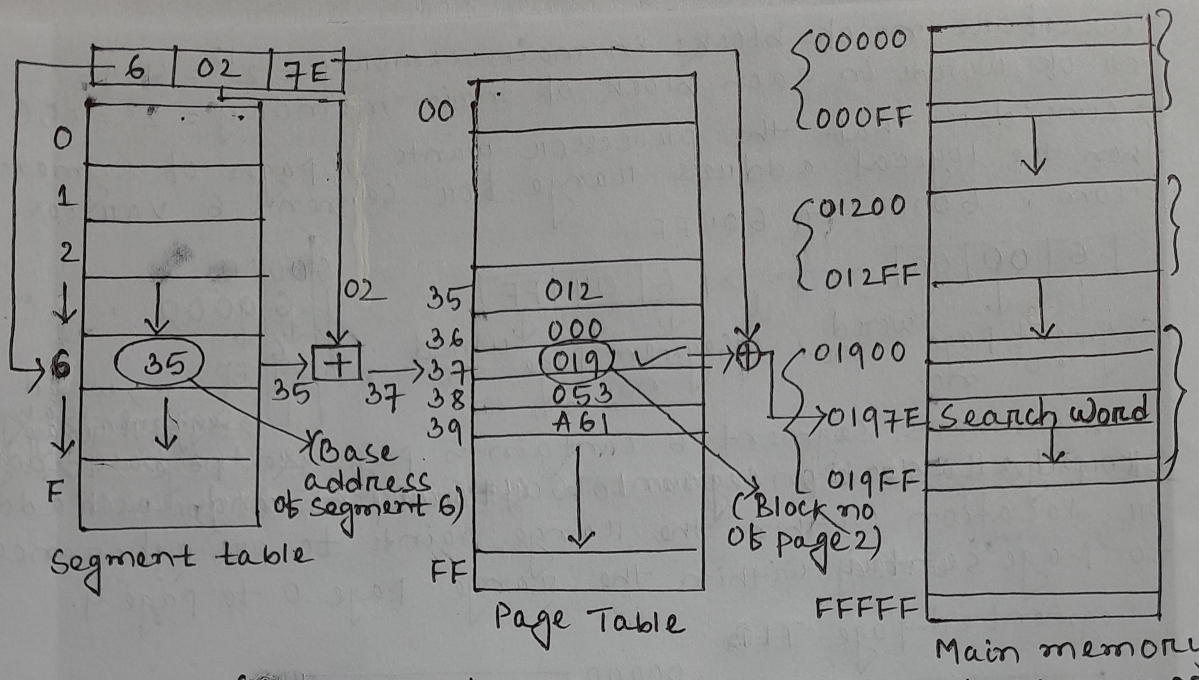
NOTE :-
using 20 bit total address range is
00000 → FFFFF



→ Now consider the processor wants to search a word at location 6027E. This means the processor wants to read word at 7E of page 02 of segment 6 from main memory



To retrieve a searched word from M/M following steps followed, → search from segment table.
→ search from page table.
→ search from main memory.



(Diagram of segment table & page table mapping)

Cache coherency →

The cache memory used in computer system because of its advantage of reduction in average access time. Two different operations performed in cache are,

- Cache Read

  Here main memory is not affected.

- Cache write

  Here main memory is affected that is any update or write in cache memory must be reflected in main memory in its corresponding location. There are 2 different types of write operation in main memory They are ← write through, write back

write through

Both cache memory & main memory get updated for each write operation Parallely.

Write Back

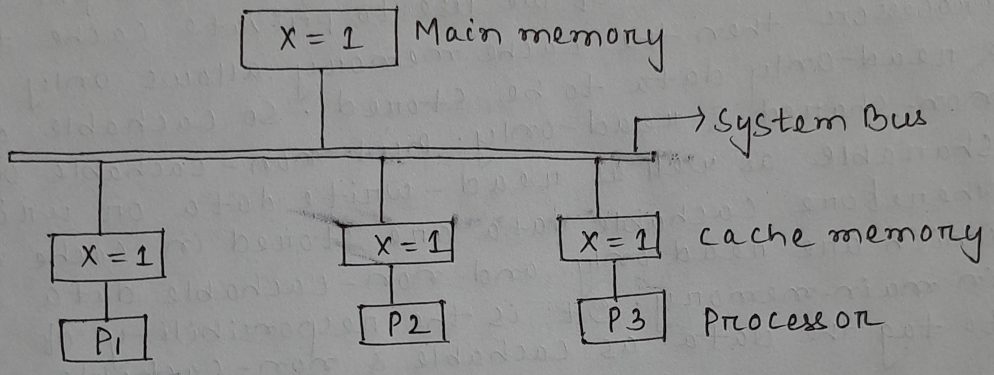
Only cache memory is updated & the corresponding location of main memory is marked so that the update can be copied later later into main memory.

→ In a shared memory multiprocessor system all the processors share a common memory & may have a local memory of its own & the cache memory may be a part of local memory or can be the local memory whole can be a cache memory.

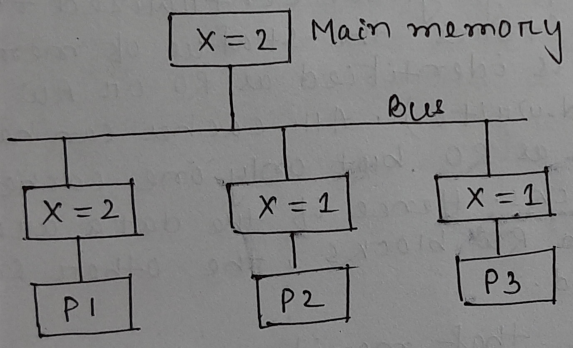
→ Hence to ensure the ability of system to executing memory operation correctly the multiple copies in the cache memories must be identical which is known as cache coherence or cache memory consistency.

→ A memory is coherent if the value returned on a load instruction is always the value given by the latest store instruction with the same address.

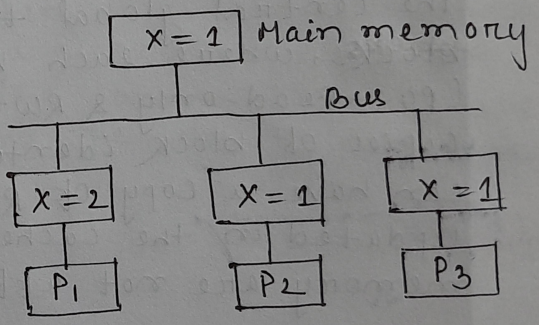
→ condition for cache coherence



Write through



Write Back



→ Cache coherence problem exist in multiprocessor system with private caches because of the need to share the writable data.

→ In write through figure the cache memory & main memory are consistent but the other two caches are inconsistent.

→ In write back figure only the cache memory 1 has update writable data. So not only the cache memory 1 & main memory are inconsistent but also the cache memory 2 as well as cache memory 3 are also inconsiste -nt.

Remedies for cache coherence →

There are several solutions to cache coherence. These are -

1. Rather than using private cache for each processor, use a shared cache memory associated with main memory

Drawback of this - It violates the closeness of cpu to cache & increase the average memory access time.

2. Using a private cache memory to each processor increase the performance. If we use to private cache to each Processor then the cache memory allows only non-shared & read-only data to be stored. So cachable data are non-shared & read-only. But non-cachable data are sharable as well as read-write data or writable data. Therefore cachable data are stored in cache memory which are read only and non-cachable data are stored in main-memory. It is the responsibility of compiler to tag the data as cachable & non-cachable.

→ To allow writable data to exist in atleast one cache is a method that employs global centralized table. The central global table stores the status of memory blocks, where each block is identified as RO or RW (RO → Read-only & RW → Read-write). All caches can have copies of block identified as RO but only one cache can have a copy of RW blocks. Hence if the data are updated in the cache with RW blocks, the other cache memory are not affected.

3. A bus controller is there that monitors the action of watching over all the caches attached to the bus known as snoopy bus controller. The snoopy bus controller employs snoopy cache protocol to solve cache coherence problem.

→ In snoopy cache protocol, write-through policy is adopted. when a word in a cache is updated by writting into it, the corresponding location in main-memory is also updated. The local snoopy controller in other cache memory check their memory space to determine if they have a copy of the word (Block) that has been overwritten. If a copy exist in remote cache then that location (block) marked as invalid (dirty block)

→ whenever a word is updated or written then the net effect is to update it in original cache & main-memory and remove it from all other caches. Therefore 2 different schemes are there,

1. Invalid scheme
2. Write-update scheme

→ If a processor access the invalid item from its cache memory, it is similar to cache miss & then the updated item transferred from main memory to cache memory

1. Invalidate scheme - when a processor writes to a block in a write through policy, the cache controller at all processors match the address to check if they have a copy of the block or not. If they have a copy of the block then they update local cache memory.

After any update to a block by a processor, the other processors when they read the same block, read the updated or valid values.

2. Snoopy write-update scheme - It is a write-back scheme. When a processor writes a block, writting of the block to main-memory is disabled and only local caches at the processors having a copy of the block are updated.