Virtual Memory

Paging

Both unequal fixed size and variable size partitions are inefficient in the use of memory. It has been observed that both schemes lead to memory wastage. Therefore we are not using the memory efficiently.

There is another scheme for use of memory which is known as paging.In this scheme,The memory is partitioned into equal fixed size chunks that are relatively small. This chunk of memory is known as frames or page frames.

Each process is also divided into small fixed chunks of same size. The chunks of a program is known as pages.
A page of a program could be assigned to available page frame.In this scheme, the wastage space in memory for a process is a fraction of a page frame which corresponds to the last page of the program.
At a given point of time some of the frames in memory are in use and some are free. The list of free frame is maintained by the operating system.

Process A , stored in disk , consists of pages . At the time of execution of the process A, the operating system finds six free frames and loads the six pages of the process A into six frames.

These six frames need not be contiguous frames in main memory. The operating system maintains a page table for each process.

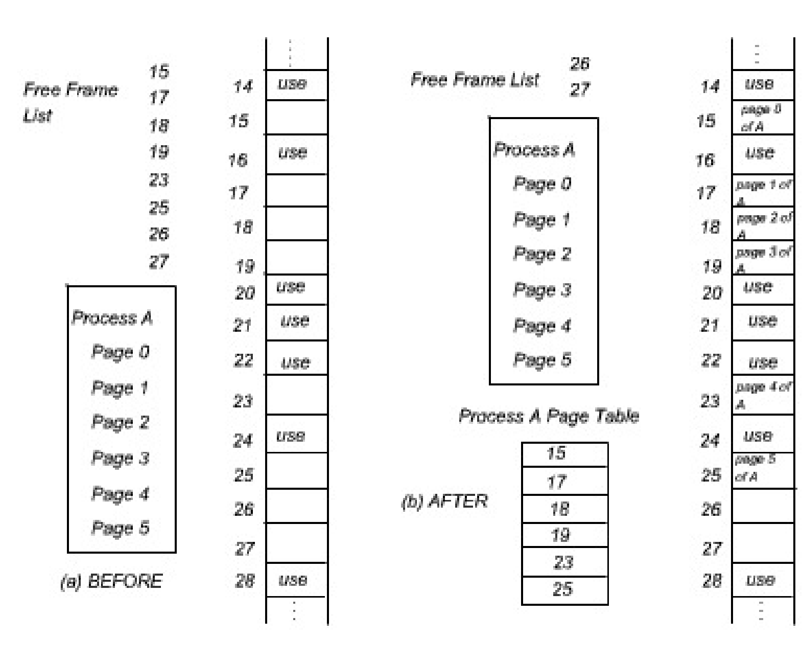
Within the program, each logical address consists of page number and a relative address within the page.

In case of simple partitioning, a logical address is the location of a word relative to the beginning of the program; the processor translates that into a physical address.

With paging, a logical address is a location of the word relative to the beginning of the page of the program, because the whole program is divided into several pages of equal length and the length of a page is same with the length of a page frame.

A logical address consists of page number and relative address within the page, the process uses the page table to produce the physical address which consists of frame number and relative address within the frame.

The Figure below shows the allocation of frames to a new process in the main memory. A page table is maintained for each process. This page table helps us to find the physical address in a frame which corresponds to a logical address within a process.



The conversion of logical address to physical address is shown in the figure for the Process A.

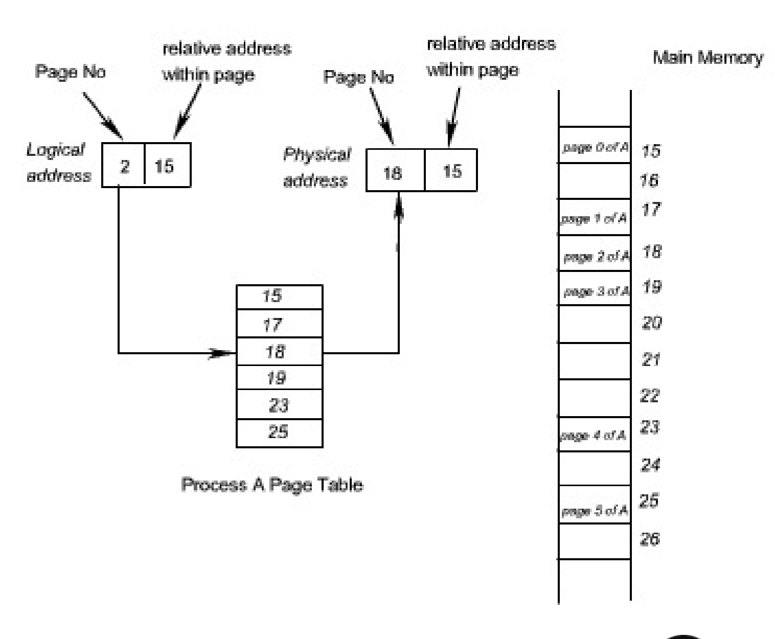


Figure : Translation of Logical Address to Physical Address

This approach solves the problems. Main memory is divided into many small equal size frames. Each process is divided into frame size pages. Smaller process requires fewer pages, larger process requires more. When a process is brought in, its pages are loaded into available frames and a page table is set up.

The translation of logical addresses to physical address is shown in the Figure above.

Virtual Memory

The concept of paging helps us to develop truly effective multiprogramming systems.

Since a process need not be loaded into contiguous memory locations, it helps us to put a page of a process in any free page frame. On the other hand, it is not required to load the whole process to the main memory, because the execution may be confined to a small section of the program. (eg. a subroutine).

It would clearly be wasteful to load in many pages for a process when only a few pages will be used before the program is suspended.

Instead of loading all the pages of a process, each page of process is brought in only when it is needed, i.e on demand. This scheme is known as demand paging .

Demand paging also allows us to accommodate more process in the main memory, since we are not going to load the whole process in the main memory, pages will be brought into the main memory as and when it is required.

With demand paging, it is not necessary to load an entire process into main memory.
This concept leads us to an important consequence – It is possible for a process to be larger than the size of main memory. So, while developing a new process, it is not required to look for the main memory available in the machine. Because, the process will be divided into pages and pages will be brought to memory on demand.
Because a process executes only in main memory, so the main memory is referred to as real memory or physical memory.

A programmer or user perceives a much larger memory that is allocated on the disk. This memory is referred to as virtual memory. The program enjoys a huge virtual memory space to develop his or her program or software.

The execution of a program is the job of operating system and the underlying hardware. To improve the performance some special hardware is added to the system. This hardware unit is known as Memory Management Unit (MMU).

In paging system, we make a page table for the process. Page table helps us to find the physical address from virtual address.

The virtual address space is used to develop a process. The special hardware unit , called Memory Management Unit (MMU) translates virtual address to physical address. When the desired data is in the main memory, the CPU can work with these data. If the data are not in the main memory, the MMU causes the operating system to bring into the memory from the disk.

A typical virtual memory organization is shown in the Figure below.

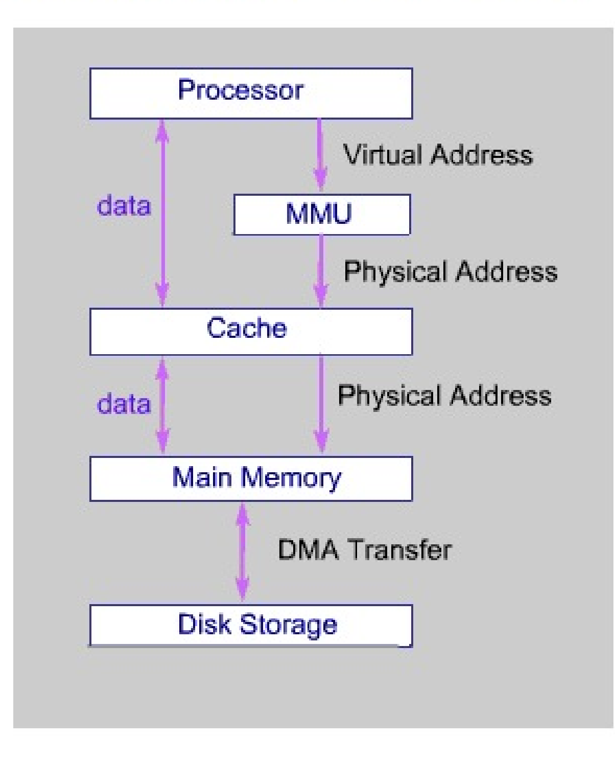


Figure : Virtual Memory Organization

Address Translation

The basic mechanism for reading a word from memory involves the translation of a virtual or logical address, consisting of page number and offset, into a physical address, consisting of frame number and offset, using a page table.

There is one page table for each process. But each process can occupy huge amount of virtual memory. But the virtual memory of a process cannot go beyond a certain limit which is restricted by the underlying hardware of the MMU. One of such component may be the size of the virtual address register.

The sizes of pages are relatively small and so the size of page table increases as the size of process increases. Therefore, size of page table could be unacceptably high.

To overcome this problem, most virtual memory scheme store page table in virtual memory rather than in real memory.

This means that the page table is subject to paging just as other pages are.

When a process is running, at least a part of its page table must be in main memory, including the page table entry of the currently executing page.

A virtual address translation scheme by using page table is shown in the Figure below.

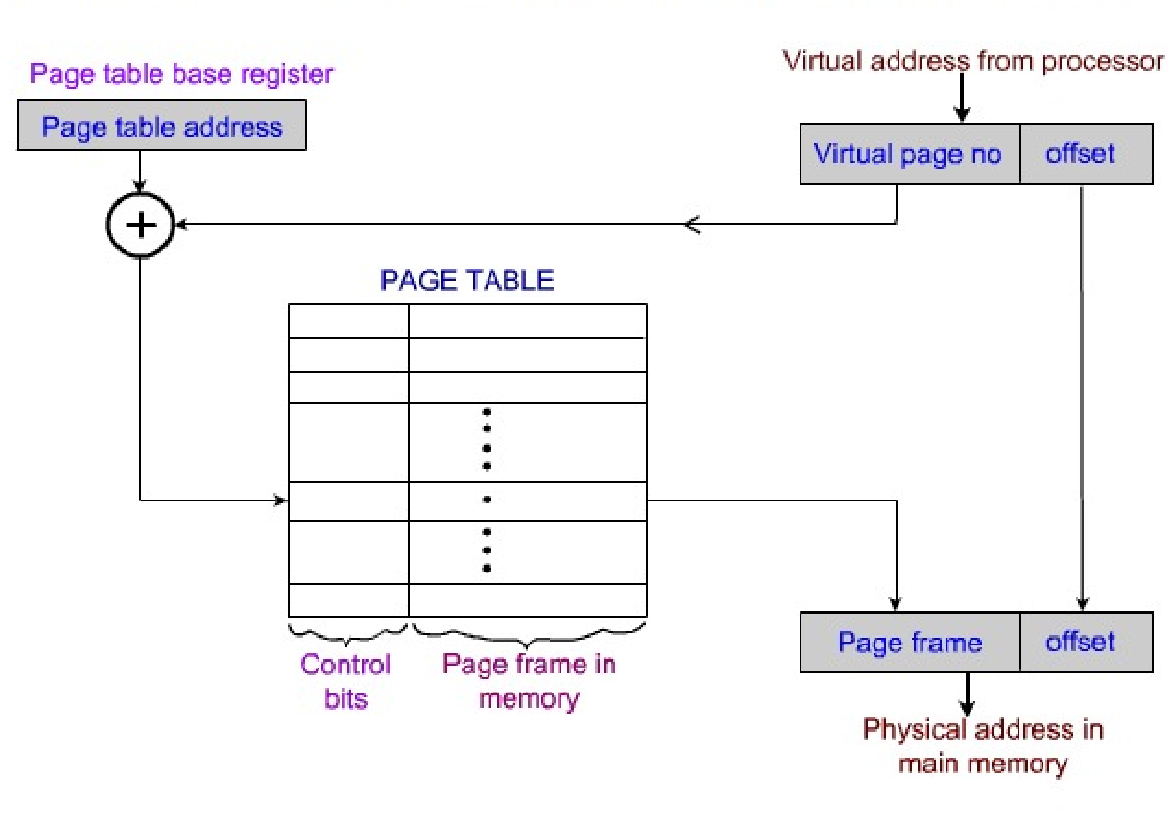


Figure : Virtual Address Translation Method
Each virtual address generated by the processor is interpreted as virtual page number (high order list) followed by an offset (lower order bits) that specifies the location of a particular word within a page. Information about the main memory location of each page kept in a page table.

Some processors make use of a two level scheme to organize large page tables.

In this scheme, there is a page directory, in which each entry points to a page table.

Thus, if the length of the page directory is X, and if the maximum length of a page table is Y, then the process can consist of up to X * Y pages.

Typically, the maximum length of page table is restricted to the size of one page frame.

Inverted page table structures

There is one entry in the hash table and the inverted page table for each real memory page rather than one per virtual page.

Thus a fixed portion of real memory is required for the page table, regardless of the number of processes or virtual page supported.

Because more than one virtual address may map into the hash table entry, a chaining technique is used for managing the overflow.

The hashing techniques results in chains that are typically short – either one or two entries.
The inverted page table structure for address translation is shown in the Figure below.

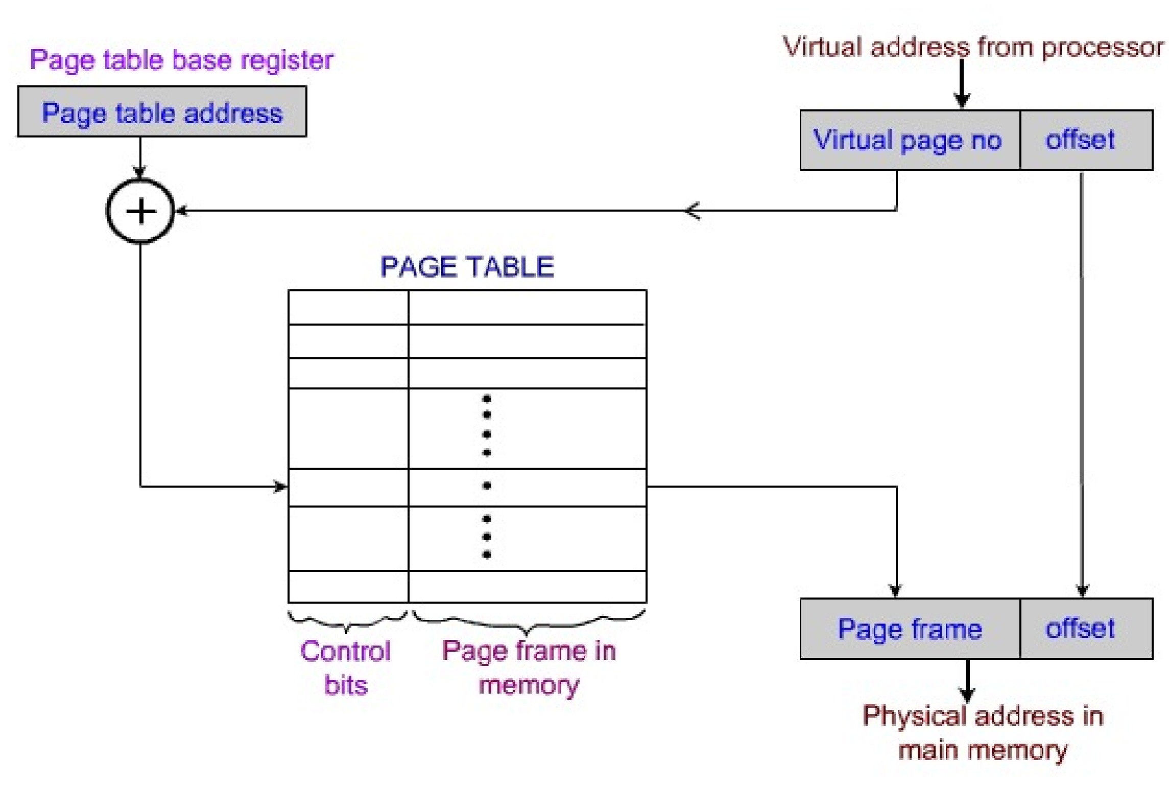


Figure : Inverted Page table structure
Translation Lookaside Buffer (TLB)

1. Every virtual memory reference can cause two physical memory accesses.
2. One to fetch the appropriate page table entry
3. One to fetch the desired data.
4. Thus a straight forward virtual memory scheme would have the effect of doubling the memory access time.
5. To overcome this problem, most virtual memory schemes make use of a special cache for page table entries, usually called Translation Lookaside Buffer (TLB).
6. This cache functions in the same way as a memory cache and contains those page table entries that have been most recently used.
7. In addition to the information that constitutes a page table entry, the TLB must also include the virtual address of the entry.

The Figure below shows a possible organization of a TLB whwere the associative mapping technique is used.

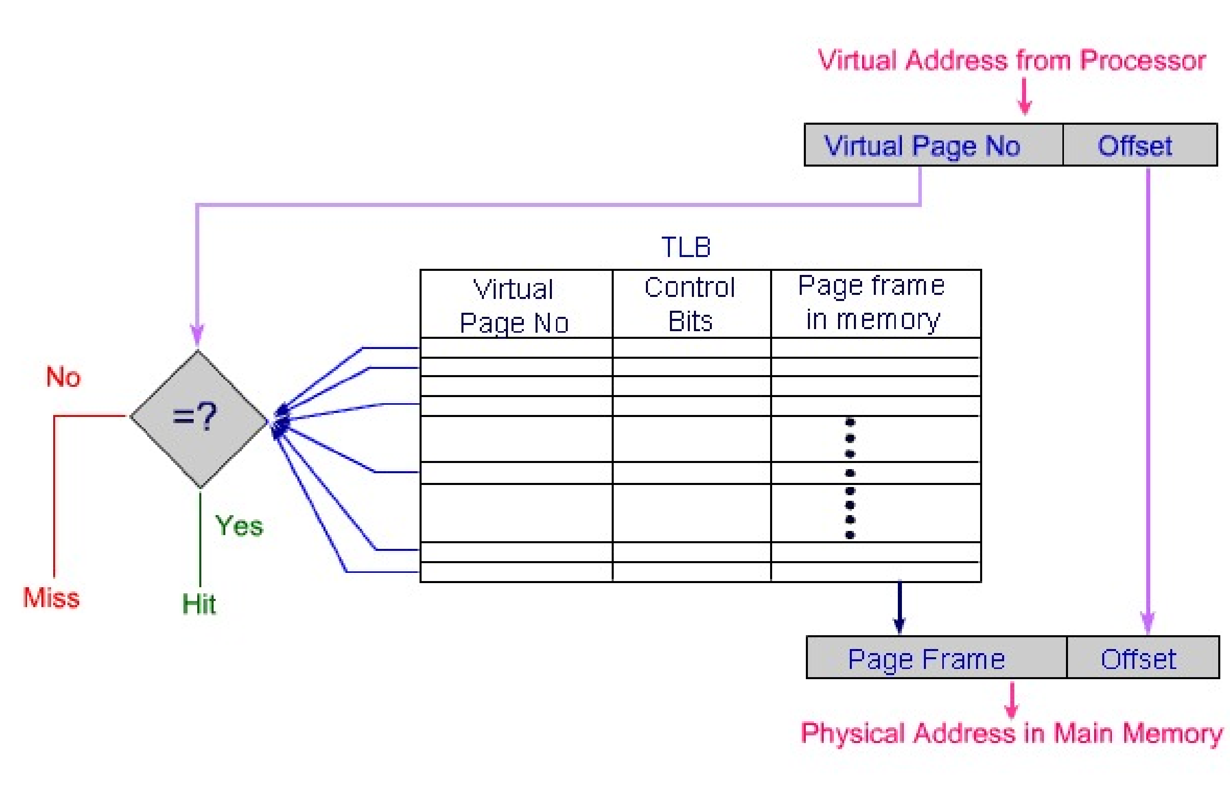


Figure : Use of an associative mapped TLB

Set-associative mapped TLBs are also found in commercial products.

An essential requirement is that the contents of the TLB be coherent with the contents of the page table in the main memory.

When the operating system changes the contents of the page table it must simultaneously invalidate the corresponding entries in the TLB. One of the control bits in the TLB is provided for this purpose.

Address Translation proceeds as follows:

1. Given a virtual address, the MMU looks in the TLB for the reference page.
2. If the page table entry for this page is found in the TLB, the physical address is obtained immediately.
3. If there is a miss in the TLB, then the required entry is obtained from the page table in the main memory and the TLB is updated.
4. When a program generates an access request to a page that is not in the main memory, a page fault is said to have occurred.
5. The whole page must be brought from the disk into the memory before access can proceed.
6. When it detects a page fault, the MMU asks the operating system to intervene by raising an exception.(interrupt).
7. Processing of active task is interrupted, and control is transferred to the operating system.
8. The operating system then copies the requested page from the disk into the main memory and returns control to the interrupted task. Because a long delay occurs due to a page transfer takes place, the operating system may suspend execution of the task that caused the page fault and begin execution of another task whose page are in the main memory.