University Institute of Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Unit- 2

Subject Name : Operating System

Chapter : Memory Management- (Part-2)

Prepared By: Er. Inderjeet Singh(e8822)

Outline

- Memory Management and its Need
- Address Binding
- Logical v/s Physical address space
- Swapping
- Types of Memory Allocation
- Fragmentation
- Paging
- Exercise on Paging

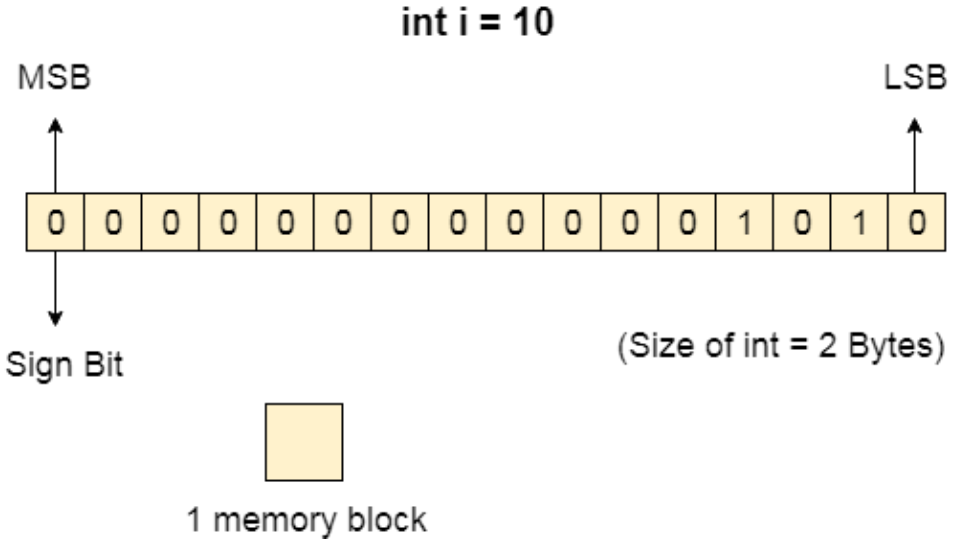
Memory Management

Memory Management is the process of controlling and coordinating computer memory, assigning portions called blocks to various running programs to optimize overall system performance.

- When the program requests a block of memory, a part of the memory manager called the allocator assigns that block to program.
- When a program no longer needs the data in previously allocated memory blocks, these blocks become available for reassignment. This task can be done automatically by memory manager or manually by the user.

Need of Memory Management

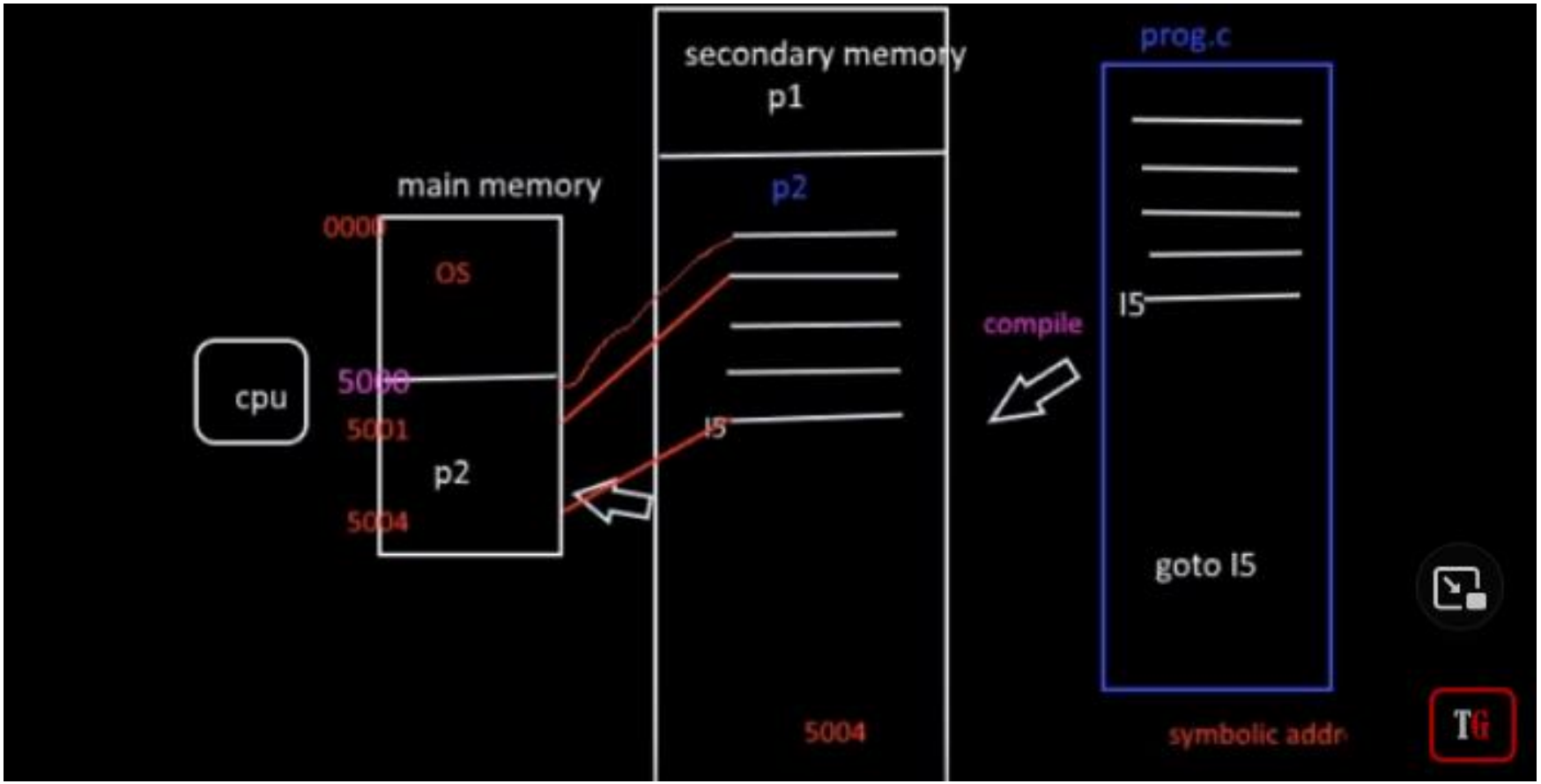
- Memory management is an essential element of all modern computing systems
- With the continued used of virtualization and the need to optimize resource utilization, memory is constantly being allocated, removed, segmented, used and re-used.
- With memory management techniques, memory management errors, that can lead to system and application instability and failures can be mitigated



Address Binding

- Address binding is the process of mapping from one address space to another address space.
- Logical address is an address generated by the CPU during execution, whereas Physical Address refers to the location in the memory unit(the one that is loaded into memory).
- The logical address undergoes translation by the MMU or address translation unit in particular. The output of this process is the appropriate physical address or the location of code/data in RAM.
- Addresses in the Source program are generally symbolic.
- A compiler binds these symbolic addresses to relocatable addresses.
- The loader binds these relocatable addresses to absolute addresses.

Types of Address Binding



Types of Address Binding

Compile Time –

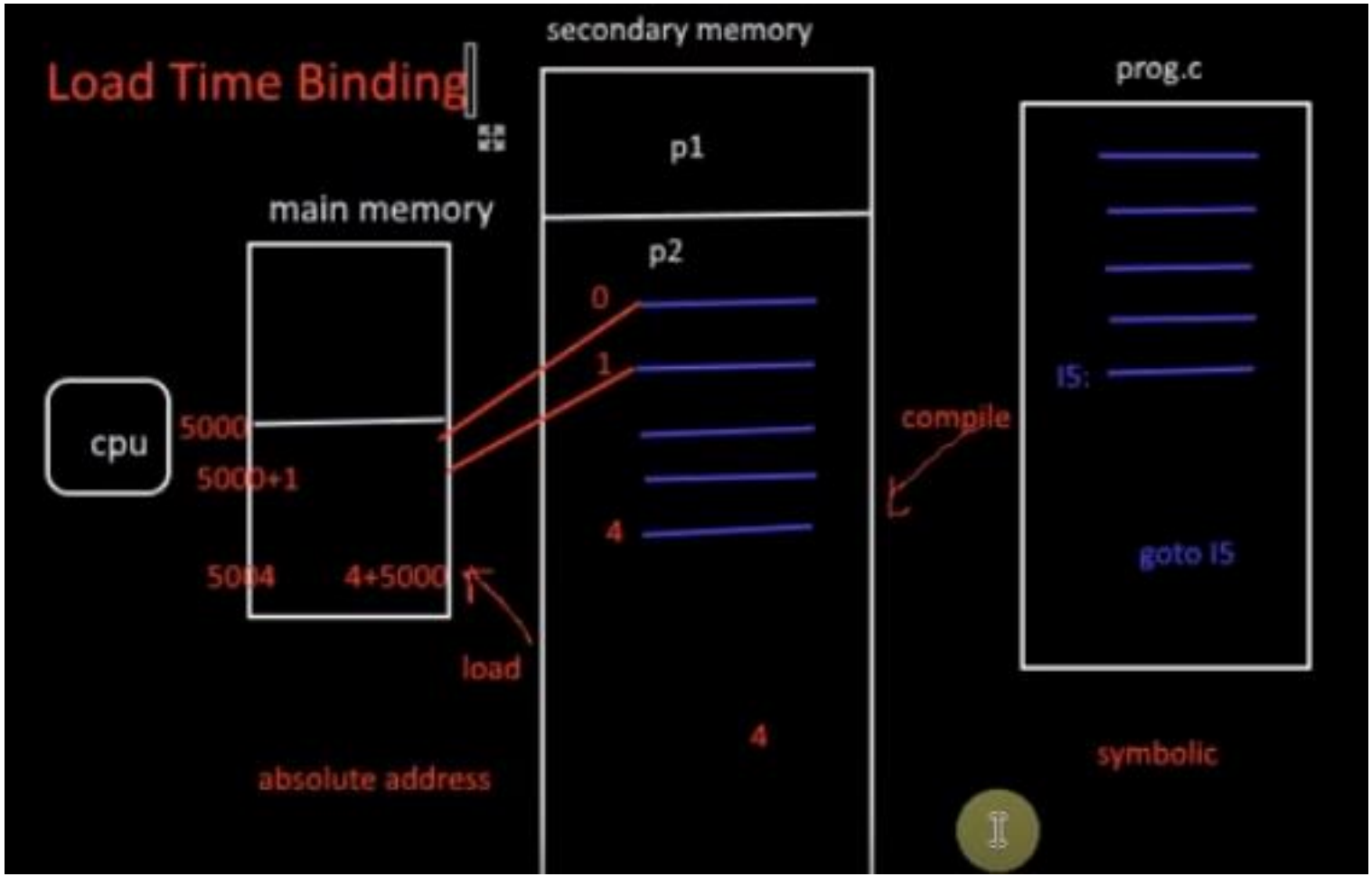
If you know that during compile time, where process will reside in memory, then an absolute address is generated. i.e. The physical address is embedded to the executable of the program during compilation.

Loading the executable as a process in memory is very fast. But if the generated address space is preoccupied by other processes, then the program crashes and it becomes necessary to recompile the program to change the address space.

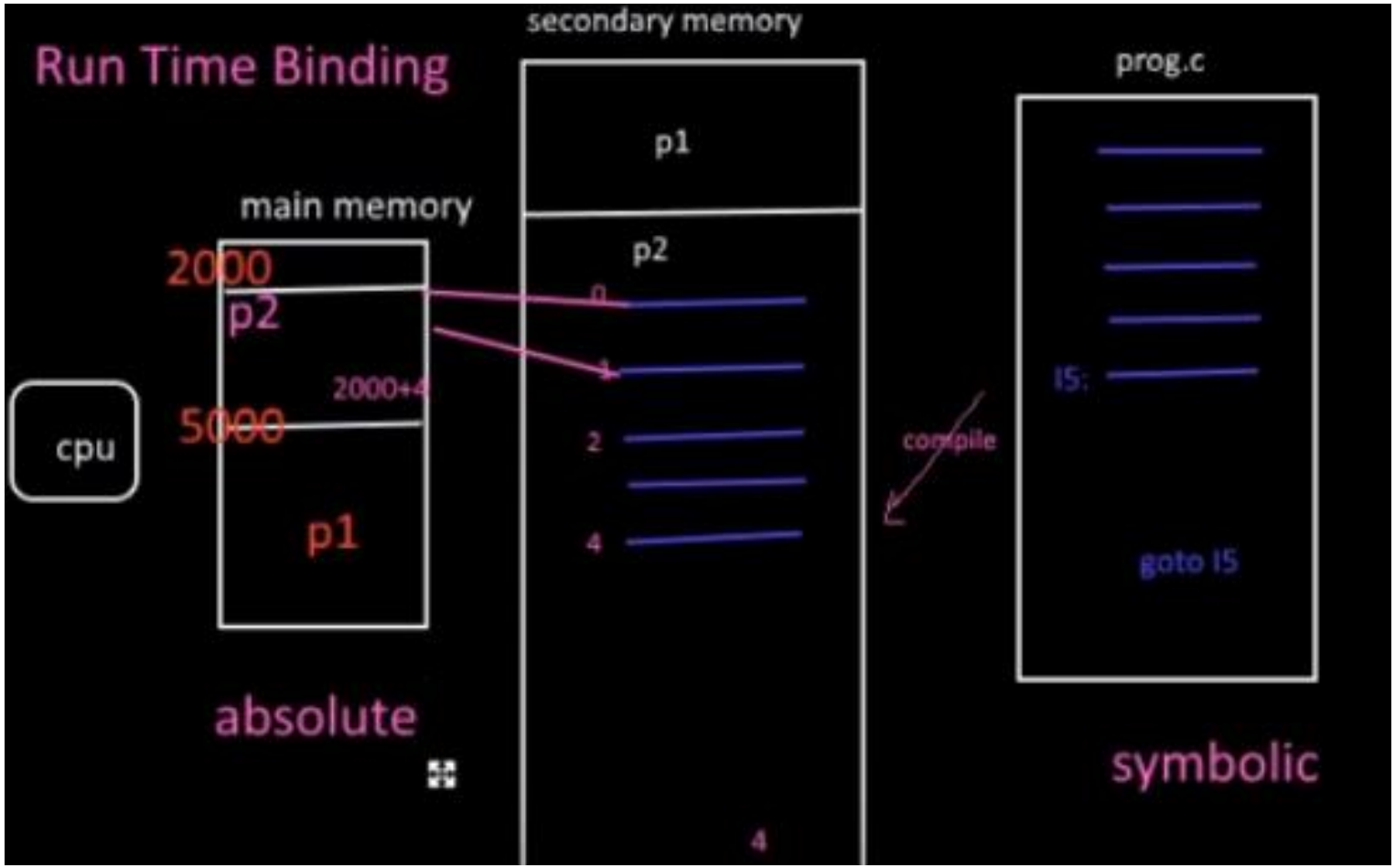
Advantage: Minimum setup time.

Disadvantage: Program to get loaded into the main memory is targeting to a memory location is occupied by any other program, then collision will occur. The current program will override.

Types of Address Binding



Types of Address Binding



Types of Address Binding

- **Load time –** **Must Generate re-locatable code if memory location is not known at compile time.**
- If it is not known at the compile time where the process will reside, then a relocatable address will be generated. The loader translates the relocatable address to an absolute address. The base address of the process in main memory is added to all logical addresses by the loader to generate an absolute address. In this, if the base address of the process changes, then we need to reload the process again.
- **Execution time** – The instructions are in memory and are being processed by the CPU. Additional memory may be allocated and/or deallocated at this time. This is used if a process can be moved from one memory to another during execution(dynamic linking-Linking that is done during load or run time). e.g – Compaction.

Address Binding:(contd.)

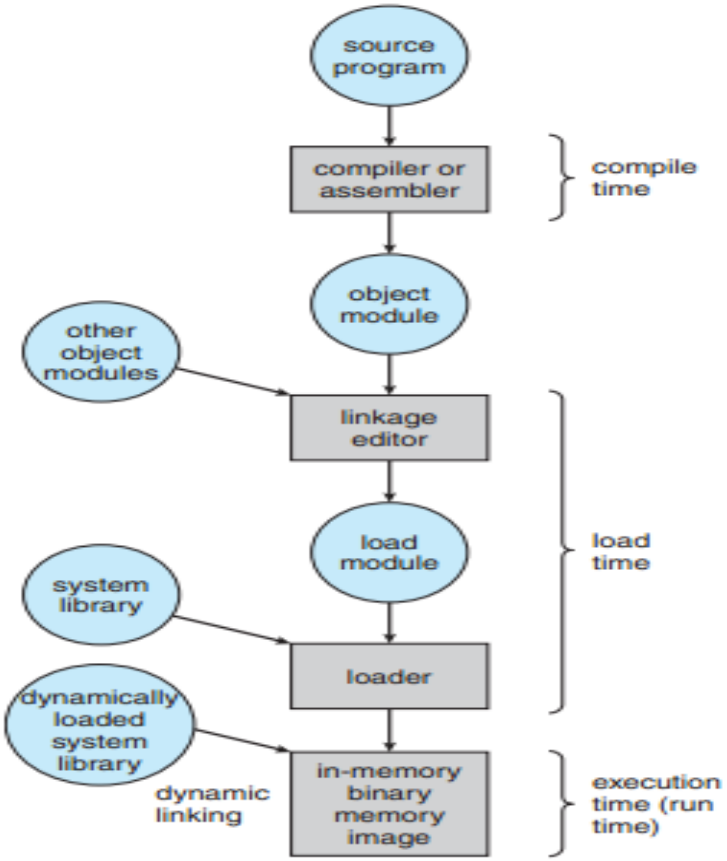


Figure 8.3 Multistep processing of a user program.

Mapping Virtual Addresses to Physical Addresses

- In Contiguous memory allocation mapping from virtual addresses to physical addresses is not a difficult task, because if we take a process from secondary memory and copy it to the main memory, the addresses will be stored in a contiguous manner, so if we know the base address of the process, we can find out the next addresses.
- The Memory Management Unit is a combination of 2 registers –

1. Base Register (Relocation Register)
2. Limit Register.

- **Base Register –** contains the starting physical address of the process.
- **Limit Register** -mentions the limit relative to the base address on the region occupied by the process.

Memory Protection

- The relocation-register scheme provides an effective way to allow the operating system's size to change dynamically. This flexibility is desirable in many situations.

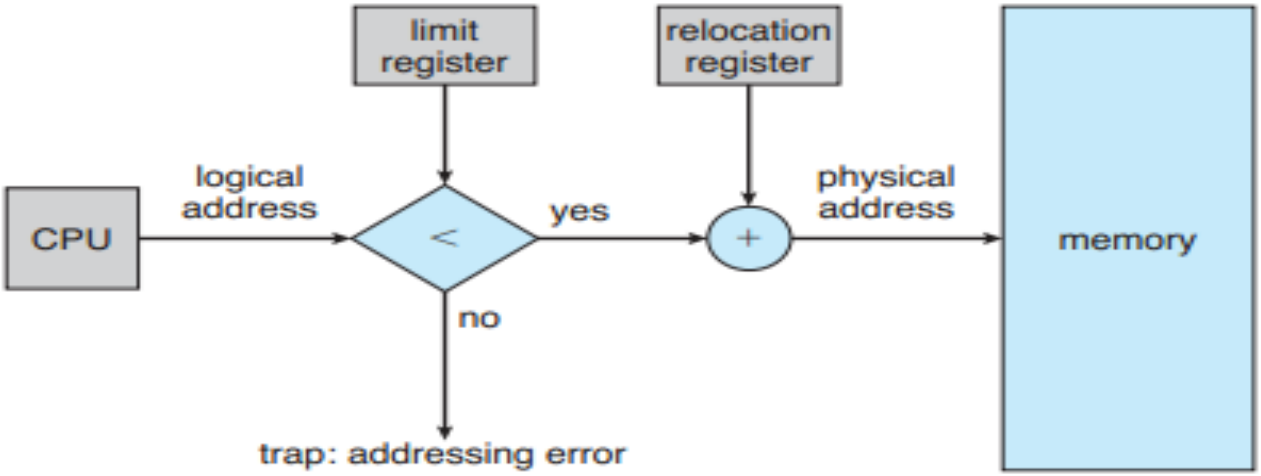


Figure 8.6 Hardware support for relocation and limit registers.

Memory Protection

- If we have a system with a relocation register, together with a limit register, we accomplish our goal. The relocation register contains the value of the smallest physical address; the limit register contains the range of logical addresses (for example, relocation = 100040 and limit = 74600). Each logical address must fall within the range specified by the limit register. The MMU maps the logical address dynamically by adding the value in the relocation register. This mapped address is sent to memory (Figure 8.6).
- When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch.

Logical Versus Physical Address Space

- An address generated by the CPU is commonly referred to as a **logical address**
- An address seen by the memory unit — that is, the one loaded into the memory-address register of the memory — is commonly referred to as a **physical address.**
- The compile-time and load-time address-binding methods generate identical logical and physical addresses. However, the execution-time address-binding scheme results in differing logical and physical addresses.
- We usually refer to the logical address as a virtual address.

Logical Versus Physical Address Space

- The set of all logical addresses generated by a program is a **logical address space** .
- The set of all physical addresses corresponding to these logical addresses is a **physical address space.** Thus, in the execution-time address-binding scheme, the logical and physical address spaces differ.
- The run-time mapping from virtual to physical addresses is done by a hardware device called the **memory-management unit(MMU).**
- We illustrate this mapping with a simple MMU scheme that is a generalization of the base-register scheme. The base register is now called a relocation register.
- The value in the relocation register is added to every address generated by a user process at the time the address is sent to memory (see Figure 8.4)

Logical Versus Physical Address Space(Contd.)

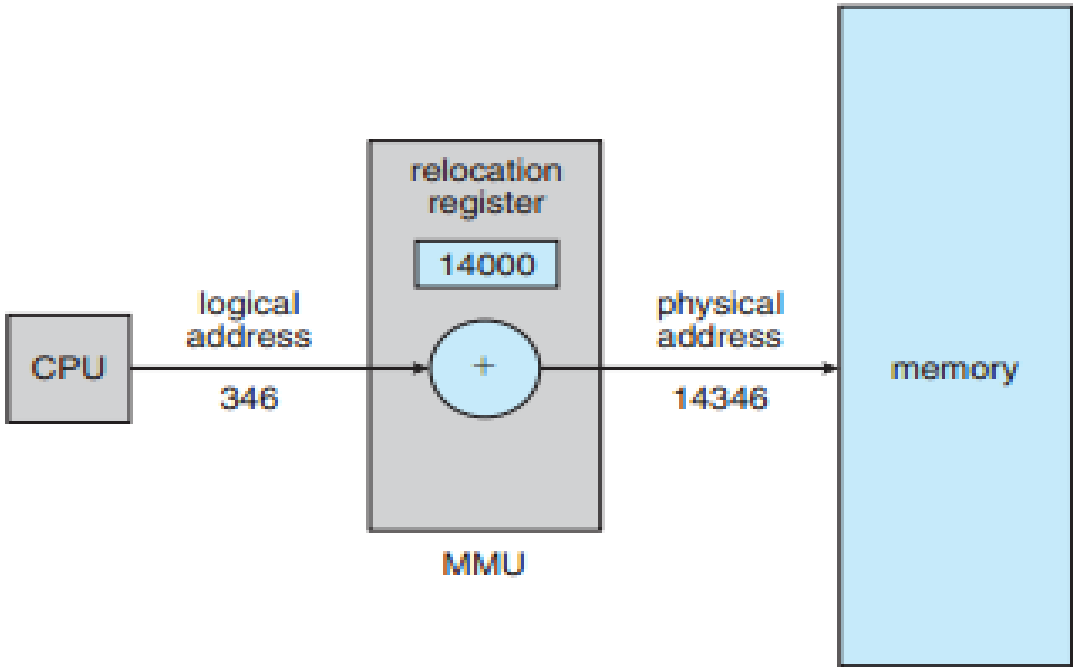


Figure 8.4 Dynamic relocation using a relocation register.

Logical Versus Physical Address Space(Contd.)

Parameter	LOGICAL ADDRESS	PHYSICAL ADDRESS
Basic	generated by CPU	location in a memory unit
Address Space	Logical Address Space is set of all logical addresses generated by CPU in reference to a program.	Physical Address is set of all physical addresses mapped to the corresponding logical addresses.
Visibility	User can view the logical address of a program.	User can never view physical address of program.
Generation	generated by the CPU	Computed by MMU
Access	The user can use the logical address to access the physical address.	The user can indirectly access physical address but not directly.

Swapping

- A process must be in memory to be executed. A process, however, can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution
- Standard swapping involves moving processes between main memory and a backing store.
- The backing store is commonly a fast disk. It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images.
- The system maintains a ready queue consisting of all processes whose memory images are on the backing store or in memory and are ready to run.
- Whenever the CPU scheduler decides to execute a process, it calls the dispatcher.

Swapping(Contd.)

- The dispatcher checks to see whether the next process in the queue is in memory. If it is not, and if there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process. It then reloads registers and transfers control to the selected process.

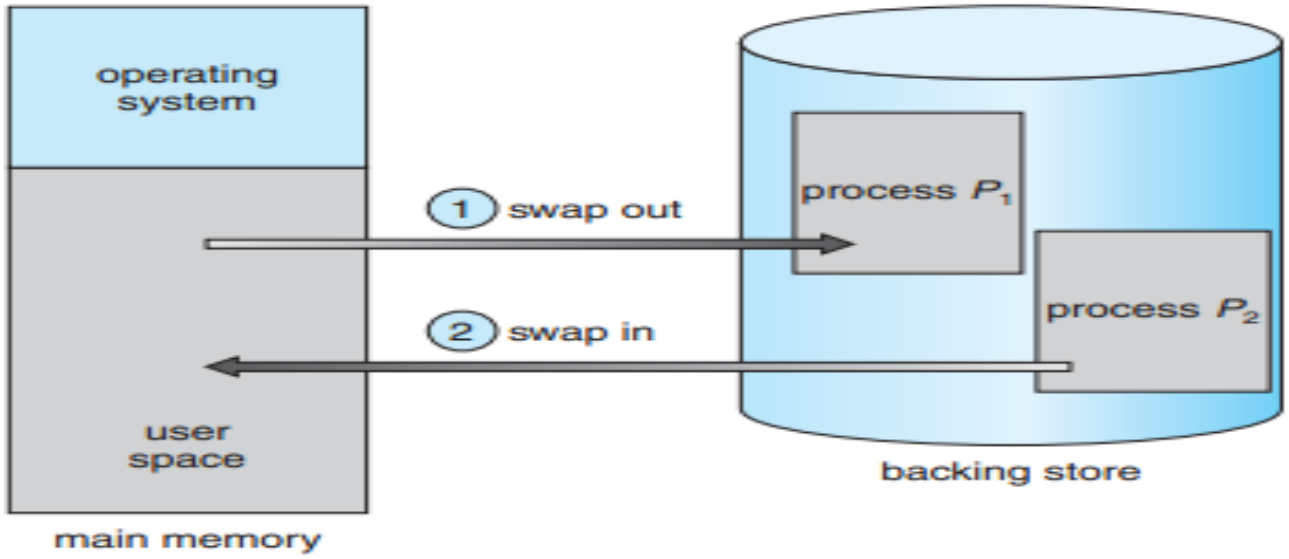


Figure 8.5 Swapping of two processes using a disk as a backing store.

Contiguous Memory Allocation

- The memory is usually divided into two partitions: one for the resident operating system and one for the user processes. We can place the operating system in either low memory or high memory. The major factor affecting this decision is the location of the interrupt vector. Since the interrupt vector is often in low memory, programmers usually place the operating system in low memory as well.
- Thus, in this text, we discuss only the situation in which the operating system resides in low memory.
- One of the simplest methods for allocating memory is to divide memory into several **single/fixed-sized partitions**. Each partition may contain exactly one process. Thus, the degree of multiprogramming is bound by the number of partitions.

Contiguous Memory Allocation(Contd.)

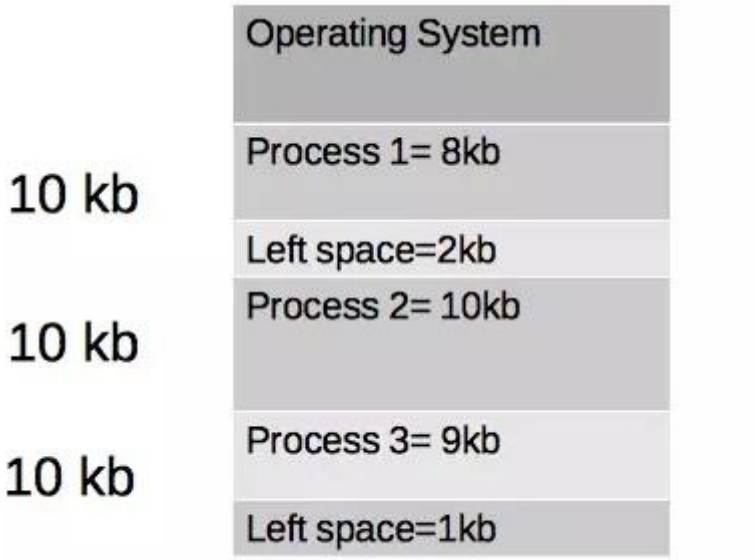


Fig. Single Partition/Fixed Size Partition

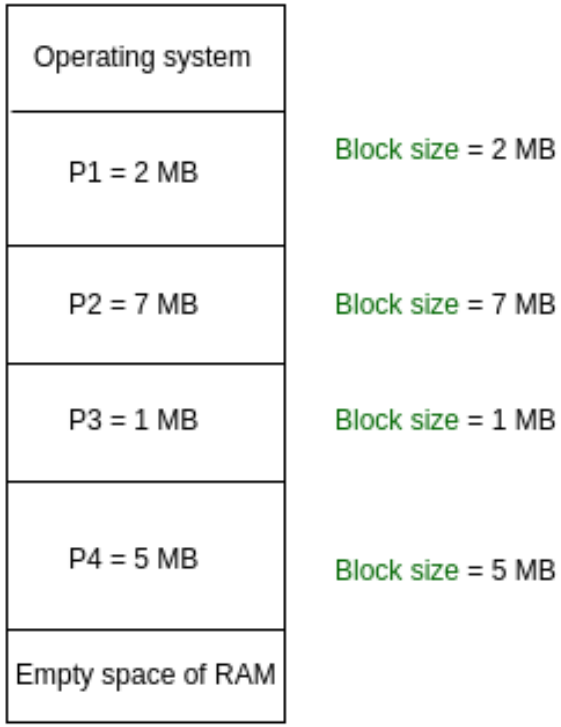
Contiguous Memory Allocation(Contd.)

Variable Partitioning – It is a part of Contiguous allocation technique. It is used to alleviate the problem faced by Fixed Partitioning. In contrast with fixed partitioning, partitions are not made before the execution or during system configure. Various **features** associated with variable Partitioning-

- Initially RAM is empty and partitions are made during the run-time according to process's need instead of partitioning during system configure.
- The size of partition will be equal to incoming process.
- The partition size varies according to the need of the process so that the internal fragmentation can be avoided to ensure efficient utilisation of RAM.
- Number of partitions in RAM is not fixed and depends on the number of incoming process and Main Memory's size.

Contiguous Memory Allocation(Contd.)

Dynamic partitioning



Partition size = process size
So, no internal Fragmentation

NON-CONTIGUOUS/ DYNAMIC STORAGE ALLOCATION

- **First fit.** Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.
- **Best fit.** Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.
- **Worst fit.** Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach.

Problem 01

- 1. Given five memory partitions of 100Kb, 500Kb, 200Kb, 300Kb, 600Kb (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of 212 Kb, 417 Kb, 112 Kb, and 426 Kb (in order)? Which algorithm makes the most efficient use of memory?

Solution 01

First-fit:

- 212K is put in 500K partition
- 417K is put in 600K partition
- 112K is put in 288K partition (new partition 288K = 500K - 212K)
- 426K must wait

Best-fit:

- 212K is put in 300K partition
- 417K is put in 500K partition

Solution 01(Contd.)

- 112K is put in 200K partition
- 426K is put in 600K partition
- **Worst-fit:**
- 212K is put in 600K partition
- 417K is put in 500K partition
- 112K is put in 388K partition
- 426K must wait

- In this example, best-fit turns out to be the best.

Problem 02

Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB and 250 KB. These partitions need to be allocated to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order.

Perform the allocation of processes using-

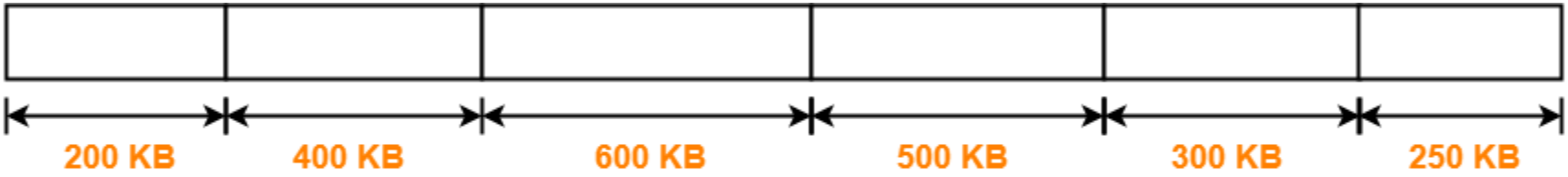
- First Fit Algorithm
- Best Fit Algorithm
- Worst Fit Algorithm

Problem 02

Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB and 250 KB. These partitions need to be allocated to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order.

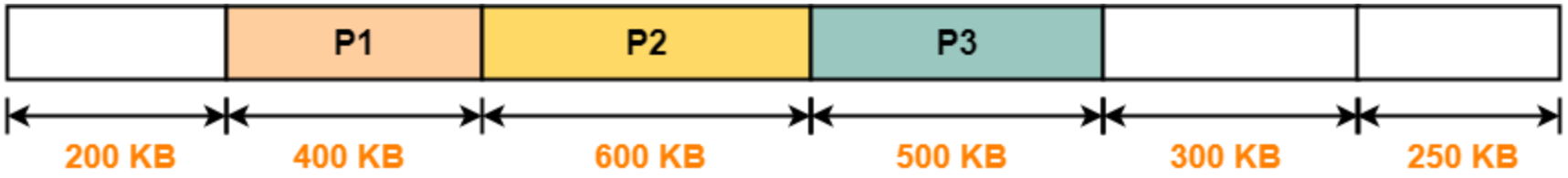
Problem 02

Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB and 250 KB. These partitions need to be allocated to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order.



Main Memory

Allocation Using First Fit Algorithm-



Main Memory

Problem 02

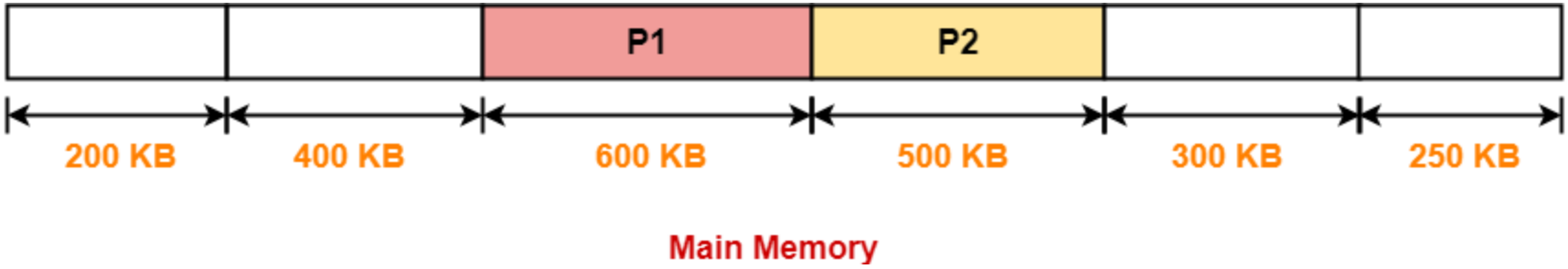
Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB and 250 KB. These partitions need to be allocated to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order.

Allocation Using Best Fit Algorithm-

Problem 02

Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB and 250 KB. These partitions need to be allocated to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order.

Allocation Using Worst fit Algorithm-

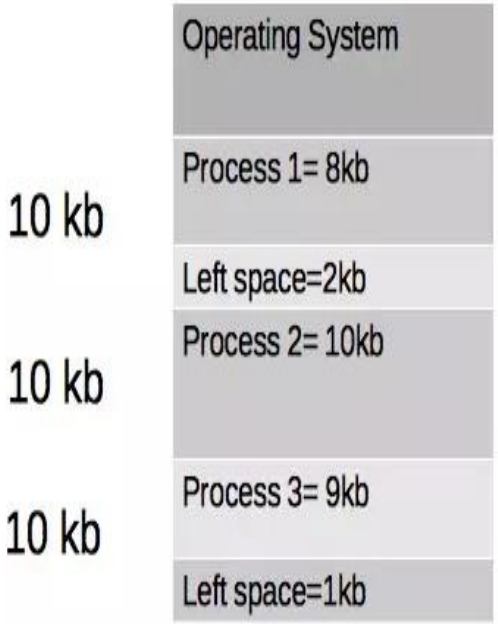


Contiguous v/s Non- Contiguous Memory Allocation

S.NO	Contiguous Memory Allocation	Non-Contiguous Memory Allocation
1.	Contiguous memory allocation allocates consecutive blocks of memory to a file/process.	Non-Contiguous memory allocation allocates separate blocks of memory to a file/process.
2.	Faster in Execution.	Slower in Execution.
3.	It is easier for the OS to control.	It is difficult for the OS to control.
4.	Overhead is minimum as not much address translations are there while executing a process.	More Overheads are there as there are more address translations.
5.	Both Internal fragmentation and external fragmentation occurs in Contiguous memory allocation method.	External fragmentation occurs in Non-Contiguous memory allocation method.
6.	It includes single partition allocation and multi-partition allocation.	It includes paging and segmentation.
7.	Wastage of memory is there.	No memory wastage is there.

Internal Fragmentation

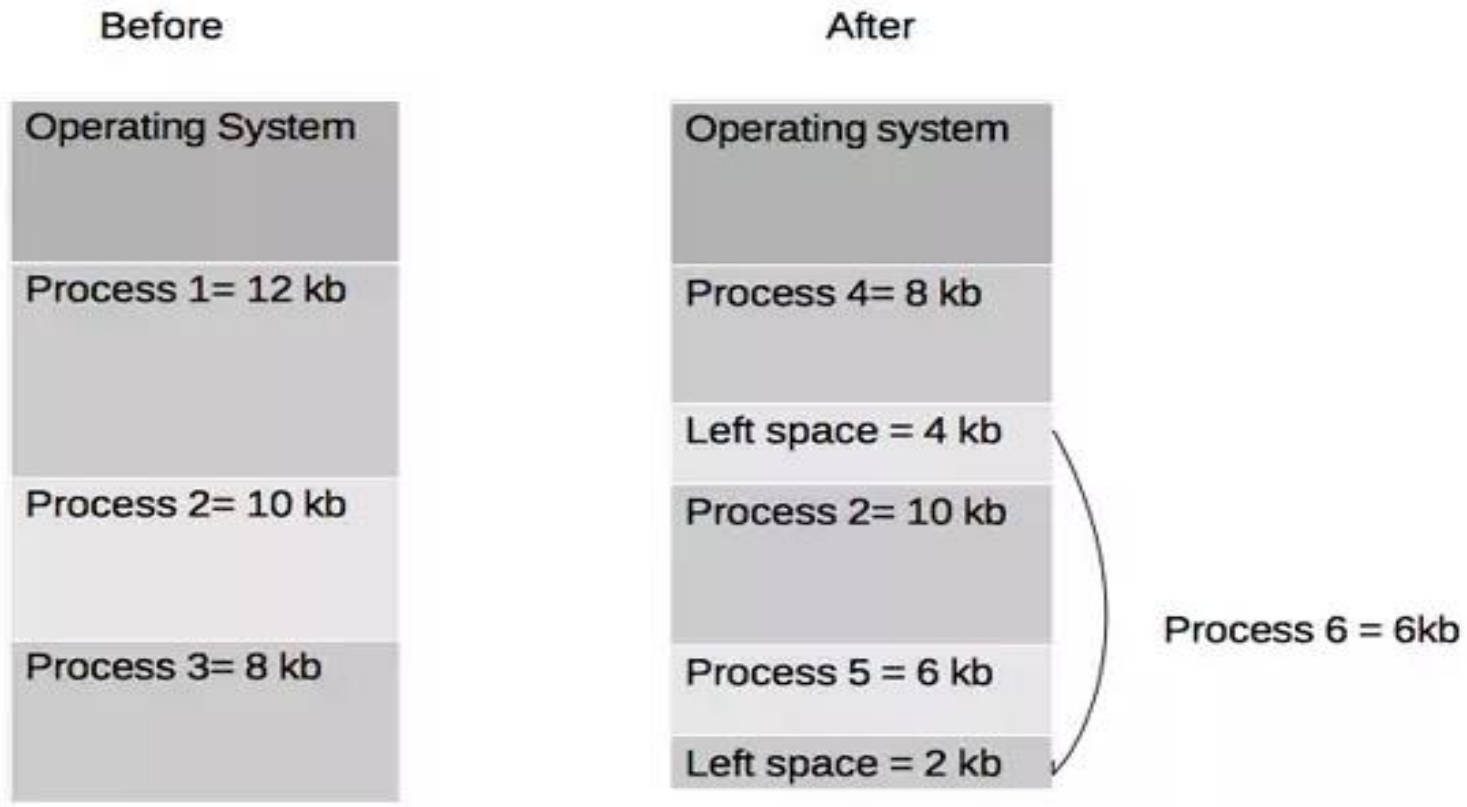
- It arises when we use fixed sized partitioning.
- Some part of the memory is kept for operating system and the rest is available for user space.
- In this case the user space is divided into blocks of 10 KB each.
- When process 1 with size 8 KB is allocated a block of 10 KB, 2 KB space is left unused. When process 2 with size 10 KB is allocated a !0 KB block no space is left unused. When process 3 with size 9KB is allocated a 10 KB block, 1KB is left unused.
- Here 2 processes are allocated space more than required and this unused space is so small to store a new process and is wasted. This is called as INTERNAL FRAGMENTATION.



External Fragmentation

- It arises when dynamic partitioning technique is used.
- Here memory is allocated to the processes dynamically based on their size.
- So in the above example, the user space contains processes 1,2 & 3 Out of which process 1 & 3 complete their execution and are swapped out and two other processes, process 4 & 5 are swapped in their places.
- Process 4 takes place of process 1 but as its size is only 8 KB, it is allocated only 8 KB and rest is left unused.
- Process 5 takes place of process 3. It is allocated 6 KB space and 8 KB is left unused.
- Now suppose a new process, process 6 wants to be swapped in and its size is 6 KB. Though we have total 6 KB space but we cannot service this request as these blocks are not contiguous(adjacent). This is called as **EXTERNAL FRAGMENTATION**.

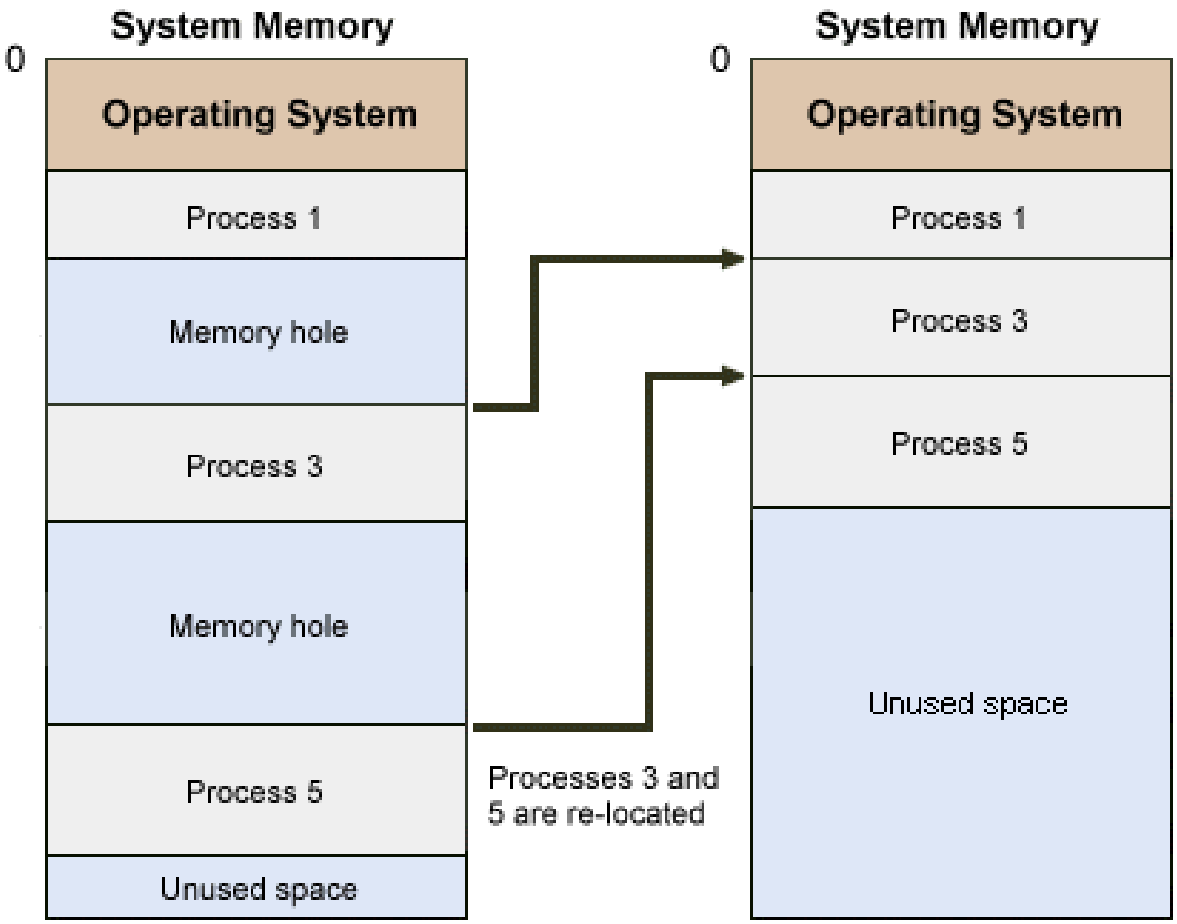
External Fragmentation



Compaction

- We got to know that the dynamic partitioning suffers from external fragmentation. However, this can cause some serious problems.
- To avoid compaction, we need to change the rule which says that the process can't be stored in the different places in the memory.
- We can also use compaction to minimize the probability of external fragmentation. In compaction, all the free partitions are made contiguous and all the loaded partitions are brought together.
- By applying this technique, we can store the bigger processes in the memory. The free partitions are merged which can now be allocated according to the needs of new processes. This technique is also called defragmentation.

Compaction


System Memory
0
Operating System
Process 1
Memory hole
Process 3
Memory hole
Process 5
Unused space
Processes 3 and 5 are re-located
System Memory
0
Operating System
Process 1
Process 3
Process 5
Unused space

Comparison Between External & Internal Fragmentation

BASIS FOR COMPARISON	INTERNAL FRAGMENTATION	EXTERNAL FRAGMENTATION
Basic	It occurs when fixed sized memory blocks are allocated to the processes.	It occurs when variable size memory space are allocated to the processes dynamically.
Occurrence	When the memory assigned to the process is slightly larger than the memory requested by the process this creates free space in the allocated block causing internal fragmentation.	When the process is removed from the memory, it creates the free space in the memory causing external fragmentation.
Solution	The memory must be partitioned into variable sized blocks and assign the best fit block to the process.	Compaction, paging and segmentation.

Exercise: External Fragmentation

Assume 140K, 260K, 60K memory is free. What is the total external fragmentation arises for the following requests 110K, 30K, 210K, 50K using Best-fit policy assume no compaction

(A) 120 K (B) 110 K (C) 60 K (D) 30 K

Exercise: External Fragmentation

Assume 140K, 260K, 60K memory is free. What is the total external fragmentation arises for the following requests 110K, 30K, 210K, 50K using Best-fit policy assume no compaction

(A) 120 K (B) 110 K (C) 60 K (D) 30 K

Total External Fragmentation= (140-110)+(60-30)= 60 k

dynamic partitions allow more than one process in same block of partition.so answer is 60K only. If we have to calculate Internal Fragmentation then Answer will be 110 k because 50k process will not be occupied in same partition

Paging

- **Paging** Segmentation permits the physical address space of a process to be non-contiguous.
- Paging is another memory-management scheme that offers this advantage.
- However, paging avoids external fragmentation and the need for compaction, whereas segmentation does not.
- It also solves the considerable problem of fitting memory chunks of varying sizes onto the backing store.

Basic Method: Paging

- The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages.
- When a process is to be executed, its pages are loaded into any available memory frames from their source (a file system or the backing store).
- The backing store is divided into fixed-sized blocks that are the same size as the memory frames or clusters of multiple frames
- Every address generated by the CPU is divided into two parts: a page number (p) and a page offset (d).
- The page number is used as an index into a page table. The page table contains the base address of each page in physical memory.
- This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

Paging Hardware

Address generated by CPU is divided into

- **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
- **Page offset(d):** Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

Physical Address is divided into

- **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
- **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

Paging Hardware

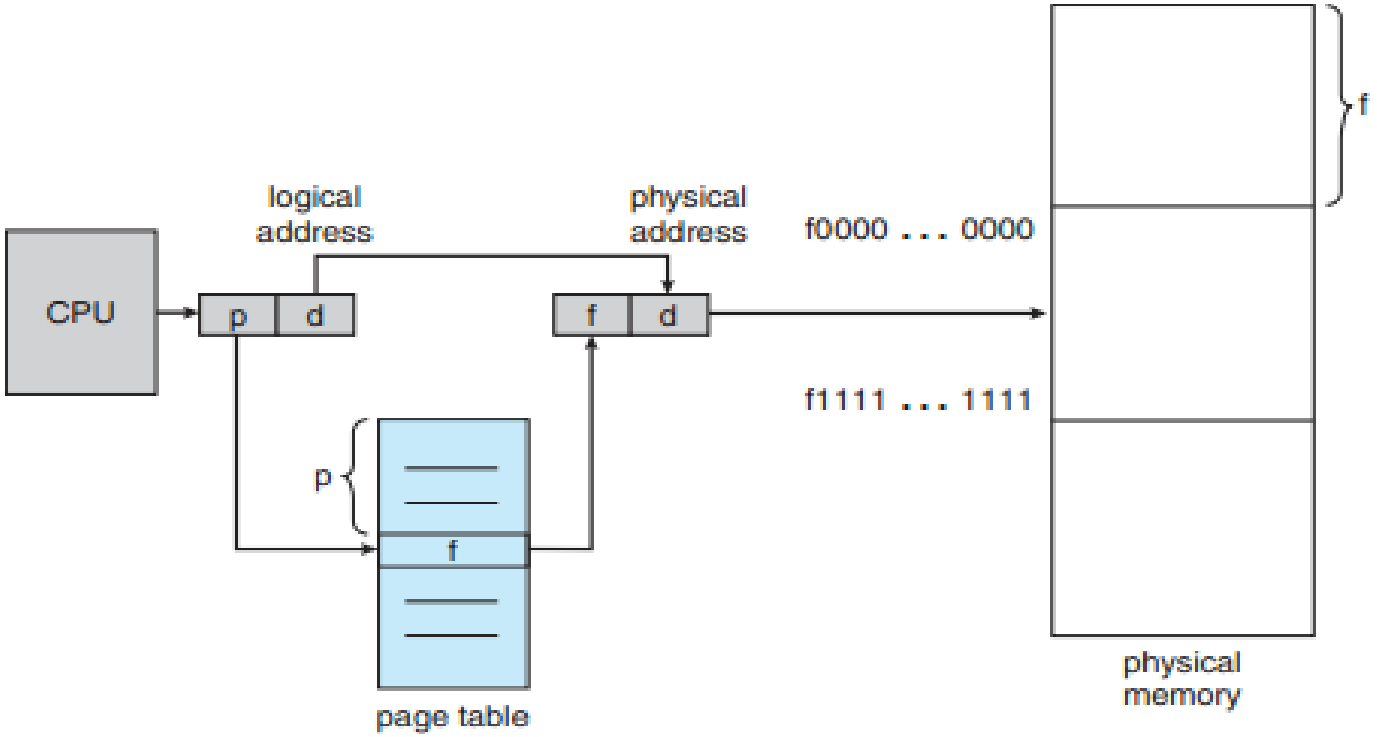


Figure 8.10 Paging hardware.

Paging

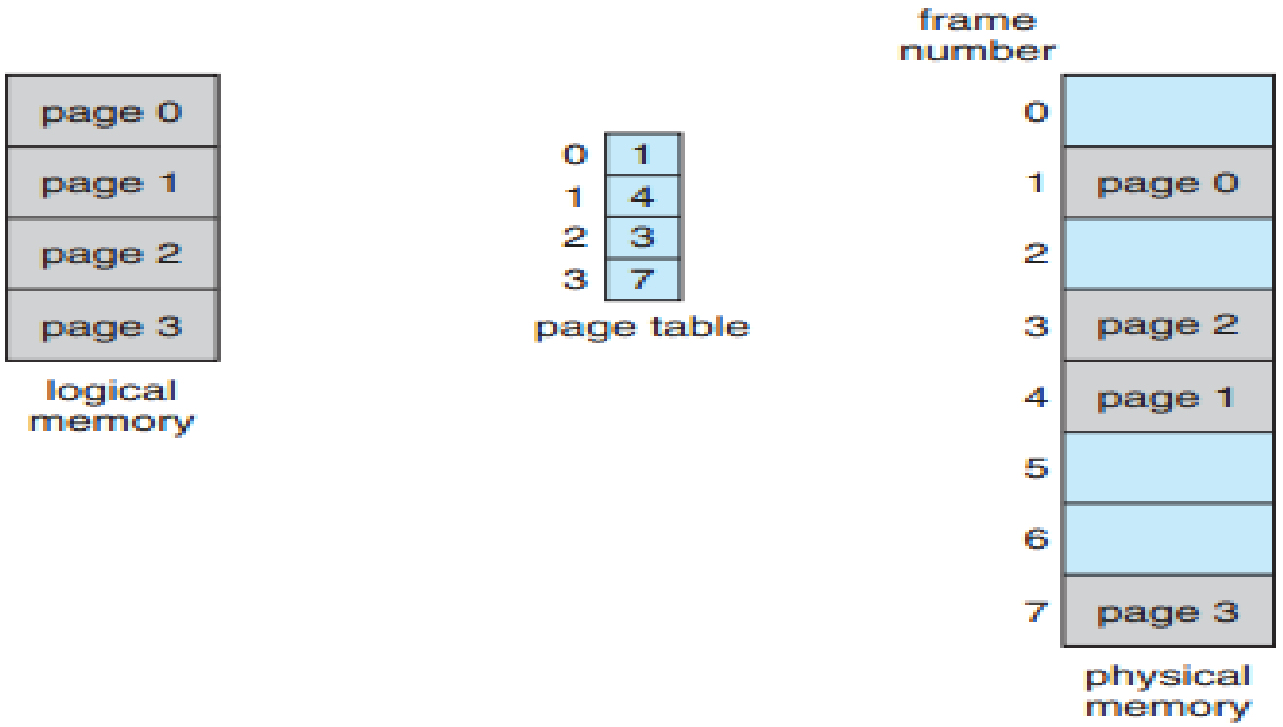


Figure 8.11 Paging model of logical and physical memory.

Points to remember

For Main Memory-

- Physical Address Space = Size of main memory
- Size of main memory = Total number of frames x Page size
- Frame size = Page size
- If number of frames in main memory = 2^X, then number of bits in frame number = X bits
- If Page size = 2^X Bytes, then number of bits in page offset = X bits
- If size of main memory = 2^X Bytes, then number of bits in physical address = X bits

Points to remember

- 1 byte = 8 bits
- 1 kilobyte (K / Kb) = 2^10 bytes = 1,024 bytes
- 1 megabyte (M / MB) = 2^20 bytes = 1,048,576 bytes
- 1 gigabyte (G / GB) = 2^30 bytes = 1,073,741,824 bytes
- 1 terabyte (T / TB) = 2^40 bytes = 1,099,511,627,776 bytes
- 1 petabyte (P / PB) = 2^50 bytes = 1,125,899,906,842,624 bytes
- 1 exabyte (E / EB) = 2^60 bytes = 1,152,921,504,606,846,976 bytes

Problems on Paging

Q1) Calculate the size of memory if its address consists of 22 bits and the memory is 2-byte addressable.

Ans. We have-

Number of locations possible with 22 bits = 2^{22} locations

It is given that the size of one location = 2 bytes

Size of main memory = Total number of frames x Page size

Thus, Size of memory

= 2^{22} x 2

= 2^{23} bytes

= 8 MB

Problems on Paging

Q:2: Calculate the number of bits required in the address for memory having size of 16 GB. Assume the memory is 4-byte addressable.

Solution-

Let 'n' number of bits are required.

Size of main memory = Total number of frames x Page size

Then, Size of memory = 2^n x 4 bytes.

Since, the given memory has size of 16 GB, so we have-

2^n x 4 bytes = 16 GB

2^n x 4 = 16 G

2^n x 2^2 = 2^{34}

2^n = 2^{32}

∴ n = 32 bits

Points to remember

For Process-

- Virtual Address Space = Size of process
- Number of pages the process is divided = Process size / Page size
- If process size = 2^X bytes, then number of bits in virtual address space = X bits

Problems on Paging

Q2) Evaluating the maximum number of pages needed If a system supports 16 bit address line and 1K page size.

Ans. Virtual Address space= 2^{16} bytes

Page size= 1k = 2^{10} bytes

Number of pages= Virtual Address space/page size

= $2^{16}/2^{10}$

=2^{6} =64 pages

Points to remember

For Page Table-

- Size of page table = Number of entries in page table x Page table entry size
- Number of entries in pages table = Number of pages the process is divided
- Page table entry size = Number of bits in frame number + Number of bits used for optional fields if any

Points to remember

NOTE-

- In general, if the given address consists of 'n' bits, then using 'n' bits, 2^n locations are possible.
- Then, size of memory = 2^n x Size of one location.
- If the memory is byte-addressable, then size of one location = 1 byte.
- Thus, size of memory = 2^n bytes.
- If the memory is word-addressable where 1 word = m bytes, then size of one location = m bytes.
- Thus, size of memory = 2^n x m bytes.
- Let's say, Page size = 4 KB = 2 ^ 12 Bytes then Page offset = 12

Problems on Paging

Q4) Consider a system with byte-addressable memory, 32 bit logical addresses, 4 kilobyte page size and page table entries of 4 bytes each. The size of the page table in the system in megabytes is _____.

Ans. Given-Number of bits in logical address = 32 bits

Page size = 4KB

Page table entry size = 4 bytes

Process Size-

Number of bits in logical address = 32 bits

Thus,

Process size

$= 2^{32}$ B

= 4 GB

Problems on Paging(contd.)

Number of Entries in Page Table-

Number of pages the process is divided
= Process size / Page size
= 4 GB / 4 KB
= 2^{20} pages

Thus, Number of entries in page table = 2^{20} entries

Page Table Size-

Page table size
= Number of entries in page table x Page table entry size
= 2^{20} x 4 bytes
= 4 MB

Problems on Paging

Q: Consider a machine with 64 MB physical memory and a 32 bit virtual address space. If the page size is 4 KB, what is the approximate size of the page table?

16 MB

8 MB

2 MB

24 MB

Problems on Paging

Q: Consider a machine with 64 MB physical memory and a 32 bit virtual address space. If the page size is 4 KB, what is the approximate size of the page table?

Ans

Given-

Size of main memory = 64 MB

Number of bits in virtual address space = 32 bits

Page size = 4 KB

We will consider that the memory is byte addressable.

Number of Bits in Physical Address-

Size of main memory

= 64 MB

= 2^{26} B

Thus, Number of bits in physical address = 26 bits

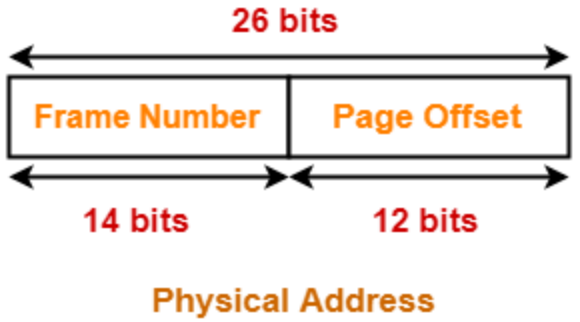
Problems on Paging(contd.)

Number of Frames in Main Memory-

Number of frames in main memory
= Size of main memory / Frame size
= 64 MB / 4 KB
= 2^{26} B / 2^{12} B
= 2^{14}
Thus, Number of bits in frame number = 14 bits

Number of Bits in Page Offset-

We have,
Page size
= 4 KB
= 2^{12} B
Thus, Number of bits in page offset = 12 bits



Problems on Paging(contd.)

Process Size-

Number of bits in virtual address space = 32 bits

Thus,

Process size

= 2^{32} B

= 4 GB

Number of Entries in Page Table-

Number of pages the process is divided

= Process size / Page size

= 4 GB / 4 KB

= 2^{20} pages

Thus, Number of entries in page table = 2^{20} entries

Problems on Paging(contd.)

Page Table Size-

Page table size
= Number of entries in page table x Page table entry size
= Number of entries in page table x Number of bits in frame number
= 2^{20} x 14 bits
= 2^{20} x 16 bits (Approximating 14 bits ≈ 16 bits)
= 2^{20} x 2 bytes
= 2 MB

Thus, Option (C) is correct.

Problems on Paging

Q: Consider a logical space of 64 pages of 1024 words each mapped onto a physical memory of 32 frames.

a. how many bits are there in the logical address

b. how many bits are there in physical address

Ans.

The logical address requires 6 bits for the page number because there are $64 = 2^6$ of them. Then, the logical address requires 10 bits of the offset because there are $1024 = 2^{10}$ of them. So, the logical address requires a total is 16 bits.

The physical address requires 5 bits for the frame number because there are $32 = 2^5$ of them. Then, the physical address also requires 10 bits of the offset because there are $1024 = 2^{10}$ of them. So, the physical address requires a total is 15 bits.

Disadvantage Of Paging

One major disadvantage of paging is-

- It increases the **effective access time** due to increased number of memory accesses.
- One memory access is required to get the frame number from the page table.
- Another memory access is required to get the word from the page.

References

- "Operating System Concepts" by Avi Silberschatz and Peter Galvin
- "Operating Systems: Internals and Design Principles" by William Stallings
- www.tutorialpoint.com
- www.javapoint.com

Thank You

For any Query, you can contact

Er. Inderjeet Singh(e8822)

Ph. No: 86-99-100-160

Email id: inderjeet.e8822@cumail.in

University Institute of Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Unit- 2

Subject Name : Operating System

Chapter : Memory Management- (Part-2)

Prepared By: Er. Inderjeet Singh(e8822)

DISCOVER . LEARN . EMPOWER

Outline

- Paging with TLB
- Exercise on Paging with TLB
- Multi-Level Paging
- Problems on Multi-Level Paging
- Segmentation and Exercise Problems

Paging with TLB:

- Translation Look-Aside Buffer, a table in the processors memory that contains information about the pages in memory the processor has accessed recently. The TLB enables faster computing because it allows the address processing to take place independent of the normal address-translation pipeline.

- The TLB is associative, high-speed memory. Each entry in the TLB consists of two parts: a key (or tag) and a value. When the associative memory is presented with an item, the item is compared with all keys simultaneously. If the item is found, the corresponding value field is returned. The search is fast; a TLB lookup in modern hardware is part of the instruction pipeline, essentially adding no performance penalty. To be able to execute the search within a pipeline step, however, the TLB must be kept small. It is typically between 32 and 1,024 entries in size.

Paging with TLB:

The logical address generated by the CPU is translated into the physical address using following steps-

Step-01:

CPU generates a logical address consisting of two parts-

- Page Number
- Page Offset

Step-02:

- TLB is checked to see if it contains an entry for the referenced page number.
- The referenced page number is compared with the TLB entries all at once.

Paging with TLB:

Now, two cases are possible-

Case-01: If there is a TLB hit-

- If TLB contains an entry for the referenced page number, a TLB hit occurs.
- In this case, TLB entry is used to get the corresponding frame number for the referenced page number.

Case-02: If there is a TLB miss-

- If TLB does not contain an entry for the referenced page number, a TLB miss occurs.
- In this case, page table is used to get the corresponding frame number for the referenced page number.
- Then, TLB is updated with the page number and frame number for future references.

Paging with TLB:

Step-03:

- After the frame number is obtained, it is combined with the page offset to generate the physical address.
- Then, physical address is used to read the required word from the main memory.

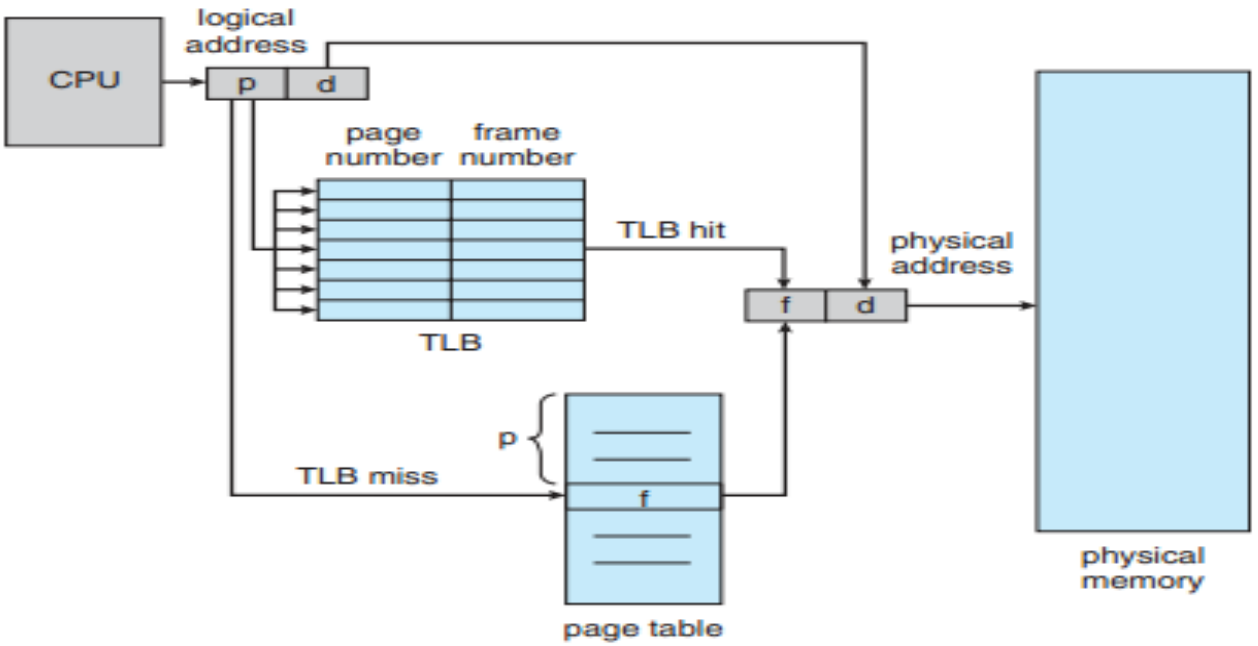


Figure 8.14 Paging hardware with TLB.

Paging with TLB:

- The TLB is used with page tables in the following way. The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB. If the page number is found, its frame number is immediately available and is used to access memory.
- In addition, we add the page number and frame number to the TLB, so that they will be found quickly on the next reference. If the TLB is already full of entries, an existing entry must be selected for replacement.

Paging with TLB:

The **advantages of using TLB are-**

- TLB reduces the effective access time.
- Only one memory access is required when TL

Disadvantages- A major disadvantage of using TLB is-

- After some time of running the process, when TLB hits increases and process starts to run smoothly, a context switching occurs.
- The entire content of the TLB is flushed.
- Then, TLB is again updated with the currently running process. This happens again and again.

Effective Access Time

- In a single level paging using TLB, the effective access time is given as-

Effective Access Time =

Hit ratio of TLB x { Access time of TLB + Access time of main memory }

+

Miss ratio of TLB x { Access time of TLB + 2 x Access time of main memory }

Problems on Paging with TLB

Problem 1: A paging scheme uses a Translation Lookaside buffer (TLB). A TLB access takes 10 ns and a main memory access takes 50 ns. What is the effective access time (in ns) if the TLB hit ratio is 90% and there is no page fault?

a) 54

b) 60

c) 65

d) 75

Solution: Given-

TLB access time = 10 ns

Main memory access time = 50 ns

TLB Hit ratio = 90% = 0.9

Problems on Paging with TLB

Calculating TLB Miss Ratio-

TLB Miss ratio
= 1 – TLB Hit ratio
= 1 – 0.9
= 0.1

Calculating Effective Access Time-

Substituting values in the above formula, we get-
Effective Access Time
= 0.9 x { 10 ns + 50 ns } + 0.1 x { 10 ns + 2 x 50 ns }
= 0.9 x 60 ns + 0.1 x 110 ns
= 54 ns + 11 ns
= 65 ns
Thus, Option (C) is correct.

Problems on Paging with TLB

Problem-02:A paging scheme uses a Translation Lookaside buffer (TLB). The effective memory access takes 160 ns and a main memory access takes 100 ns. What is the TLB access time (in ns) if the TLB hit ratio is 60% and there is no page fault?

a) 54

b) 60

c) 20

d) 75

Problems on Paging with TLB

Given-
Effective access time = 160 ns
Main memory access time = 100 ns
TLB Hit ratio = 60% = 0.6

Calculating TLB Miss Ratio-

TLB Miss ratio
= 1 – TLB Hit ratio
= 1 – 0.6
= 0.4

Calculating TLB Access Time-

Let TLB access time = T ns.
Substituting values in the above formula, we get-
160 ns = 0.6 x { T + 100 ns } + 0.4 x { T + 2 x 100 ns }
160 = 0.6 x T + 60 + 0.4 x T + 80
160 = T + 140
T = 160 – 140
T = 20

Multilevel Paging

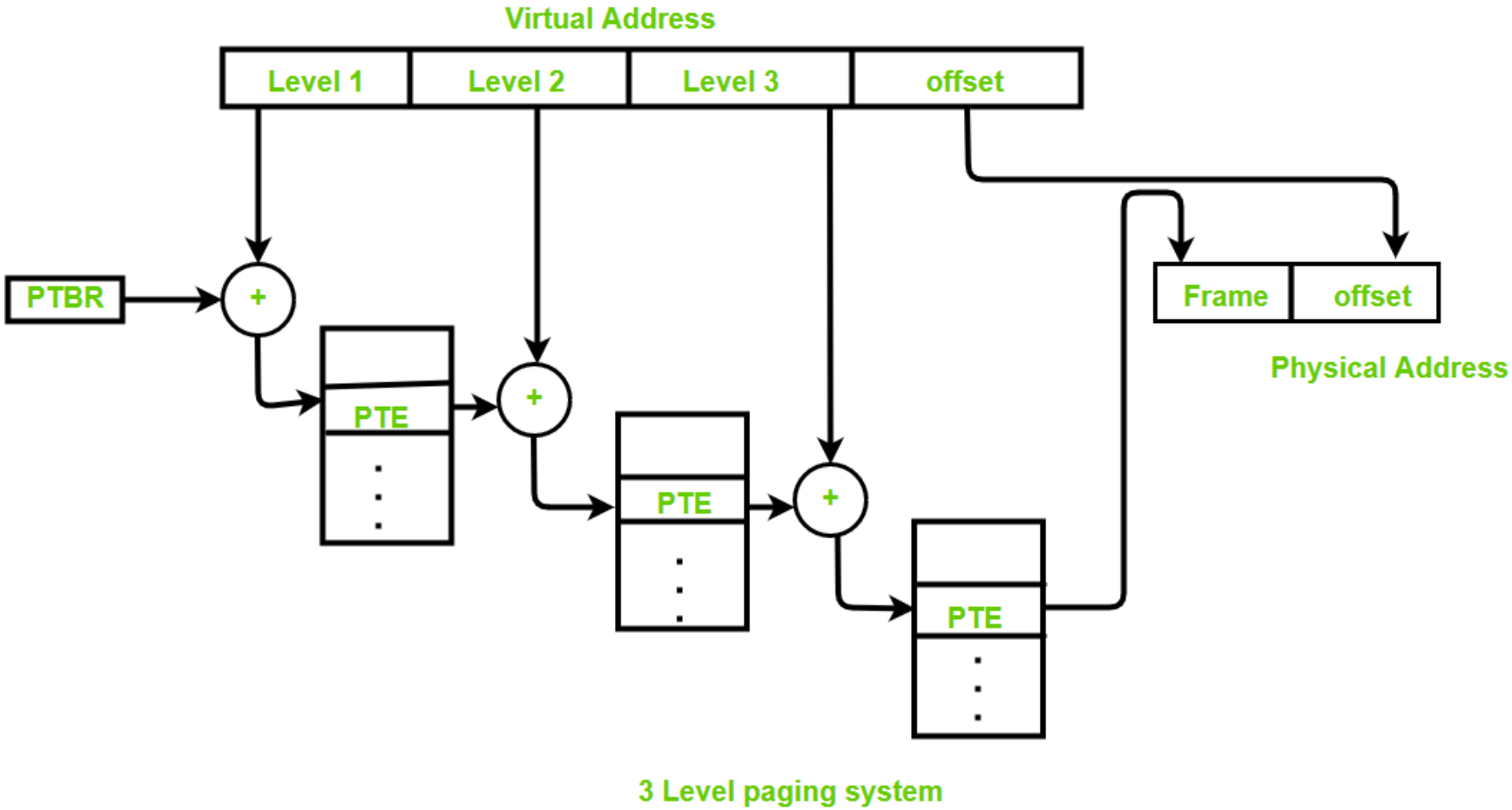
- **Multilevel Paging** is a paging scheme which consist of two or more levels of page tables in a hierarchical manner.
- It is also known as hierarchical paging.
- The entries of the level 1 page table are pointers to a level 2 page table and entries of the level 2 page tables are pointers to a level 3 page table and so on.
- The entries of the last level page table are stores actual frame information.
- Level 1 contain single page table and address of that table is stored in PTBR (Page Table Base Register).

Multilevel Paging

- Virtual Addresses

- In multilevel paging whatever may be levels of paging all the page tables will be stored in main memory.
- So it requires more than one memory access to get the physical address of page frame.
- One access for each level needed. Each page table entry except the last level page table entry contains base address of the next level page table.

Multilevel Paging



3 Level paging system

Multilevel Paging

- **Reference to actual page frame:**

- Reference to PTE in level 1 page table = PTBR value + Level 1 offset present in virtual address.
- Reference to PTE in level 2 page table = Base address (present in Level 1 PTE) + Level 2 offset (present in VA).
- Reference to PTE in level 3 page table= Base address (present in Level 2 PTE) + Level 3 offset (present in VA).
- Actual page frame address = PTE (present in level 3).

- Generally the page table size will be equal to the size of page.

Multilevel Paging

Assumptions:

Byte addressable memory, and n is the number of bits used to represent virtual address.

Important formula:

Number of entries in page table: = (virtual address space size) / (page size)= Number of pages

Virtual address space size: = 2^nB

Size of page table: <>= (number of entries in page table)*(size of PTE)

If page table size > desired size then create 1 more level.

Illustration of Multilevel Paging

Consider a system using paging scheme where-

Logical Address Space = 4 GB

Physical Address Space = 16 TB

Page size = 4 KB

Now, let us find how many levels of page table will be required.

Illustration of Multilevel Paging

Number of Bits in Physical Address-

Size of main memory =Physical Address Space

= 16 TB = 2^{44} B

Thus, Number of bits in physical address = 44 bits

Number of Frames in Main Memory-

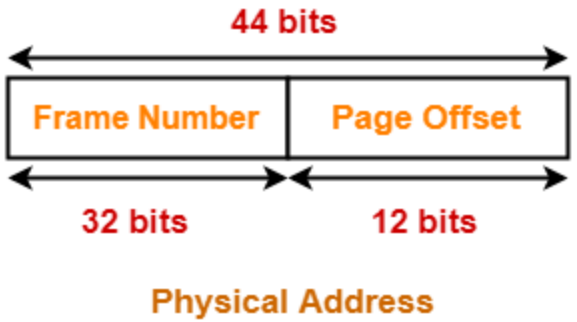
Number of frames in main memory

= Size of main memory / Frame size

= 16 TB / 4 KB

= 2^{32} frames

Thus, Number of bits in frame number = 32 bits



Number of Bits in Page Offset-

We have, Page size= 4 KB

= 2^{12} B

Thus, Number of bits in page offset = 12 bits

Illustration of Multilevel Paging

Number of Pages of Process-

Number of pages the process is divided = Process size / Page size

= 4 GB / 4 KB = 2^{20} pages

Inner Page Table Size-

Inner page table keeps track of the frames storing the pages of process.

Inner Page table size = Number of entries in inner page table x Page table entry size

= Number of pages the process is divided x Number of bits in frame number

= 2^{20} x 32 bits

= 2^{20} x 4 bytes

= 4 MB

Now, we can observe-

The size of inner page table is greater than the frame size (4 KB).

Thus, inner page table can not be stored in a single frame.

So, inner page table has to be divided into pages.

Illustration of Multilevel Paging

Number of Pages of Inner Page Table-

Number of pages the inner page table is divided = Inner page table size / Page size
= 4 MB / 4 KB
= 2^{10} pages

Now, these 2^{10} pages of inner page table are stored in different frames of the main memory.

Number of Page Table Entries in One Page of Inner Page Table-

Number of page table entries in one page of inner page table
= Page size / Page table entry size
= Page size / Number of bits in frame number
= 4 KB / 32 bits
= 4 KB / 4 B
= 2^{10}

Illustration of Multilevel Paging

Number of Bits Required to Search an Entry in One Page of Inner Page Table-

One page of inner page table contains 2^{10} entries.

Thus, Number of bits required to search a particular entry in one page of inner page table = 10 bits

Outer Page Table Size-

Outer page table is required to keep track of the frames storing the pages of inner page table.

Outer Page table size = Number of entries in outer page table x Page table entry size

= Number of pages the inner page table is divided x Number of bits in frame number

= 2^{10} x 32 bits

= 2^{10} x 4 bytes

= 4 KB

Illustration of Multilevel Paging

Now, we can observe-

The size of outer page table is same as frame size (4 KB).

Thus, outer page table can be stored in a single frame.

So, for given system, we will have two levels of page table.

Page Table Base Register (PTBR) will store the base address of the outer page table.

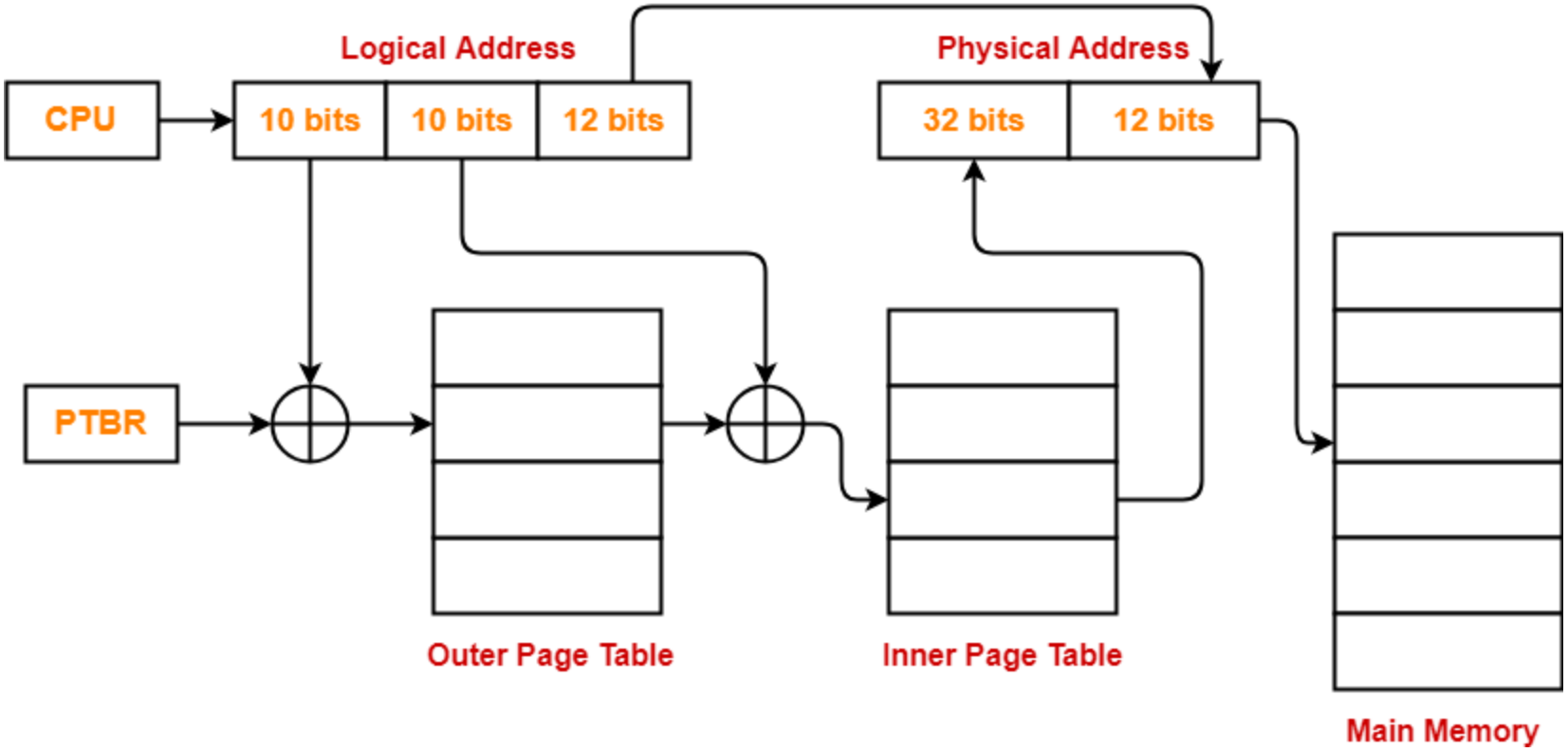
Number of Bits Required to Search an Entry in Outer Page Table-

Outer page table contains 2^{10} entries.

Thus,

Number of bits required to search a particular entry in outer page table = 10 bits

Illustration of Multilevel Paging



Segmentation

- Segmentation is a memory-management scheme that supports this programmer view of memory.
- A logical address space is a collection of segments.Each segment has a name and a length.
- The addresses specify both the segment name and the offset within the segment.
- The programmer therefore specifies each address by two quantities: a segment name and an offset.
- For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name.
- Thus, a logical address consists of a two tuple: <segment-number, offset>

Segmentation Hardware

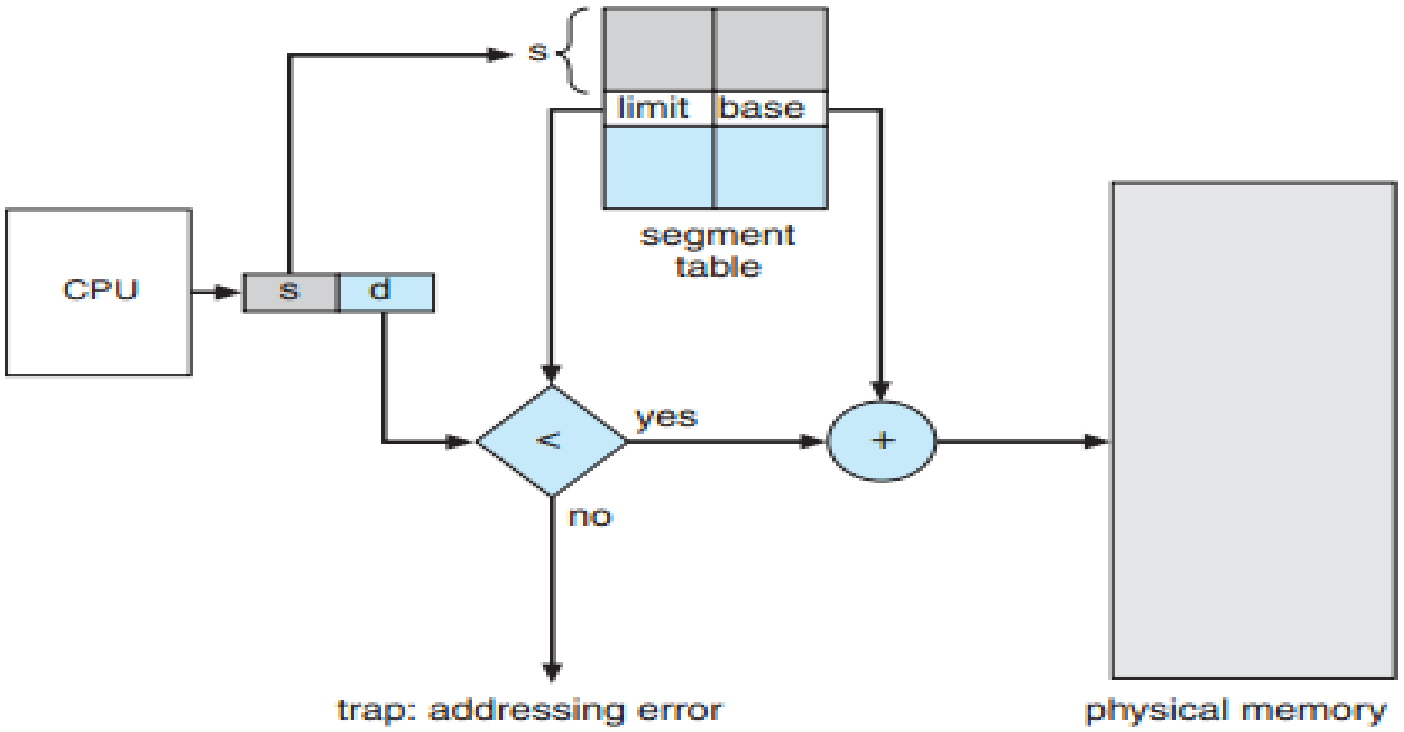


Figure 8.8 Segmentation hardware.

Segmentation(Contd.)

- Each entry in the segment table has a segment base and a segment limit.
- The segment base contains the starting physical address where the segment resides in memory, and the segment limit specifies the length of the segment.
- The use of a segment table is illustrated in Figure 8.8.
- A logical address consists of two parts: a segment number, s, and an offset into that segment, d.
- The segment number is used as an index to the segment table.
- The offset d of the logical address must be between 0 and the segment limit.
- If it is not, we trap to the operating system (logical addressing attempt beyond end of segment).

Segmentation (Contd.)

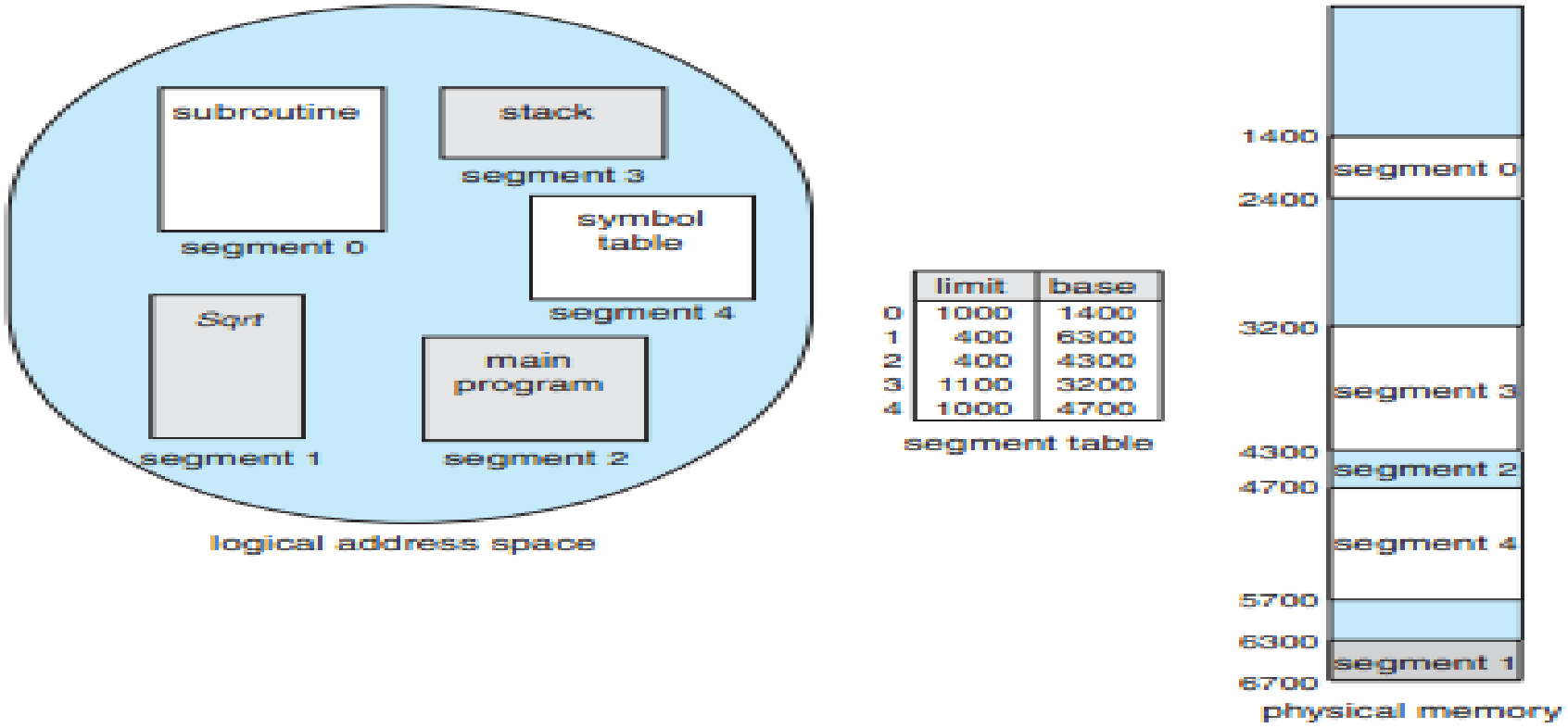


Figure 8.9 Example of segmentation.

Segmentation (Contd.)

- Consider the situation shown in Figure 8.9. We have five segments numbered from 0 through 4.The segments are stored in physical memory as shown.
- The segment table has a separate entry for each segment, giving the beginning address of the segment in physical memory (or base) and the length of that segment (or limit).
- For example, segment 2 is 400 bytes long and begins at location 4300. Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 + 53 = 4353.
- A reference to segment 3, byte 852, is mapped to 3200 (the base of segment 3) + 852 = 4052.
- A reference to byte 1222 of segment 0 would result in a trap to the operating system, as this segment is only 1,000 bytes long.

Advantages of SEGMENTATION

- In the Segmentation technique, the segment table is mainly used to keep the record of segments. Also, the segment table occupies less space as compared to the paging table.
- There is no Internal Fragmentation.
- Segmentation generally allows us to divide the program into modules that provide better visualization.
- Segments are of variable size.

Disadvantages of SEGMENTATION

- Maintaining a segment table for each process leads to overhead
- This technique is expensive.
- The time is taken in order to fetch the instruction increases since now two memory accesses are required.
- Segments are of unequal size in segmentation and thus are not suitable for swapping.
- This technique leads to external fragmentation as the free space gets broken down into smaller pieces along with the processes being loaded and removed from the main memory then this will result in a lot of memory waste.

S.NO	Paging	Segmentation
1.	In paging, program is divided into fixed or mounted size pages.	In segmentation, program is divided into variable size sections.
2.	For paging operating system is accountable.	For segmentation compiler is accountable.
3.	Page size is determined by hardware.	Here, the section size is given by the user.
4.	Paging could result in internal fragmentation.	Segmentation could result in external fragmentation.
5.	In paging, logical address is split into page number and page offset.	Here, logical address is split into section number and section offset.
6.	Paging comprises a page table which encloses the base address of every page.	While segmentation also comprises the segment table which encloses segment number and segment offset.
7.	Page table is employed to keep up the page data.	Section Table maintains the section data.
8.	In paging, operating system must maintain a free frame list.	In segmentation, operating system maintain a list of holes in main memory.
9.	Paging is invisible to the user.	Segmentation is visible to the user.
10.	In paging, processor needs page number, offset to calculate absolute address.	In segmentation, processor uses segment number, offset to calculate full address.

Exercise on SEGMENTATION

Problem- Consider the following segment table-

Segment No.	Base	Limit
0	1219	700
1	2300	14
2	90	100
3	1327	580
4	1952	96

Which of the following logical address will produce trap addressing error?

0, 430
1, 11
2, 100
3, 425
4, 95

Exercise on SEGMENTATION

Solution-

In a segmentation scheme, the generated logical address consists of two parts-

Segment Number

Segment Offset

We know-

Segment Offset must always lie in the range [0, limit-1].

If segment offset becomes greater than or equal to the limit of segment, then trap addressing error is produced.

Exercise on SEGMENTATION

Solution-

Option-A: 0, 430-

Here,
Segment Number = 0
Segment Offset = 430

We have,
In the segment table, limit of segment-0 is 700.
Thus, segment offset must always lie in the range = [0, 700-1] = [0, 699]

Now,
Since generated segment offset lies in the above range, so request generated is valid.
Therefore, no trap will be produced.
Physical Address = 1219 + 430 = 1649

Exercise on SEGMENTATION

Solution-

Option-B: 1, 11-

Here,
Segment Number = 1
Segment Offset = 11

We have,
In the segment table, limit of segment-1 is 14.
Thus, segment offset must always lie in the range = [0, 14-1] = [0, 13]

Now,
Since generated segment offset lies in the above range, so request generated is valid.
Therefore, no trap will be produced.
Physical Address = 2300 + 11 = 2311

Exercise on SEGMENTATION

Solution-

Option-C: 2, 100-

Here,
Segment Number = 2
Segment Offset = 100

We have,
In the segment table, limit of segment-2 is 100.
Thus, segment offset must always lie in the range = [0, 100-1] = [0, 99]

Now,
Since generated segment offset does not lie in the above range, so request generated is invalid.
Therefore, trap will be produced.

Exercise on SEGMENTATION

Solution-

Option-D: 3, 425-

Here,
Segment Number = 3
Segment Offset = 425

We have,
In the segment table, limit of segment-3 is 580.
Thus, segment offset must always lie in the range = [0, 580-1] = [0, 579]

Now,
Since generated segment offset lies in the above range, so request generated is valid.
Therefore, no trap will be produced.
Physical Address = 1327 + 425 = 1752

Exercise on SEGMENTATION

Solution-

Option-E: 4, 95-

Here,
Segment Number = 4
Segment Offset = 95

We have,
In the segment table, limit of segment-4 is 96.
Thus, segment offset must always lie in the range = [0, 96-1] = [0, 95]

Now,
Since generated segment offset lies in the above range, so request generated is valid.
Therefore, no trap will be produced.
Physical Address = 1952 + 95 = 2047

PAGING with SEGMENTATION

Major Limitation of Single Level Paging

- A big challenge with single level paging is that if the logical address space is large, then the page table may take up a lot of space in main memory.
- For instance, consider that logical address is 32 bit and each page is 4 KB, the number of pages will be 2^20 pages. The page table without additional bits will be of the size 20 bits * 2^{20} or 2.5 MB.
- Since each process has its own page table, a lot of memory will be consumed when single level paging is used.
- A solution to the problem is to use segmentation along with paging to reduce the size of page table.

PAGING with SEGMENTATION

- Process is first divided into segments and then each segment is divided into pages.
- These pages are then stored in the frames of main memory.
- A page table exists for each segment that keeps track of the frames storing the pages of that segment.
- Each page table occupies one frame in the main memory.
- Number of entries in the page table of a segment = Number of pages that segment is divided.
- A segment table exists that keeps track of the frames storing the page tables of segments.
- Number of entries in the segment table of a process = Number of segments that process is divided.
- The base address of the segment table is stored in the segment table base register.

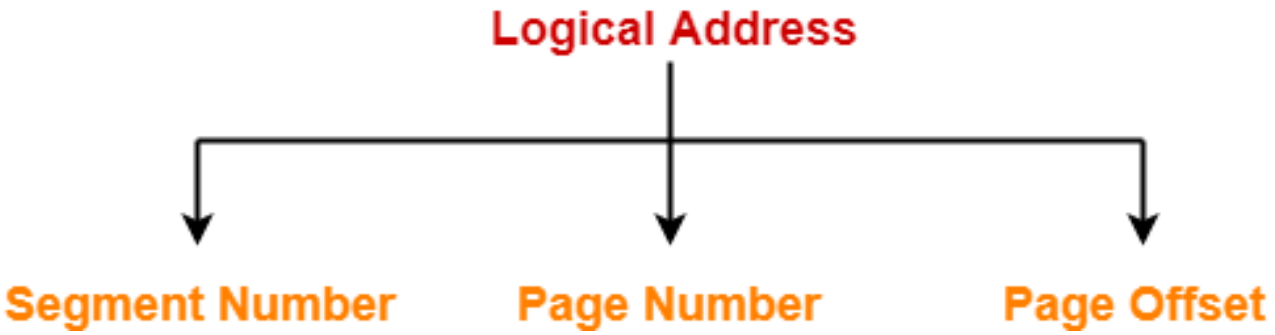
PAGING with SEGMENTATION

- Following steps are followed to translate logical address into physical address-

Step-01:

CPU generates a logical address consisting of three parts-

- Segment Number
- Page Number
- Page Offset



PAGING with SEGMENTATION

Step-02:

- For the generated segment number, corresponding entry is located in the segment table.
- Segment table provides the frame number of the frame storing the page table of the referred segment.
- The frame containing the page table is located.

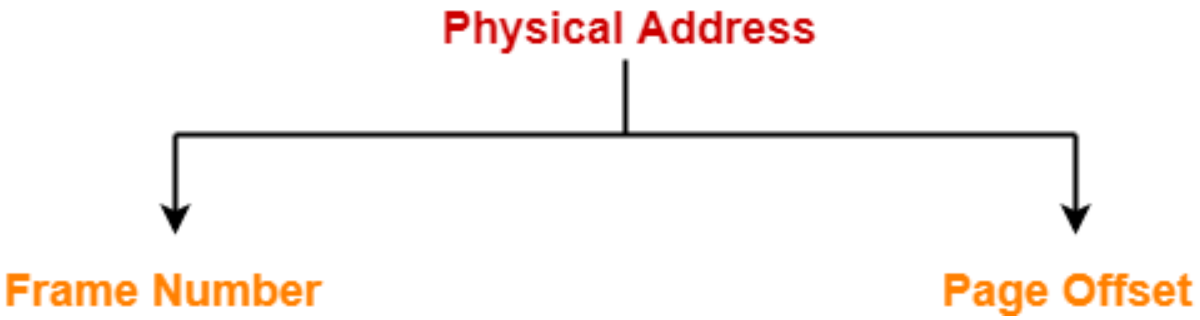
Step-03:

- For the generated page number, corresponding entry is located in the page table.
- Page table provides the frame number of the frame storing the required page of the referred segment.
- The frame containing the required page is located.

PAGING with SEGMENTATION

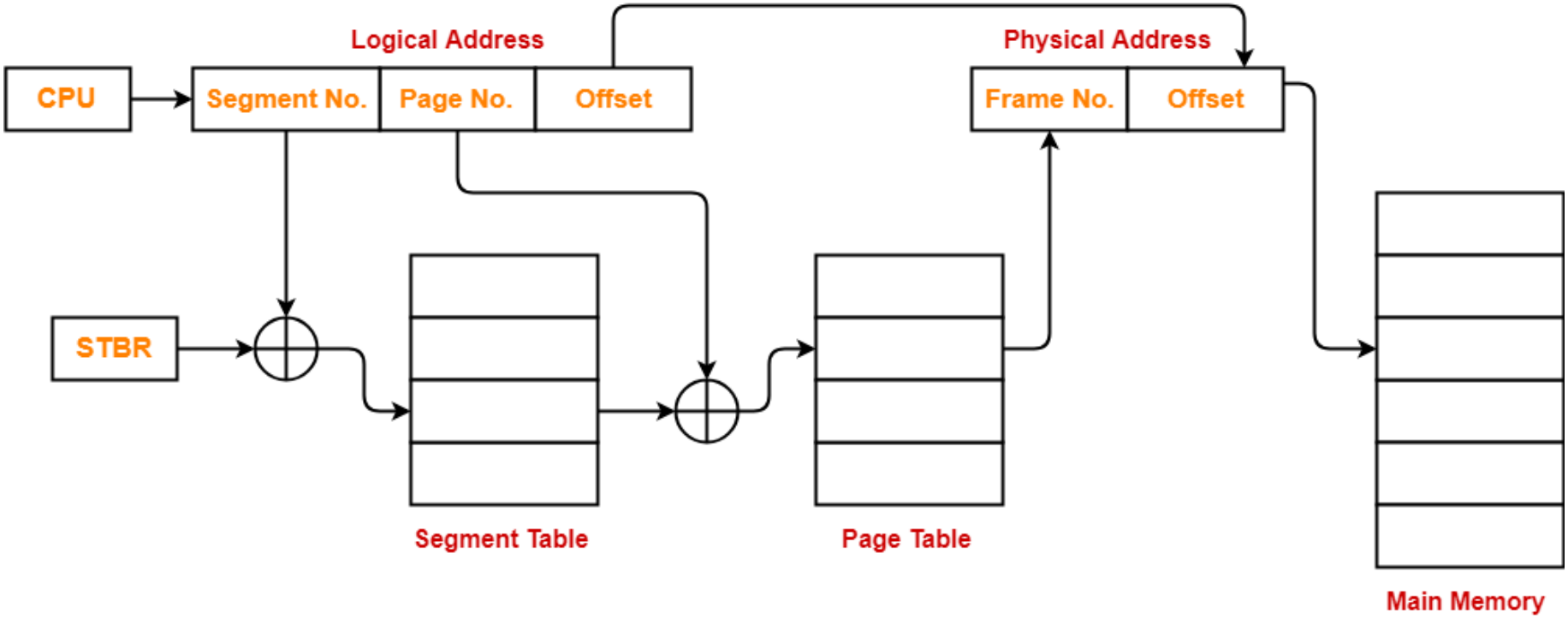
Step-04:

- The frame number combined with the page offset forms the required physical address.
- For the generated page offset, corresponding word is located in the page and read.



PAGING with SEGMENTATION

The following diagram illustrates the above steps of translating logical address into physical address-



Translating Logical Address into Physical Address

PAGING with SEGMENTATION

Advantages-

- Segment table contains only one entry corresponding to each segment.
- It reduces memory usage.
- The size of **Page Table** is limited by the segment size.
- It solves the problem of external fragmentation.

Disadvantages-

- Segmented paging suffers from internal fragmentation.
- The complexity level is much higher as compared to paging.

References

- "Operating System Concepts" by Avi Silberschatz and Peter Galvin
- "Operating Systems: Internals and Design Principles" by William Stallings
- www.tutorialpoint.com
- www.javapoint.com

Thank You

For any Query, you can contact

Er. Inderjeet Singh(e8822)

Ph. No: 86-99-100-160

Email id: inderjeet.e8822@cumail.in