Chapter: Memory Management

Paging / Paged Memory Management

– Paging is a memory management scheme that allows processe's physical memory to be discontinuous, and eliminates problems with fragmentation by allocating memory in equal sized blocks known as ***pages***.

Paging / Paged Memory Management

- Every process is divided in to number of pages
- Memory is divided into partitions whose size is same as page size : **frames**
- Each frame has frame no.
- Page size is same as frame size
- Put any page in any free frame
- Paging allows non-contiguous memory allocation
- Page numbers, frame numbers and frame sizes are determined by the machine architecture
- **Paging leads to Internal Fragmentation**

Pages size and Frame size is always in powers of two

Paging / Paged Memory Management

- CPU generates logical address and put the pages into random frames
- **Mapping** is required to map which page is stored in which page number
- From frame no. physical address could be found

Paging / Paged Memory Management

- Physical address space of a process in non- contiguous
- **Implementation:**
  - **Frames: Fixed sized blocks of the physical memory**
  - **Pages: Fixed sized slots of the logical memory**
- When a process is to be executed, its pages are loaded into any available memory frames.
- **Page Table: is a data structure.** Used to translate logical address to physical address
- CPU generated logical addresses to fetch the instructions.

Address Translation

– Address generated by CPU is divided into:

a) Page Number (p): used as an **index into page table,** which contains base address of each page in the physical memory.

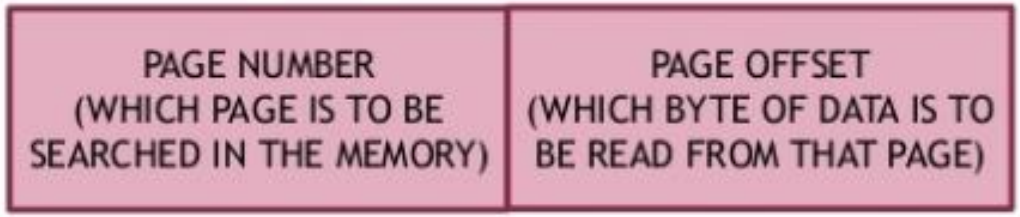
b) Page Offset (d): combined with base address to **define physical memory address** that is sent to the memory unit.

Actual address of any byte in page or frame (Position of instruction in page or frame)

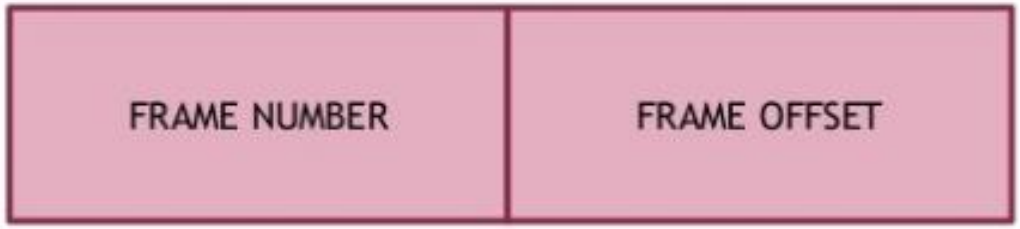
- Address of physical memory, where page resides.
- The number of bits in the offset determines the maximum size of each page, and should correspond to the system frame size.

Address Translation

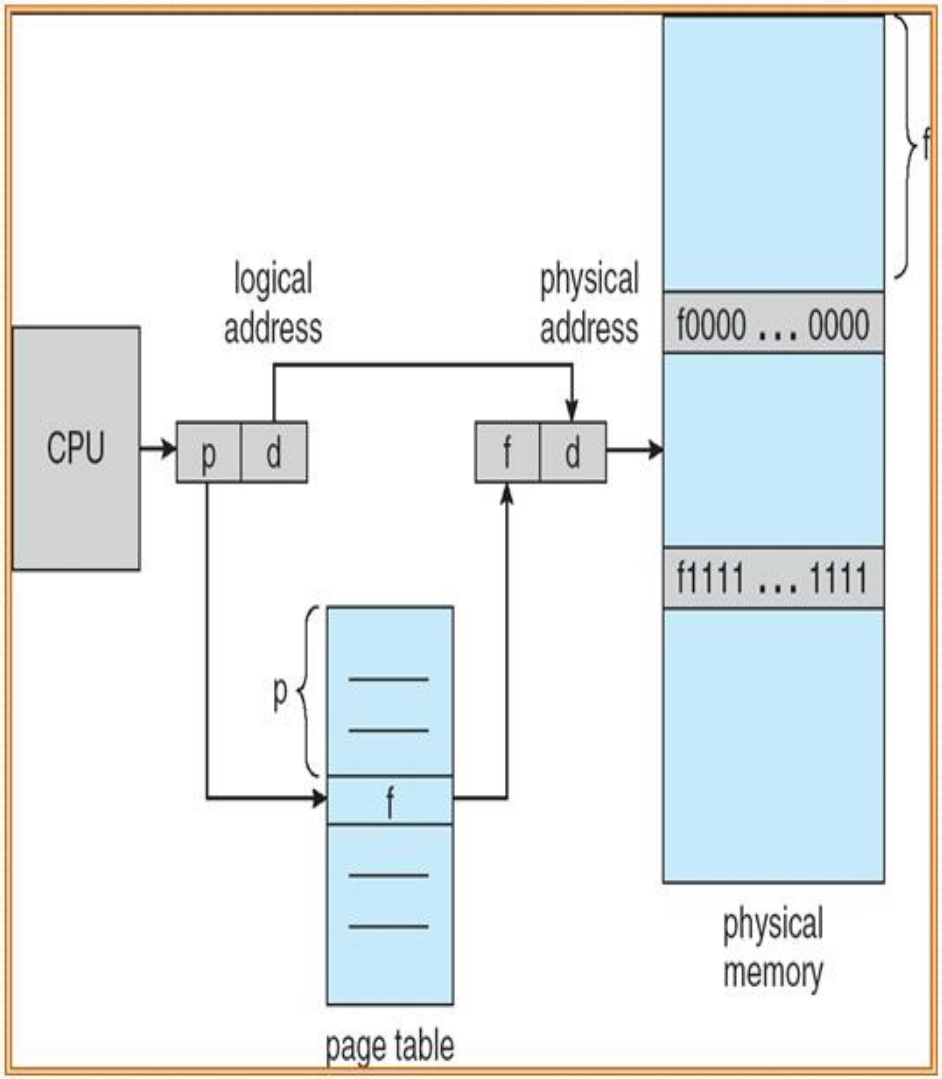
LOGICAL ADDRESS

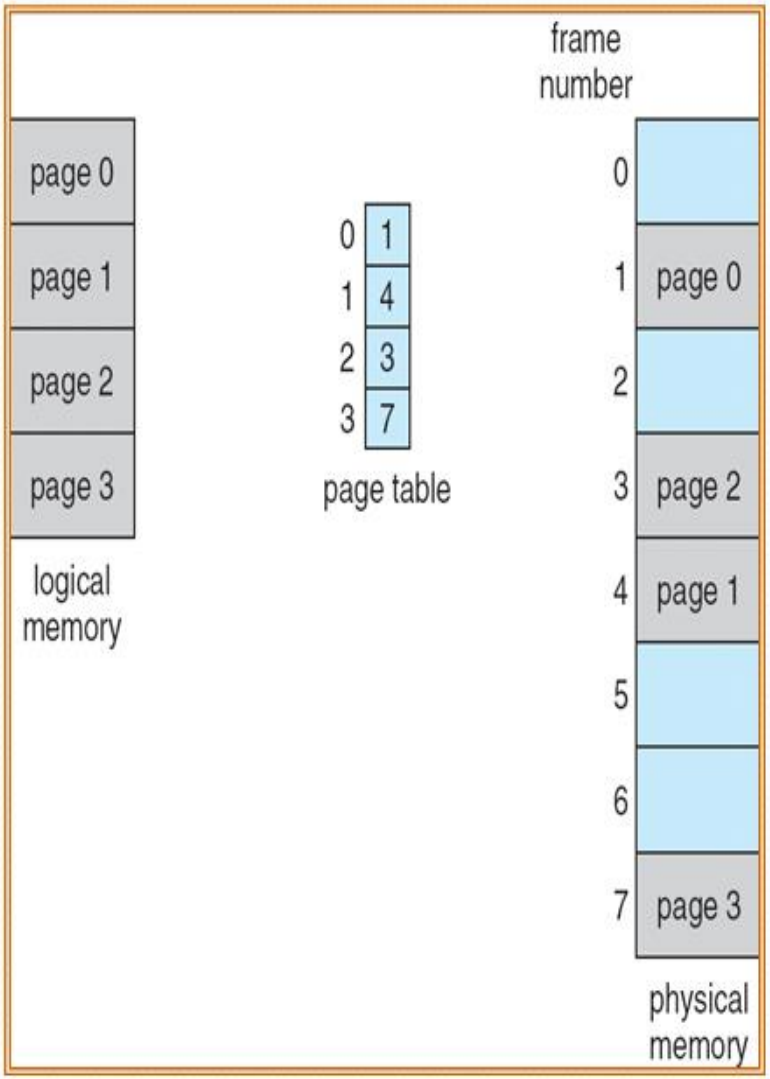


PHYSICAL ADDRESS



Address Translation





Translation of Address to memory

How many address combinations can be generated from bits:

1 bit → 2^1=2 i.e 0 or 1

2 bit → 2^2=4 i.e 00,01,10,11 ……..so on

n bits → 2^n combination of address locations can be generated.

If we have n bit address, and system is byte addressable (every location/partition is of 1 Byte) it will support memory of:

2^n x 1Byte

Translation of Address to memory

a) 2^{10} means 1K i.e 1Kilo (1024)

b) 2^{20} means 1M i.e 1Mega

c) 2^{30} means 1G i.e 1Giga

d) 2^{40} means 1T i.e 1Tera

e) 2^{50} means 1P i.e 1Peta

Translation of Address to memory

Translate n bit address to memory

Q1. Assume memory is byte addressable and address is of size 14bit. Compute size of memory.

Ans

Using 14 bits how many address combinations we can generate? 2^14.

2^4 x 2^10 =16 K *1 Byte

= 16KB

No. of bits required to address memory

If memory size is given then compute how many bits are required to address that memory.

Memory Size= Total no. of locations x size of each location

Total no. of locations= Memory Size/ size of each location

No. of bits required to address memory

Q1. Memory size is of 64KB. System is Byte addressable. How many bits are required to represent 64KB?

Ans: 64KB/1B
= 64K

$$2^6 \times 2^{10} = 2^{16}$$

Total 16 bits are required to rep. memory of 64 KB

Address Translation

How to break logical address into page number and page offset?

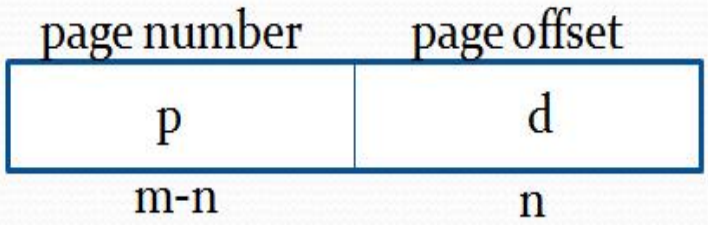
If address space size = 2^m

Page size = 2^n

Then

Page number = higher order $(m-n)$ bits

Page offset = n low-order bits



Physical Address= frame x page size + offset

Example

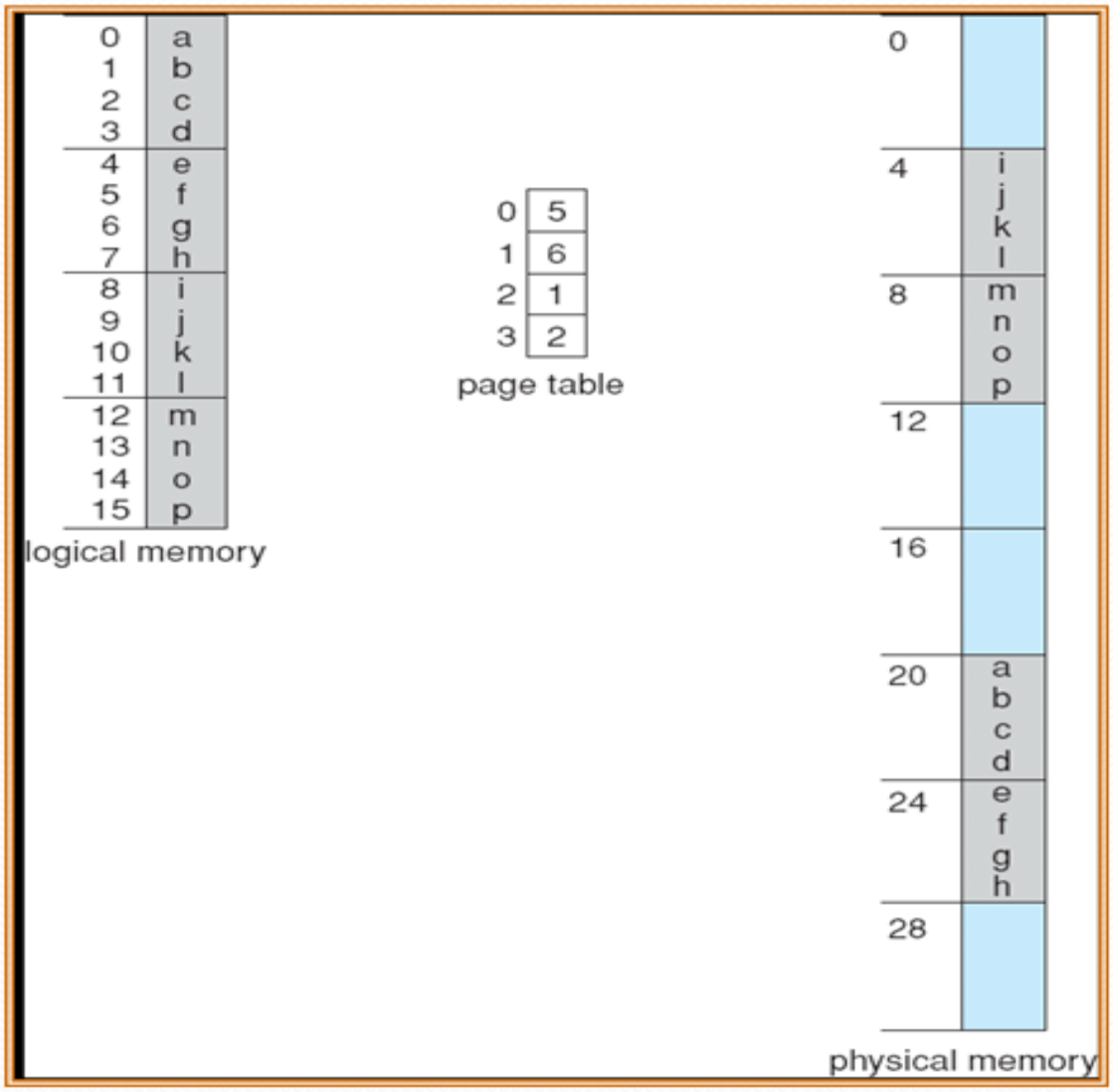
Q. Using a page size of 4 bytes and physical memory of 32 bytes, find the physical address if logical address is:

a) 4

b) 10

Solution:

Example- Page Table



Example

Q. Using a page size of 4 bytes and physical memory of 32 bytes, find the physical address if logical address is:

a) 4

b) 10

Solution:

a) Page size = 4bytes = 2^2 (i.e. n=2)
Address space = 32 = 2^5 (i.e. m=5)
Logical address = 4
In binary = 00100
Page number=(higher)(m-n)bits=5-2=3 bits
i.e. 001 = 1
Offset = (lower)n bits = 2 bits i.e. 00 = 0
From page table, if page number = 1
Then frame = 6
Physical address = frame*page size+offset
= 6*4+0 = 24

Example

1. if logical address space can hold 2^13 addresses:

Means

a) logical address is 13 bit long.

b) logical address space is 2^13= 8KB if system is Byte addressable

Example

Q1. LA=24bit

PA=16bit

Page Size =1KB. **System is byte addressable.**

Find total number of pages and frames.

Sol:

We need to compute 4 things in sequence:

a) **Total addressable locations from L.A and P.A**

b) **Total locations and offset from page size and total bits to represent page size.**

c) **Total pages.**

d) **Total frames.**

Example

Sol: Total addressable locations from 24 bits

$2^{24} = 2^4 \times 2^{20} = 16M$

Every location is byte addressable

So Total memory:

16Mx1B= 16MB

Total addressable locations from 16 bits

$2^{24} = 2^6 \times 2^{10} = 64K$

Every location is byte addressable

So Total memory:

64Kx1B= 64KB

c) Page size= 1KB

total addressable locations and offset = Page size/ size of each location

= 1KB/1B

=1K locations

Example

c) Page size= 1KB

total addressable locations and offset = Page size/ size of each location

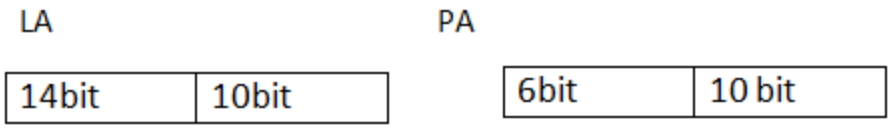
= 1KB/1B

=1K locations

Total bits required to represent 1K locations:

2^{10} i.e. 10 bits

So offset is 10bit.



d) Total pages with 14bit

$2^{14} = 2^4 \times 2^{10} = 16K$ pages

e) Total frames with 6bit

$2^6 = 64$ frame

Example

Q2. LA=33bit

PA=24bit

Page Size =2KB. **System is byte addressable.**

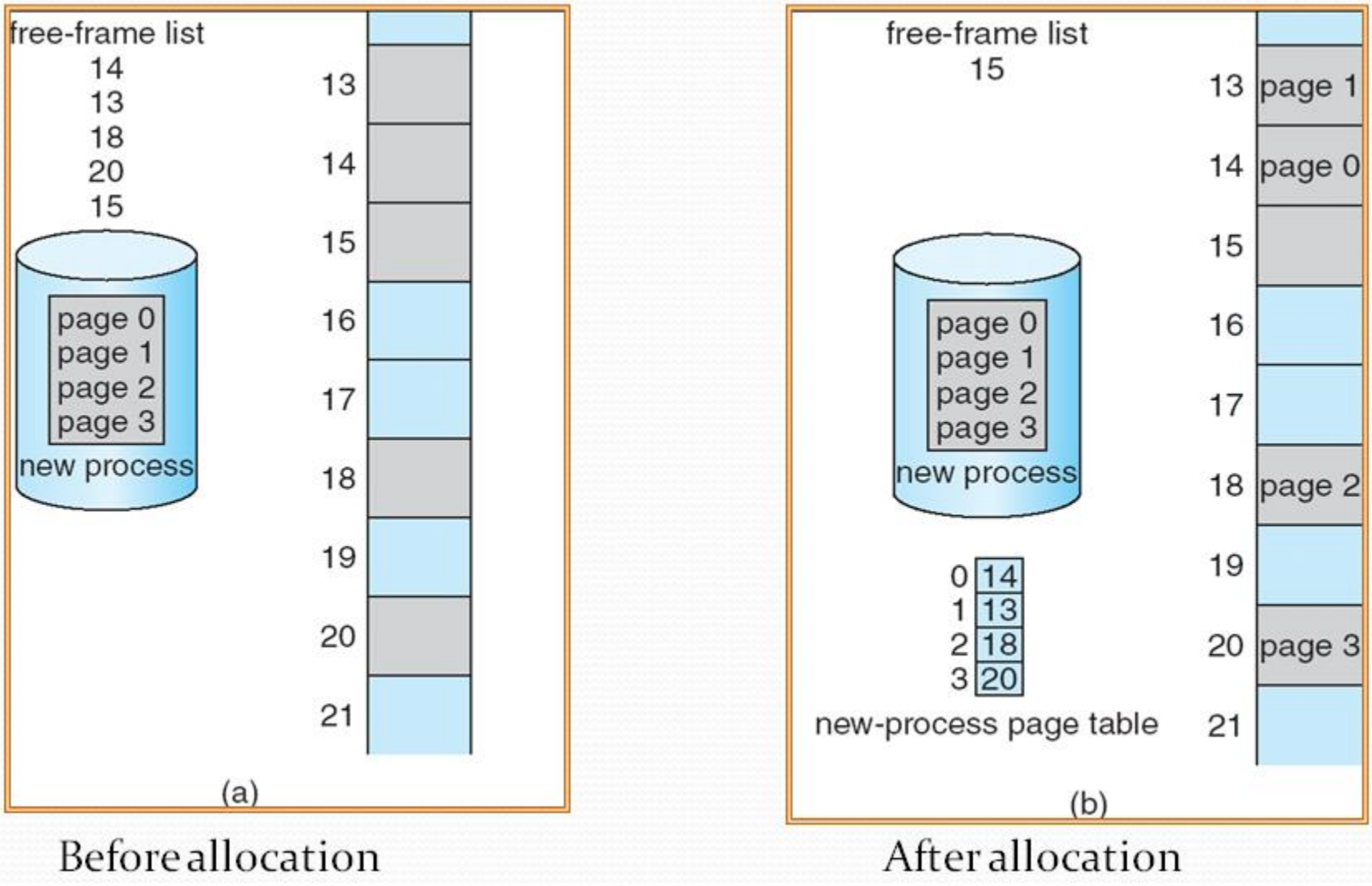
Find total number of pages and frames.

Sol:

a) **Total pages = 4M**

b) **Total frames= 8K**

Frame Allocation



Hardware Support: for Page Table Implementation

Hardware implementation of page table: Methods:

1. Use Registers

- One option is to **use a set of registers for the page table**
- The use of registers for the page table is satisfactory if the page table is reasonably small (for example, 256 entries).

Hardware Support: for Page Table Implementation

2. An alternate option is to:

– Store the page table in main memory, and to use a single register (called the ***page-table base register, PTBR***) to record where in memory the page table is located.

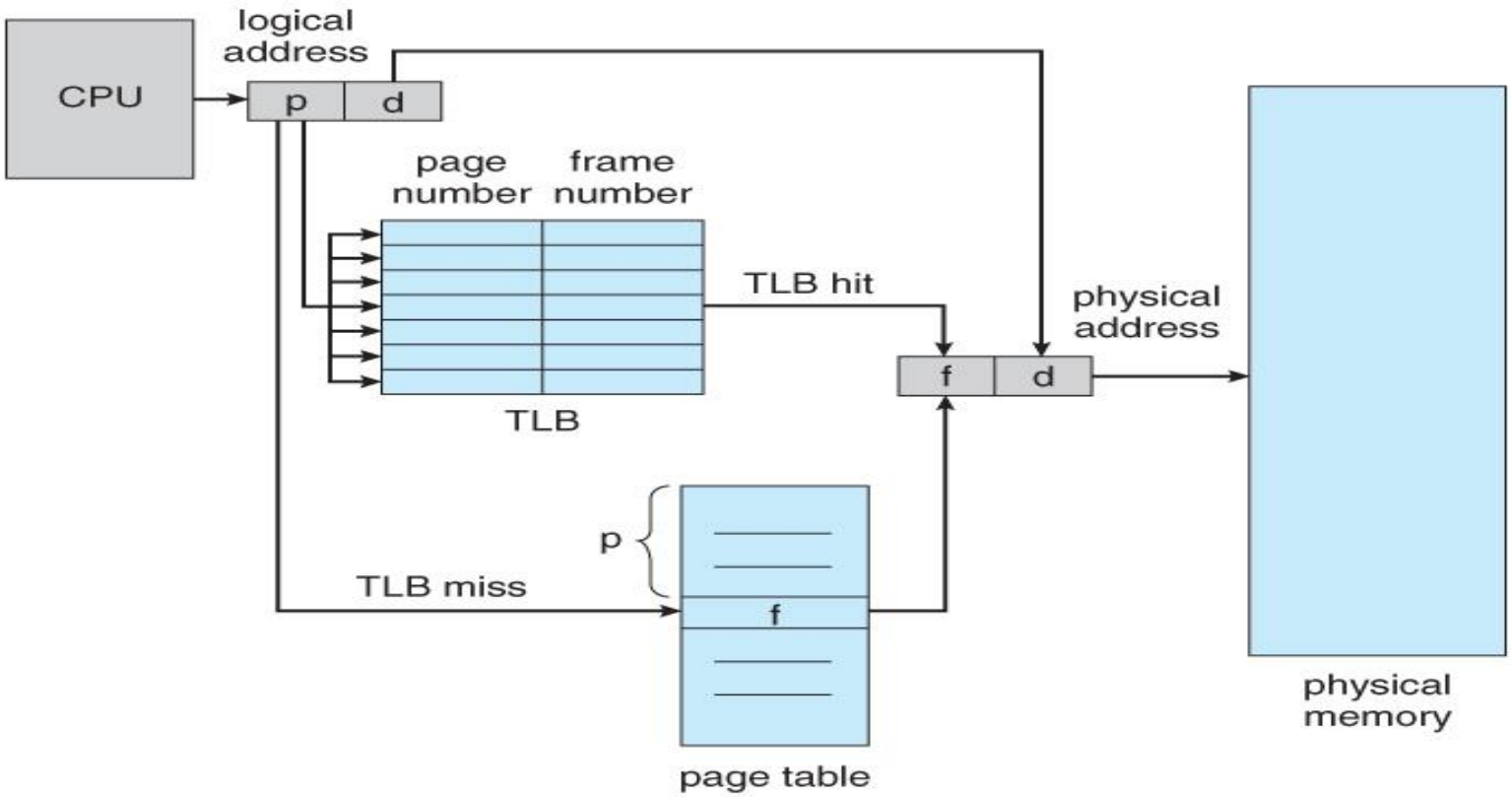
Hardware Support: for Page Table Implementation

- Use a very special high-speed memory device called the ***translation look-aside buffer, TLB.***
- Each entry in the TLB consists of two parts:
- a key (or tag) and a value.
- If the item is found, the corresponding value field is returned.
- The benefit of the TLB is that it can search an entire table for a key value in parallel, and if it is found anywhere in the table, then the corresponding lookup value is returned.

Hardware Support: for Page Table Implementation

- TLB is not large enough to hold the entire page table.
- So used as a cache device.
- The percentage of time that the desired information is found in the TLB is termed the ***hit ratio***.
- The percentage of time that the desired information is not found in the TLB is termed the ***miss ratio***.

Paging hardware with TLB



Example

Q. If it takes 20 nanoseconds to search the TLB and 100 nanoseconds to access memory, then find the effective access time if the hit-ratio is 80%.

Solution:

Time taken to access data if page is found in TLB =

20+100=120

Time taken to access data if page is not found in TLB =

20+100+100 = 220

Effective access time = .80*120 + .20*220

= 140 nanoseconds

Effective access time = hit ratio x time taken for TLB hit + miss ratio x time taken for TLB miss

Example

Q. If it takes 20 nanoseconds to search the TLB and 100 nanoseconds to access memory, then find the effective access time if the hit-ratio is 80%.

Explanation:

- TLB hit takes 120 nanoseconds total (20 to find the frame number and then another 100 to go get the data)
- TLB miss takes 220 (20 to search the TLB, 100 to go get the frame number, and then another 100 to go get the data.

- ***Effective access time = hit ratio x time taken for TLB hit + miss ratio x time taken for TLB miss***

Problem

Q. If it takes 20 nanoseconds to search the TLB and 100 nanoseconds to access memory, then find the effective access time if the miss-ratio is 2%.

Solution:

Effective access time = .98*120 + .02*220

= 122 nanoseconds

Segmentation

- It is a memory management scheme in which the memory allocated to the process is non contiguous
- Logical address space is divided into number of small blocks called segments
- Segments are of variable sized.

A C compiler might create separate segments for the following:

1. The code
2. Global variables
3. The heap, from which memory is allocated
4. The stacks used by each thread
5. The standard C library

Segmentation

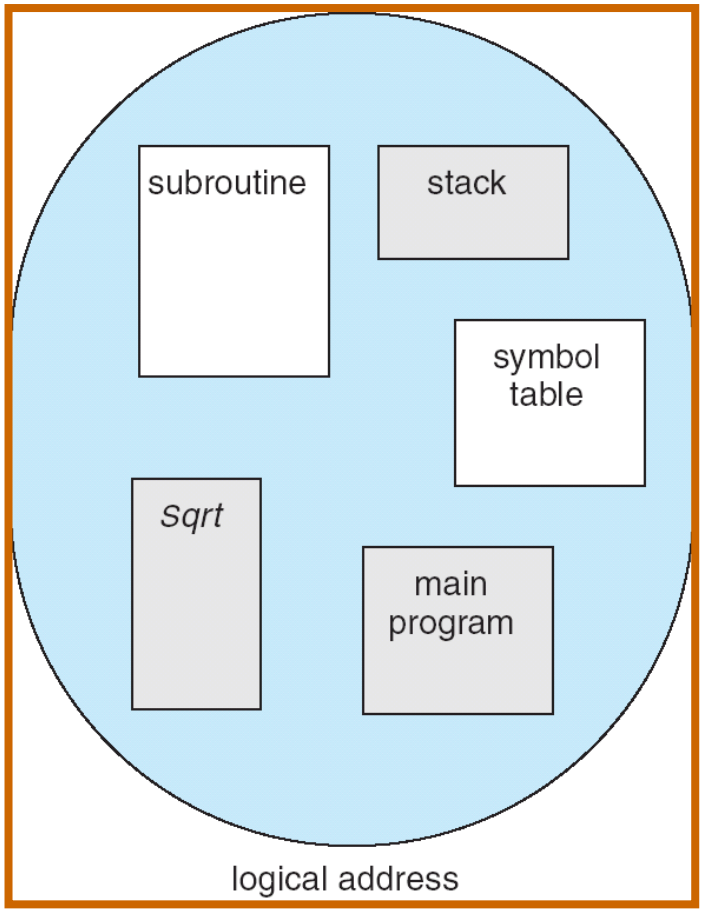
- Users view memory as a collection of variable size segments. With no necessary ordering of these segments

- Segmentation is memory-management scheme that supports user view of memory

- A program is a collection of segments. A segment is a logical unit such as:

  main program, function, object, local variables, global variables, data structures : stack, symbol table, arrays

User's View of a Program



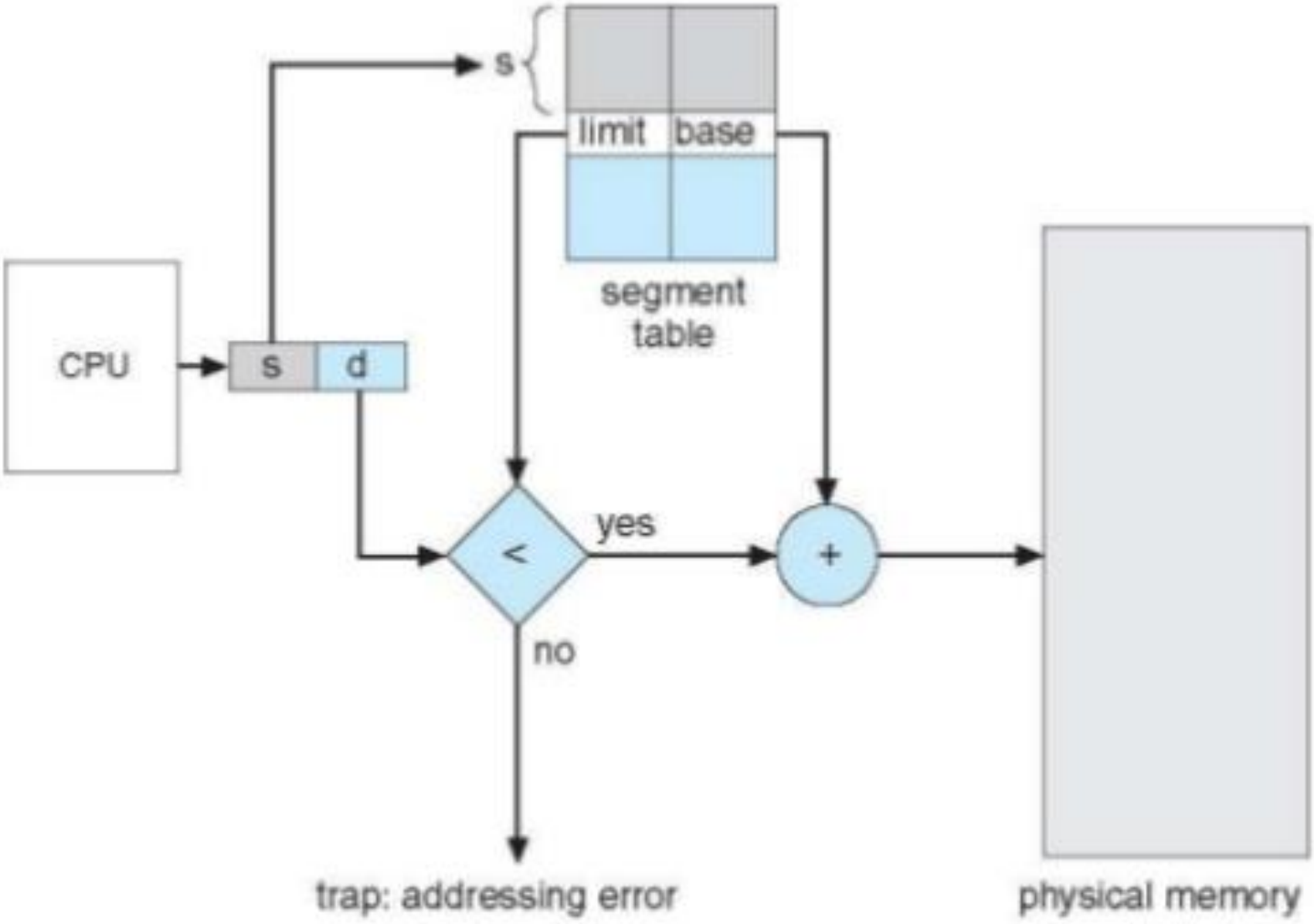
Segmentation Architecture

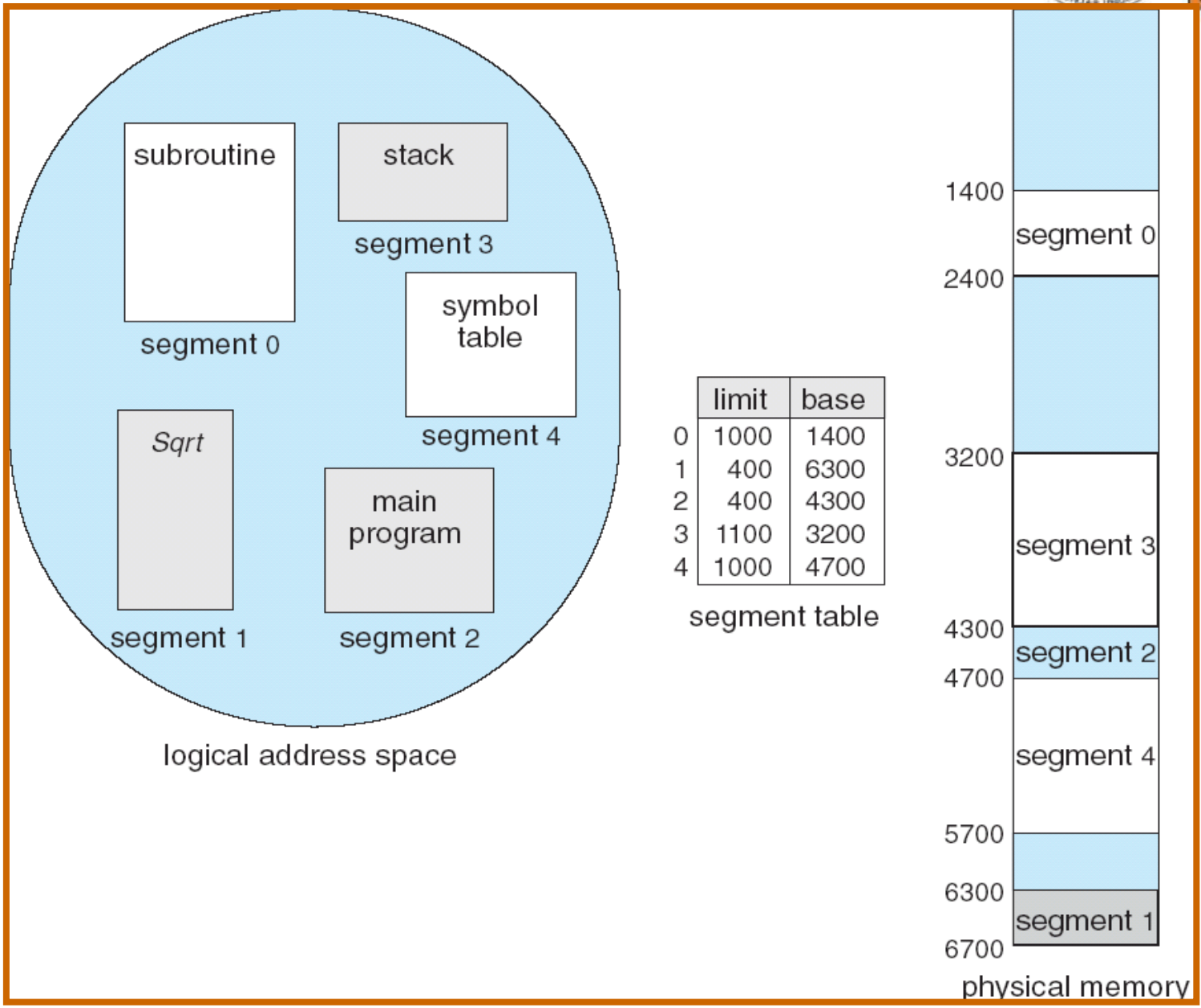
- Each segment has a name and its length.
- Logical address consists of a two tuple:

  <segment-number, offset>,
- **Segment table** – maps physical addresses; each table entry has:
  - base – contains the starting physical address where the segments reside in memory
  - *limit* – specifies the length of the segment
- ***Segment-table base register (STBR)*** points to the segment table's location in memory
- ***Segment-table length register (STLR)*** indicates number of segments used by a program;

segment number *s* is legal if *s* < STLR

Address Translation: Segmentation Hardware




subroutine
stack
segment 3
segment 0
symbol table
segment 4
Sqrt
main program
segment 1
segment 2
logical address space
limit
base
0 1000 1400
1 400 6300
2 400 4300
3 1100 3200
4 1000 4700
segment table
1400
segment 0
2400
3200
segment 3
4300
segment 2
4700
segment 4
5700
6300
segment 1
6700
physical memory

Problems

Q1.if Segment 2 is 400 bytes long and begins at location 4300, then a reference to byte 53 of segment 2 is mapped onto location?

Ans: base address: 4300

Limit: 400

53<400 (I.e limit)

4300+53=4353.

Problems

Q2. A reference to byte 852 of segment 3 is mapped to?

Ans: 3200+852=4052.

Q3. A reference to byte 1222 of segment 0 is mapped to?

Ans: would result in a trap to OS, as this segment is only 1000 bytes long.