

قسم هندسة تقنيات الحاسوب

Department of Computer Engineering Techniques (Stage: 4)

Advance Computer Technologies

Dr.: Mayas Aljibawi

Chapter 8: Main Memory

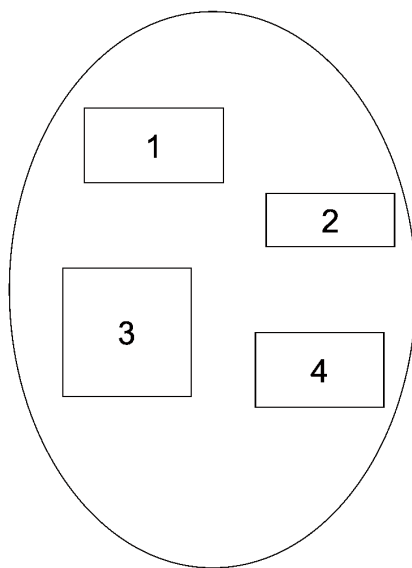


Operating System Concepts – 9th Edition

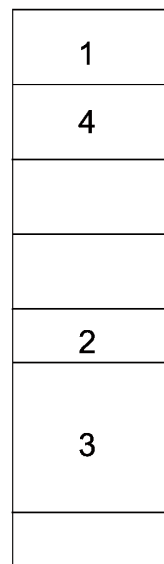
Silberschatz, Galvin and Gagne ©2013



Logical View of Segmentation



user space



physical memory space





Segmentation Architecture

- Logical address consists of a two tuple:
 <segment-number, offset>,
- **Segment table** – maps two-dimensional 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;





Segmentation Architecture (Cont.)

- Protection

With each entry in segment table associate:

validation bit = 0 \Rightarrow illegal segment

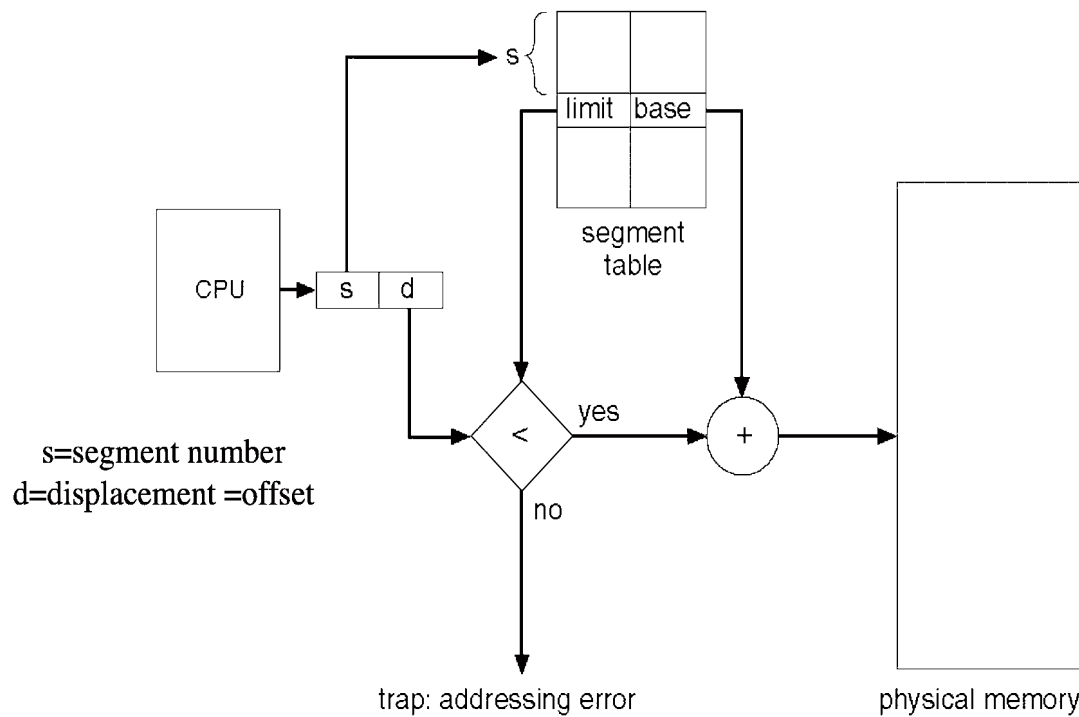
read/write/execute privileges

- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
- A segmentation example is shown in the following diagram





Segmentation Hardware





Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
 - Avoids external fragmentation
 - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames**
 - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Backing store likewise split into pages
 - hard-disk





Address Translation Scheme

- Address generated by CPU is divided into:

Page number (p) – used as an index into a **page table** which contains base address of each page in physical memory

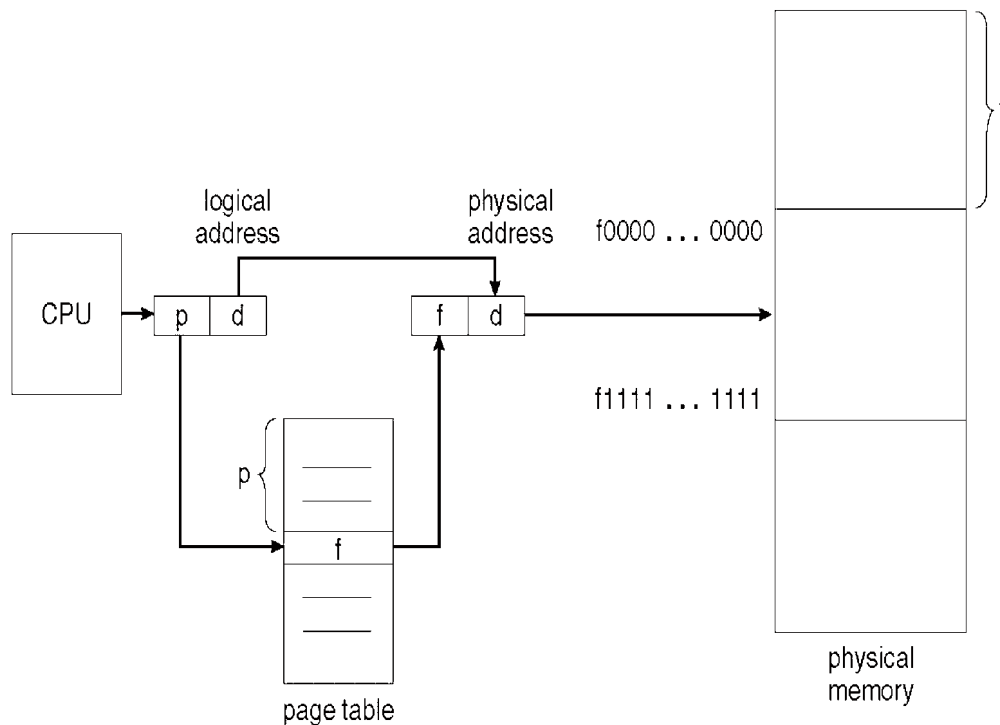
Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit

page number	page offset
p	d
$m - n$	n

For given logical address space 2^m and page size 2^n



Paging Hardware



P is used to get frame no.

F combined with displacement (offset) (d) to reach physical memory

Main difference between segmentation is there is no protection



Paging Model of Logical and Physical Memory

page 0
page 1
page 2
page 3

logical
memory

0	1
1	4
2	3
3	7

page table

frame
number

0	
1	page 0
2	
3	page 2
4	page 1
5	
6	
7	page 3

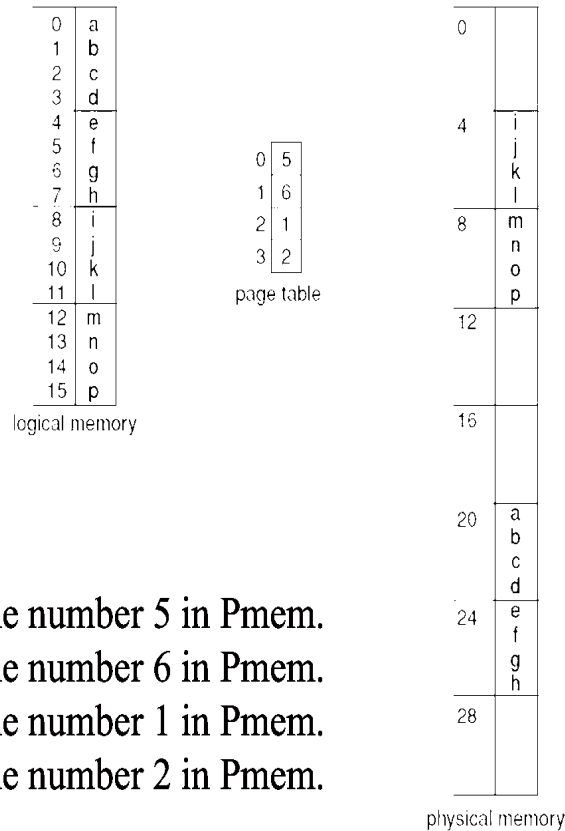
physical
memory

Translation from 0 to frame number 1
Translation from 1 to frame number 4
Translation from 2 to frame number 3
Translation from 3 to frame number 7





Paging Example



Translation from 0 to frame number 5 in Pmem.

Translation from 1 to frame number 6 in Pmem.

Translation from 2 to frame number 1 in Pmem.

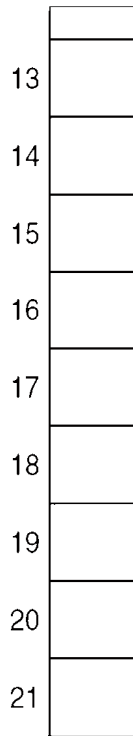
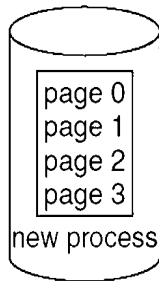
Translation from 3 to frame number 2 in Pmem.

$n=2$ and $m=4$ 32-byte memory and 4-byte pages



Free Frames

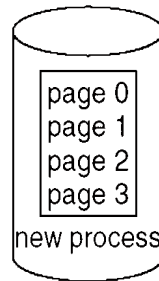
free-frame list
14
13
18
20
15



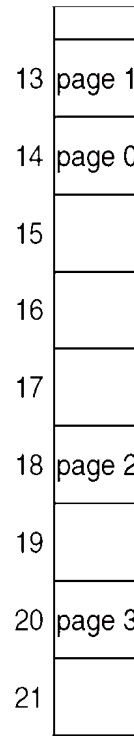
(a)

Before allocation

free-frame list
15



new-process page table
0 14
1 13
2 18
3 20



(b)

After allocation



Implementation of Page Table

- Page table is kept in main memory
- **Page-table base register (PTBR)** points to the page table
- **Page-table length register (PTLR)** indicates size of the page table
- In this scheme every data/instruction access requires memory accesses
- and
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**





Implementation of Page Table (Cont.)

- Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry – uniquely identifies each process to provide address-space protection for that process

Otherwise need to at every context switch

- TLBs typically small (64 to 1,024 entries)
- On a TLB miss, value is loaded into the TLB for faster access next time

Replacement policies must be considered

Some entries can be **wired down** for permanent fast access





Associative Memory

- Associative memory – parallel search

Page #	Frame #

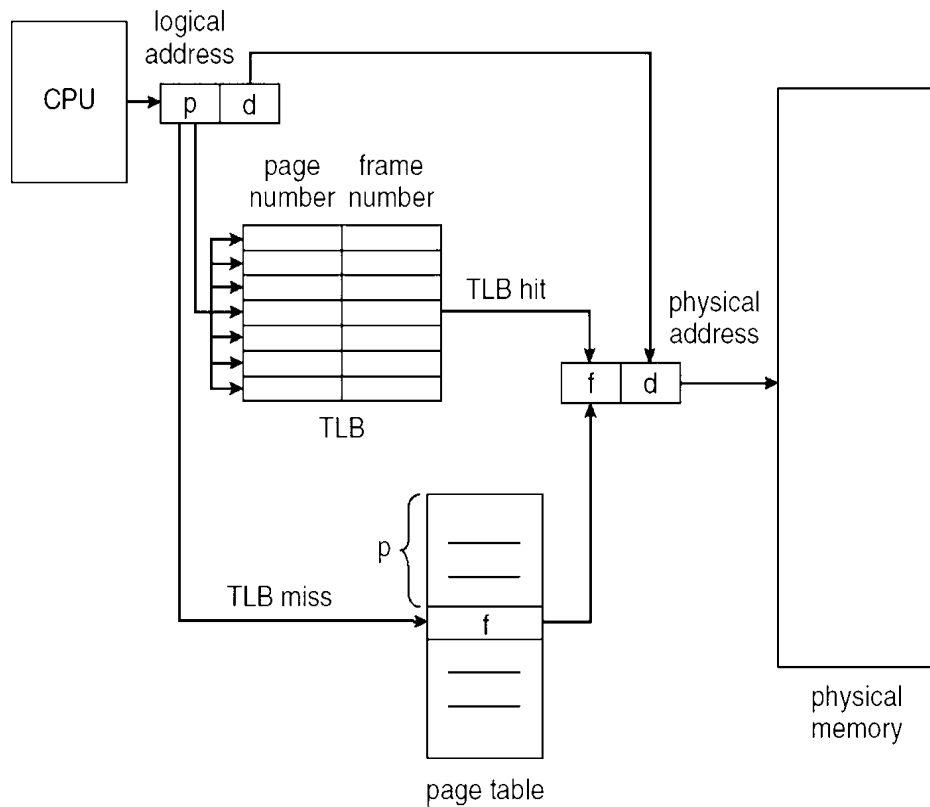
- Address translation (p, d)

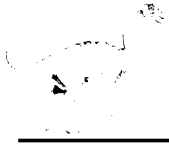
If p is in associative register, get frame # out

Otherwise get frame # from page table in memory



Paging Hardware With TLB





Effective Access Time

- Associative Lookup = ε time unit

Can be < 10% of memory access time

- Hit ratio = α

Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers

- Consider $\alpha = 80\%$, $\varepsilon = 20\text{ns}$ for TLB search, 100ns for memory access

- **Effective Access Time (EAT)**

$$\begin{aligned} \text{EAT} &= (1 + \varepsilon) \alpha + (2 + \varepsilon)(1 - \alpha) \\ &= 2 + \varepsilon - \alpha \end{aligned}$$

- Consider $\alpha = 80\%$, $\varepsilon = 20\text{ns}$ for TLB search, 100ns for memory access

$$\text{EAT} = 0.80 \times 100 + 0.20 \times 200 = 120\text{ns}$$

- Consider more realistic hit ratio -> $\alpha = 99\%$, $\varepsilon = 20\text{ns}$ for TLB search, 100ns for memory access

$$\text{EAT} = 0.99 \times 100 + 0.01 \times 200 = 101\text{ns}$$





Memory Protection

- Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed

Can also add more bits to indicate page execute-only, and so on

- **Valid-invalid** bit attached to each entry in the page table:

“valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page

“invalid” indicates that the page is not in the process’ logical address space

Or use **page-table length register (PTLR)**

- Any violations result in a trap to the kernel

