19 Virtual Memory

Creation, Propagation, and Destruction of Addresses

```
; ex1901.a Creation, propagation, and destruction of addresses
          ld r0, a    ; address now in r0
          st r0, b    ; address now at b
          ld r1, c
          st r1, a    ; address at a overlaid with a constant
          halt
a:        .word d     ; assembled to 16-bit address of d
b:        .word 3
c:        .word 5
d:        .word 17    ; the address of this word is 0008
```

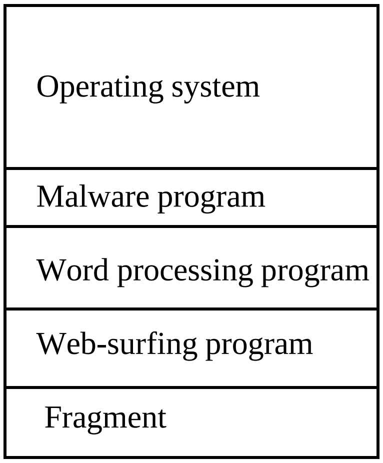
Observation: On the LCC, once a program starts executing, it cannot stop, move to a new location, andresume executing.

Why?

Because address adjustment required by the new load point cannot be performed because where the addresses are is unknown. The A entries indicate where the addresses are ONLY before execution starts.

Memory Fragmentation Problem

Suppose word processing program terminates:



Operating system

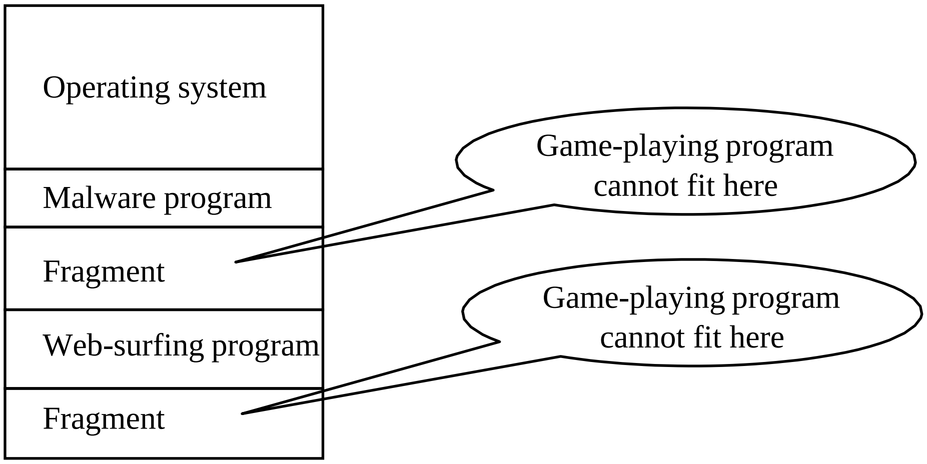
Malware program

Word processing program

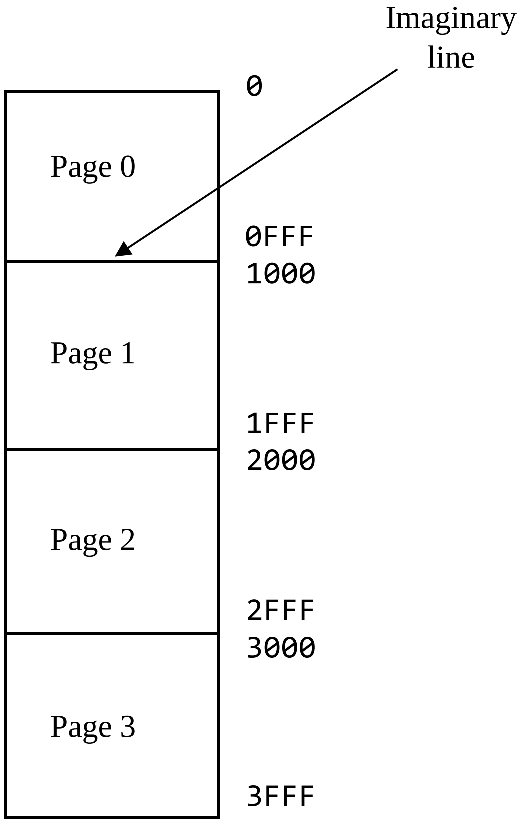
Web-surfing program

Fragment

Game playing program cannot fit into either fragment although the total amount of available memory is sufficient.

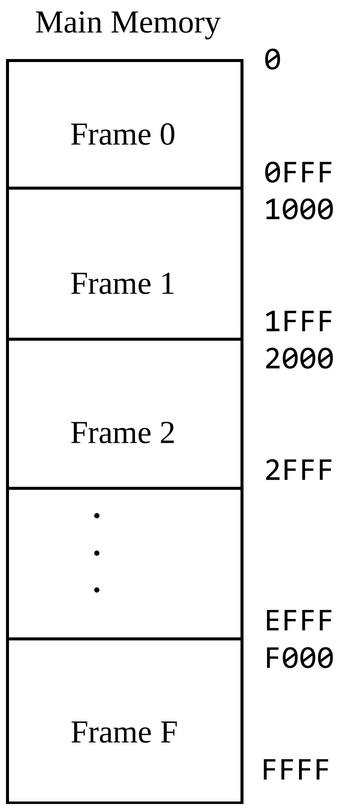

Operating system
Malware program
Fragment
Web-surfing program
Fragment
Game-playing program cannot fit here
Game-playing program cannot fit here

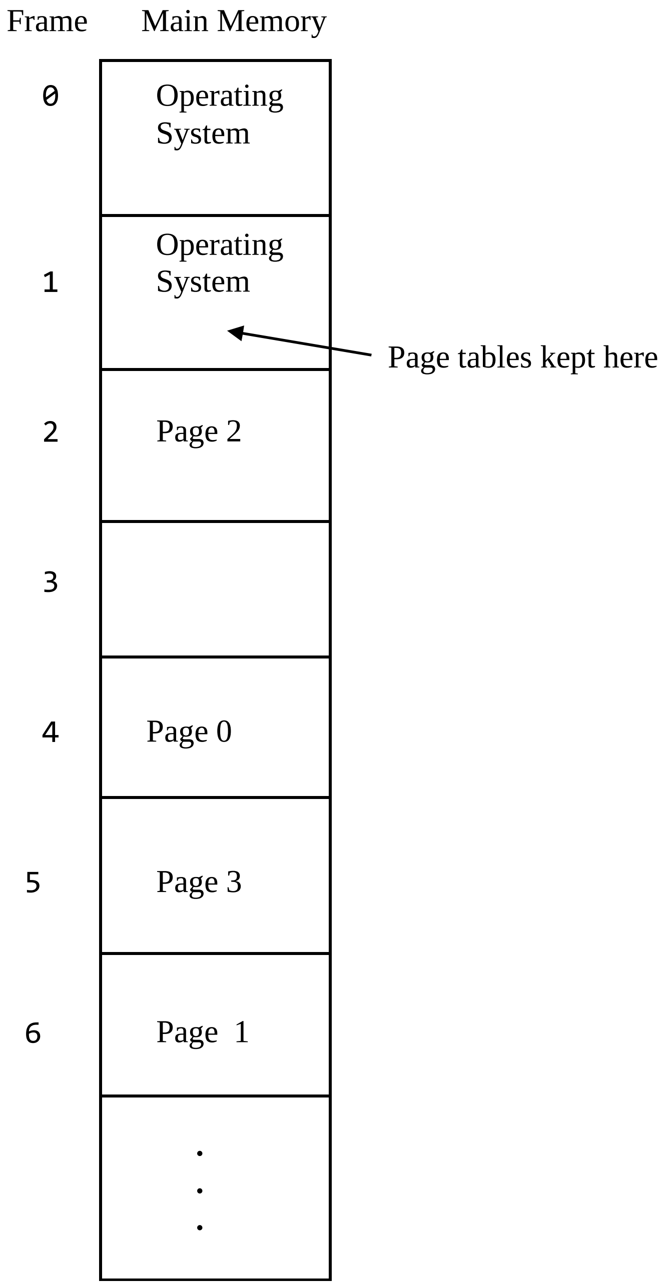
Simple Paging



Page	Address Range (hex)
0	0000 to 0FFF
1	1000 to 1FFF
2	2000 to 2FFF
3	3000 to 3FFF

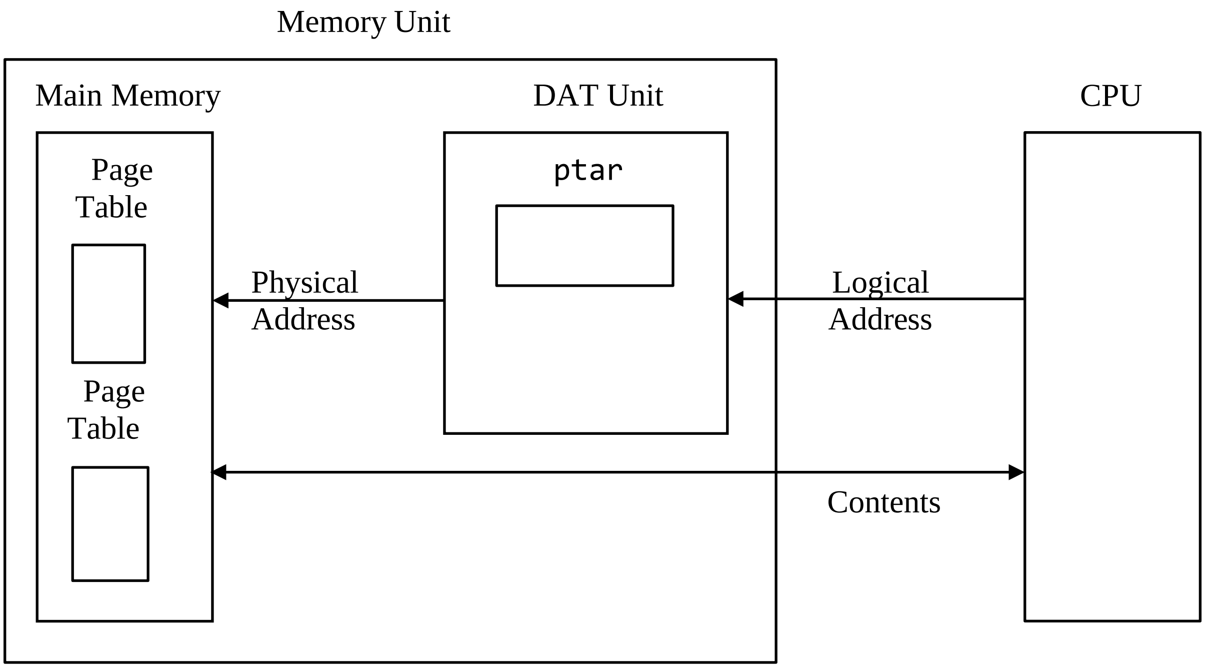
$$\overbrace{\underbrace{\mathtt{2}}_{\text{Page Number}}\qquad\underbrace{\mathtt{1}\qquad\mathtt{3}\qquad\mathtt{C}}_{\text{Displacement Within a 4K Page}}}^{\text{16-bit address}}$$


Main Memory
0
Frame 0
0FFF
1000
Frame 1
1FFF
2000
Frame 2
2FFF
.
.
.
EFFF
F000
Frame F
FFFF

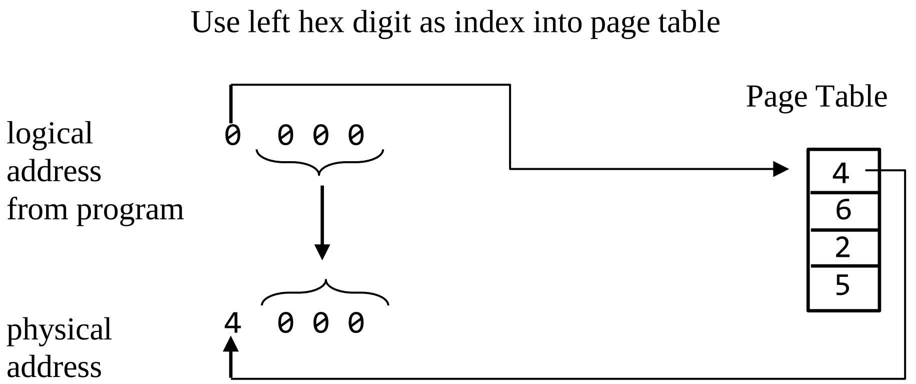

Frame
Main Memory
0
Operating System
1
Operating System
Page tables kept here
2
Page 2
3
4
Page 0
5
Page 3
6
Page 1
.
.
.

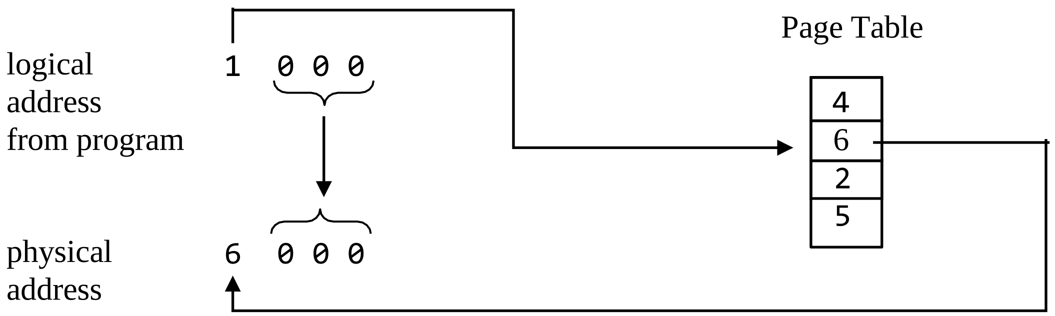
Page Table

Page Number	Frame Number
0	4
1	6
2	2
3	5


Memory Unit
Main Memory
Page Table
Page Table
DAT Unit
ptar
Physical Address
Logical Address
Contents
CPU

Dynamic Address Translation

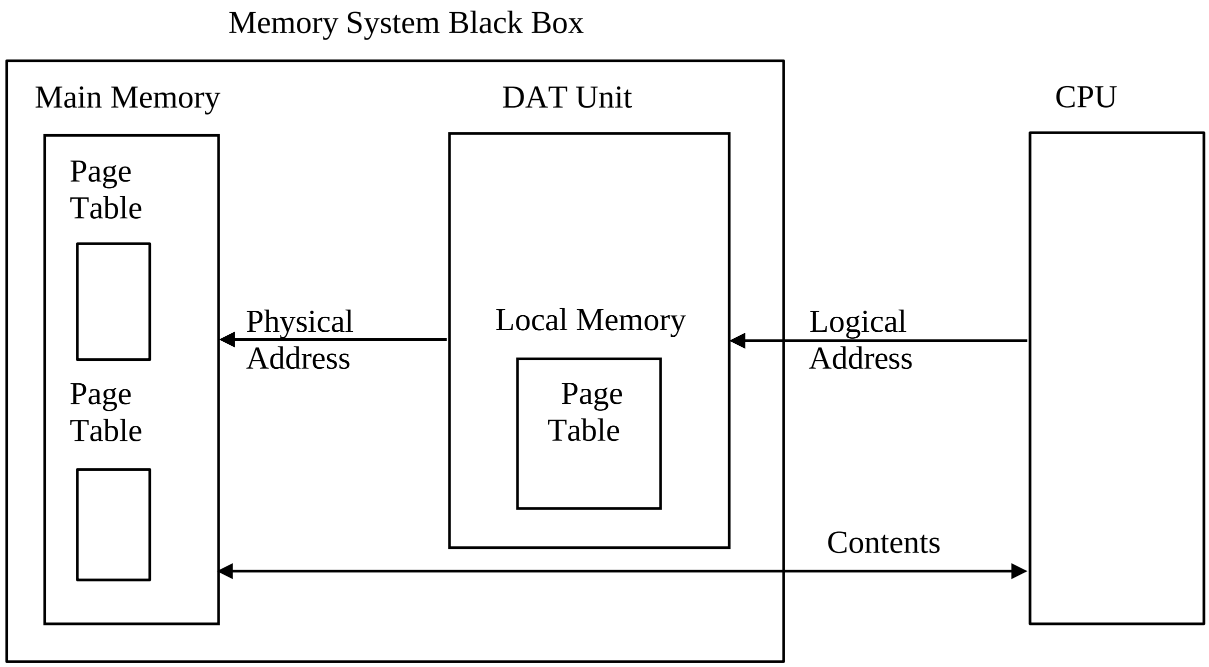

Use left hex digit as index into page table
logical address from program
0 000
physical address
4 000
Page Table
4
6
2
5



Computation of physical address requires a *substitution*, which can be performed quickly.

Associative Memory

The obvious solution to the problem with paging—two reads performed every time the CPU fetches an item from memory—is to keep a copy of the page table in a local memory area within the DAT unit:

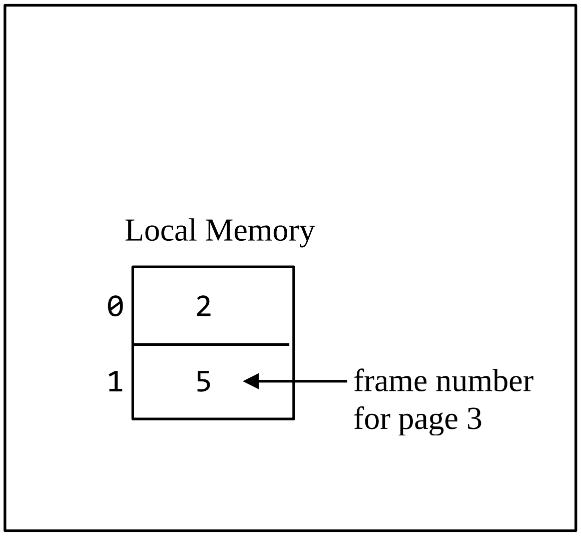


Problem if Subset of Page Table in DAT Unit

Page Table

Page Number	Frame Number
0	4
1	6
2	2
3	5

DAT Unit

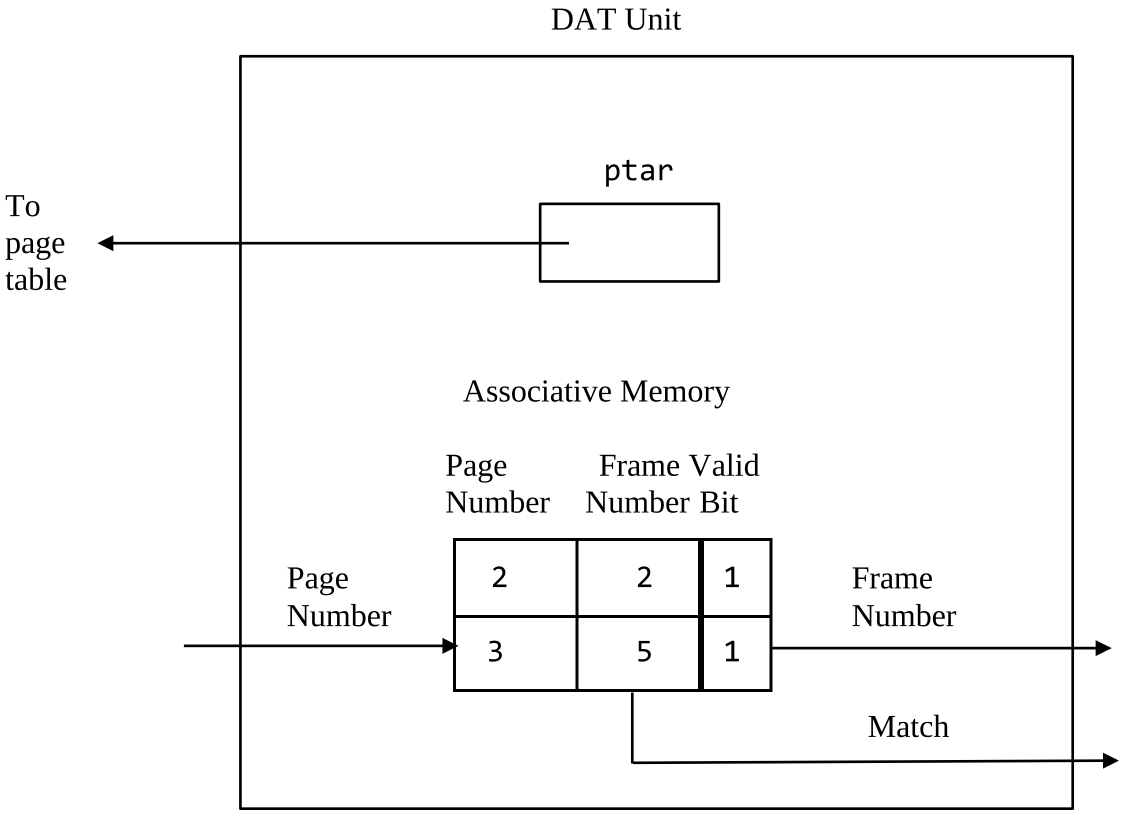


Local Memory

0	2
1	5

frame number for page 3

Solution



DAT Unit

ptar

To page table

Associative Memory

Page Number	Frame Number	Valid Bit
2	2	1
3	5	1

Page Number

Frame Number

Match

Virtual Memory: Demand Paging

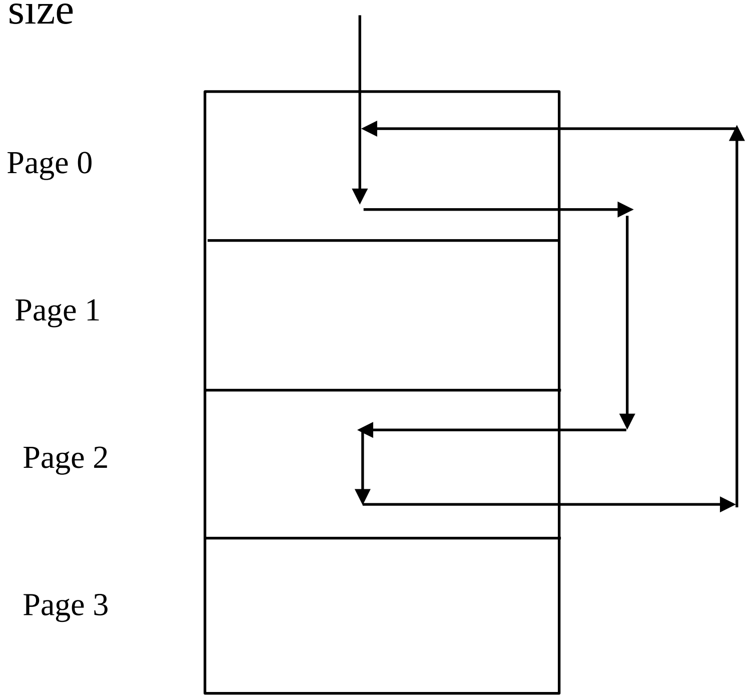
Page Number	Frame Number	Valid Bit
0	0	0
1	0	0
2	2	1
3	5	1

Page Replacement Policies for Demand Paging

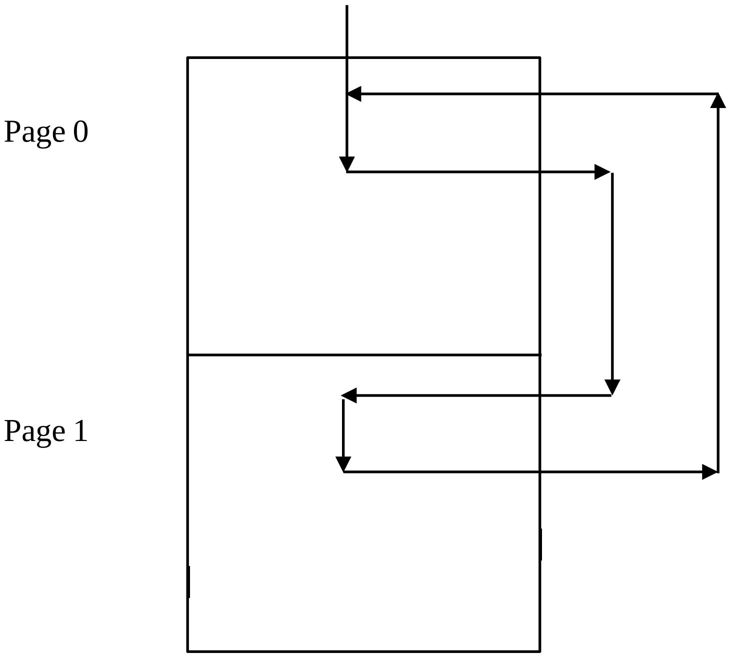
- FIFO
- LRU
- NUR

Page Size Considerations

Small page size



Big page size



Supervisor/User Modes

- Machine instructions that a user program is not allowed to execute *privileged instructions*.

- Privileged instructions are implemented by means of two CPU running modes: a supervisor mode and a user mode.

Memory Protection

Paging provides a simple memory protection mechanism.

Segmentation with Paging

- To solve the problem of page tables that are excessively large because they map unused gaps in the logical address space, we can divide our program into functional segments.

- Each page table would be sufficiently large to map only its corresponding segment.

- The unused gap in the address space would not be represented by any page table. Thus, the combined size of all the page tables would be minimized.

Advantages of Segmentation with Paging

1. It allows the specification of a privilege level (the level of memory protection) and an access mode (i.e, permissible accesses, such as read/write, read-only, execute, etc.) for each segment that is tailored to that segment.

2. Because each segment has, in effect, its own virtual memory, it simplifies the mechanics of dynamically increasing or decreasing the size of a segment.

3. Because segments are logical units of a program, sharing of segments among users is simplified.