澳 門 理 工 學 院
Instituto Politécnico de Macau
Macao Polytechnic Institute

School of Applied Sciences (B.Sc. in Computing)

Notes #6: Memory Management

COMP 213

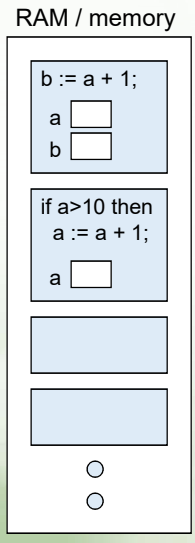
(211/212)

Operating Systems

2019-2020 1st Semester

Memory Management

- Subdivide memory to accommodate multiple processes
- Allocate memory efficiently – pack as many processes into memory as possible



What We'll Learn ...

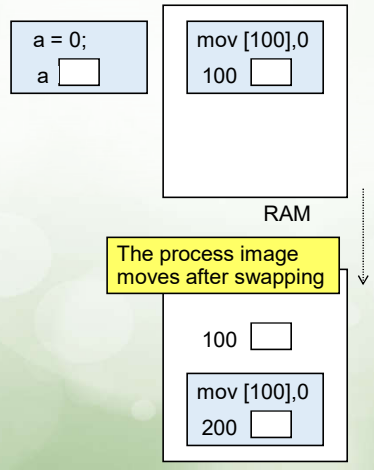
- Memory management requirement
- Fixed partitioning
- Dynamic partitioning
- Paging
- Segmentation
- Textbook: Chapters 7.1 to 7.5
- And … virtual memory, in next chapter

Memory Management Requirements

- Relocation
- Protection
- Sharing
- Logical organization
- Physical organization

Memory Management: Relocation

- When a program is compiled, the address of the code and data are fixed
- The memory range a process occupied may change in run time, e.g. due to swapping
- The program may no longer work afterward



Memory Management: Protection

- Other processes cannot reference memory of a process without permission
- Cannot be checked in compile time, must be checked during execution

Memory Management: Sharing

- Allow several processes to access the same portion of memory
- Examples:
  - Several processes running the same program
  - Several processes sharing a common library
  - Shared memory for data transfer between processes

Memory Management: Logical Organization

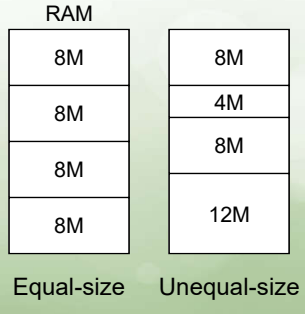
- Programs are written in modules
- Different degrees of protection given to modules (read-only, execute-only)
- Share modules (e.g. DLL)

Memory Management: Physical Organization

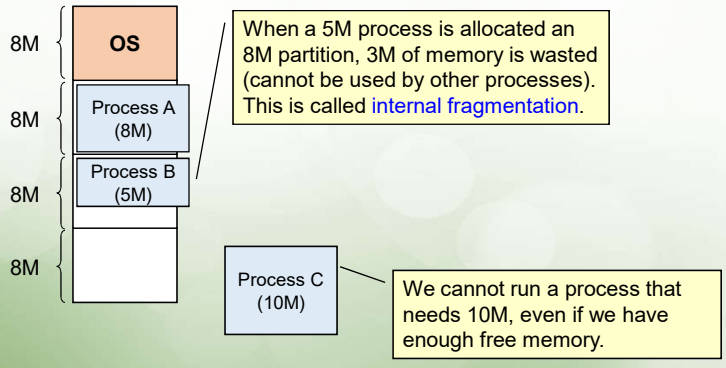
- Memory available may be insufficient
- Secondary memory cheaper, larger capacity, and permanent
- How to use secondary memory to 'simulate' primary memory?
  - Overlay
  - Memory-mapped file
  - Virtual memory (discussed later)

Fixed Partitioning

- Partition available memory into regions with fixed boundary (e.g. at system boot up)
- Each process is allocated ONE partition



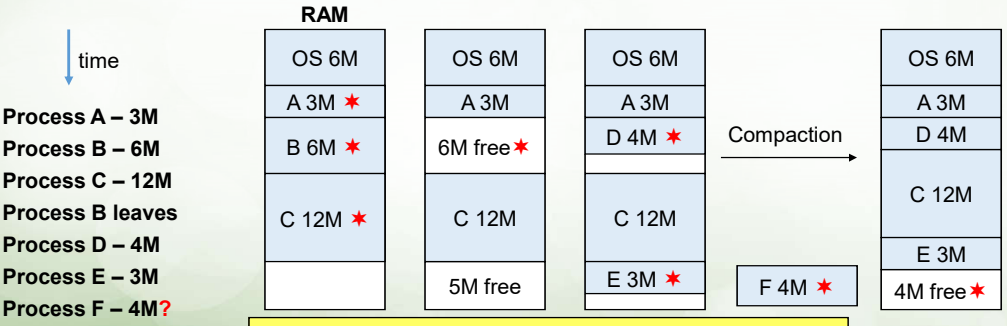
Internal Fragmentation



Dynamic Partitioning

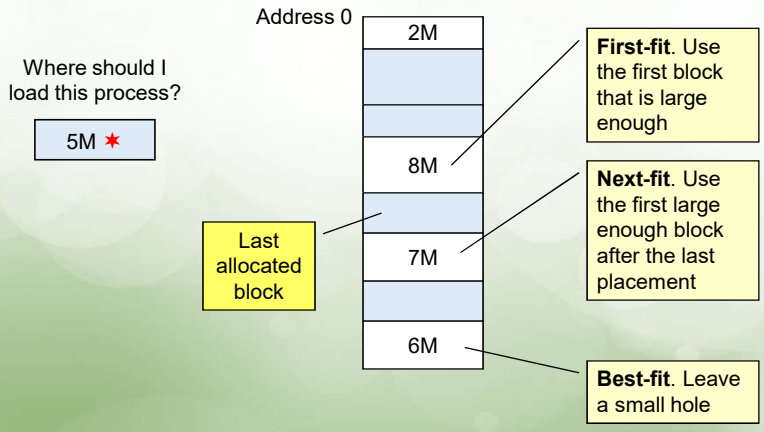
- Partitions are of variable length and number
- Process is allocated exactly as much memory as required
- No internal fragmentation, but there is still external fragmentation
- Must use compaction to shift processes so that all free memory is in one block

External Fragmentation



Holes appear as processes are created and terminated. This is called external fragmentation. In this example, we have totally 4M free memory. But there is no contiguous region large enough for a new process. We need to perform compaction.

Placement Algorithm



Placement Algorithms: Best-Fit Algorithm

- Chooses the block that is **closest in size to the request**
- Worst performer overall
- Small blocks are left all around; compaction must be done more often

Placement Algorithms: First-Fit Algorithm

- Starts scanning memory **from the beginning** and chooses the first available block that is large enough
- May have many processes loaded in the front end of memory that must be searched over when trying to find a free block

Placement Algorithms: Next-Fit Algorithm

- Starts scanning memory **from the location of the last placement** and chooses the next available block that is large enough

Can We Do Better?

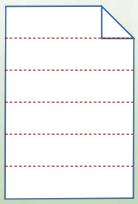
- Basic memory management techniques
  - Partitioning and placement algorithms
- More advanced
  - Paging
  - Segmentation
  - Mixed

Key concepts in Paging

- Physical memory (RAM), frame, physical address
- Addressing space of each process, page, logical address
- Page table
- Address translation

Paging in Pentium

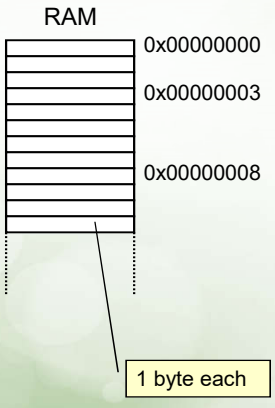
- Pentium is used as an example
- Pentium uses 32 bit address and address bus
- Sizes of a frame and a page are both 4K (4096) bytes ← 12-bit



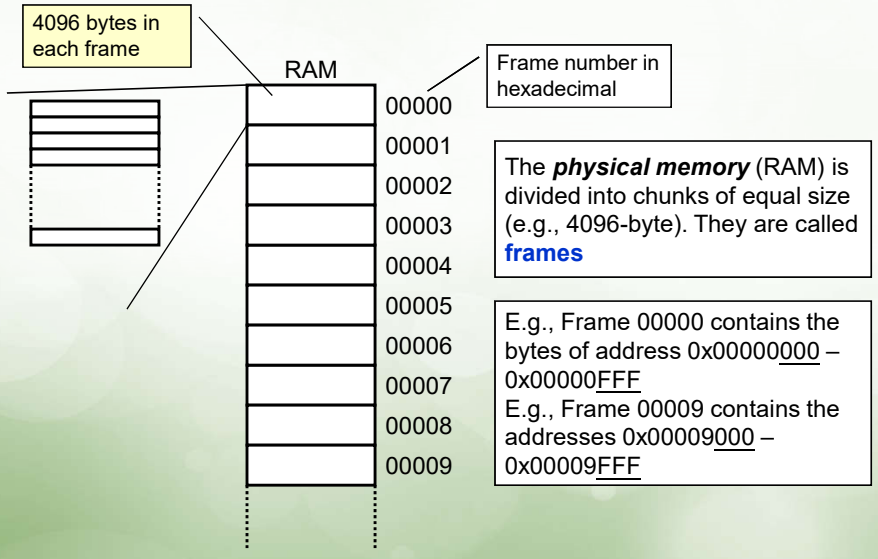
A file of 24KB in size
6 pages, each 4KB in size

Physical Memory / RAM

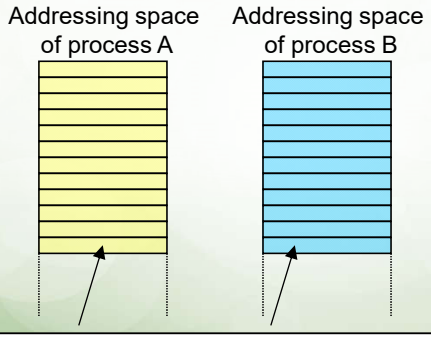
- Each byte in the RAM is identified by a 32 bit **physical address**
- Hence, can plug in at most 2^{32} = 4G bytes of RAM
- The CPU cannot address RAM above 4G
- The address range is 0x00000000 – 0xFFFFFFFF



Frame



Addressing Space



Each process has its own 'memory' called **addressing space**.

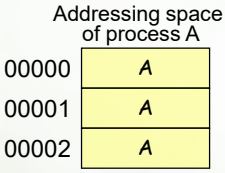
It is an illusion created by the hardware and the OS. Some bytes may be in RAM, some may be in hard disk (virtual memory, discussed later), and some may not exist yet.

Each byte of the addressing space is located by a 32-bit **logical address**. So each process can access at most 2^{32} = 4G bytes

The addressing spaces of process *A* and *B* are separate!

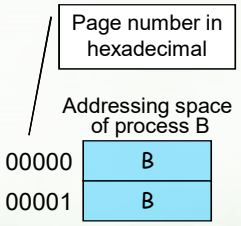
E.g. this byte is not the same as that byte, although *A* and *B* access the bytes using the same address.

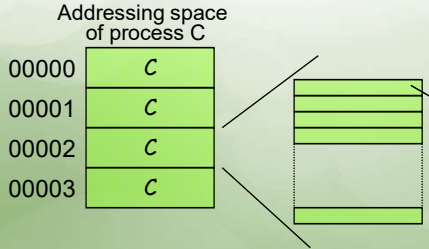
Page



The addressing space of each process is divided into chunks of equal size (same size as the frame size).

They are called **pages**.





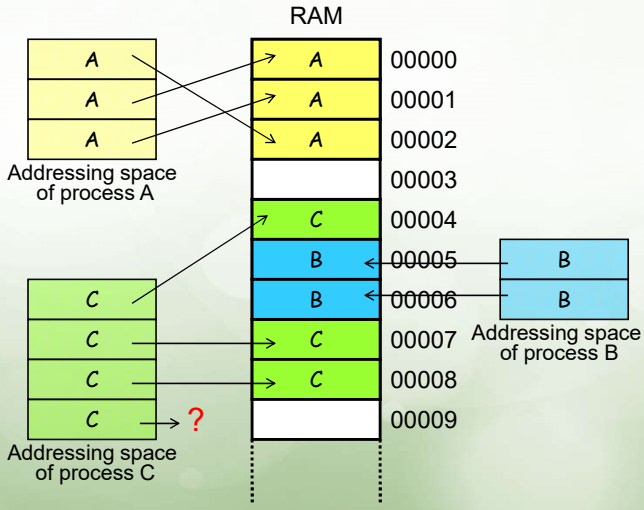
For example, the byte 00002000 is the first byte in the page 00002.

Put Pages into Frames

The pages in the addressing space of each process are mapped to the frames in the physical memory.

Note that contiguous pages may be mapped to separate frames.

Some pages might not be in RAM at the moment. More about this later



Page Table

Page table of process A

00000	
00001	
00002	

Addressing space of process A

00000	A
00001	A
00002	A

RAM

A_1	00000
A_2	00001
A_0	00002
	00003
C_0	00004
B_0	00005
B_1	00006
C_1	00007
C_2	00008
C_3	00009

The OS maintains a data structure called **page table** to store the mapping between the pages and frames.

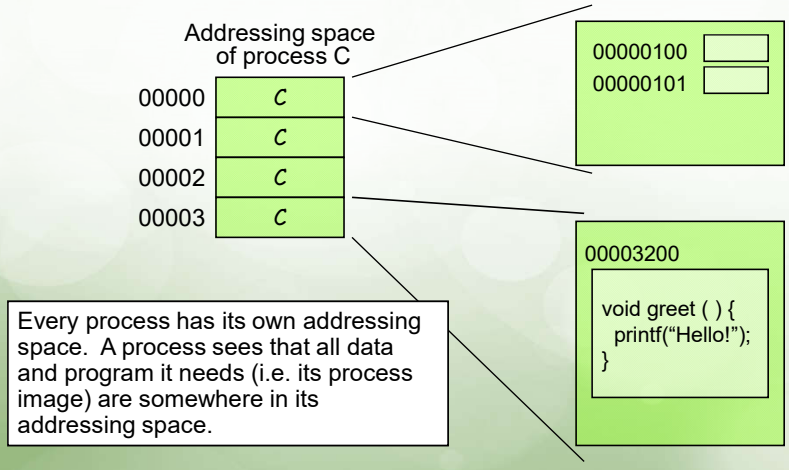
Page table of process B

00000	00005
00001	00006

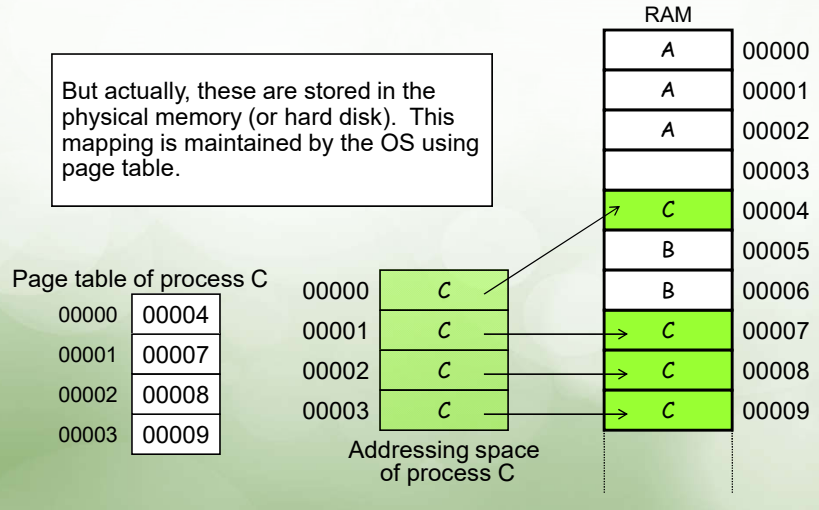
Page table of process C

00000	00004
00001	00007
00002	00008
00003	?

Address Translation (1)

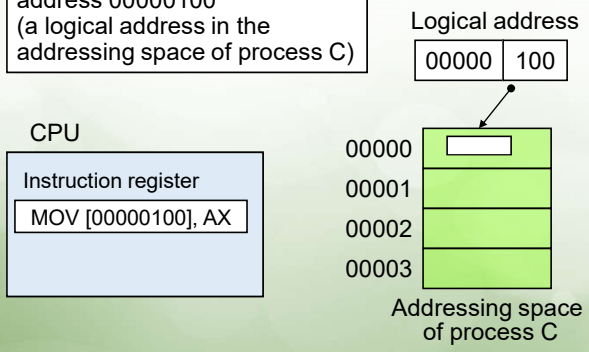


Address Translation (2)



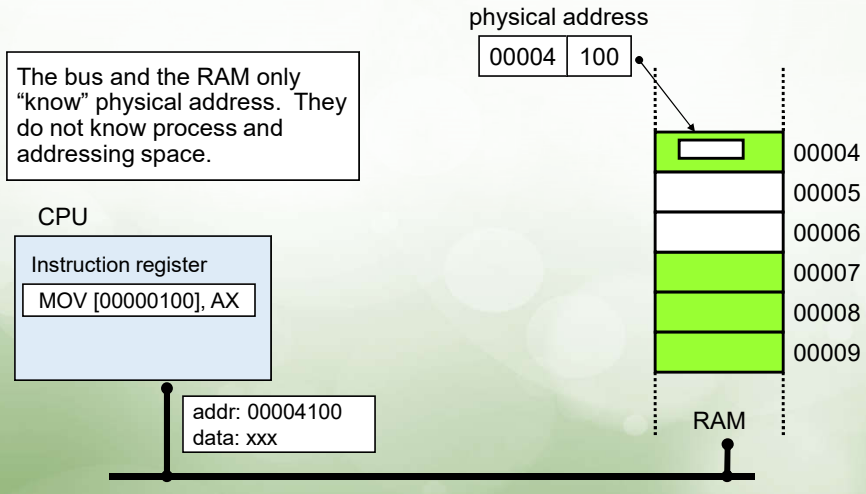
Address Translation (3)

Suppose process C wants to write a global variable at address 00000100 (a logical address in the addressing space of process C)



Address Translation (4)

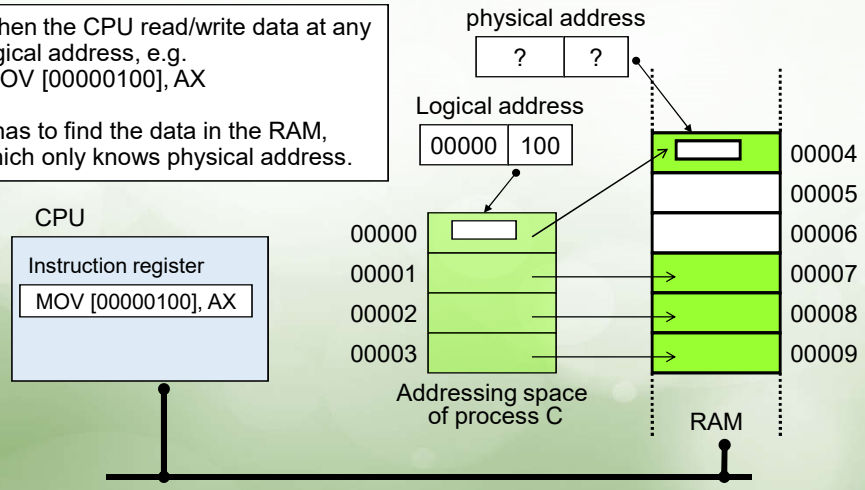
The bus and the RAM only "know" physical address. They do not know process and addressing space.



Address Translation (5)

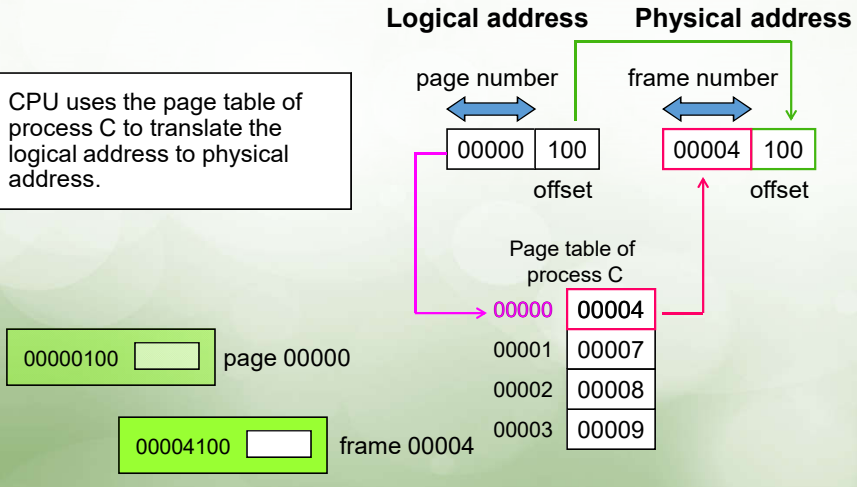
When the CPU read/write data at any logical address, e.g.
MOV [00000100], AX

it has to find the data in the RAM, which only knows physical address.



Address Translation (6)

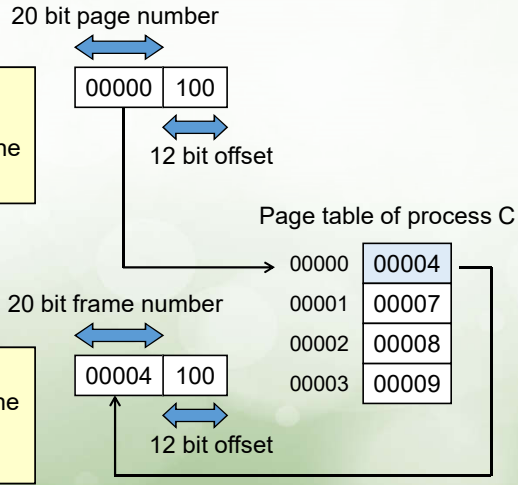
CPU uses the page table of process C to translate the logical address to physical address.



Address Translation (7)

Logical address is the address as seen by a process. The process does not know where in the RAM the byte actually resides in.

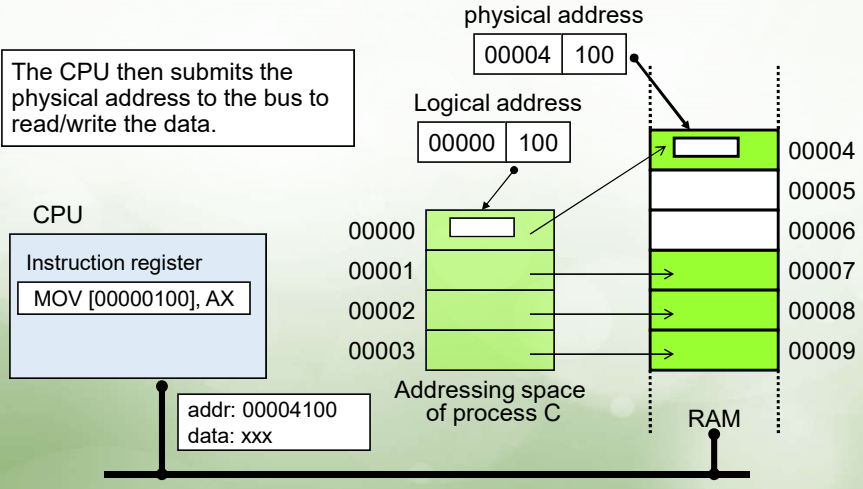
Physical address is the address actually used by the CPU to read the physical memory (RAM). It is the actual address of the byte in RAM.



Eddie Law 33

Address Translation (8)

The CPU then submits the physical address to the bus to read/write the data.



Eddie Law 34

Address Translation (9)

Before executing any instruction in the process, the CPU fetches that instruction. E.g. when the process C calls the function greet( ), the CPU fetches the instruction at the logical address 00003200.

MOV "instruction register", "instruction at 00003200"

The CPU again needs to translate the logical address to physical address before fetching the instruction through the bus.

page 00003 frame 00009

On Address Translation

- All addresses seen by a process is logical address
- To read some data or code from RAM, the CPU has to submit the physical address (the real one) onto the bus
- The CPU uses the page table of the running process to translate every address used at runtime

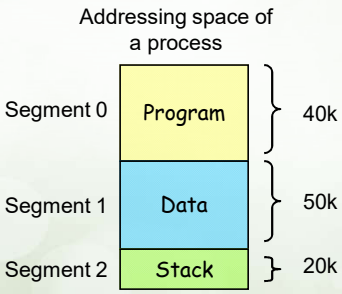
On Address Translation (cont'd)

- … is done every time the CPU needs to access data/code in RAM
  - when the CPU fetches an instruction
  - when the CPU executes an instruction that access data in RAM (not present in cache)

Segments

The addressing space of a process is divided into parts of unequal length called **segment**s.
Each segment usually holds 'data' of a certain type, e.g. program code, global data, and stack. Each segment is identified by a number.

Segments in Pentium is implemented with segment registers (CS, DS, SS, etc) and implicit or explicit association with address register. In concepts, this is similar to what we study here



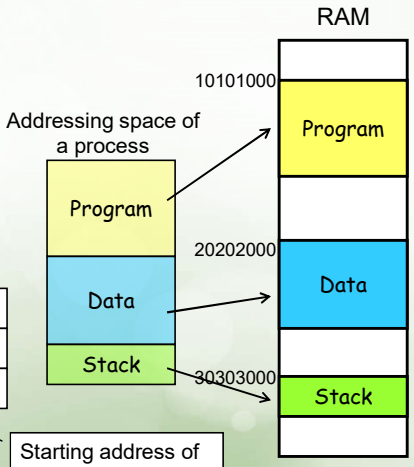
Segment Table

The segments are mapped to the RAM through a **segment table**. It marks the base address and length of the segment.

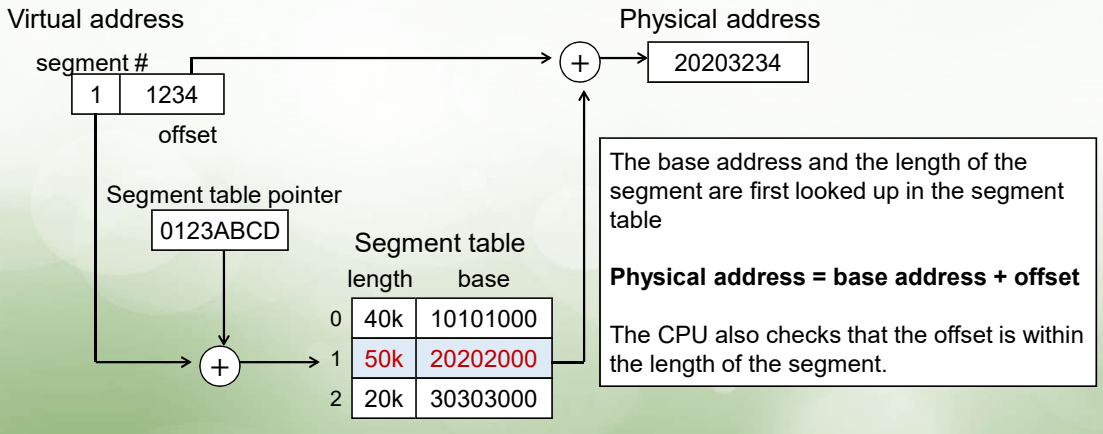
Segment table

	P	M	length	base
0	1	0	40k	10101000
1	1	1	50k	20202000
2	1	1	20k	30303000

Starting address of the segment



Address Translation



The base address and the length of the segment are first looked up in the segment table

Physical address = base address + offset

The CPU also checks that the offset is within the length of the segment.

Segmentation

- Simplifies handling of growing data structures
- Allows programs to be altered and recompiled independently
- Used for sharing data among processes
- Lends itself to protection

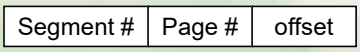
Combined Paging and Segmentation

- Paging is transparent to the programmer
- Paging eliminates external fragmentation
- Segmentation is visible to the programmer
- Segmentation allows for growing data structures, modularity, and support for sharing and protection
- Each segment is broken into fixed-size pages

Address Translation

- The process has a page table for each segment
- The CPU uses the segment # to get a page table
- The CPU generates the physical address using the page #, offset and the page table

Virtual address



Remark

- Different partitioning techniques
- Paging and segmentation
- Next – virtual memory