# Chapter 3: Memory management

## Operationg System

@ptit.edu.vn

Posts and Telecommunications
Institute of Technology
Faculty of Information Technology 1


IT1 Faculty
PTIT

August 15, 2023

# Mục lục


- Address and related issues
  - Logical address and physical address
- Some ways to organize the program
  - Loading during runtime
  - Dynamic linking and shared libraries
- Memory management requirements
  - Reissue
  - Protect
  - Share
  - Logical structure and physical structure
- Memory division

# Content of chapter 3

1. Address and related issues
2. Some ways to organize the program
3. Memory management requirements
4. Memory chaptering
5. Memory paging
6. Memory segment
7. Virtual memory

# Memory management

Bộ nhớ *is the second most important resource after the CPU in a computer system, consisting of bytes and addressed memory words.*

- Memory is where the process and data for the process are stored
- Memory management affects the processing speed and computing power of the entire system
- Tasks related to memory management include:
  - Manage free memory
  - Allocate and release memory for processes
  - Prevent unauthorized access to memory areas
  - Address mapping between logical addresses and physical addresses

# Address and related issues

- Computer memory is addressed in bytes or words

| Dữ liệu hoặc lệnh | Địa chỉ byte (theo Hexa) |
|---|---|
| byte (8-bit) | **0x0000 0000** |
| byte | 0x0000 0001 |
| byte | 0x0000 0002 |
| byte | 0x0000 0003 |
| byte | **0x0000 0004** |
| byte | 0x0000 0005 |
| byte | 0x0000 0006 |
| byte | 0x0000 0007 |
| . | |
| . | |
| . | |
| byte | 0xFFFF FFFB |
| byte | **0xFFFF FFFC** |
| byte | 0xFFFF FFFD |
| byte | 0xFFFF FFFE |
| byte | 0xFFFF FFFF |

$2^{32}$ bytes

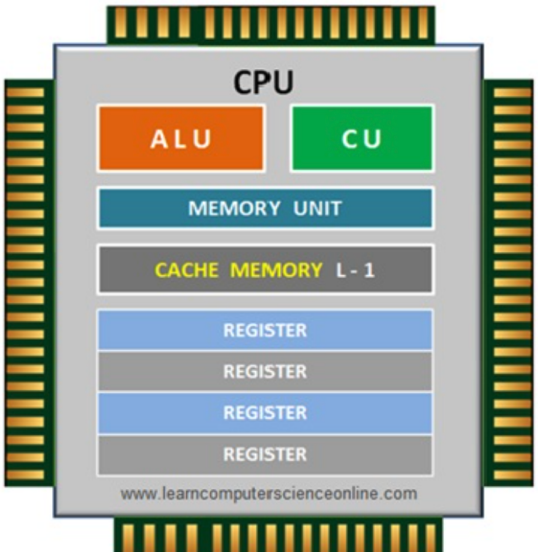
| Dữ liệu hoặc lệnh | Địa chỉ word (theo Hexa) |
|---|---|
| word (32-bit) | 0x0000 0000 |
| word | 0x0000 0004 |
| word | 0x0000 0008 |
| word | 0x0000 000C |
| word | 0x0000 0010 |
| word | 0x0000 0014 |
| word | 0x0000 0018 |
| . | |
| . | |
| . | |
| word | 0xFFFF FFF4 |
| word | 0xFFFF FFF8 |
| word | 0xFFFF FFFC |

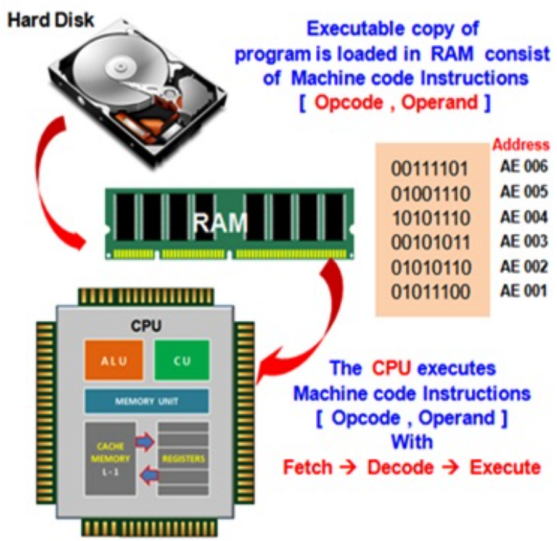
$2^{30}$ words

Computer Architecture

# Address and related issues (cont.)

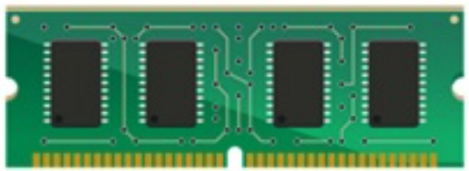
- The CPU process reads instructions from memory one by one and executes these instructions
- Data and instructions are both localised
- The CPU will use addresses to identify specific instructions and data



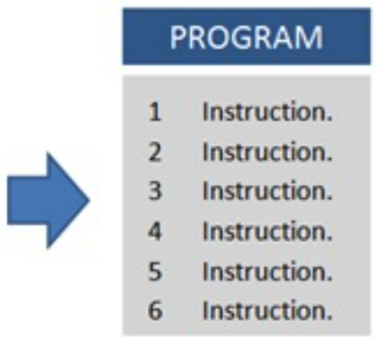



# Address and related issues (cont.)

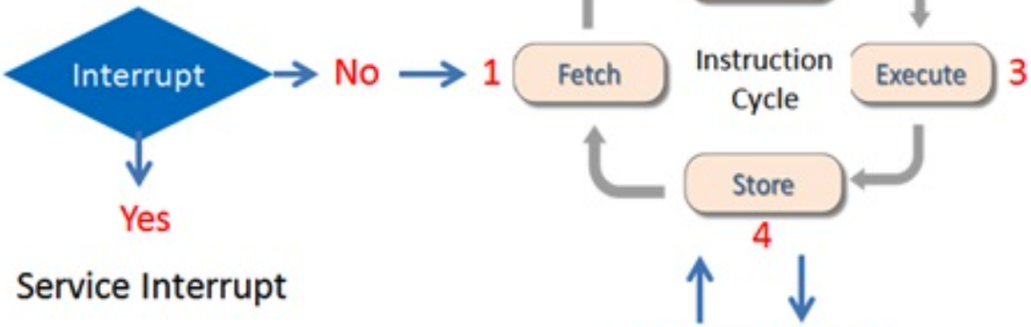

Disk Memory
Secondary Memory
Operating System Loads
Program Into RAM



Primary Memory
Main Memory
RAM




Program Is Set Of
Instructions Stored In The
Main Memory RAM



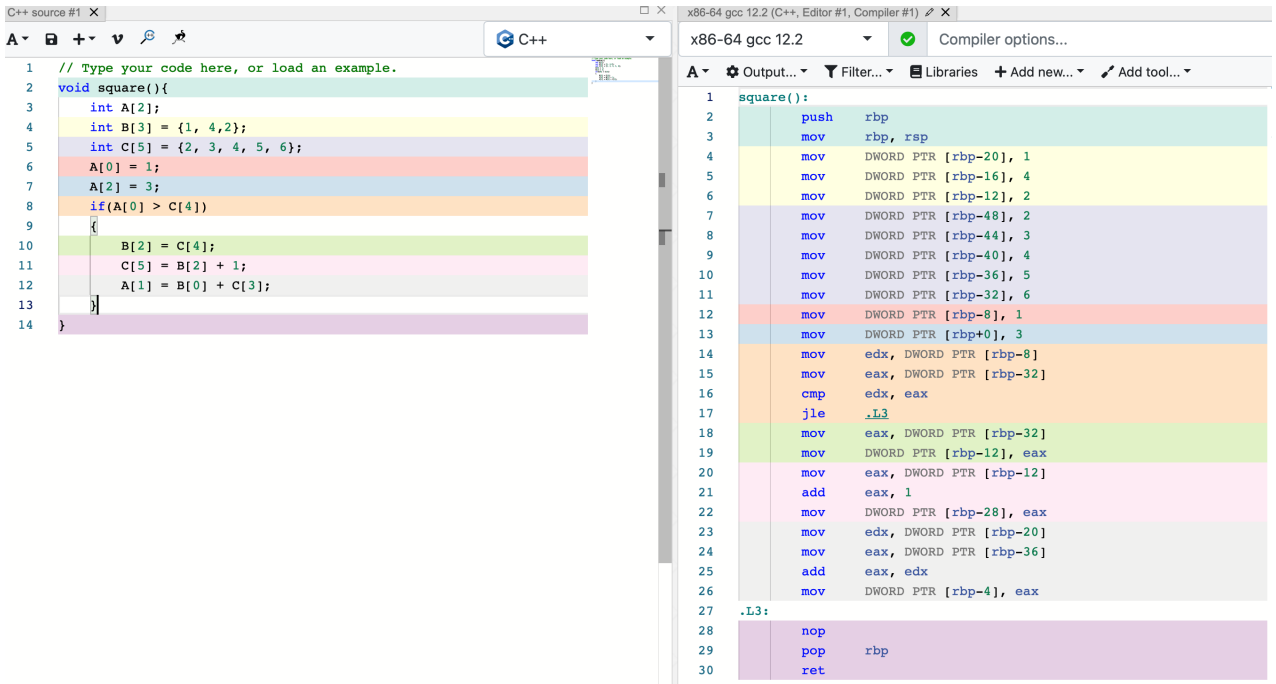

Service Interrupt


Processor CPU


Instruction Format

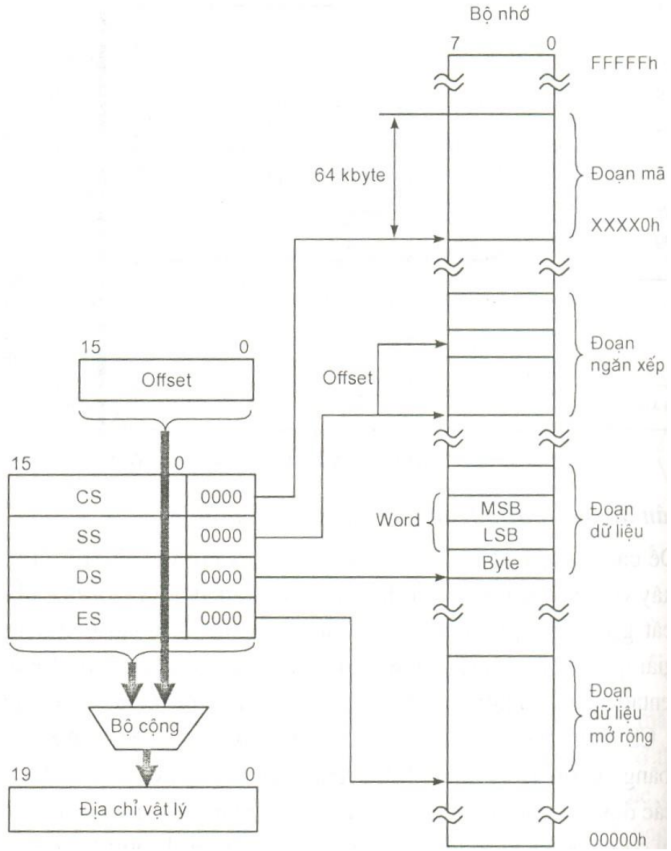
# Address and related issues (cont.)

- Programs use addresses as names (variables, functions)
- When translating, the translation program maps names to relative addresses from the beginning of the obj file
- The link program maps addresses to relative addresses from the beginning of the program



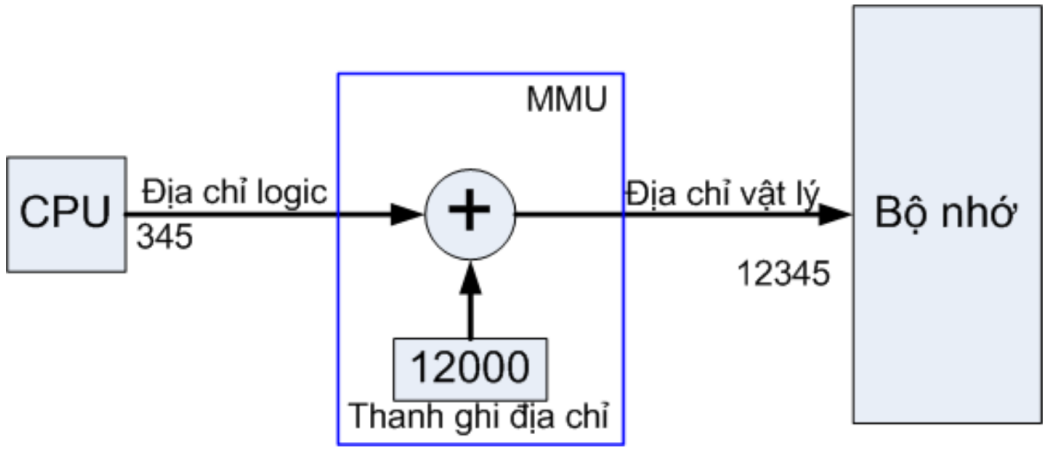

Godbolt

VXL 8086 uses 20 bits to encode 1MB addresses from 00000h - FFFFFh




20-bit physical address of a memory cell = 16-bit segment address shifted 4 bits left (multiplied by 16) + 16-bit instruction address.

- Logical address
  - Assignment to instructions and data does not depend on the specific location of the process in memory. Spawned in process (CPU launched)
  - CTUD only cares about logical addresses and physical addresses are determined by the Operating System
  - When executing a program, the CPU "sees"and uses this logical address to point to different parts of instructions and data.
  - To access memory, logical addresses need to be mapped to physical addresses and converted to physical addresses by the memory management unit (MMU) when accessing objects in the program.




For example: JMP 010A: Jump to memory location 010Ah at the same code (CS - Code Segment)
If CS = 1555h, will go to position: 1555h * 10h + 010Ah = 1560Ah

- Physical address:
  - Is the exact address in computer memory
  - Memory circuits are used to access programs and data

- Logical addresses are converted to physical addresses using the address mapping block. (MMU=Memory Mapping Unit)

# Some ways to organize the program

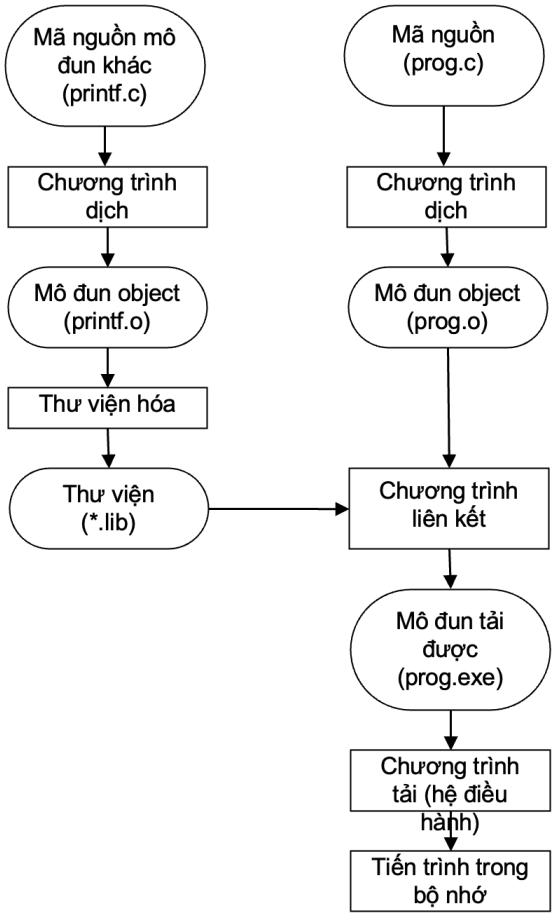
Important issues in program organization and memory management:

- Reduce the space the program takes up on disk and memory
- Use memory space effectively
- Techniques for organizing and using memory effectively:
  - Loading during runtime
  - Dynamic linking and shared libraries

The entire program is normally loaded into memory for execution.
For large programs, during one session, some parts of the program may not be used. These functions will take up unnecessary memory and increase the initial program load time.
**Solution:**

- A function that has not been called has not been loaded into memory
- The main program is loaded into memory and run
- When there is a function call:
  - The program will check if the function is loaded.
  - If not, the program will proceed to load then map the function addresses into the program's common space and change the address table to record those mappings.
- The programmer is responsible for checking and loading functions. The operating system only provides library functions for the module

- Programs must go through a translation and linking process before becoming downloadable and executable.

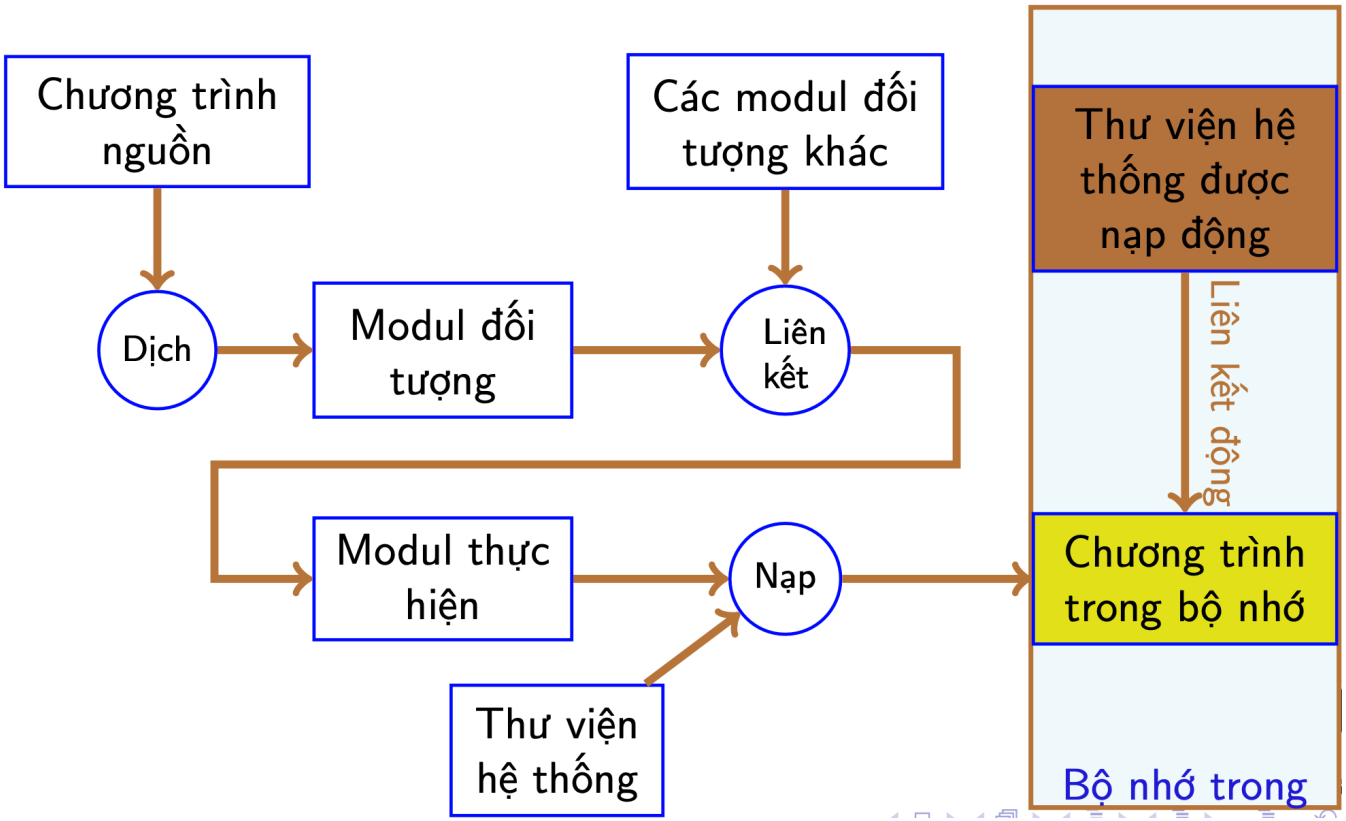


During static linking, functions and libraries are always linked into the program code
*Program size = Size of newly compiled program + Size of library functions*

- Functions will appear repeatedly in programs
- Waste of space both on disk and internal memory

**Solution** use **dynamic linking technique** : *During the linking phase, do not connect library functions into the program but only insert information about that library function (stub).*

- Programs must go through a translation and linking process before becoming downloadable and executable.



- Library modules are linked in the implementation:
  - The process does not keep copies of library modules, but the process keeps a small piece of code containing information about the library module
  - When the small code is called, it checks whether the corresponding module is already in memory. If not, then download the rest and use it.
  - The next time you need to use it, the library module will be run directly
  - Each library module has only one copy contained in MEM
  - Need support from OS

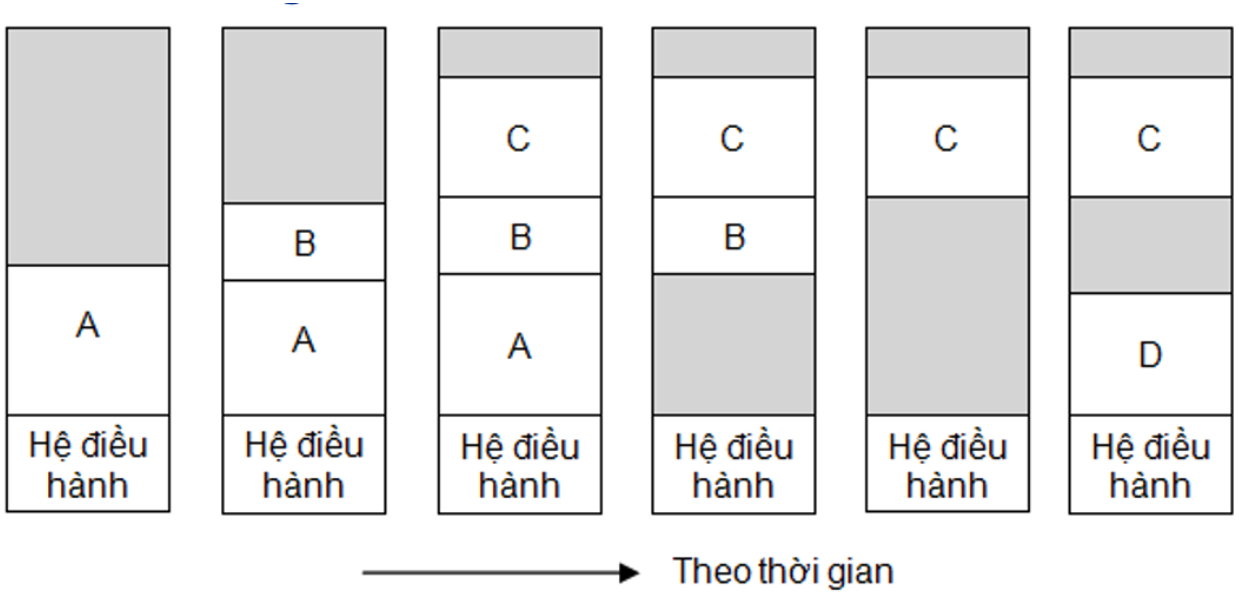
# Memory management requirements

- Memory management:
  - **Memory allocation:**for processes
  - **Reposition**,maps instruction and data addresses from the process into memory
  - **Protect the process**: Some memory for one process is not used by another process
  - **Share:** Processes can share memory spaces with each other

Requirements related to memory management include:

- **Reallocation** to processes
- **Protection**: any memory for any process cannot be used by another process
- **Shared:** processes can share memory spaces when interacting with each other
- **Structure:** logical structure and physical structure

- It is necessary to be able to swap processes in and out of MEM to maximize processor utilization
- You cannot request that a process be moved back into MEM, it must be in the same place it was used before being moved out



- Each process must be protected from unwanted references from other processes to its reserved memory
- When a memory area is used by this process, it does not allow another process to reference that memory area.
- All memory references of a process must be checked at runtime
- The operating system cannot predict every MEM reference, so the microprocessor hardware takes care of it

- Multiple processes need and are allowed to access the same memory area
- Collaborating processes need to share access to a data structure
- Allow access to shared areas

**Logical structure:**

- MEM is structured in a linear way consisting of bytes, while the program is organized into modules
- Must respond to:
  - Modules can be written and interpreted independently
  - The level of protection may vary
  - Modules can be shared between processes

**Physical structure:**

- 2-level physical structure:
  - Main memory: fast; High cost, low capacity
  - Secondary memory: large capacity, allowing to save programs and data for a long time

- The system is responsible for converting information between two levels

# Memory chaptering

- To execute the process, the Operating System needs to allocate the necessary memory space to the process.
- Memory allocation and management is an important function of the operating system
- The simplest allocation technique is for each process to be allocated a contiguous area of memory
- The operating system divides memory into continuous parts called partitions, each process will be provided with a chapter to contain its instructions and data.
- The process of dividing memory into chapters is called *memory chaptering*.
- Depending on the choice of chapter location and size, fixed and dynamic chaptering can be distinguished

# Memory chaptering (cont.)

Memory management strategies:

- Fixed chapter division
- Dynamic chapter division
- Pagination
- Segment
- Combination of segmentation - paging

# summary

## Chapter 3 Memory management

- Address and related issues
- Some ways to organize the program
- Memory management requirements

## Chapter 3 Memory management

- Memory chaptering
- Memory paging
- Memory segmentation
- Virtual memory