江苏科技大学物联网工程实践教育中心
典型多媒体课件简介

一、典型课件基本情况介绍

1、《微机原理与接口技术课件》制作人李永忠教授，曾获得教学部教育管理信息中心组织的“第十二届全国多媒体课件大赛”高教工科组三等奖。

证　书

李永忠、张明 制作的课件《微机原理与接口技术》在“第十二届全国多媒体课件大赛”中荣获高教工科组三等奖。

2、“十三五”江苏省高校中心教材《C++程序设计基础教程》课件，该教材为中心教师自编新教材，并制作配套教学的课件。

3、《计算机组成原理》课程，典型的计算机大类专业的专业基础课程，面向计算机、物联网等专业教学。

二、课件典型部分章节内容展示

1、《微机原理与接口技术》、《C++程序设计基础教程》和《计算机组成原理》等

微机原理及接口技术

计算机科学与工程学院

李永忠 教授

E-mail:
liyongzhong61@163.com
Tel: 13852946785

微机 原理 及 接口 技术

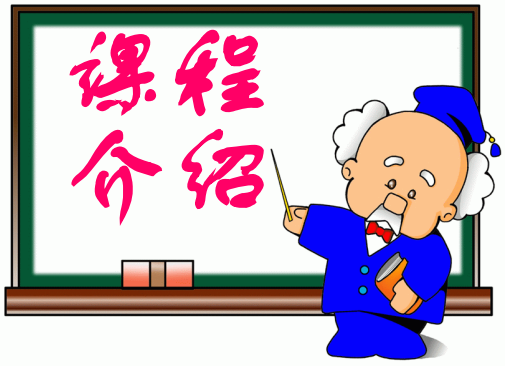
典型机型： IBM PC系列机

基本系统：8088CPU和半导体存储器

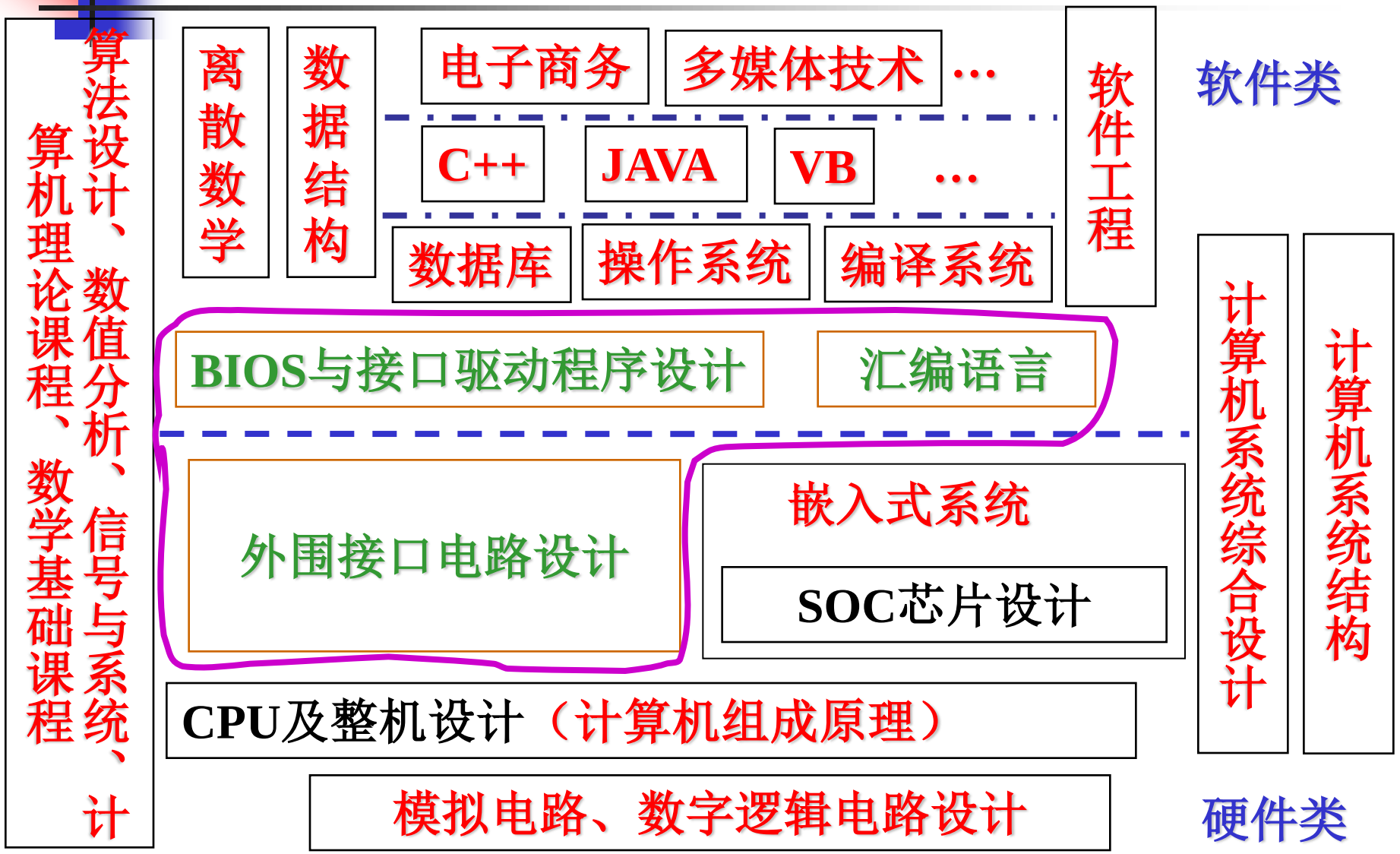
I/O接口电路及与外设的连接

硬件——接口电路原理

软件——接口编程方法

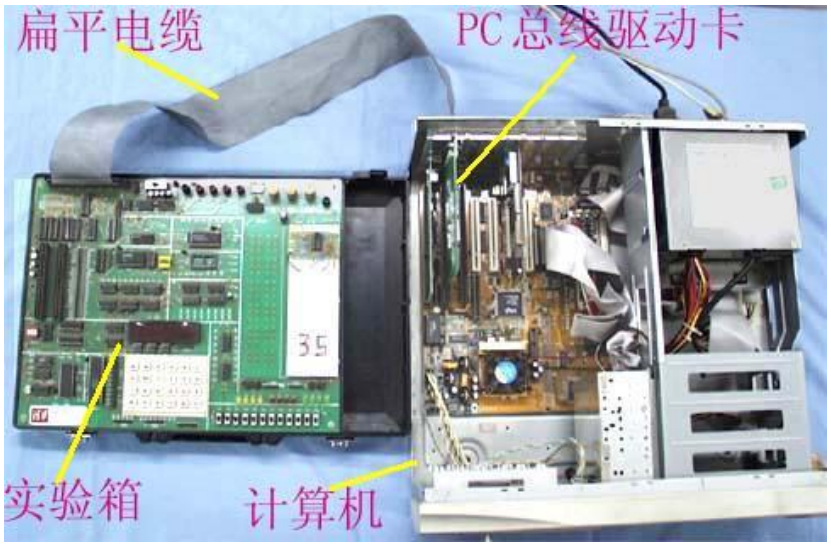


本课程在计算机课程体系中的位置



“微机原理与接口技术” 学习目的

1. 从应用角度，了解计算机的基本组成、工作原理，
   建立计算机系统的整体概念；　　(消除对计算机的畏惧心理）
2. 掌握汇编语言程序设计方法；　　(从底层了解计算机）
3. 了解计算机接口技术，初步具备计算机软、硬件开发能力。
   (为将来更好学习工作打基础）



(清华)计算机科学与技术系与硬件有关的必修课程：

1. 数字逻辑 (二年级春季， 5学分/80学时)
2. 汇编语言程序设计 (二年级春季， 5学分/80学时）
3. 计算机组成原理 (三年级春季， 5学分/80学时)

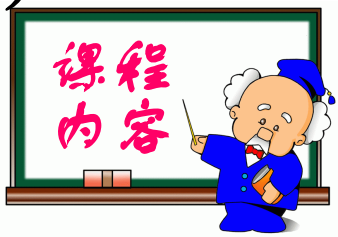
   讨论计算机各基本部件的组成原理和运行机制，及其硬件实现
4. 微机与接口技术 (三年级秋季， 4学分/64学时)

   从应用角度，详细介绍微处理器芯片，接口技术和应用编程。
5. 计算机系统结构 (四年级春季， 4学分/64学时)

   讨论计算机系统的各种基本结构，设计技术和性能定量分析方法。
6. 计算机网络原理 (四年级秋季， 4学分/64学时)

章节目录　　计划学时

- 第一章　计算机基础知识　（6学时）
- 第二章　8086/8086微处理器　（10学时）
- 第三章　汇编语言程序设计　（10学时）
- 第四章　PC机的总线结构和时序　（6学时）
- 第五章　微机接口技术概述　（6学时）
- 第六章　中断技术　（6学时）
- 第七章　并行输入/输出接口　（8学时）
- 第八章　数/模、模/ 数转换接口　（6学时）
- 第九章　半导体存储器　（4学时）
- 第十章　高档微处理器　（2学时）



章节目录（续） 计划学时

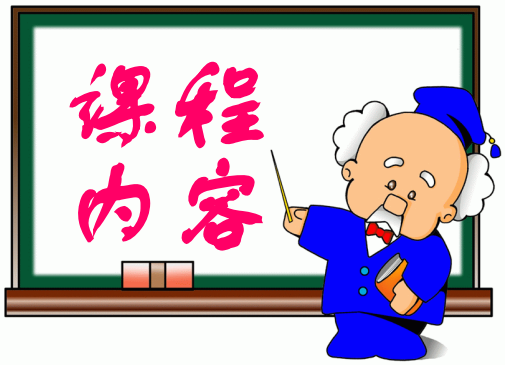
课内：56学时

实验：24学时（11实验）

教材：《微机原理与接口技术》
李永忠主编
电子工业出版社

参考：《微型计算机原理及接口技术》（第二版）
裘雪红主编
西安电子科技大学出版社

《微机原理与接口技术》
牟琦，聂建萍
清华大学出版社



参考书： 微机接口类 索引号TP36

1. IBM-PC汇编语言程序设计 沈美明 清华出版社
2. 80X86汇编语言程序设计教程 杨季文 清华出版社
3. IBM PC Assembly Language and Programming
4. 计算机硬件技术基础 张菊鹏 清华出版社(第2版)
5. 微型计算机技术及应用 戴梅萼 清华出版社
6. 计算机组成与结构 王爱英 清华出版社
7. Computer Organization and Design:

   The Hardware/Software Interface

8. 阅读“中国期刊网”相关主题的科技文章，了解应用背景与实例

“中国期刊网”可通过清华图书馆的主页进入, 使用简单方便

图书馆有很多可供做参考的书：

- 微机原理与接口技术　　雷丽文　电子工业出版
- 微型计算机系统原理及应用　　杨素行　清华出版社
- 微型计算机原理及应用　　周明德　清华出版社
- 微型计算机接口技术　　李大友　清华出版社
- 微型计算机原理与接口技术　　裘雪红　西安电子科大出版
- 微型计算机原理及应用辅导　　李伯成　西安电子科大出版
- 微型计算机原理及应用学习辅导　　朱定华　电子工业出版社

、、、、、、

涉及80x86汇编语言程序设计和微机接口技术的均可

课程性质：专业技术基础课

硬件系列课程之一

计算机组成原理

微机原理及接口技术

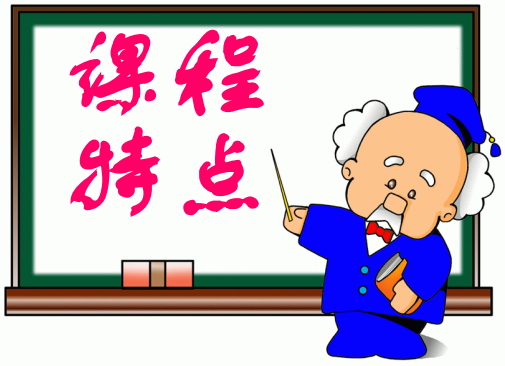
计算机体系结构

指定选修课

以技术为主

面向应用

软硬件相结合



先修课程

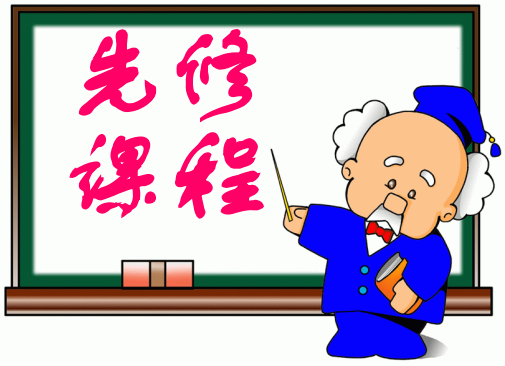
数字逻辑

提供硬件基础

计算机组成原理

确立计算机部件功能

掌握计算机工作原理



学习方法

战略上轻视它，战术上重视它

（复习并掌握选修课的有关内容）

要点： 课堂：听讲与理解、适当笔记；

课后：认真读书、完成作业

实验：充分准备、勇于实践

方法： 1. 合理安排时间，跟上教学进 度

2. 上课专心听讲，认真完成作 业

3. 重视实验环节，做好实验报 告

（利用 DEBUG 程序学习、多上机实践 ）

4. 要有充分信心，不要轻易放弃

推荐的学习风格：

学习方式　　独学 ⇒ 群学

- 知识系统　　封闭 ⇒ 开放
- 知识获取　　被动接受 ⇒ 主动建构

带着问题进行学习和实验，建构自己的知识体系

考试方式及成绩评定

1. 采用闭卷笔试、上机做实验方式考试。

2. 学习成绩由以下几部分综合给出：

期末	闭卷笔试	60%
平时	实验及报告	20%
	平时成绩（考勤＋作业）	20%

总成绩＝考试成绩 ＋实验成绩＋平时成绩

第1章

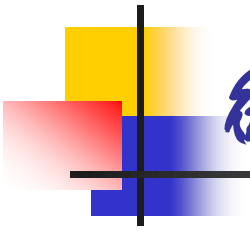
微型计算机系统概述

绪 论

第1章 微型计算机系统概述

教学重点

- 微型计算机的系统组成
- 计算机中数据的表示

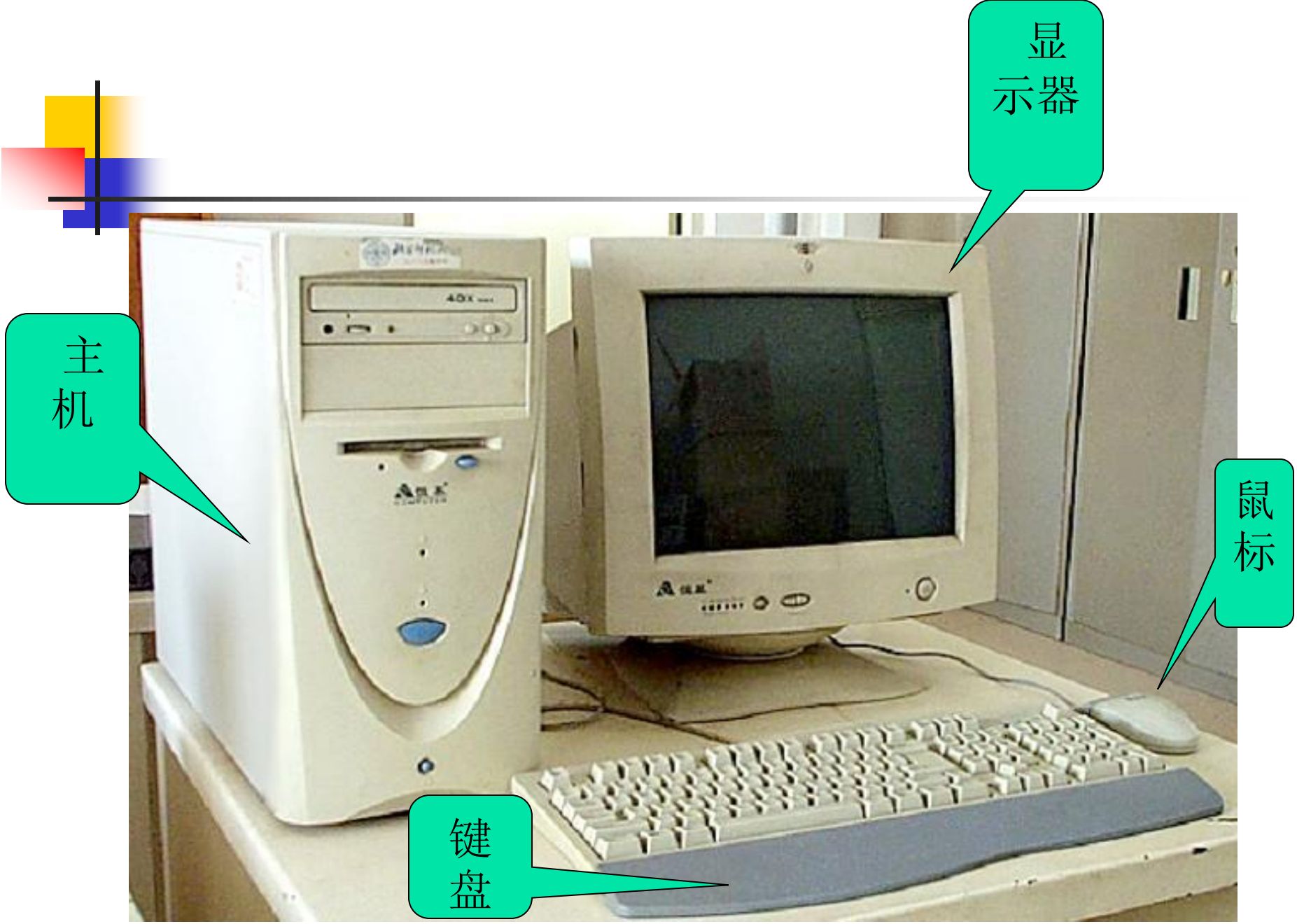


微型计算机系统

思考：

什么是微型计算机？

微机由哪几个部分构成？

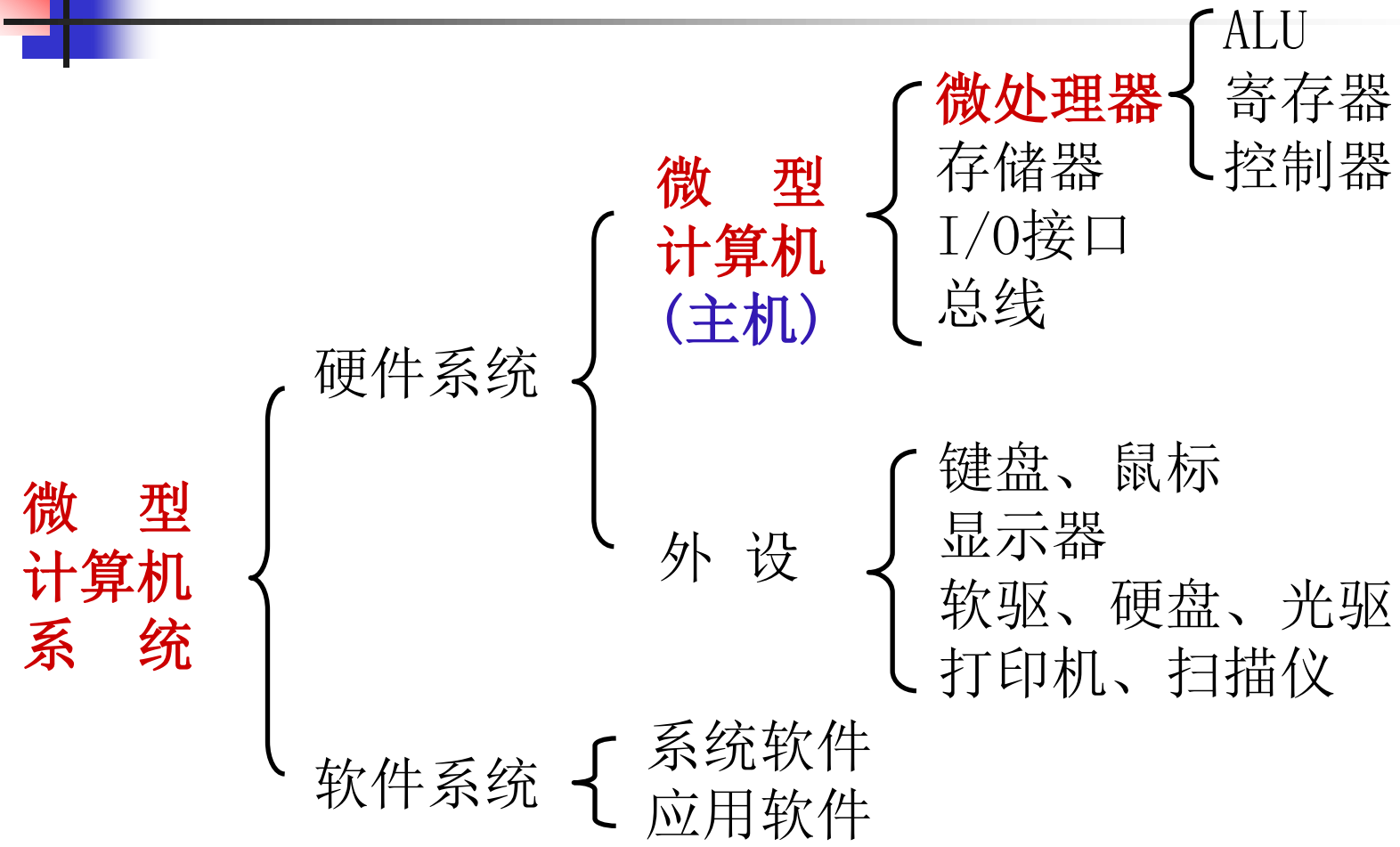

显示器
主机
鼠标
键盘

三个基本概念

微型计算机系统的三个层次

- 微处理器(Microprocessor)
- 微型计算机(Microcomputer)
- 微型计算机系统(Microcomputer System)

1.1 微型计算机系统的三个层次

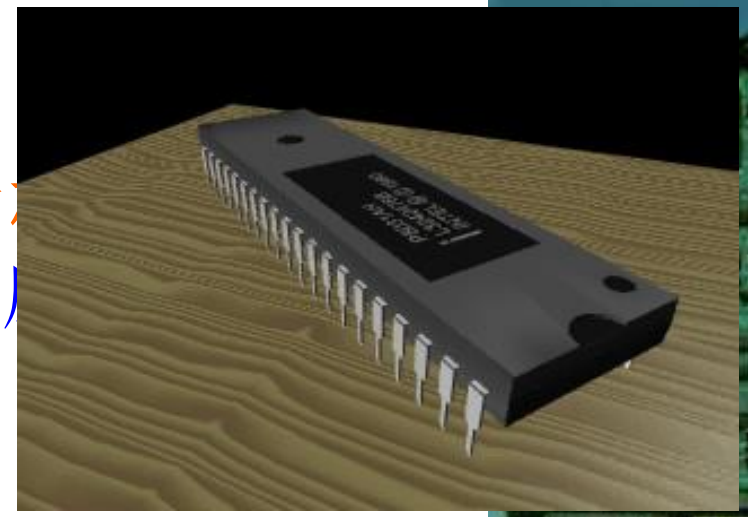


微型计算

- 微型计算机系统
  - 硬件
    - 微型计算机（主机）
    - 外围设备
  - 软件
    - 系统软件(操
    - 应用软件(科
    - 程序设计语

单片机简介

- 单片机即单片机微型计算机，是将计算机主机(CPU、内存和I/O接口)集成在一小块硅片上的微型机。
- 单片机为工业测控而设计，又称微控制器。具有三高优势(集成度高、可靠性高、性价比高)。
- 主要应用于工业检测与控制、计算机外设、智能仪器仪表、通讯设备、家用电器等。特别适合于嵌入式微型机应用系统。
- 单片机



1.2 微型计算机的发展和应用

- **1946**年，世界上出现第一台数字式电子计算机**ENIAC**（电子数据和计算器）
- 发展到以大规模集成电路为主要部件的第四代，产生了微型计算机
- **1971**年，**Intel**公司设计了世界上第一个微处理器芯片**Intel4004**，开创了一个全新的计算机时代

1.2.1 微型计算机的发展

- 第1代：4位和低档8位微机

  4004→4040→8008

- 第2代：中高档8位微机

  Z80、I8085、M6800，Apple-II微机

- 第3代：16位微机

  8086→8088→80286，IBM PC系列机

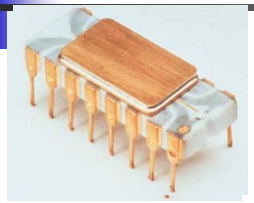
1.2.1 微型计算机的发展（续）

- 第4代：32位微机
  - 80386→80486→Pentium→Pentium II →Pentium III →Pentium 4
  - 32位PC机、Macintosh机、PS/2机
- 第5代：64位微机

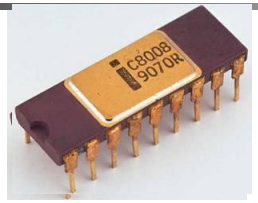
  Itanium、64位RISC微处理器芯片

  微机服务器、工程工作站、图形工作站

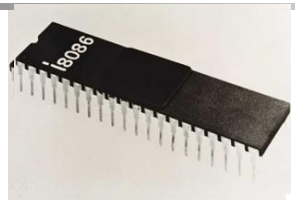
1.2 微处理器发展



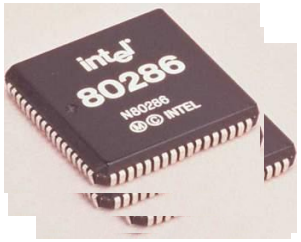
Intel 4004



Intel 8008



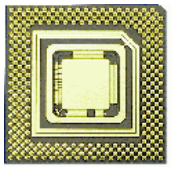
Intel 8086



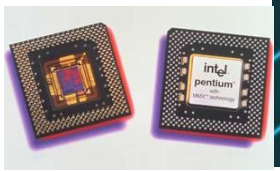
Intel 80286

Intel 80386

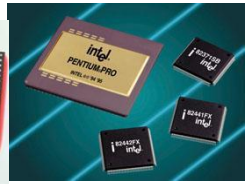
Intel 80486



Pentium

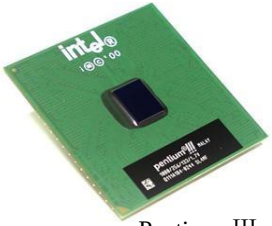


Pentium MMX



Pentium Pro

Pentium Ⅱ



Pentium Ⅲ

Pentium 4

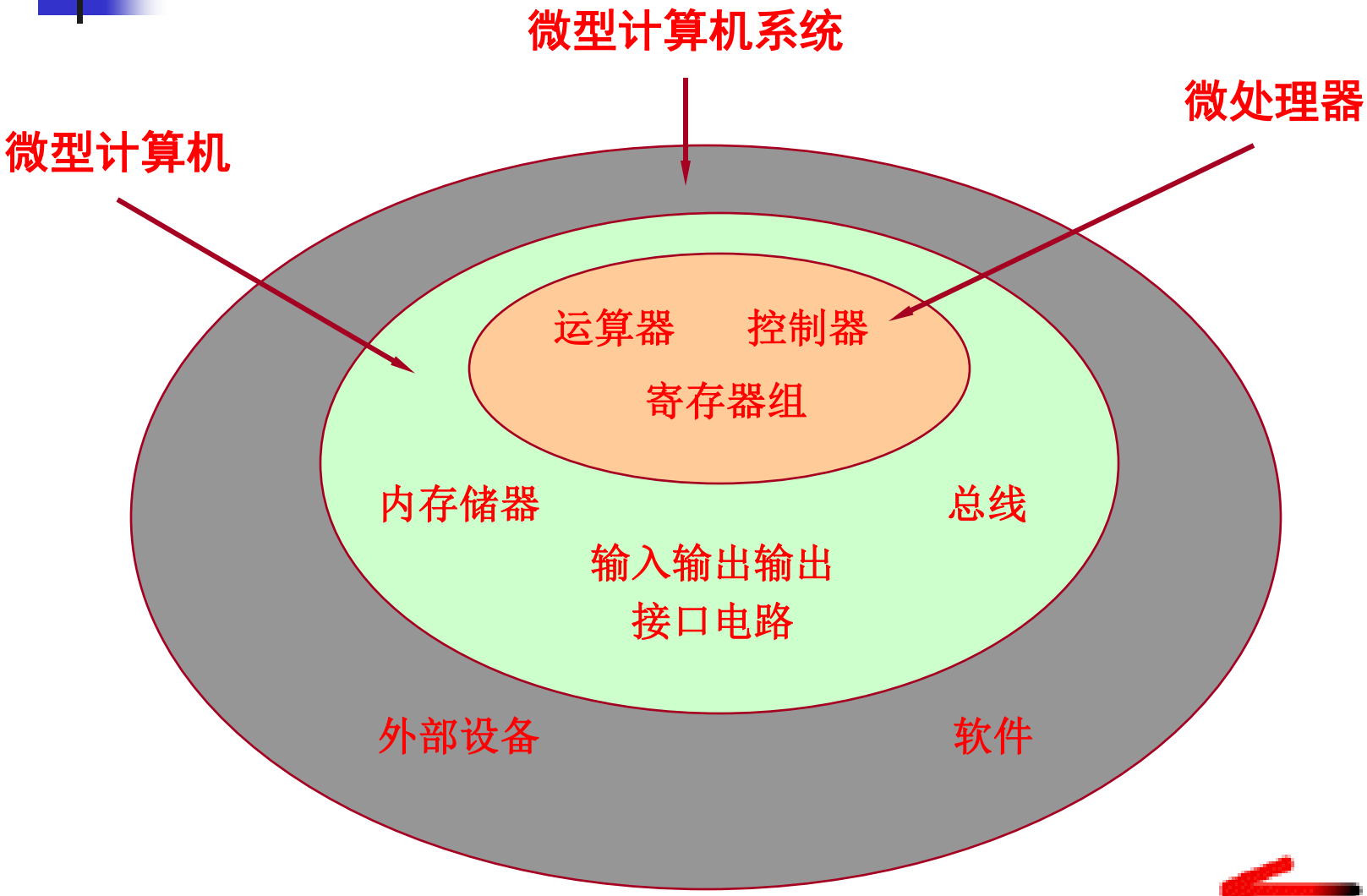
1.2.2 微型计算机的应用

- 用
  -
  -
- 用于过程控制及智能化仪
  - 专用微机，例如：单片机 工控机
  - 可靠性高、实时性强
  - 程序相对简单、处理数据量小
  - 嵌入式计算机应用系统

将**CPU**以及其他主要部件（如**ROM**、**RAM**、**I/O**接口）都集成在一个微处理器芯片中

例如：常用的**MCS-51**、**MCS-96**

1.3 微型计算机的系统组成



1.3.1 微型计算机的硬件组成

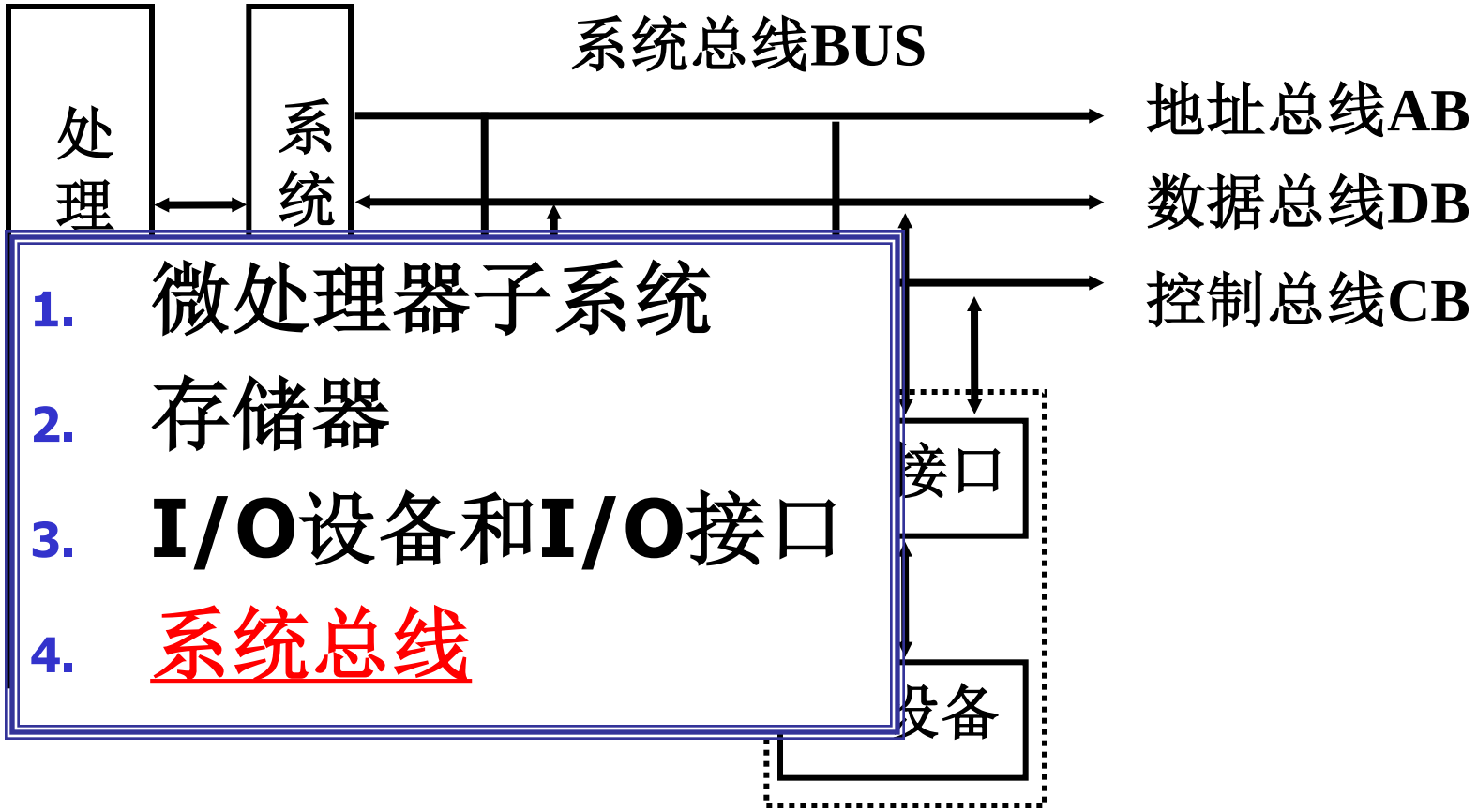


图1.1 微型计算机的系统组成

系统总线

- 总线是指传递信息的一组公用导线
- 总线是传送信息的公共通道
- 微机系统采用总线结构连接系统功能部件
- 总线信号可分成三组
  - 地址总线AB：传送地址信息
  - 数据总线DB：传送数据信息
  - 控制总线CB：传送控制信息

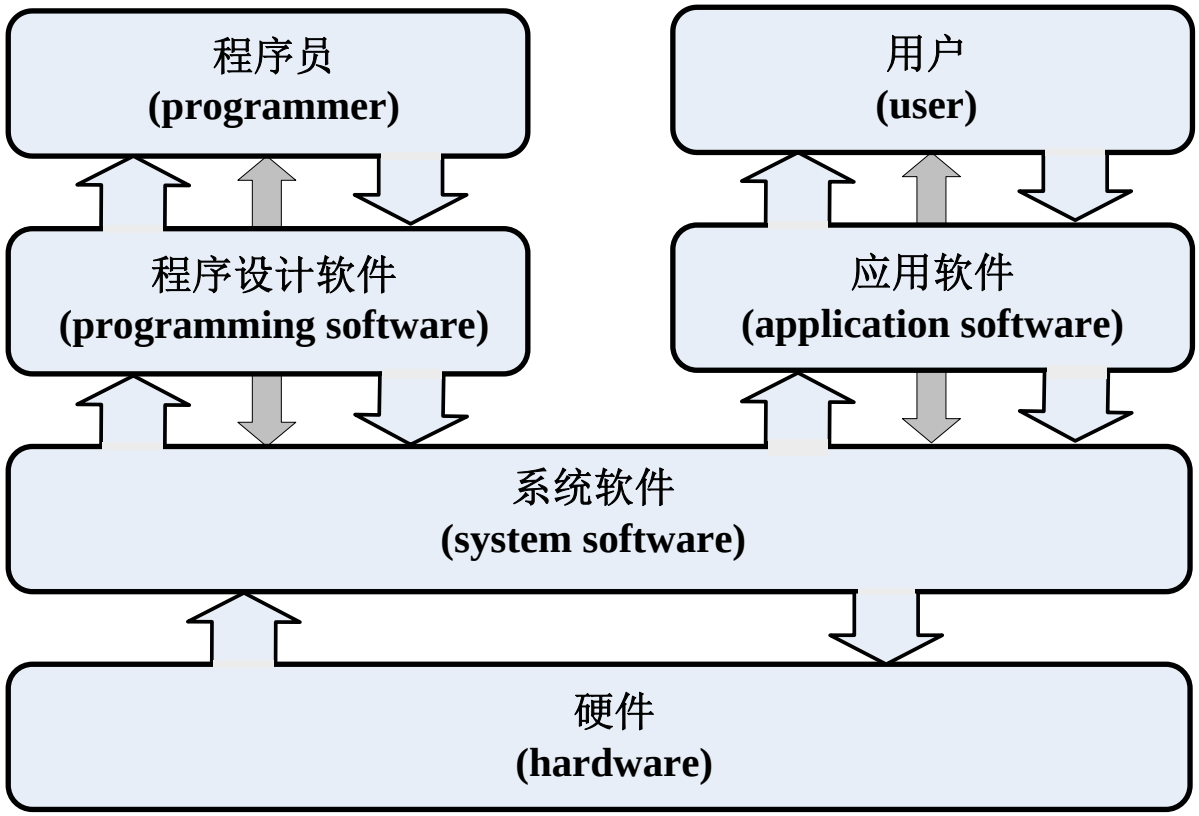
总线信号

- 地址总线AB
  - 输出将要访问的内存单元或I/O端口的地址
  - 地址线的多少决定了系统直接寻址存储器的范围
- 数据总线DB
  - CPU读操作时，外部数据通过数据总线送往CPU
  - CPU写操作时，CPU数据通过数据总线送往外部
  - 数据线的多少决定了一次能够传送数据的位数
- 控制总线CB
  - 协调系统中各部件的操作，有输出控制、输入状态等信号
  - 控制总线决定了系统总线的特点，例如功能、适应性等

1.3.2 微型计算机的软件系统

- **软件(software)**
  - 计算机硬件执行的一系列指令和相关数据
  - 杰出统计学家**John Tukey**在**1958**杜撰的术语——用来指计算机程序 **(computer program)**、程序**(program)**和代码**(code)**
- **软件系统(software systems)**
  - 软件的同义词，三种类型软件
    - 面向计算机的系统软件**(system software)**
    - 面向软件开发的程序设计软件**(programming software)**
    - 面向应用的应用软件**(application software)**

1.3.2 微型计算机的软件系统



软件分层结构

1.3.3 计算机语言

- 计算机语言(computer language)的概念
  - 可指从机器语言到高级语言的任何一种语言
  - 人造语言，用于定义在计算机上执行的指令系列
  - 计算机语言可分成五代
    - 第一代语言：机器语言
    - 第二代语言：汇编语言
    - 第三代语言：高级程序设计语言
    - 第四代语言：描述语言
    - 第五代语言：问题求解语言
- 程序设计语言(programming language)
  - 表达计算机程序的人造语言，用于定义指令序列
  - 编写的程序可被编译器等自动翻译成机器语言

1.3.3 程序设计语言

- **机器语言(machine language)**
  - 用数字表示指令的语言
    - 【例】“把数值9送到寄存器AX”，**10110000 00001001 (B)** 和 **B009 (H)**
    - 用数字表示指令的代码很容易被计算机理解、识别和执行，因此也称“机器码**(machine code)**”
  - 处理器装载和执行的语言
    - 由“**0**”和“**1**”组成连续序列
  - 由特定计算机的机器指令集组成
    - 由汇编语言或高级语言编写的程序经编译后得到
  - 人难读懂，容易出错，很难修改

1.3.3 程序设计语言

- 汇编语言(assembly language)
  - 20世纪50年代末的发明
  - 使用英文单词的缩写或称助记符号
    - 每条语句对应一条单独的机器指令
    - 通过“编译器” 翻译成机器语言
  - 与特定的处理器相关，称为“低级语言”
    - 执行速度快
    - 机器语言和汇编语言都属于低级语言
  - 可让程序员直接与系统硬件打交道

指令说明	汇编语言指令	机器语言指令	
		操作码	操作数
把数值9送到寄存器AX	MOV AX，9	10110000	00001001

1.3.3 程序设计语言

- 高级语言(high-level language)
  - 不针对特定计算机结构和操作系统的程序设计语言
  - 有点像“自然语言(natural language)”
    - 【例】求A、B、C的平均值X，用高级语言写成

      X = (A+B+C)/3
  - 在汇编语言之上的任何一种程序设计语言都是高级语言，如Basic，C和Java
  - 需用编译器翻译成计算机可执行的机器语言

1.3.3 程序设计语言

- **BASIC语言**
  - **Beginner‘s All purpose Symbolic Instruction Code**—初学者通用符号指令码
  - **20**世纪**60**年代中期由美国的**John Kemeny**和**Thomas Kurtz**两位计算机科学家开发的高级程序设计语言
  - 属于代数语言，指令集不大，并有交互功能，被认为是最容易学习的编程语言之一
- **VB(Visual Basic)语言**
  - 微软公司开发的**Basic**高级语言
    - 支持可视化编程，可用来开发在**Windows**使用的应用软件
  - **VB**语言是一种解释语言
    - 用**VB**解释程序可把**VB**语言编写的程序转换成机器码

1.3.3 程序设计语言

- **C语言**
  - 贝尔实验室的**Dennis Ritchie**于**1972**年开发的结构化程序设计语言，具有简洁、有效和适应性强等特点
  - **C**语言是编译语言，在它内置的函数集中，有一部分是与机器有关的函数，其余的函数是与机器无关的；
  - 应用场合
    - 由于**C**和**C++**的功能强大，但较复杂，适合用于编写系统软件和游戏软件
    - 认为**C**语言是独立于机器的汇编语言，而不是高级语言
    - 因与**UNIX**操作系统紧密结合、广泛流行以及美国 **ANSI**和**ISO**为它进行了标准化工作，到**20**世纪**80**年代后期，**C**语言已经成为个人计算机与工作站上的标准程序设计语言

1.3.3 程序设计语言

- **Java语言**
  - **Sun Microsystems开发的面向对象的程序设计语言**
  - **Java源于1991年，经改造后命名为Java**
  - **与C++语言类似，但比C++更小、更容易移植、更容易使用和更加安全，而且能自己管理内存**
  - **Java语言是编译和解释语言**
    - **Java源程序先被编译成字节码，然后在不同平台上由Java虚拟机解释执行**
  - **用Java开发的程序称为自运行小应用程序(applet)**
    - **允许开发人员创建、传送并可在客户机上运行**
    - **小应用程序可以是程序员构思的任何东西，从电子数据表格和教学材料到交互游戏和动画**

1.3.4 系统软件

- 系统软件(system software)
  - 控制计算机和相关设备运行的程序，或定义为操作系统及其相关的程序和数据的集合
  - 基本功能
    - 管理、控制和协调计算机系统中各个部件的工作
    - 支持各种应用程序的生成和运行，如字处理软件
  - 典型系统软件的组成
    - 操作系统
    - 设备驱动程序
    - 实用软件

1.3.4 系统软件

(1) 操作系统(operating system，OS)

- Windows、Mac OS X等都是卓越的操作系统
- 控制硬件资源的分配、使用和执行各种程序
  - 硬件资源通常是指内存、CPU、磁盘空间和其他外围设备
- 提供用户和计算机之间的接口
  - 称为人机接口(Human-machine interface，HMI)
- 所有应用程序的基础，计算机系统的核心软件

(2) 设备驱动程序(device driver)

- 计算机与硬件设备通信的程序
  - 操作系统与键盘、驱动器、监视器、打印机等之间的通信
- 每个操作系统中都包含一套标准的设备驱动程序
- 没有得到操作系统认证、支持或新开发的外围设备，需安装相应设备的驱动程序

1.3.4 系统软件

(3) 实用软件(utility software)

- 也称“实用程序(utility program)”、“服务程序(service program)”、“服务例程(service routine)”、“工具(tool)”或“实用例程(utility routine)”
- 为辅助和协调计算机硬件、操作系统或应用程序而设计的常用程序。例如
  - 磁盘实用程序(如磁盘分区、碎片整理和备份等)
  - 文件管理(如拷贝、删除、列表、重命名、检索和排序等)
- 大部分实用软件已集成到操作系统

1.3.5 应用软件

- 应用软件(application software)
  - 用来帮助用户完成某种类型的工作而开发的计算机程序
- 常见应用软件
  - 办公软件(office software)，如
    - Microsoft Office
  - 互联网软件(internet software)，如
    - 即时通信软件(instant messaging，MSN，QQ)
    - 网页浏览器(browser)
    - 电子邮件软件(email software)

1.3.5 应用软件

- 媒体软件(**media software**)，如
  - 图像编辑软件
  - 声音编辑软件
  - 影视编辑软件
  - 媒体播放器
- 数据库软件(**databases software**)，如
  - **Oracle database**
  - **SQL Server**
- 统计软件(**statistical software**)，如
  - 社会科学统计软件包(**Statistical Package for the Social Science, SPSS**)

1.3.5 应用软件

- 教育软件(**educational software**)
- 医用软件(**medical software**)
- 计算机游戏软件(**computer games software**)
- 计算机辅助设计软件(**CAD**)，用于辅助产品、工程和建筑物的设计

- 软件分类和列表
  - “华军软件园”的软件分类和列表(http://www.onlinedown.net/sort/)；
  - “天空软件站”的软件分类和列表(http://www.skycn.com/sort/soft_sort.html)
  - ……

1.3.2 微型计算机的汇编语言

操作系统	**MS-DOS**
汇编程序	**MASM和LINK**
文本编辑程序	**EDIT.COM**
调试程序	**DEBUG.EXE**

1.4 IBM PC系列机系统

16位IBM PC系列机是32位微机的基础

8088CPU

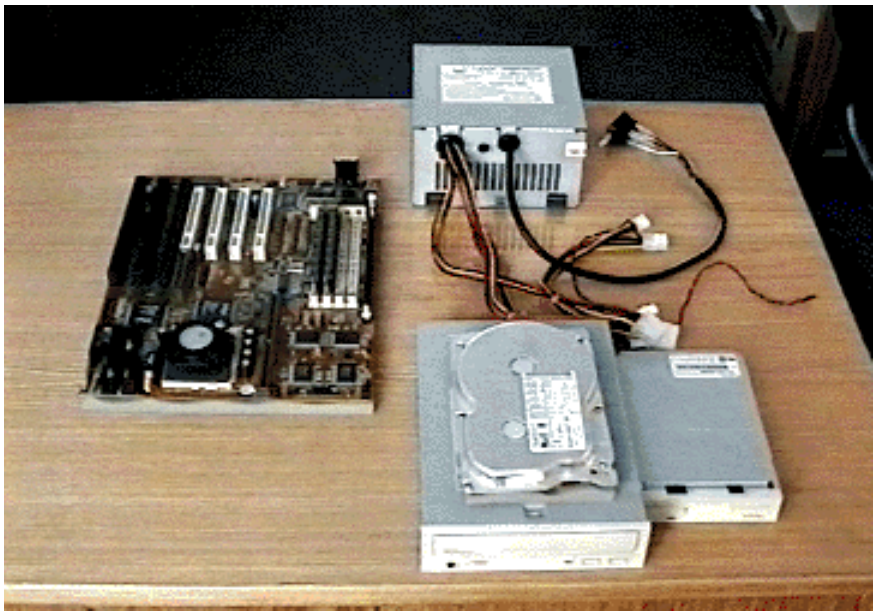
IBM PC机

IBM PC/XT机

IBM PC/AT机

1.4.1 硬件基本组成

16位和
32位PC机的基本部件相同



1.4.2 主机板组成

1. 微处理器子系统

8088：16位内部结构、8位数据总线、20位地址总线、4.77MHz主频

2. 存储器

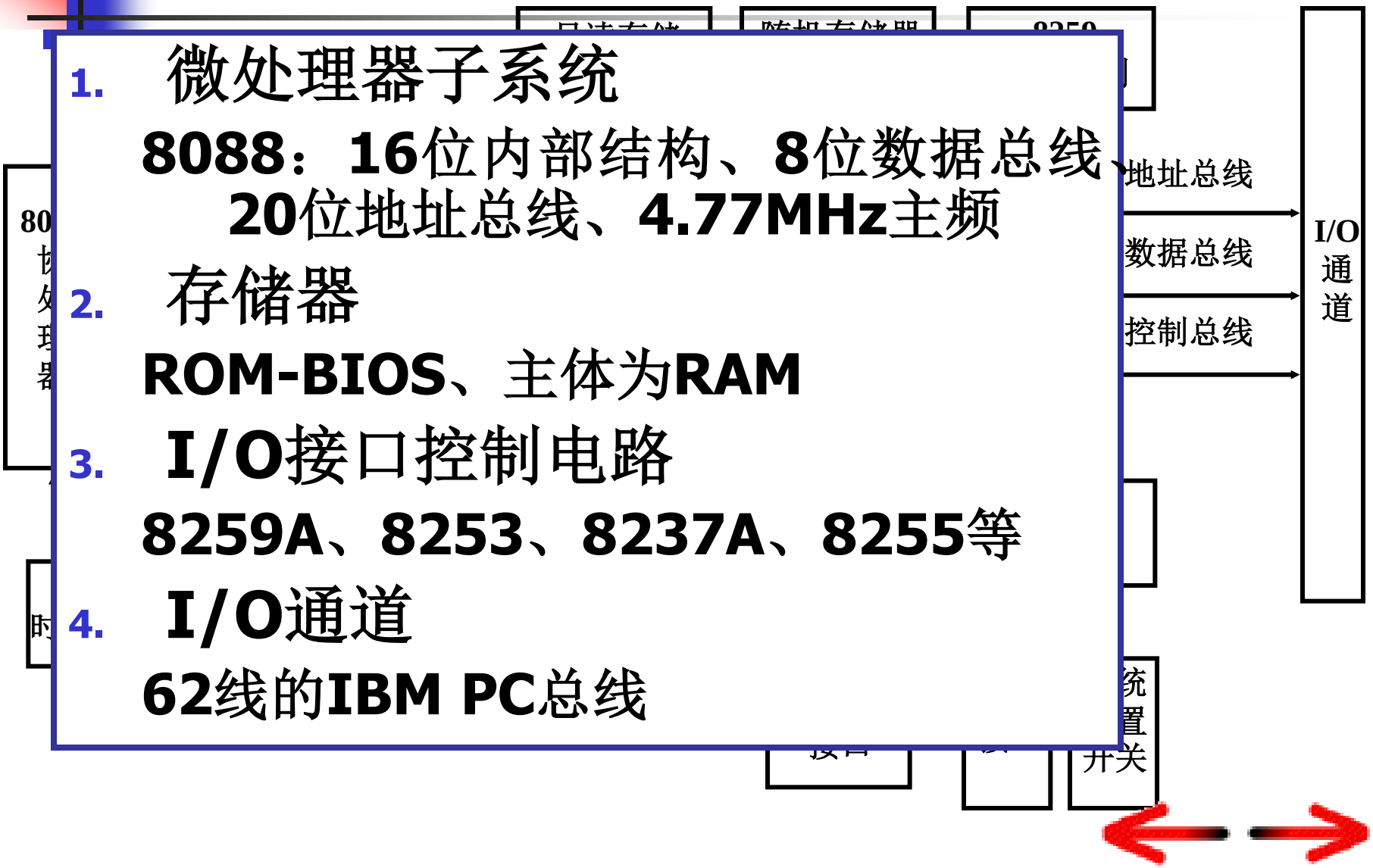
ROM-BIOS、主体为RAM

3. I/O接口控制电路

8259A、8253、8237A、8255等

4. I/O通道

62线的IBM PC总线



1.4.3 存储空间的分配

常规内存：1MB

- 基本RAM区：640KB
- 保留RAM区：128KB
- 扩展ROM区：128KB
- 基本ROM区：64KB

扩展内存：用作RAM区

图1.5 存储空间的分配

1.4.4 I/O空间的分配

系统板	000－01F	DMA控制器1
	020－03F	中断控制器1
	040－05F	定时计数器
I/O通道	278－27F	并行打印机接口LPT2
	2F8－2FF	串行通信接口COM2
	378－37F	并行打印机接口LPT1
	380－38F	SDLC通信接口
	3A0－3AF	BSC通信接口
	3B0－3BF	单色显示/打印机适配器
	3D0－3DF	彩色图形适配器CGA
	3F0－3F7	软盘适配器
	3F8－3FF	串行通信接口COM1

- 80x86访问外设时，只使用低16位A_0～A_{15}，寻址64K个8位I/O端口
- PC机仅使用低10位A_0～A_9，寻址1024个8位I/O端口

1.5 计算机中的数据表示

- 计算机处理的对象是各种数据，计算机中的数据均采用二进制形式。
- 从使用角度来看，计算机中的数据可分为两大类：

（1）数—用来直接表征量的多少，它们有大小之分，可进行各种数学运算。

（2）码—用来指代某个事物或事物的状态属性。计算机对码主要是做管理、编辑、判断、检索、转换、存储及传输等工作。

（详细内容见下一节）

第1章教学要求

1. 了解微机发展概况、熟悉典型微处理器和微机系统；

2. 明确微机两个应用方向、区别通用微机（PC机）和控制专用微机（单片机）；

3. 了解微机的硬件组成，理解总线及其应用特点、掌握地址、数据、控制总线的概念；

第1章教学要求（续）

4. 熟悉**PC**系列机的主机板、存储空间分配和**I/O**空间分配；

5. 了解微处理器基本结构、**8088/8086**的内部功能结构；

6. 复习汇编语言源程序格式。

习题**1**（第**18**页）——

1.2 1.4 1.5 1.8

计算机程序设计语言（VC++）

第 3 章

函　　数

张晓如，华伟

《C++程序设计基础教程》
人民邮电出版社，2018.05

本章内容

3.1 函数的概念和定义

3.1.1 函数的概念

- 函数是组成源程序的基本模块。
- 函数通过封装在一起的语句实现特定的功能。
- 函数可分为库函数和用户自定义函数两类。
  - 库函数由系统提供，用户只需要在程序编译预处理部分，包含相应的头文件便可直接使用；
  - 用户自定义函数通常要先定义，然后才能使用。

【例3-1】设计一个程序，求角度的正弦值。

- 库函数 sin 用来求正弦值，其参数为弧度；
- 自定义函数 f 将角度转化为弧度；
- 主函数中调用 f 函数和 sin 函数。

3.1 函数的概念和定义

【源程序代码】

```
#include<iostream>
#include<cmath>
using namespace std;
double f(double a) {
    double t;
    t=a*3.1415926/180;
    return t;
}
```

```
int main(){
    double a,r,s;
    cout<<"请输入一个角度： ";
    cin>>a;  r=f(a);
    cout<<"弧度： "<<r<<endl;
    s=sin(r);
    cout<<"正弦值： "<<s<<endl;
    return 0;
}
```

- sin 函数是数学库函数：
  - 在标准C++中应包含头文件**cmath** ；
  - 在VC6.0中应包含头文件**math.h**。
- 任何一个程序有且只能有一个主函数**main**。

3.1 函数的概念和定义

3.1.2 函数定义的基本形式

```
函数类型  函数名（形参列表）        // 函数头部
{
    语句序列                        函数体
}
```

- 函数名必须符合自定义标识符的规则。
- 函数体是用花括号括起来的语句系列，实现函数功能。
- 函数定义时的参数称为形参，在形参列表处给出。
  - 每个形参都必须包含参数类型和参数名称；
  - 函数有多个参数时，参数之间用逗号分隔；
  - 如果函数没有参数，则可以在形参列表处用**void** 表示；或者缺省，但圆括号不能省略。

3.1 函数的概念和定义

3.1.3 函数类型与返回值

1. 函数类型

- 函数运行结果（返回值）的数据类型。
- 无值型函数的函数类型为**void**，该类函数称为无值型函数，其运行结果不是一个具体的数据。
- 除void 类型以外的其他类型的函数统称为有值型函数，该类函数的运行结果为某类型的值。如函数类型为：
  - **float** 时，函数运行结果为一实数；
  - **char*** 时，函数运行结果为一字符型指针（地址）；
  - **int** 时，函数运行结果为一整数；
  - 函数的缺省类型 为 **int** 类型，即定义函数时，若未指明函数类型，则该函数的类型为 **int** 。

3.1 函数的概念和定义

3.1.3 函数类型与返回值

2. return 语句

- 结束函数的运行；若函数为主函数，则结束程序运行。
- 返回有值型函数的运行结果。
- return 语句的格式必须与函数类型一致。

（1）无值型函数

```
return ;                    // 函数运行结束，不返回值
```

（2）有值型函数

- return 表达式；// 函数运行结束,并返回表达式的值

或

- return （表达式）； // 表达式的值即函数运行结果

3.1 函数的概念和定义

3.1.3 函数类型与返回值

【例3-2】设计程序求整数的阶乘，要求输出阶乘的数学表达式和阶乘的值。

- 定义函数fac 求整数n 的阶乘，并返回其值；fac 函数的函数类型为int，具有一个整型参数（传递整数）。
- 定义函数print 输出阶乘的数学表达式，即输出：
  n！=1*2*3*…*（n-1）*n=
  print函数无返回值，其函数类型为void。
- 在主函数中调用print 函数输出阶乘表达式；调用fac 函数求阶乘，并将其运行结果（函数的返回值）输出。

3.1 函数的概念和定义

【源程序代码】

```
#include<iostream>
using namespace std;
int fac(int n){
    int t=1;
    for(int i=2;i<=n;i++)
        t*=i;
    return t;
}
```

```
int main(){
    int num;
    cout<<"请输入一个整数： ";
    cin>>num;
    print(num);                  // A
    cout<<fac(num)<< endl;  // B
    return 0;
}
```

```
void print(int n){
    cout<<n<<"!=";
    if(n>1)
        for(int i=1;i<=n;i++)
            if(i==n) cout<<i<<'=';
            else cout<<i<<'*';
}
```

程序运行结果

请输入一个正整数：4

4!=1*2*3*4=24

注意：

- A行为无值型函数的调用。
- B行为有值型函数的调用。

3.1 函数的概念和定义

3.1.3 函数类型与返回值

◆ 分析下列函数的定义

```
void f1(){return 0;}
float f2(int x){ cout<<x<<endl;}
char *f3(char *s){return s;} // *s 或 &s皆为语法错误
f4(float x,float y){ if(x>y) return x; return y; }
```

- 一个函数中可以有一条或者多条return 语句，但只能有一条return 语句被执行，即只能返回一个值；
- 函数类型应该与return 后面表达式的数据类型相同；若不同，则将return 后面的表达式转换成函数类型指定的数据类型后返回，如f4(3.5,5.5)的运行结果为5；若无法转换则报语法错误。

3.2 函数的调用

3.2.1 函数调用的基本形式

函数名(实参列表)

- 调用函数时，函数名前没有类型。
- 函数调用时的参数称为实际参数，简称实参。
  - 实参只有名称，没有类型；
  - 多个参数时，各实参之间用逗号隔开；
  - 实参可以是变量、常量、表达式或函数调用表达式；
  - 每个实参通常应有确定的值；
  - 实参与形参的个数、类型和顺序通常应该保证一致；
- 调用无参函数时，无需提供参数，但“（）”不能少。
- 注意有值型函数与无值型函数的调用方式的区别。

3.2 函数的调用

3.2.1 函数调用的基本形式

◆ 分析下列函数的调用

```
double f1(int x,int y){ return x+y; }
void f2(float x,float y){ cout<<x+y<<endl; }
int main(){
    cout<<f1(3.5,5.5)<<endl;        // A有值型函数的调用方式
    f2 (3.5,5.5);                   // B无值型函数的调用方式
}
```

- A 行的运行结果是输出实数8，而不是9。
- 有值型函数的调用通常作为表达式的一部分，如A行；若写成“f1(3.5,5.5);”，虽然不是语法错误，但没有实际意义。
- 无值型函数的调用只能作为一条单独语句，不能参与输出等运算；若将B 行写成“cout<<f2(3.5,5.5)<<endl;”，则是语法错误。

3.2 函数的调用

3.2.1 函数调用的基本形式

【例3-3】 设计程序求3 个整数的最大公约数。

- 穷举法：从所给整数中的最小者开始，依次向下遍历，直到找到最大公约数为止。
- 辗转相除法（欧几里德算法）：不断地用两数相除得到的余数去除除数，直到余数为0时，除数即为最大公约数。

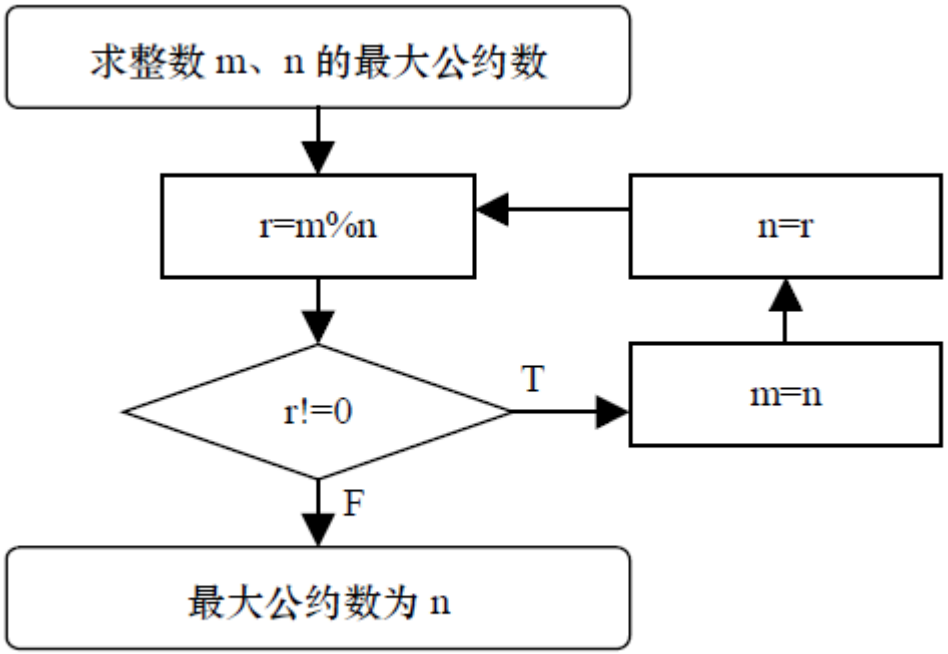


图 3-1 欧几里德算法

3.2 函数的调用

3.2.1 函数调用的基本形式

【例3-3】 设计程序求3 个整数的最大公约数。

- 欧几里德算法只能直接求两个整数的最大公约数。
- 定义gcd 函数求两个整数m 和n 的最大公约数，即gcd 函数有两个整型形参m 和n。
- 求3 个整数a、b、c 的最大公约数时：
  - 首先调用gcd 函数求a 和b 的最大公约数g；
  - 再调用gcd 函数，求出g 和c 的最大公约数；
  - 则可求出3 个整数的最大公约数。

3.2 函数的调用

【源程序代码】

```
#include<iostream>
using namespace std;
int gcd(int m,int n){
    int r;
    while(r=m%n){  // A
    m=n;
    n=r;
}
```

```
int main(){
    int a,b,c,g;
    cout<<"输入3 个正整数： ";
    cin>>a>>b>>c;
    g=gcd(a,b);          // B
    g=gcd(g,c);          // C
    cout<<g<<endl;    // D
    return 0;
}
```

- A行将m 和n 相除得到的余数r作为循环条件。
- B行和C行可用一条调用语句实现，即“g=gcd(gcd(a,b),c);”；此时，程序首先执行内层的函数调用gcd(a,b)。
- B行、C行和D行可用一条语句 “cout<<gcd(gcd(a,b),c)<<endl ;”实现。

3.2 函数的调用

3.2.2 函数的嵌套调用

- 函数不允许嵌套定义，即不能在函数体内再定义函数。
- 函数可以嵌套调用，即在调用函数的过程中再调用函数。

【例3-4】设计程序求代数式$1^k+2^k+……+n^k$ 的值，其中k 为整数。

- 该程序求的是n 项的和，而每项的值为i^k。
- 设计函数sum求n 项的和，设计函数powers 求i^k。
- 主函数在调用sum 函数的过程中再调用powers 函数，这属于函数的嵌套调用。

3.2 函数的调用

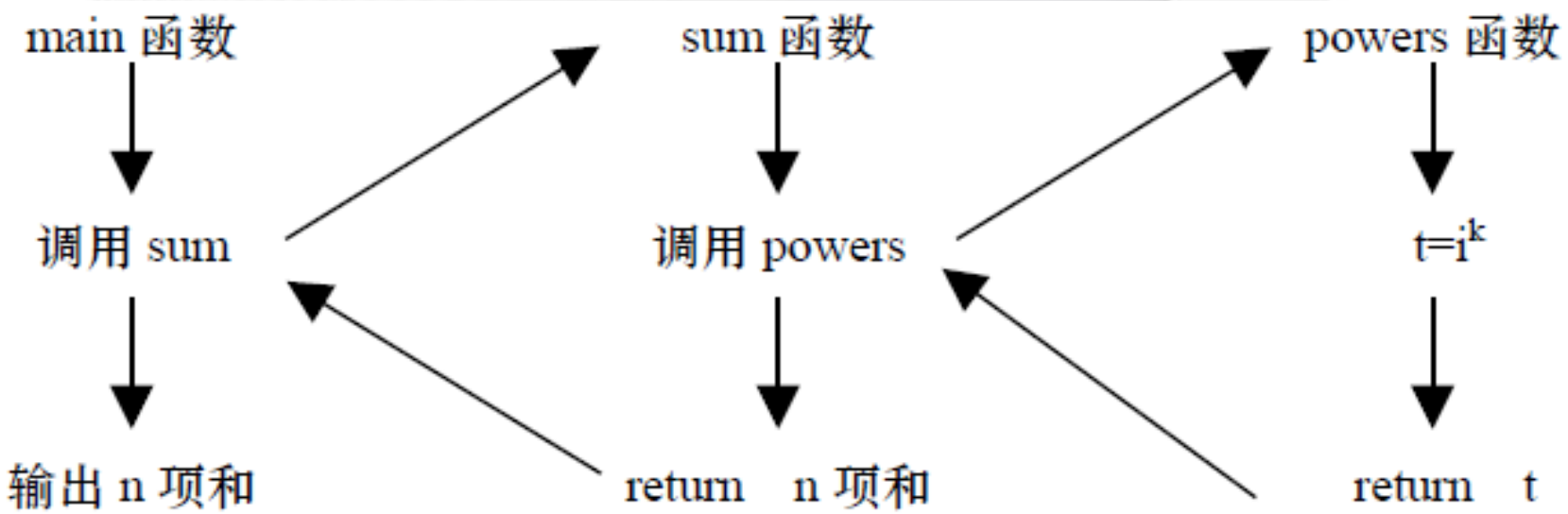


图 3-2 函数调用过程

- 函数powers 的函数类型为int，有两个整型参数i 和k；
- 函数sum的函数类型为int，有两个整型参数n 和k。

3.2 函数的调用

【源程序代码】

```
#include<iostream>
using namespace std;
int powers(int i,int k) {
    int t=1,j;
    for(j=1;j<=k;j++) t*=i;
    return t;
}                          // 累乘求i k
```

```
int sum(int n,int k) {
    int s=0;
    for(int i=1;i<=n;i++)
        s+=powers(i,k);
    return s;
}              // 累加求n 项的和
```

```
int main(){
    int n,k;  cout<<"Input n and k: ";   cin>>n>>k;
    cout<<"Sum of "<<k<<" powers from 1 to "<<n<<" = ";
    cout<<sum(n,k)<<endl;
    return 0;
}
```

3.2 函数的调用

3.2.3 函数的递归调用

- 在函数的调用过程中直接或间接地调用自身函数。

1. 直接递归

- 在一个函数的函数体内直接调用自身函数。

【例3-5】编程用递归法求n!。

◆ n!的递归定义：

$$n!=\begin{cases}1 & n=0\text{或}n=1\\ n\times(n-1)! & n>1\end{cases}$$

◆ 设计函数long f (int n)求n!。

➢ 函数体中通过调用语句f(n-1)求(n-1)!，得n* f(n-1)；

➢ f(n-1)的值为(n-1)*f(n-2)；以此类推，直至f(1)为1。

3.2 函数的调用

【源程序代码】

```
#include<iostream>
using namespace std;
long f(int n) {                          // 求n!的递归函数
    if(n==1||n==0)return 1;                // 递归结束条件
    else return n*f(n-1);                      // 递归公式
}
```

```
int main(){
    int n;
    cout<<"请输入一个正整数： "; cin>>n;
    cout<<n<<"!="<<f(n)<<endl;
    return 0;
}
```

3.2 函数的调用

3.2.3 函数的递归调用

- 递归算法是将原问题转换为用同样方法解决的规模更小的问题，其关键有两点：
  - 递归公式：原问题与新问题之间的关系；
  - 结束条件：转换到何时结束。
- 例3-5中，原问题是求n!，新问题是求(n-1)！，其关系是n!=n×(n-1)!，即f(n)=n*f(n-1)。
- 例3-5中，有两个递归结束条件，即n为1或n为0，其阶乘为确定值1，此时无需再递归调用。
- 递归算法通常可用下列范式实现：

  if(递归结束条件) 确定操作；
  else 递归公式;

3.2 函数的调用

3.2.3 函数的递归调用

- 递归调用的过程通常可分为递推和回归两个阶段
  - 递推：将原问题不断分解为新问题，逐渐从未知结果向已知方向推测的过程，最终到达已知条件，即递归结束条件；
  - 回归：从已知的条件出发，按照递推的逆过程，逐一求值回归，最后到达递推的开始处。
- 以递归函数中的递归语句为界，递归语句之前的语句在递推阶段执行，递归语句之后的语句在回归阶段执行，且执行时的顺序是相反的。如例3-5中参数n值的变化：
  - 递推：n→n-1→n-2→… →2→1；
  - 回归：1→2→…→ n-2→n-1→n 。

3.2 函数的调用

3.2.3 函数的递归调用

- 分析下列函数的调用过程

```
long f(int n){
    int t;
    cout<<n<<"!="<<n<<'*'<<n-1<<"!\n";
    if(n==1||n==0)t=1;
    else t=n*f(n-1);
    cout<<n<<"!="<<t<<'\n';
    return t;
}
int main(){  cout<<f(4)<<endl; }
```

3.2 函数的调用

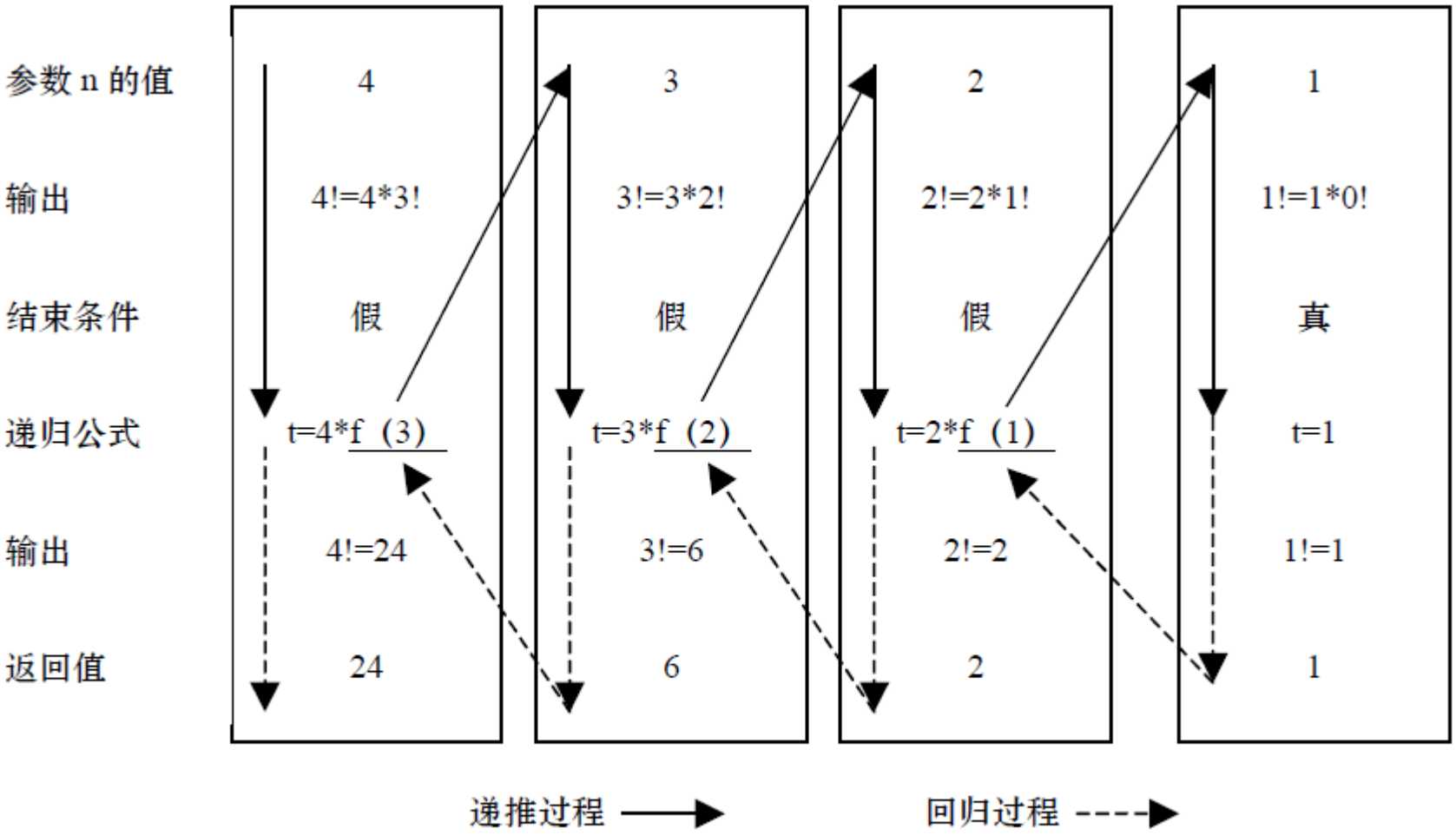


图 3-3　f(4)的递归调用过程

3.2 函数的调用

3.2.3 函数的递归调用

2. 间接递归

- 在一个函数的函数体内调用其他函数，而其他函数再调用该函数。

【例3-6】阅读程序，指出程序的输出结果。

```
#include <iostream>
using namespace std;
void f1(int);   // f1函数原型说明
void f2(int);   // f2 函数原型说明
int main(){  f1(123456);  cout<<endl;  return 0;  }
```

3.2 函数的调用

【源程序代码】

```
void f1(int n){
    cout<<n%10;
    if(n>0)f2(n/10);
}
```

```
void f2(int n) {
    if(n>0)f1(n/10);
    cout<<n%10;
}
```

递归调用过程

f1：n=123456，输出6

↓

f2：n=12345 (回归时输出5)

↓

f1：n=1234，输出4

↓

f2：n=123 (回归时输出3)

↓

f1：n=12，输出2

↓

f2：n=1(回归时输出1)

↓

f1：n=0，输出0，进入回归阶段，依次输出1、3、5

程序运行结果：6420135

3.3 函数的参数传递

函数的参数用于在主调函数与被调函数之间传递数据，根据所传递的数据形式，可以将C++语言中的参数传递方式分为值传递、地址传递和引用传递3 种类型。

3.3.1 函数的值传递

- 形参为基本类型变量、结构体类型变量和类类型变量等。
- 实参为相应的变量、常量或表达式等。
- 值传递是一种单向传递，即只能把实参传递给形参，对形参的操作不能改变实参的值。

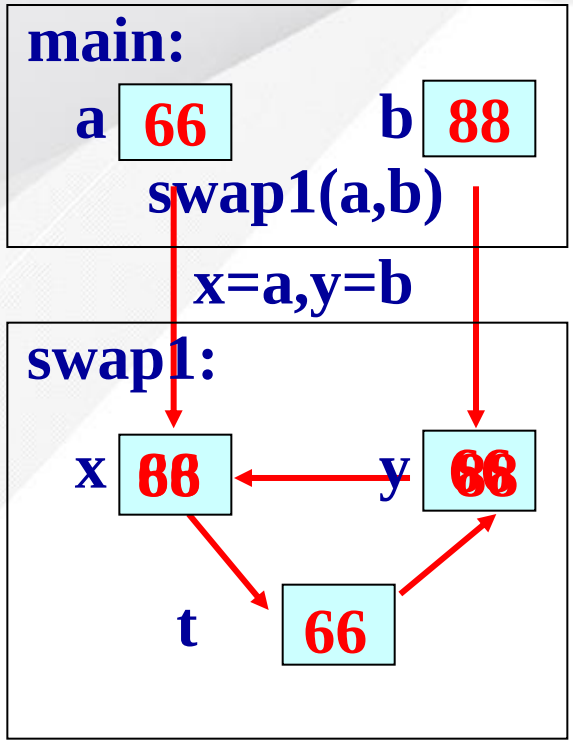
【例3-7】 分析下列程序的输出结果。

3.3 函数的参数传递

【源程序代码】

```
void swap1(int x, int y) {
    int t;
    t=x; x=y; y=t;
    cout<<x<<','<<y<<'\n';
}
```

```
int main(void){
    int a=66,b=88;
    cout<<a<<','<<b<<'\n';
    swap1(a,b);
    cout<<a<<','<<b<<'\n';
    return 0;
}
```



程序运行结果

66, 88
88, 66
66, 88

3.3 函数的参数传递

3.3.2 函数的地址传递

- 形参为某种类型的指针。
- 实参为相应的地址：
  - 变量的地址；
  - 保存了某个地址的指针变量。
- 被调用函数中：
  - 既可以操作指针；
  - 也可以操作指针所指的内存空间。
- 地址传递可以改变实参的值。

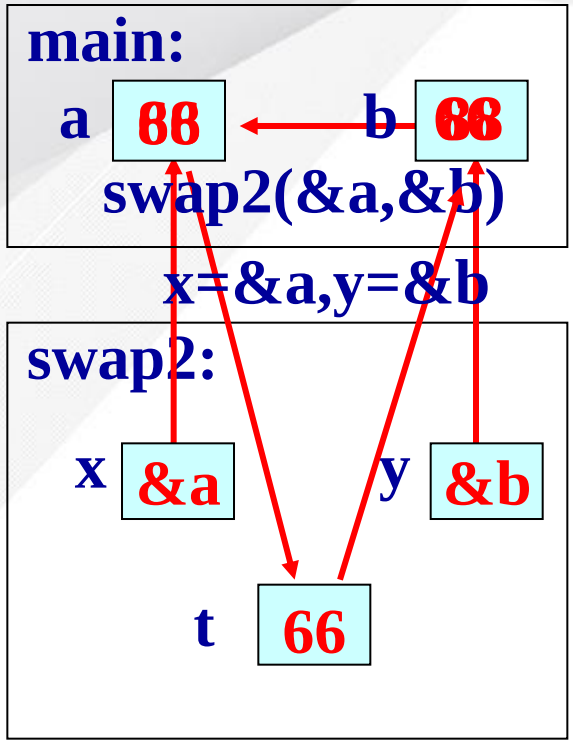
【例3-8】 分析下列程序的输出结果。

3.3 函数的参数传递

【源程序代码】（交换*x和*y）

```
void swap2(int *x, int *y) {
    int t;
    t=*x; *x=*y; *y=t;
    cout<<*x<<','<<*y<<'\n';
}
```

```
int main(void){
    int a=66,b=88;
    cout<<a<<','<<b<<'\n';
    swap2(&a,&b);
    cout<<a<<','<<b<<'\n';
    return 0;
}
```



程序运行结果

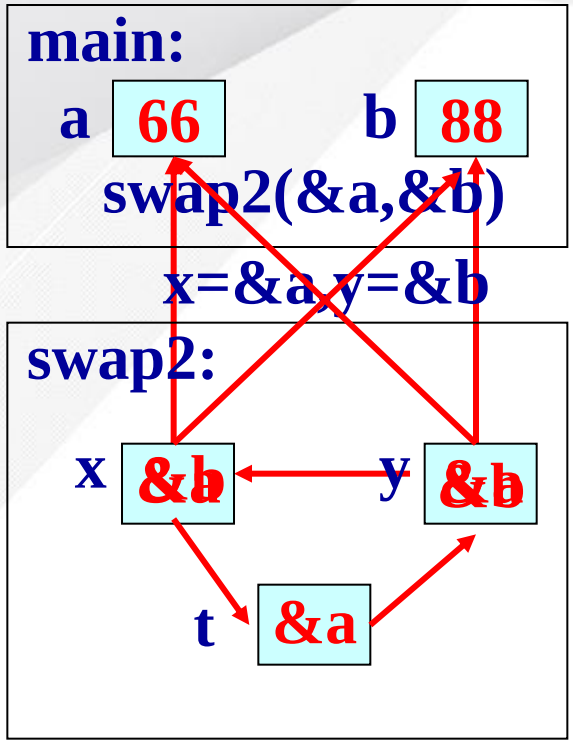
66, 88
88, 66
88, 66

3.3 函数的参数传递

【修改swap2函数】 （交换x和y）

```
void swap2(int *x, int *y) {
    int *t;
    t=x; x=y; y=t;
    cout<<*x<<','<<*y<<'\n';
}
```

```
int main(void){
    int a=66,b=88;
    cout<<a<<','<<b<<'\n';
    swap2(&a,&b);
    cout<<a<<','<<b<<'\n';
    return 0;
}
```



程序运行结果

66, 88
88, 66
66, 88

3.3 函数的参数传递

3.3.3 函数的引用传递

- 形参为某种类型的引用。
- 实参为相应的变量。
- 引用传递：
  - 形参是对实参的重新命名；
  - 形参和实参是同一个内存空间的两个名称；
  - 对形参的操作就是对实参的操作。

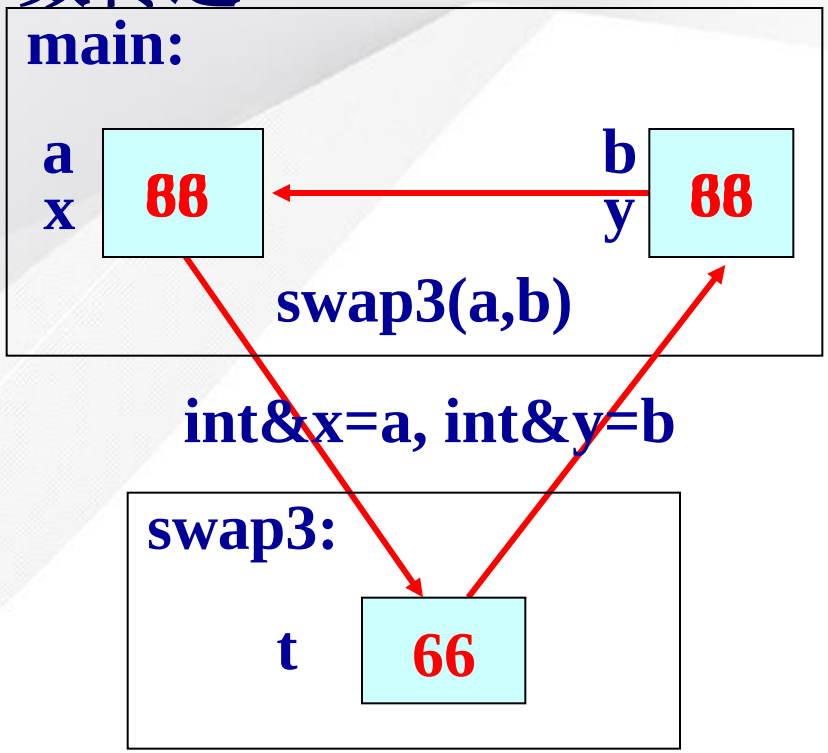
【例3-9】 分析下列程序的输出结果。

3.3 函数的参数传递

【源程序代码】

```
void swap3(int &x, int &y) {
    int t;
    t=x; x=y; y=t;
    cout<<x<<','<<y<<'\n';
}
```

```
int main(void){
    int a=66,b=88;
    cout<<a<<','<<b<<'\n';
    swap3(a,b);
    cout<<a<<','<<b<<'\n';
    return 0;
}
```



程序运行结果

```
66,  88
88,  66
88,  66
```

3.3 函数的参数传递

3.3.3 函数的引用传递

- 指针传递时，形参前的运算符*是指针运算符，表示形参是一个指针变量；其对应的实参必须是相同类型的地址。
- 引用传递时，形参前的运算符&是引用运算符，不是取地址运算符，表示形参是一个引用变量；其对应的实参必须是相同类型的变量，不能是地址。
- 三种传递方式：
  - 值传递一定不会改变实参的值；
  - 引用传递一定会改变实参的值；
  - 地址传递可能会改变实参的值。
- 若通过参数带回操作结果，应该选用引用传递或地址传递；且地址传递时，必须通过取值运算操作内存空间。

3.4 函数的其他特性

3.4.1 函数参数的默认值

- 给自定义函数的参数指定一个默认值。
  - 在定义函数时，列出参数的默认值；
  - 在说明函数原型时，列出参数的默认值；
  - 具有默认值的参数都必须位于形参列表的右侧。
- 调用具有默认参数的函数时：
  - 若提供了实参，则以提供的实参调用函数；
  - 若没有提供实参，则以默认值作为实参调用函数；
  - 提供的实参个数不能少于没有默认值的参数个数。

【例3-10】 分析下列程序的输出结果。

3.4 函数的其他特性

【源程序代码】

```
#include <iostream>
using namespace std;
void f(double x=9.9) {  //A
    cout<<"x="<<x<<endl;
}
```

```
int main(void){
    f(8.8);   // B
    f();      // C,等同于f(9.9)
    void f(double=7.7);  // D
    f(6.6);   // E
    f();      // F,等同于f(7.7)
    return 0;
}
```

程序运行结果

```
x=8.8    // B行输出
x=9.9    // C行输出
x=6.6    // E行输出
x=7.7    // F行输出
```

在不同的作用域内，可以设置不同的默认值；调用时默认值由最近的定义或说明决定。

3.4 函数的其他特性

3.4.2 函数的重载

- 函数重载是指一组名称相同，参数不同的函数。
  参数个数、或参数类型、或参数次序不同。
- 函数类型不同不能作为函数重载的依据。
- 调用重载函数时，根据提供的实参确定调用哪个函数。

【例3-11】定义重载函数，分别求解三角形、矩形和圆的面积。

- 函数名称皆为area、函数类型都为double ；
- 函数参数不同：三角形为double x, double y, double z；矩形为double x, double y；圆为 double x。

3.4 函数的其他特性

【源程序代码】

```
double area(double x, double y, double z) {          // A
    double s=(x+y+z)/2;
    return sqrt(s*(s-x)*(s-y)*(s-z));
}
double area(double x, double y) {  return x*y;  }   // B
double area(double x) {  return PI*x*x;  }           // C
```

```
int main(){
    cout<<area(3)<<endl;        // 调用C行1个参数的area函数
    cout<<area(3,4)<<endl;      // 调用B行2个参数的area函数
    cout<<area(3,4,5)<<endl;    // 调用A行3个参数的area函数
    return 0;
}
```

3.4 函数的其他特性

3.4.2 函数的重载

- 在运用重载函数时，要避免调用的二义性。

```
void f(int,float=5.5);    // A
void f(int);              // B
void fun(float);          // C
void fun(double);         // D
f(1);                     // E
fun(1);                   // F
```

A行和B行的f函数、C行和D行的fun函数都符合函数重载规则，但E 行调用f 函数和F 行调用fun 函数时，都不能确定调用哪个函数。

3.4 函数的其他特性

3.4.3 内联函数

- 将简单函数定义为内联函数，可在编译时将函数代码直接插入调用处，以提高程序的运行效率。
- 内联函数定义的一般格式

  inline 函数类型 函数名（形参列表）

  {

  语句序列

  }
- 内联函数定义时：
  - 内联函数仅限于简单的函数，函数体内不应包含循环语句、switch 分支语句和复杂嵌套的if 语句。
  - 用户指定的内联函数，系统不一定处理为内联函数。

3.4 函数的其他特性

3.4.4 exit 函数和abort 函数

- 库函数，头文件<cstdlib>；
- 功能是终止程序的执行。

1. exit函数的一般格式

exit（表达式）；

2. abort函数的一般格式

abort（）；

3.4 函数的其他特性

3.4.5 指向函数的指针

1. 定义指向函数的指针

   函数类型 (*指针变量名)(形参列表);

   - 函数类型和形参列表必须与所指向的函数相同;
   - 必须用括号将指针变量名括起来。

2. 指针指向函数

   指针变量名=函数名;

3. 指针调用函数

   指针变量名(实参列表);

   或

   (*指针变量名)(实参列表);

3.4 函数的其他特性

3.4.5 指向函数的指针

【例3-12】设计一个程序，执行两个实数的加、减、乘、除运算。

- 定义4 个函数Add、Sub、Mul 和Div 分别实现两个双精度数的加、减、乘、除运算。
- 上述4 个函数的函数类型都为double，且都有两个double型的参数，故可以定义一个指向函数的指针操作它们：

  double(*fp)(double, double);
- 根据键盘读入的运算符，使指针指向相应的函数，然后调用函数完成运算。

3.4 函数的其他特性

【源程序代码】

```
#include <iostream>
using namespace std;
double Add(double x, double y){
    return x+y;
}
double Sub(double x, double y){
    return x-y;
}
double Mul(double x, double y){
    return x*y;
}
double Div(double x, double y){
    return x/y;
}
```

3.4 函数的其他特性

【源程序代码】

```
int main(){
    double num1,num2;
    char op;
    double(*fp)(double,double);
    cout<<“请输入表达式（操作数1 运行符 操作数 2）： ";
    cin>>num1>>op>>num2;
    switch(op){
        case '+':fp=Add;break;
        case '-':fp=Sub;break;
        case '*':fp=Mul;break;
        case '/':if(num2){ fp=Div; break; }
                 else { cout<<"除数不能为0！ \n";exit(2); }
        default:cout<<"输入错误！ \n";exit(1);   //A
    }
    cout<<num1<<op<<num2<<'='<<fp(num1,num2)<<endl;
    return 0;
}
```

3.5 编译预处理

编译预处理：进行编译之前，对编译预处理指令所做的加工处理工作，包括文件包含、宏定义和条件编译。

- 一律由符号#开头。
- 以回车符结束。
- 通常每一条预处理指令占一行。

3.5.1 文件包含

将预处理指令指定的头文件包含到当前源程序中。

#include <文件名>

或

#include“文件名”

3.5 编译预处理

3.5.2 宏定义

- 宏定义：用一个标识符(宏名)来代替一个字符序列(宏内容)，字符序列可以是字符串、常量或表达式等。
- 宏调用（宏代换、宏扩展）：将宏名替换为字符序列。
- 宏定义有无参宏和有参宏两种形式。

1. 无参宏

#define 宏名 字符序列

- 宏扩展时只做简单的替换，不做计算。
- 字符串中与宏名相同的字符序列不作为宏对待。
- 若终止宏的运行，可使用#undef 命令:

#undef 宏名

3.5 编译预处理

【例3-13】 分析下列程序的输出结果。

```
#define M  (i+i)
#define N  i+i
#define CHINA  "People's Republic of China"
#define China  "CHINA“
int main (void){
    int x,y,i=1;
    x=2*M+3*M;
    y=2*N+3*N;
    cout<<"x="<<x<<endl;
    cout<<"y="<<y<<endl;
    cout<<CHINA<<endl;
    return 0;
}
```

```
x=2*M+3*M
 =2*（i+i）+3*（i+i）
y=2*N+3*N
 =2*i+i+3*i+i
```

程序运行结果

```
x=10
y=7
People’s Republic of China
```

3.5 编译预处理

3.5.2 宏定义

2. 有参宏

- 定义格式

  #define 宏名(形参列表) 字符序列

- 调用格式

  宏名(实参列表)

```
#define MAX(x,y) x*y
int a=5,b=3,c;
c=MAX(a-b,a+b);
```

```
c=MAX(a-b,a+b)
 = x*y
 =a-b*a+b
 =-7
```

宏定义时的参数为形式参数，宏调用时的参数为实际参数。

（1）定义时，宏名和形参列表之间不能有空格。

（2）形参和实参都只有名称而没有类型。

（3）函数调用是参数传递，而宏调用只做简单替换。

3.5 编译预处理

3.5.3 条件编译

将宏是否定义作为编译的条件，对程序进行编译。有两种形式：

```
#ifdef 宏名
    程序段
#endif
```

或

```
#ifdef 宏名
    程序段 1
#else
    程序段 2
#endif
```

3.5 编译预处理

【例3-14】 分析下列程序的输出结果。

- 宏A没有定义,不编译A行;
- B行不在条件编译范围内;
- 宏B已定义,编译C行和D行;不编译E行。

程序运行结果

2

3 4

```
#define B
int main(){
    #ifdef A
        cout<<1<<'\t';    // A
    #endif
        cout<<2<<endl;  // B
    #ifdef B
        cout<<3<<'\t';    // C
        cout<<4<<'\t';    // D
    #else
        cout<<5<<endl;    //E
    #endif
        cout<<endl;
}
```

3.6 变量的作用域与存储类型

3.6.1 变量的作用域

- 变量的作用域是指变量的有效使用范围，包括块作用域、文件作用域、函数原型作用域、函数作用域和类作用域。
- 按作用域的不同，可将变量分为局部变量（块作用域）和全局变量（文件作用域）两类 。

1. 块作用域

- 块内定义的变量具有块作用域，只能在该语句块内使用。
- 同一个块内不允许定义名称相同的变量。
- 函数体是一个块，函数形参的作用域为函数体。

【例3-15】 分析下列程序的输出结果。

3.6 变量的作用域与存储类型

【源程序代码】

```
#include <iostream>
using namespace std;
void f(int);
int main( ){
    int x=1;                    // A
    {
        int x=2;                // B
        cout<<x<<'\t';          // C
    }
    f(x);                       // D
    return 0;
}
```

```
void f(int a){
    int b=3;                    // E
    {
        int c=b++;              // F
        cout<<c<<'\t';
    }
    cout<<a<<'\t';
    cout<<b<<endl;              // G
}
```

程序运行结果

2 3 1 4

3.6 变量的作用域与存储类型

3.6.1 变量的作用域

- 在for 语句表达式1 位置说明的变量，表达式3 位置使用的变量，其作用域为包含for 语句的块。
- 循环体中定义的变量，其作用域为循环体。

```
void f(){
    for(int i=0; i<3;i++)
        for (int j=0; j<3;j++)
            cout<<i*j;                          // 输出000012024
    for(int i=0; i<3;i++)     // 语法错误，变量i重复定义
        for (j=0; j<3;j++)    // 语法错误，变量j没有定义
            cout<<i*j;
}
```

3.6 变量的作用域与存储类型

3.6.1 变量的作用域

2. 文件作用域

- 函数外定义的变量是全局变量，具有文件作用域，又称外部变量。
- 全局变量通常也应该先定义后使用；若使用在前，定义在后，则使用前要用关键字**extern** 说明。
- 全局变量具有默认初始值**0**。
- 当局部变量与全局变量同名时，按照局部优先原则，默认使用的是局部变量；要使用全局变量可在变量名称前加作用域运算符“**::**”。

【例3-16】分析下列程序的输出结果。

3.6 变量的作用域与存储类型

【源程序代码】

```
#include <iostream>
using namespace std;
int m=3;   // 全局变量
void f(){
    extern int n;
    // A 全局变量说明
        // 不能初始化
    m++;
    n++;
}
int n;        // 全局变量
```

```
int main() {
    int m=6; // 局部变量
    cout<<m<<'\t'<<n<<endl;   // B
    f();
    cout<<::m<<'\t'<<n<<endl; // C
    return 0;
}
```

程序运行结果

```
6  0
4  1
```

3.6 变量的作用域与存储类型

3.6.1 变量的作用域

3. 函数原型作用域

- 函数原型说明语句中形参的作用域仅限于该原型说明语句，属于函数原型作用域。
- 函数原型说明语句中可以省略函数形参的名称，只需说明形参的类型。

```
int fun(float x, float y);
```

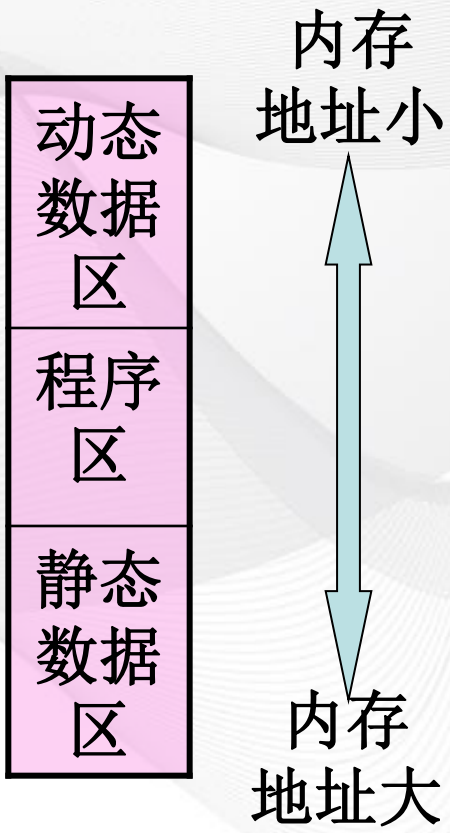
等价于：

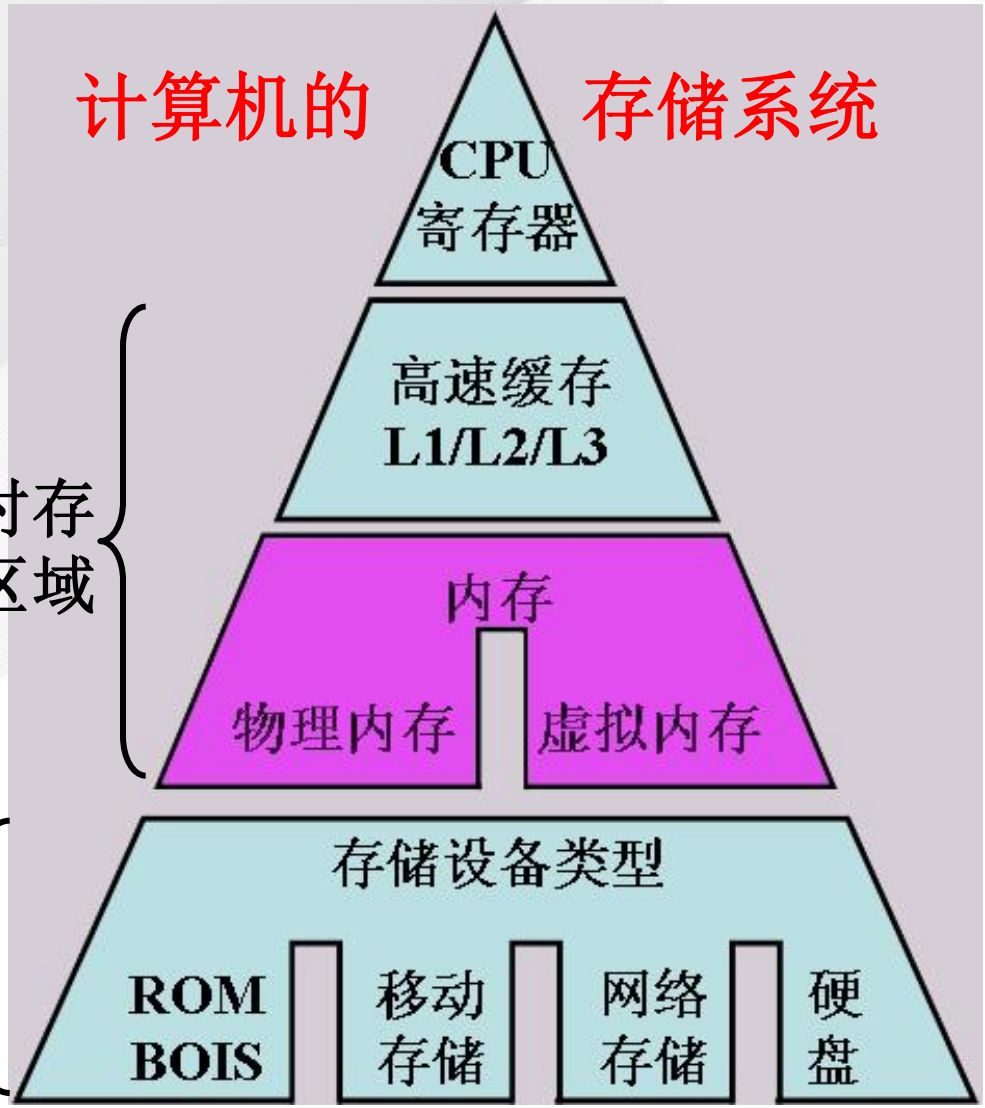
```
int fun(float, float);
```

4. 函数作用域与类作用域

3.6 变量的作用域与存储类型

3.6.2 变量的存储类型





3.6 变量的作用域与存储类型

3.6.2 变量的存储类型

一个完整的变量说明除了包括数据类型和变量名称外，还应包括变量的存储区域，变量的存储区域通过变量的存储类型加以说明。变量定义的完整格式为：

存储类型 数据类型 变量名;

- 静态存储类型：有默认初始值0，生命期为整个源程序
  - 静态类型
  - 外部类型
- 动态存储类型：无默认初始值，生命期为作用域
  - 自动类型
  - 寄存器类型

3.6 变量的作用域与存储类型

3.6.2 变量的存储类型

1. 自动类型变量

- 存储类型的关键字为**auto**；
- 局部变量的默认存储类型为自动类型。

```
void f(int a){
    float x;
}
```

等同于：

```
void f(auto int a){
    auto float x;
}
```

3.6 变量的作用域与存储类型

3.6.2 变量的存储类型

2. 静态类型变量

- 存储类型的关键字为**static**；
- 可分为局部静态变量和全局静态变量。
  - 局部静态变量：块中用**static** 说明的变量，具有块作用域，作用域外存在但不可用，进作用域延续前值。
  - 全局变量总是静态存储，用**static** 说明的全局变量仅限于在定义它的源程序中使用，而未使用**static** 说明的全局变量可以被其他文件使用。

【例3-16】分析下列程序的输出结果。

3.6 变量的作用域与存储类型

【源程序代码】

```
# include <iostream>
using namespace std;
void f(){
    auto int i=0;        // A
    static int j,k=1;    // B
    i++;
    j++;
    k++;
    cout<<i<<'\t'<<j<<'\t'<<k<<'\n';
}
```

```
int main() {
    f();
    f();
    return 0;
}
```

程序运行结果

```
1  1  2
1  2  3
```

3.6 变量的作用域与存储类型

3.6.2 变量的存储类型

3. 寄存器类型变量

- 存储类型的关键字为**register**；
- 建议系统在寄存器分配存储空间。

4. 外部变量

- 函数外部说明的变量，存储类型的关键字为**extern** 。
- 在同一源程序中，使用在前、定义在后的全局变量，使用前要说明为外部变量；
- 在一个源程序文件中，使用其他文件中定义的全局变量时，要说明为外部变量。

3.7 程序举例

【例3-18】 分析下列程序的输出结果。

```
void f(int x){
    extern int i;
    static int j=20;
    int k=30;
    i+=x;
    j+=x;
    k+=x;
    cout<<i<<'\t'<<j<<'\t'<<k<<endl;
}
int i=10;
```

```
int main() {
    for(int i=1;i<3;i++){
        int i=5;  f(i);
    }
    cout<<i<<'\t'<<::i<<endl;
    return 0;
}
```

程序运行结果

```
15  25  35
20  30  35
3  20
```

3.7 程序举例

【例3-19】 分析下列程序的输出结果。

```
int* f(int x1,int &x2,int *x3,int *x4){
    x1=x2;        x2=*x3;
    *x3=*x4;      x4=&x1;
    cout<<x1<<',<<x2<<',<<*x3<<',<<*x4<<endl;
    return &x2;
}
int i=10;
```

程序运行结果

2, 3, 4, 2

3

1, 3, 4, 4

```
int main(){
    int a(1),b(2),c(3),d(4);
    cout<<*(f(a,b,&c,&d))<<endl;
    cout<<a<<',<<b<<',<<c<<',<<d<<endl;
    return 0;
}
```

3.7 程序举例

【例3-20】分析下列程序的输出结果。

```
#include <iostream>
using namespace std;
#define FX(x,y) x/y
double fx(double x,double y){
    return x/y;
}
int main() {
    cout<<FX(4+2,3+1)<<endl;     // A
    cout<<fx(4+2,3+1)<<endl;
    return 0;
}
```

```
  FX(4+2,3+1)
= x/y
= 4+2/3+1

  fx(4+2,3+1)
= fx(6,4)
```

程序运行结果

```
5
1.5
```

3.7 程序举例

【例3-21】 编程实现两个分数的加法运算，并将结果约分为最简分数形式。

- 分数用分子和分母表示。
- 设计函数f 进行两个分数的加法运算：
  - void f(int m1,int d1,int m2,int d2,int &m3,int &d3);
  - 每个分数用一组（两个）整数表示；
  - 第3组表示结果，为引用，以通过参数带回。
- 设计函数f1 求两个整数的最大公约数，以约分：

  int f1(int x,int y)； // x 和y表示要约分分数的分子和分母
- 设计函数f2 求两个整数（分母）的最小公倍数，以通分：
  - int f2(int x,int y)； //x和y表示两个要相加分数的分母
  - f2(x,y)=x*y/f1(x,y)。

3.7 程序举例

【源程序代码】

```
#include <iostream>
using namespace std;
void f(int m1,int d1,int m2,int d2,int &m3,int &d3);
int f1(int x,int y){          // 用穷举法求x 和y 的最大公约数t
    int t=x>y?y:x;
    while(x%t||y%t)t--;
    return t;
}
int f2(int x,int y){              // 返回参数x 和y 的最小公倍数
   return x*y/f1(x,y);
}
```

3.7 程序举例

【源程序代码】

```
int main(){
    int m1,d1,m2,d2,m3,d3;
    cout<<"请输入第一个分数（分子分母）： ";
    cin>>m1>>d1;
    cout<<"请输入第二个分数（分子分母）： ";
    cin>>m2>>d2;
    f(m1,d1,m2,d2,m3,d3);
    cout<<m1<<'/'<<d1<<'+'<<m2<<'/'<<d2
        <<'='<<m3<<'/'<<d3<<endl;
    return 0;
}
```

3.7 程序举例

【源程序代码】

```
void f(int m1,int d1,int m2,int d2,int &m3,int &d3){
    int t;
    d3=f2(d1,d2);
    //求m1/d1 与m2/d2 相加时的分母d3
    m3=m1*d3/d1+m2*d3/d2;
    //求m1/d1 与m2/d2 相加时的分子m3
    t=f1(m3,d3);
    m3/=t;
    d3/=t;
}
```

程序运行结果

```
请输入第一个分数(分子 分母)：1 6
请输入第二个分数(分子 分母)：1 3
1/6+1/3=1/2
```

3.7 程序举例

【例3-22】 设计程序求[n1，n2]区间内的所有素数。要求判断一个整数是否为素数和输出分别用一个函数实现，并按每行5 个素数的方式输出。

- 设计函数f 判断参数n 是否为素数：
  - 若参数n 是素数，则f 函数返回1，否则返回0；
  - int f(int n);
- 设计函数show 输出n1 至n2 区间的素数：

  void show(int n1,int n2);
- 主函数中定义并输入区间范围；然后调用show函数输出其中的素数；在show函数中再调用f函数判断所处理的整数是否为素数，若是则按指定格式输出。

3.7 程序举例

【源程序代码】

```
#include <iostream>
#include <cmath>
using namespace std;
int f(int n){
    for(int i=2;i<=sqrt(n);i++)
        if(n%i==0)return 0;
    return 1;     // A
}
/*A行为循环结束后，若没有返回0，则返回1；前面不能加
  else，否则第1次循环时，不返回0，就返回1，不能实现2
  与 √n 之间所有整数的遍历。*/
```

3.7 程序举例

【源程序代码】

```
void show(int n1,int n2){
    int count=0;
    while(n1<=n2){
        if(f(n1)){  //等同于f(n1)==1，若n1是素数，则操作：
            cout<<n1<<'\t';
            count++;
            if(count%5==0)
                cout<<endl;
        }
        n1++;
    }
    cout<<endl;
}
```

3.7 程序举例

【源程序代码】

```
int main(){
    int number1,number2;
    cout<<"请输入区间范围(n1 n2)： ";
    cin>>number1>>number2;
    show(number1,number2);
    return 0;
}
```

程序运行结果

```
请输入区间范围(n1 n2)： 100 200
101 103 107 109 113
127 131 137 139 149
151 157 163 167 173
179 181 191 193 197
199
```

3.7 程序举例

【例3-23】利用函数求e^x的近似解，要求最后一项小于10^{-6}，其中，求e^x的近似公式如下。

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^{n-1}}{(n-1)!} + \frac{x^n}{n!}$$

- 定义递归函数int f(int n)，求n!。
- 定义递归函数int fun(int x,int n)，求x^n。
- 定义函数double sum(int x)，求各项的和，即e^x的值。
- 主函数中输入x的值调用sum函数求e^x的值。在sum函数中调用fun函数求通项的分子，即x^n；调用f函数求通项的分母，即n！。

3.7 程序举例

【源程序代码】

```
#include <iostream>
using namespace std;
int f(int n){                    // 求n!
    if(n==0||n==1) return 1;
    else return n*f(n-1);
}
int fun(int x,int n){      // 求x^n
    if(n==0) return 1;
    else if(n==1)return x;
        else return fun(x,n/2)*fun(x,n-n/2);
}
```

3.7 程序举例

【源程序代码】

```
double sum(int x){
    double s=0,t;
    int n=0;
    do{
        t=1.0*fun(x,n)/f(n);
        s+=t;
        n++;
    }while(t>1e-6);
    return s;
}
/*函数fun和f的返回值皆为整型，为防止整除，分子首先乘
   1.0*/
```

3.7 程序举例

【源程序代码】

```
int main(){
    int x;
    cout<<"请输入正整数x： ";
    cin>>x;
    cout<<'e'<<'^'<<x<<'='<<sum(x)<<'\n';
    return 0;
}
```

程序运行结果
请输入正整数x： 2
e^2=7.38867

通项$x^n/n!$，可用下列函数实现：

```
double fun(int x,int n){
    if(n==0) return 1;
    else return fun(x,n-1)* x/n;
}
```

$$\frac{x^n}{n!}=\begin{cases}1 & n=0\\ \dfrac{x^{n-1}}{(n-1)!}\times\dfrac{x}{n} & n>0\end{cases}$$

3.8 习题

1. 设计程序用迭代法求方程$3x^3-2x^2+5x-7=0$在1附近的一个根，精确达到10^{-6}。牛顿迭代公式为x=x-f(x)/f’(x)。要求定义两个函数分别求f(x)和f’(x)。

2. 设计程序求100以内的孪生素数对，要求用一个函数判断某一正整数是否为素数。所谓孪生素数对，是指差为2的一对素数。

3. 设计程序求组合数C(m,r)。其中C(m,r)=m!/(r!(m-r)!)，m和r为正整数，且m>r。要求设计两个函数分别求阶乘和组合数。

3.8 习题

4. 设计程序用递归法将十进制整数转换为十六进制整数。用递归法将十进制整数n 转换为十六进制整数的方法是求出n 与16 相除的余数t（t=n%16），并逆序输出；然后以n/16 作为参数调用递归函数，直到参数小于10 为止。为了实现逆序输出，应将输出语句置于递归语句之后；将大于等于10 的余数t 转换为相应十六进制数的方法是“char('A'+t-10)”。

5. 设计程序，分别用宏定义和函数求圆的面积，其中圆的半径可以为表达式。

计算机组成原理

第 2 版

江苏科技大计算机学院

第 1 章 计算机系统概论

1.1 计算机系统简介

一、 计算机的软硬件概念

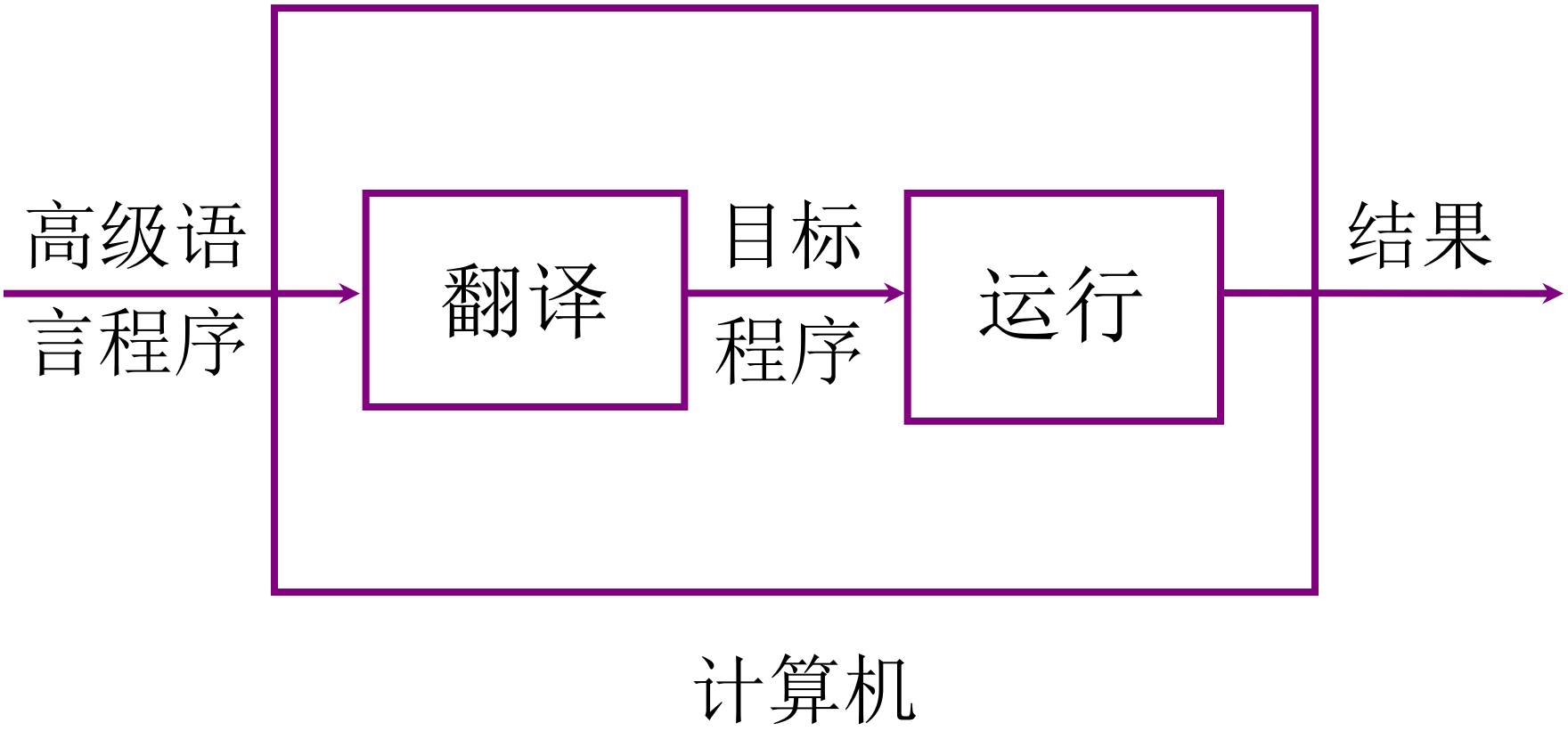
1. 计算机系统

计算机系统
- 硬件：计算机的实体，如主机、外设等
- 软件：由具有各类特殊功能的信息（程序）组成

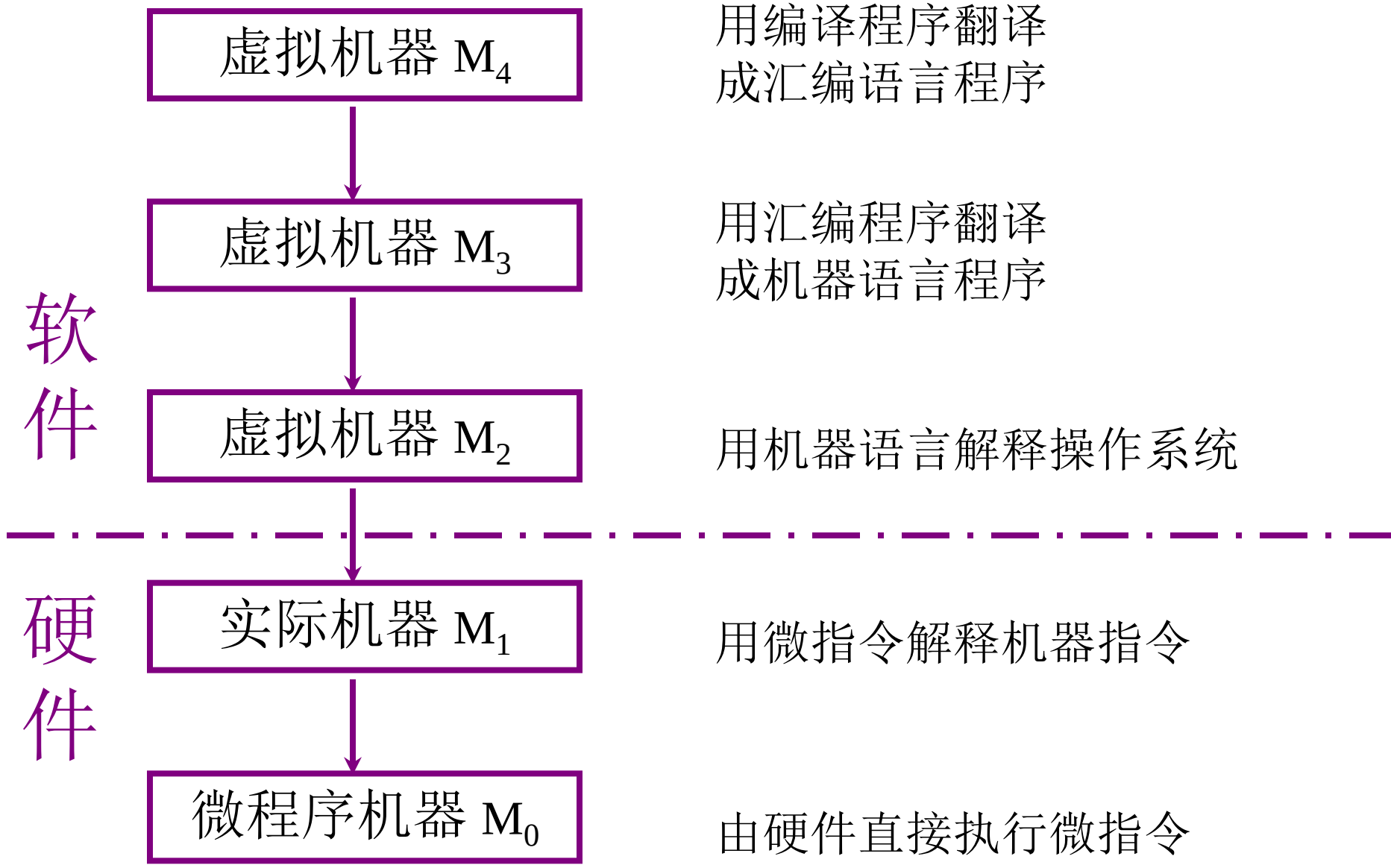
软件

- 系统软件　用来管理整个计算机系统
  - 语言处理程序
  - 操作系统
  - 服务性程序
  - 数据库管理系统
  - 网络软件
- 应用软件　按任务需要编制成的各种程序

2. 计算机的解题过程



二、计算机系统的层次结构


虚拟机器 M_4
用编译程序翻译
成汇编语言程序
虚拟机器 M_3
用汇编程序翻译
成机器语言程序
软件
虚拟机器 M_2
用机器语言解释操作系统
硬件
实际机器 M_1
用微指令解释机器指令
微程序机器 M_0
由硬件直接执行微指令

三、计算机体系结构和计算机组成

有无乘法指令

计算机体系结构　程序员所见到的计算机系统的属性
概念性的结构与功能特性

（指令系统、数据类型、寻址技术、I/O机理）

计算机组成　实现计算机体系结构所体现的属性

（具体指令的实现）

如何实现乘法指令

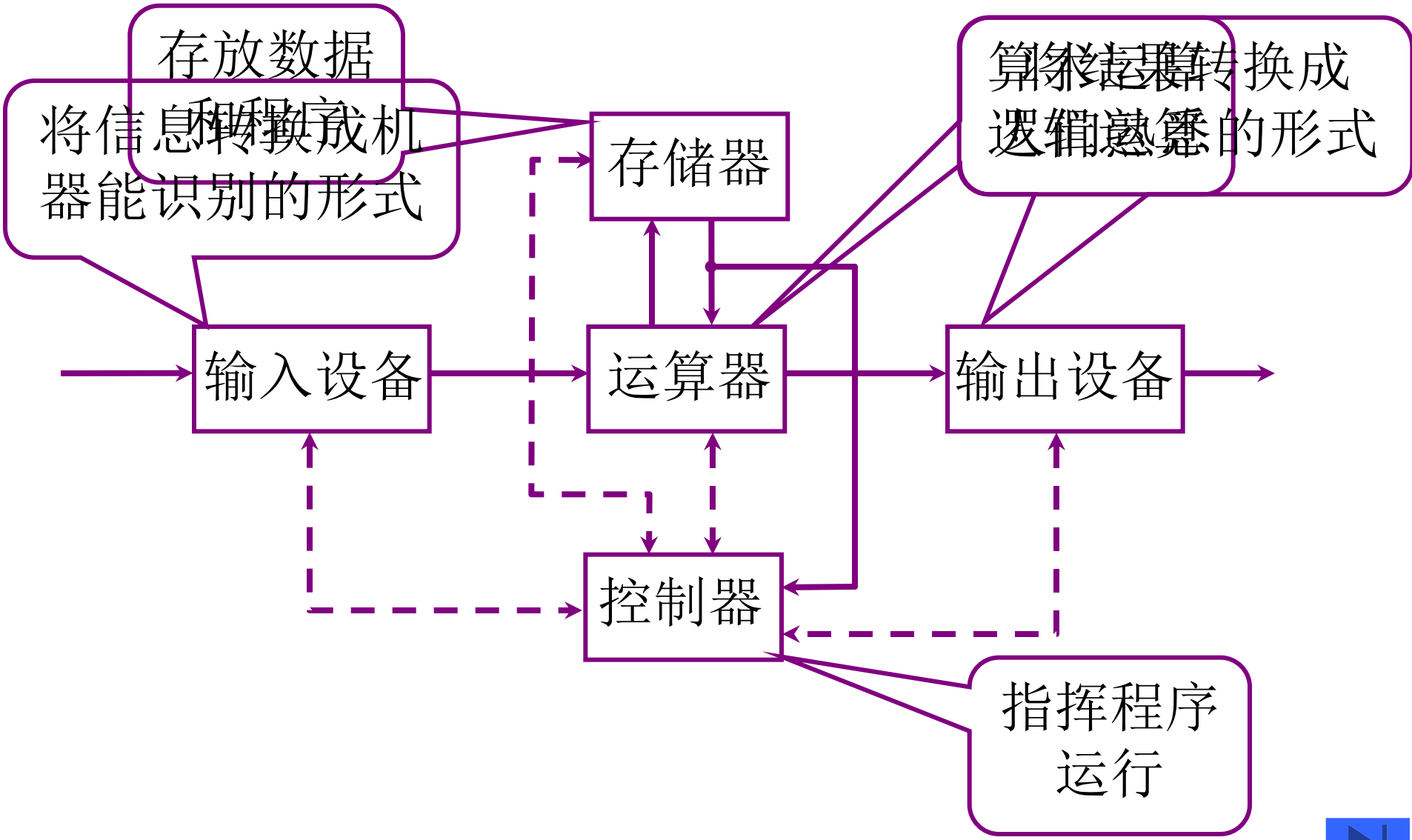
1.2 计算机的基本组成

一、冯·诺依曼计算机的特点

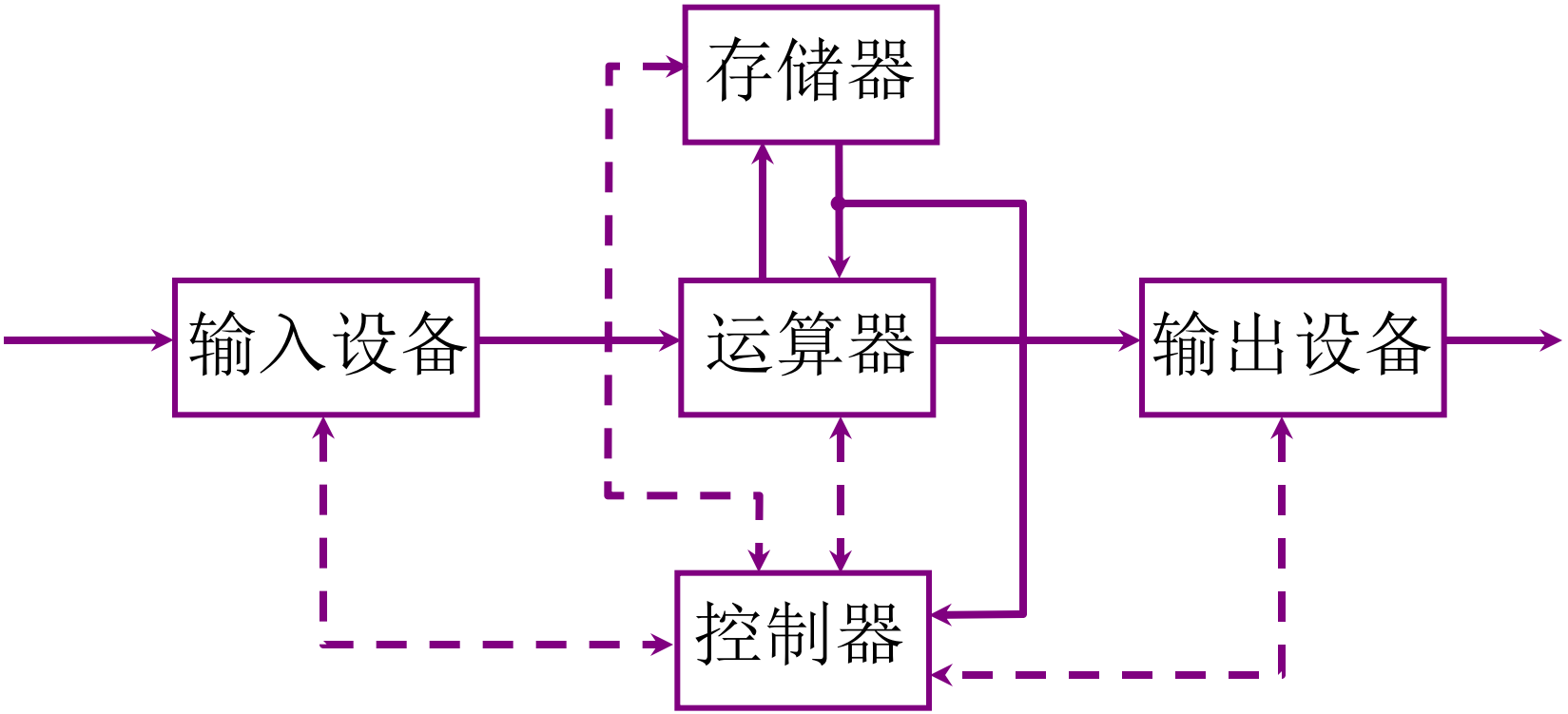
1. 计算机由五大部件组成
2. 指令和数据以同等地位存于存储器，可按地址寻访
3. 指令和数据用二进制表示
4. 指令由操作码和地址码组成
5. 存储程序
6. 以运算器为中心

冯·诺依曼计算机硬件框图

1.2

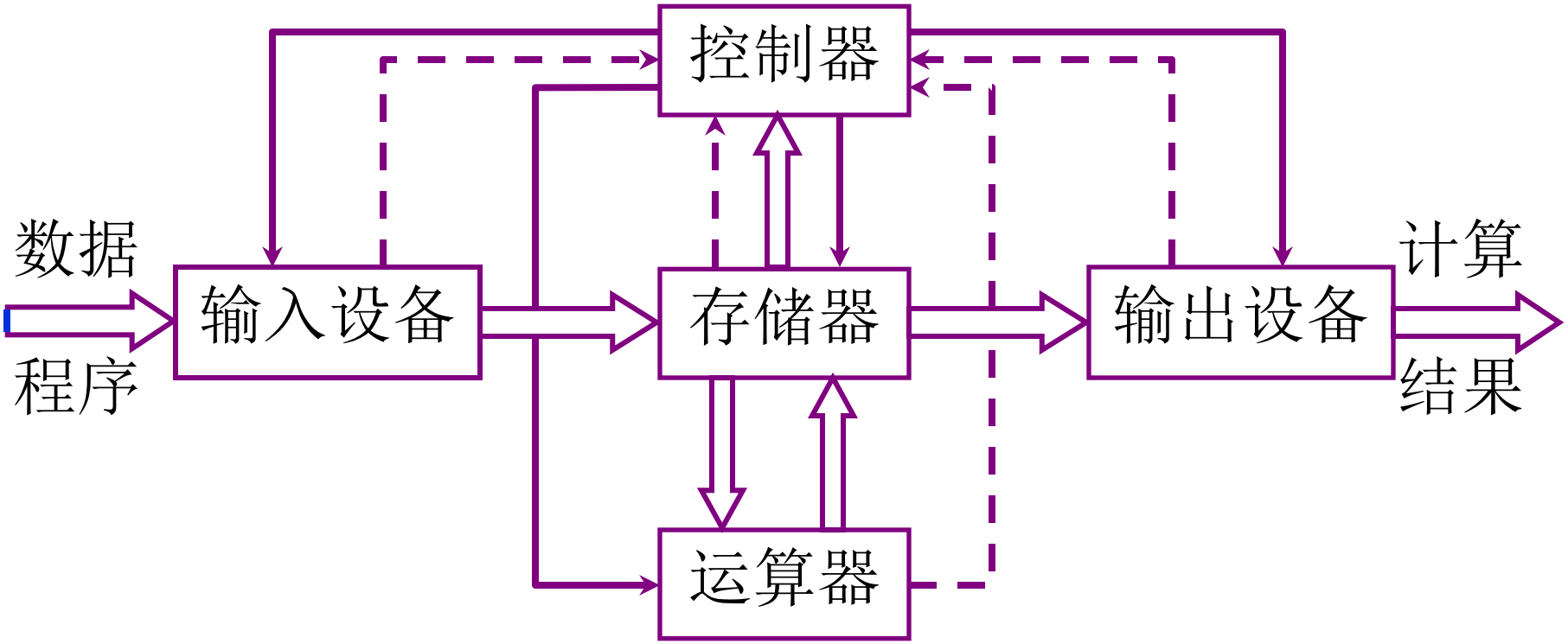


冯·诺依曼计算机硬件框图



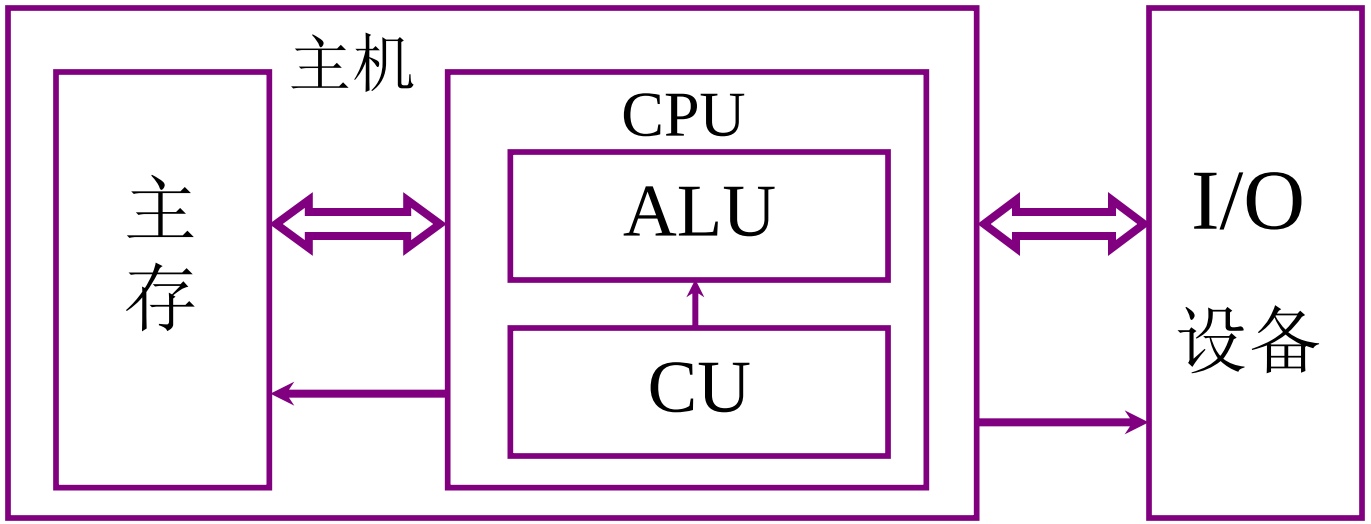
二、计算机硬件框图

1. 以存储器为中心的计算机硬件框图



2.现代计算机硬件框图

运算器 ALU
控制器 CU
} CPU
} 主机
存储器 { 主存 / 辅存
输入设备
输出设备
} I/O设备
} 硬件



三、计算机的工作步骤

1.上机前的准备

- 建立数学模型
- 确定计算方法

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots$$

$$\sqrt{x} = \frac{1}{2}\left(y_n + \frac{x}{y_n}\right)\left(n = 0, 1, 2, \cdots\right)$$

- 编制解题程序

程序 —— 运算的 全部步骤

指令 —— 每 一个步骤

编程举例

计算 $ax^2 + bx + c = (ax + b)x + c$

取x 至运算器中
乘以x 在运算器中
乘以a 在运算器中
存ax^2 在存储器中
取b 至运算器中
乘以x 在运算器中
加ax^2 在运算器中
加c 在运算器中

取x 至运算器中
乘以a 在运算器中
加b 在运算器中
乘以x 在运算器中
加c 在运算器中

指令格式举例

操作码	地址码

取数　α　　[α]　ACC

000001　0000001000

存数　β　　[ACC]　β

加　γ　　[ACC]+[γ]　ACC

乘　δ　　[ACC]×[δ]　ACC

打印　σ　　[σ]　打印机

停机

计算 $ax^2 + bx + c$ 程序清单

1.2

指令和数据存于主存单元的地址	指令		注释
	操作码	地址码	
0	000001	0000001000	取数*x*至ACC
1	000100	0000001001	乘*a*得*ax*,存于ACC中
2	000011	0000001010	加*b*得*ax*+*b* ,存于ACC中
3	000100	0000001000	乘*x*得（*ax*+*b*)*x*,存于ACC中
4	000011	0000001011	加*c*得$ax^2 + bx + c$,存于ACC
5	000010	0000001100	将$ax^2 + bx + c$,存于主存单元
6	000101	0000001100	打印
7	000110		停机
8	*x*		原始数据*x*
9	*a*		原始数据*a*
10	*b*		原始数据*b*
11	*c*		原始数据*c*
12			存放结果

2.计算机的解题过程

(1)存储器的基本组成

存储体 – 存储单元 – 存储元件 （0/1）

大楼 – 房间 – 床位 （无人/ 有人）

存储单元 存放一串二进制代码

存储字 存储单元中二进制代码的组合

存储字长 存储单元中二进制代码的位数

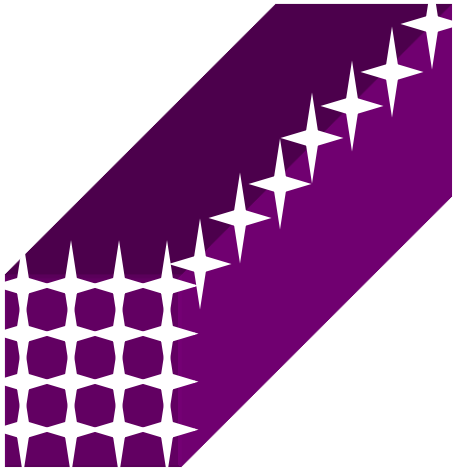
每个存储单元赋予一个地址号

按地址寻访

(1)存储器的基本组成

MAR 存储器地址寄存器
反映存储单元的个数

MDR 存储器数据寄存器
反映存储字长

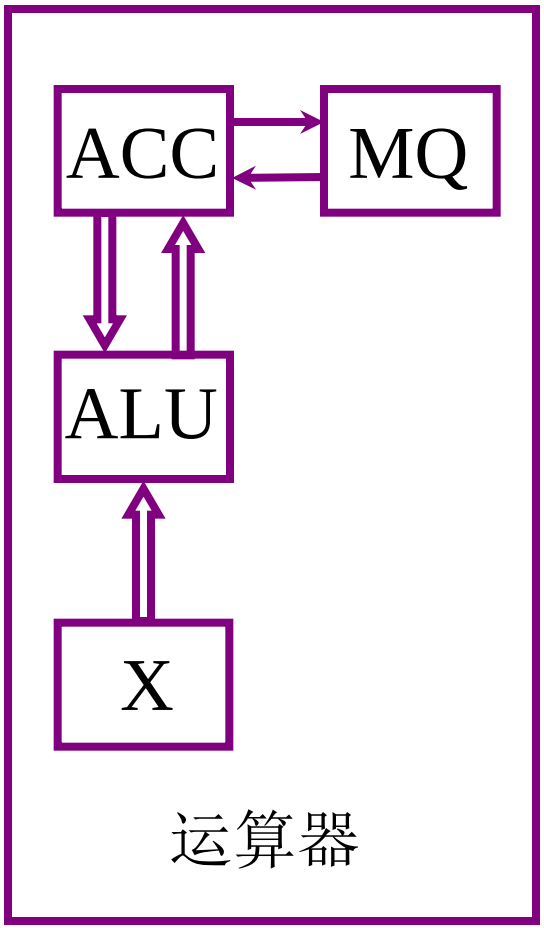


设 MAR=4 位
MDR=8 位

存储单元个数 16

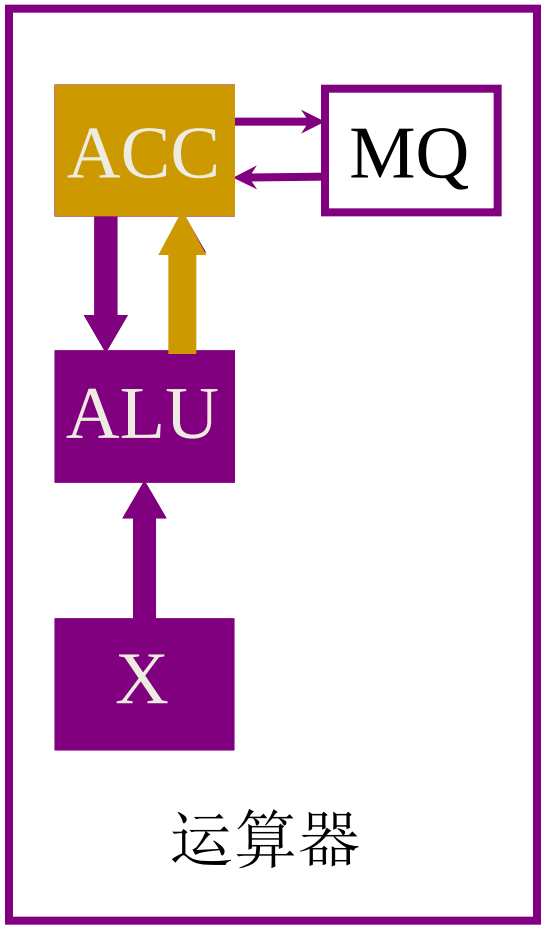
存储字长 8

(2)运算器的基本组成及操作过程



	ACC	MQ	X
加法	被加数 和		加数
减法	被减数 差		减数
乘法	乘积高位	乘数 乘积低位	被乘数
除法	被除数 余数	商	除数

① 加法操作过程



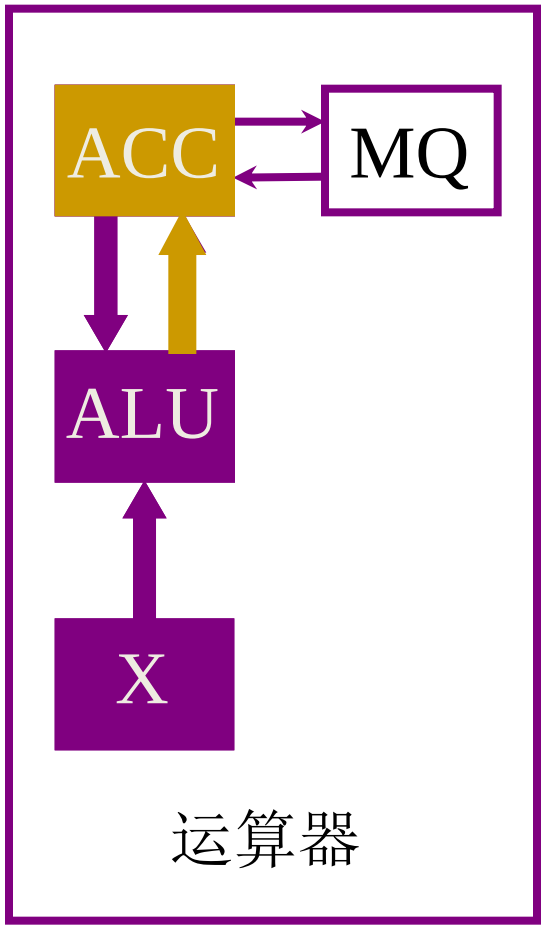
指令 | 加 | M

初态　ACC　被加数

[M] —X→

[ACC]+[X] ⟶ ACC

② 减法操作过程



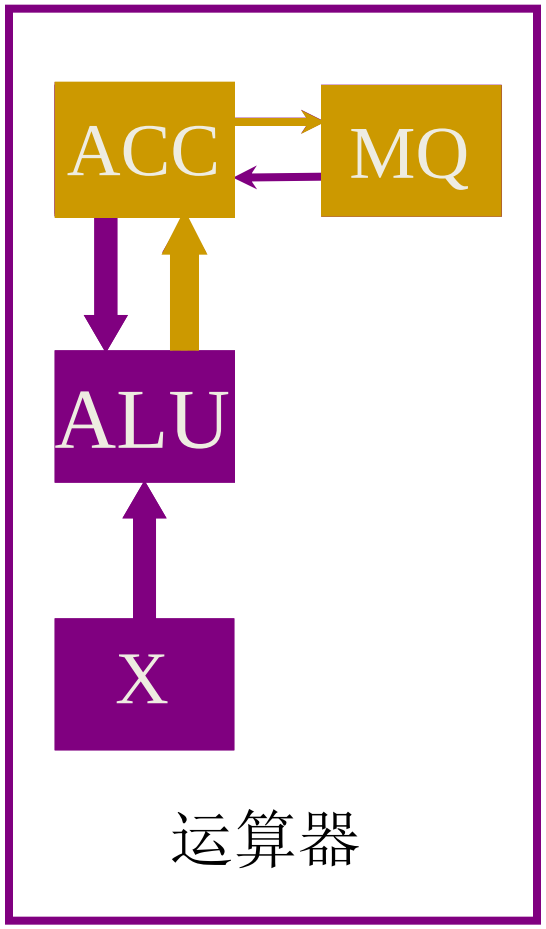
指令 | 减 | M

初态　ACC　被减数

[M] —X→

[ACC]-[X] ⟶ ACC

③ 乘法操作过程



指令 | 乘 | M

初态　ACC　被乘数

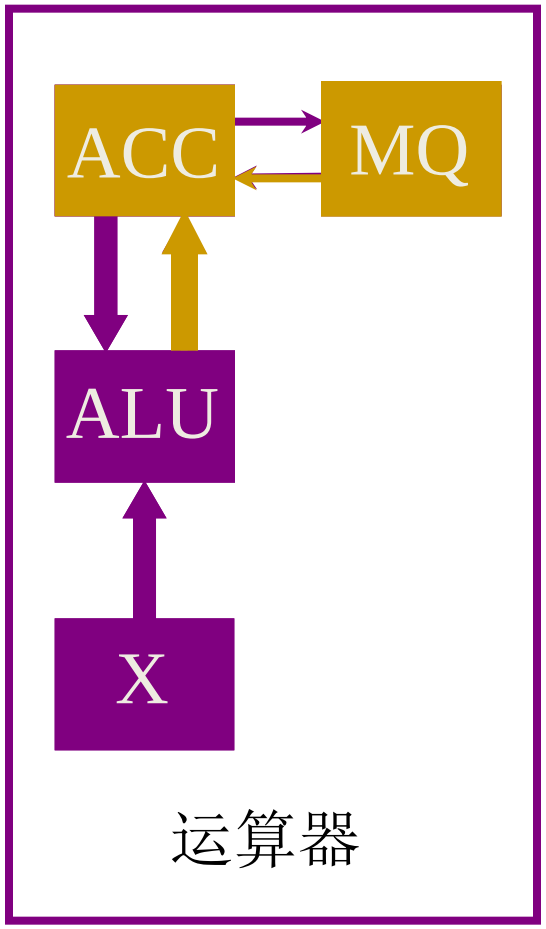
[M] ⟶ MQ

[ACC] ⟶ X

0 ⟶ ACC

[X]×[MQ] ⟶ ACC // MQ

④ 除法操作过程



指令

除	M

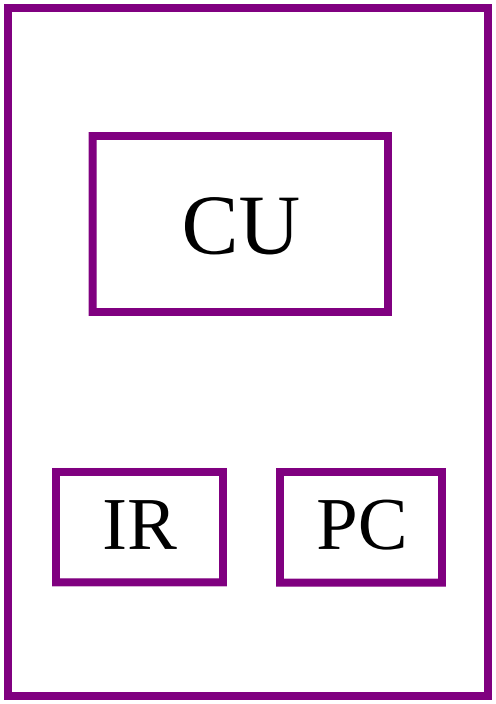
初态　ACC　被除数

$[M] \longrightarrow X$

$[ACC] \div [X] \longrightarrow MQ$

余数在ACC中

(3)控制器的基本组成



完成一条指令：

- 取指令　PC
- 分析指令　IR

（取指　访存）

- 执行指令　CU

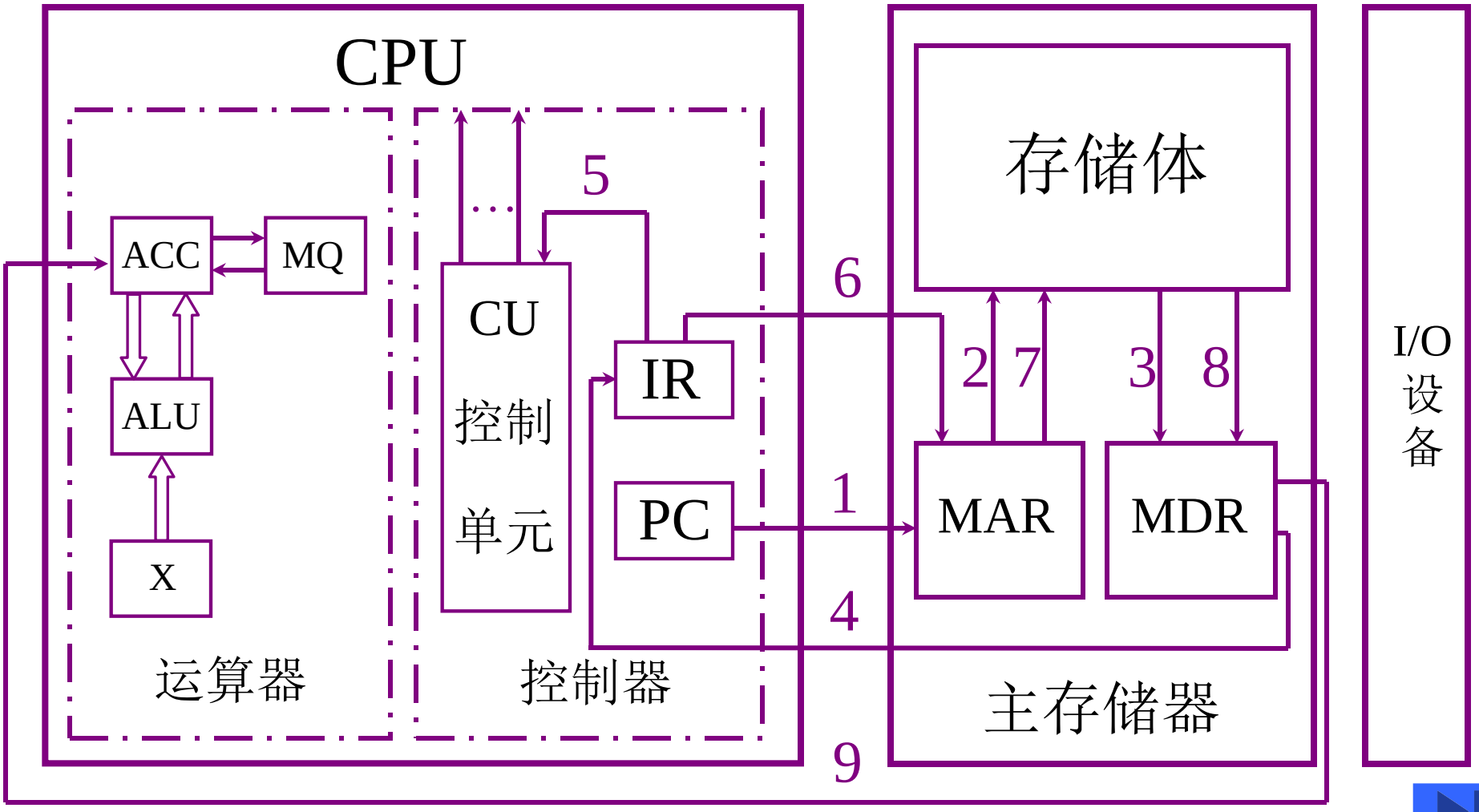
（执行　访存）

PC 存放当前欲执行指令的地址，具有计数功能（PC）+1　PC

IR 存放当前欲执行的指令

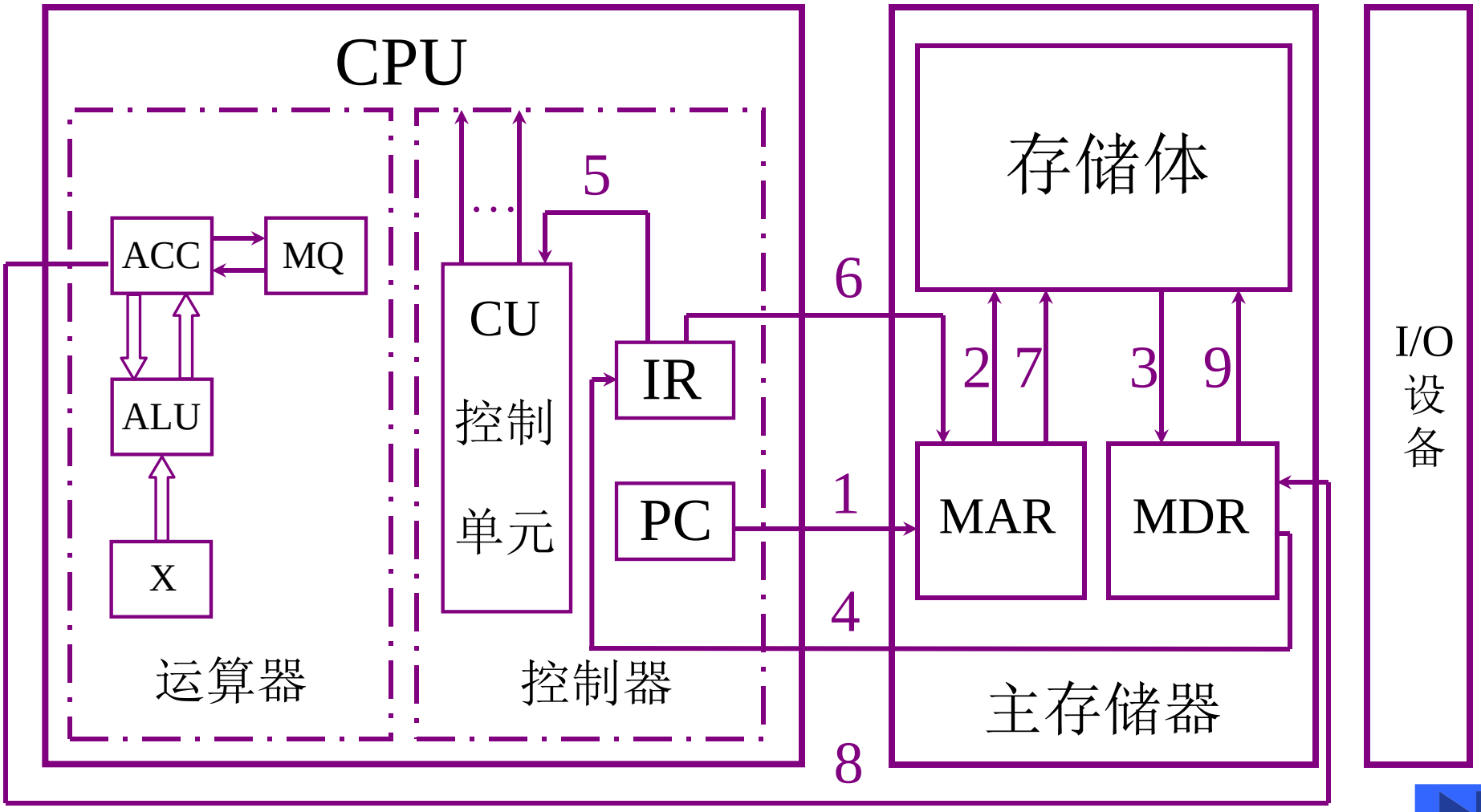
(4)主机完成一条指令的过程

以取数指令为例



(4)主机完成一条指令的过程

以存数指令为例



(5) $ax^2 + bx + c$ 程序的运行过程

- 将程序通过输入设备送至计算机
- 程序首地址 ⟶ PC
- 启动程序运行
- 取指令 PC → MAR → M → MDR → IR ,(PC)+1 PC
- 分析指令 OP(IR) → CU
- 执行指令 Ad(IR) → MAR → M → MDR → ACC

  ⋮
- 打印结果
- 停机

1.3 计算机硬件的主要技术指标

1.机器字长　CPU一次能处理数据的位数
与CPU中的寄存器位数有关

2.运算速度
- 主频
- 吉普森法　$T_M = \sum_{i=1}^{n} f_i t_i$
- MIPS　每秒执行百万条指令
- CPI　执行一条指令所需时钟周期数
- FLOPS　每秒浮点运算次数

3.存储容量　　存放二进制信息的总位数

主存容量

存储单元个数×存储字长

如 MAR　MDR　容量

10　8　　1 K×8位

16　32　　64 K×32位

1K = 2^{10}

字节数

如　2^{13} =1 KB

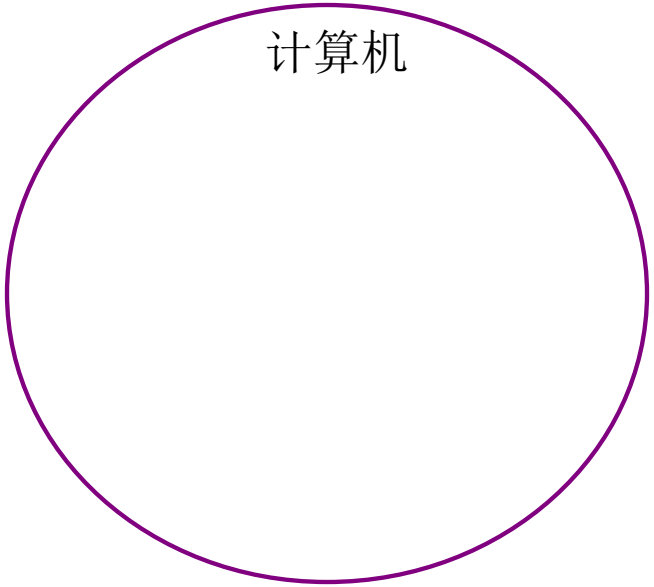
1B = 2^3b

2^{21} =256 KB

辅存容量　　字节数　80 GB

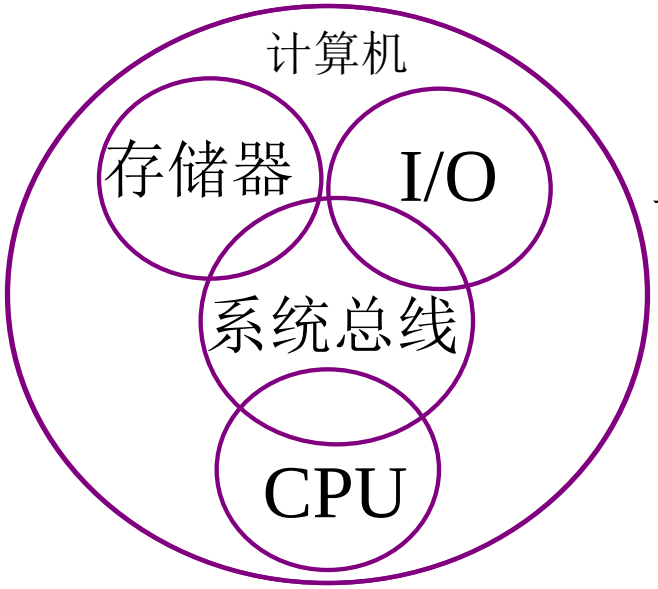
1GB = 2^{30}b

1.4 本书结构



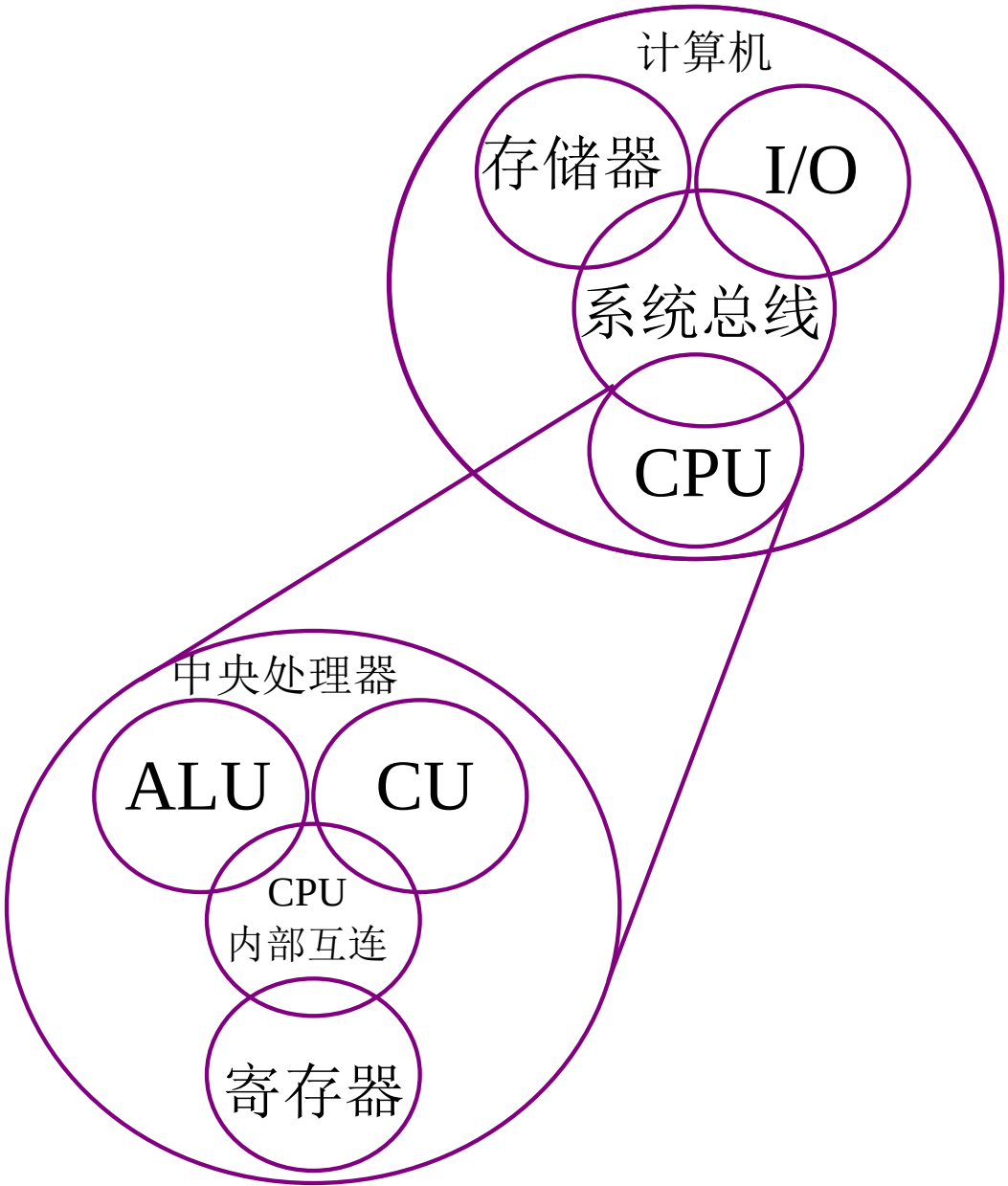
第 1 篇 概论

1.4 本书结构



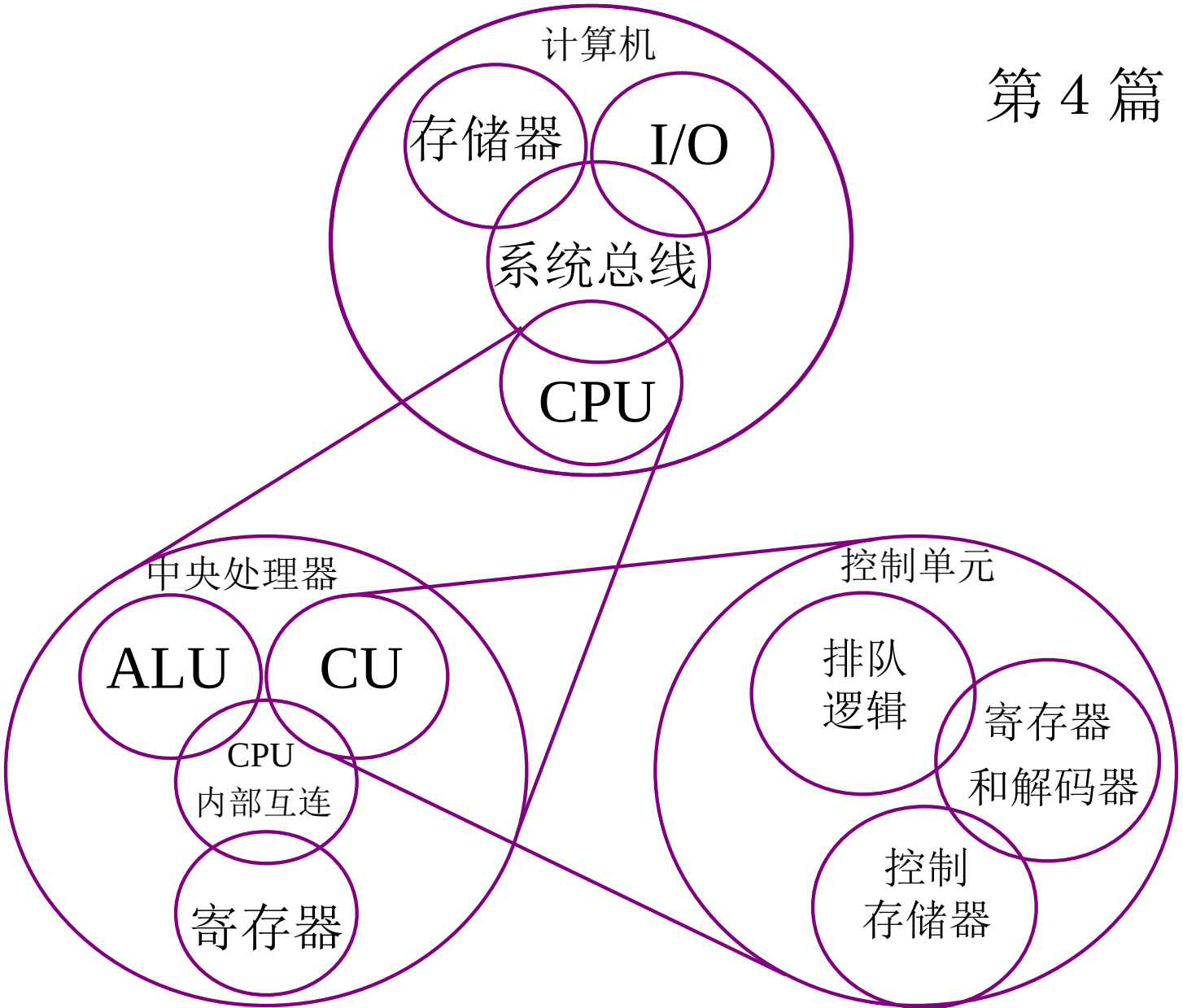
第 2 篇
计算机系统的硬件结构

1.4 本书结构



第 3 篇 CPU

1.4 本书结构



第 4 篇 CU