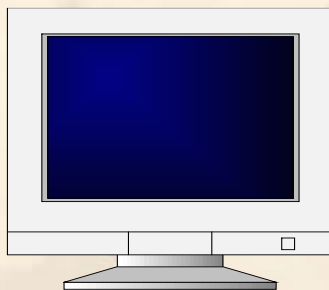


汇编语言与微机接口技术



计算机科学与工程学院 廖建明



教学安排



- **总学时数为48，其中教学内容46学时，课堂练习2学时。**
- **成绩评定方法：平时考核（含作业及课堂练习）占30%，期末考试占70%。**
- **自编讲义：《汇编语言与微机接口技术》. 唐勇,邢建川,廖建明编**
- **参考教材：**
 1. **《汇编语言：基于x86处理器》.基普·欧文著.机器工业出版社**
 2. **《汇编语言程序设计》. 廖建明主编.清华大学出版社**
 3. **《微机原理与接口技术》. 吴宁,乔亚男主编. 清华大学出版社**
- **关于实验：开设了“汇编语言与微机接口技术综合实验”课程，16学时，可以选修。**



课程目标

■ 掌握：

- 微处理器的基本工作原理
- Win32汇编语言基本程序设计方法
- 微型计算机的基本接口技术



为什么要学习汇编语言？



1. 学习和使用汇编语言可以从根本上认识、理解计算机的工作过程。

- 通过用汇编语言编制程序可以更清楚地了解计算机是如何完成各种复杂的工作。
- 程序设计人员能更充分地利用机器硬件的全部功能，发挥机器的长处。

2. 在计算机系统及应用中，某些功能必须用汇编语言程序来实现。

如：计算机启动时的自检、系统初始化、设备驱动程序、加密解密、逆向工程，还有病毒、木马等代码分析和防治等。



3. 汇编语言程序的效率高于高级语言程序



- **“效率”有两个方面的含义：程序的目标代码长度和运行的时间。**
- **某些要节省内存和提高运行速度的应用场合，常用汇编语言来编制其中的部分程序。**
- **在嵌入式系统中内存容量小，速度较慢，编写其系统程序或应用程序时就必须考虑程序的“效率”。**
- **常见的嵌入式系统有空调控制系统、智能电话、汽车智能控制、机器人控制、无人机控制等等。**
- **电脑游戏软件常常要针对具体的硬件，在程序中嵌入短小而快速的汇编语言程序代码。**



学习汇编语言需注意的问题

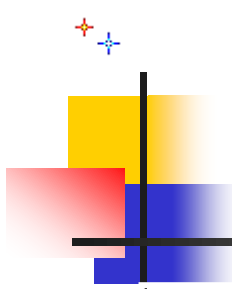




(1) 汇编语言与机器语言的关系---- “一对一”

- 一条汇编语言指令与一条机器语言指令对应
- 汇编语言程序与机器语言程序效率相同.

(2) 不同类型计算机有不同的机器指令系统和汇编语言描述，因此汇编语言不具有可移植性。

可移植性：如果用某种语言编写的源程序能够在各种不同的计算机系统中进行编译和运行，则称该语言是可移植的。

- 
- 
- 
- **为了学习和使用某种计算机的汇编语言，需熟悉该计算机的内部组成结构。**
 - **不需要像组成原理课程那样学习计算机的详细工作原理，只需掌握基本硬件结构及其功能。**
 - **机器语言的执行主要取决于该计算机的CPU，熟悉计算机内部结构就是指熟悉CPU的功能结构。**
 - **CPU功能结构包括：**
 - ◆ **CPU中包含的寄存器及其作用**
 - ◆ **CPU访问存储器的基本原理**
 - ◆ **输入输出操作的方法**



第1章 微型计算机基础概论



■ 主要内容:

- 微机系统的组成
- 微机分类
- 微机的发展历程
- 基本逻辑门及译码器 (复习)



1.1 微机系统的组成

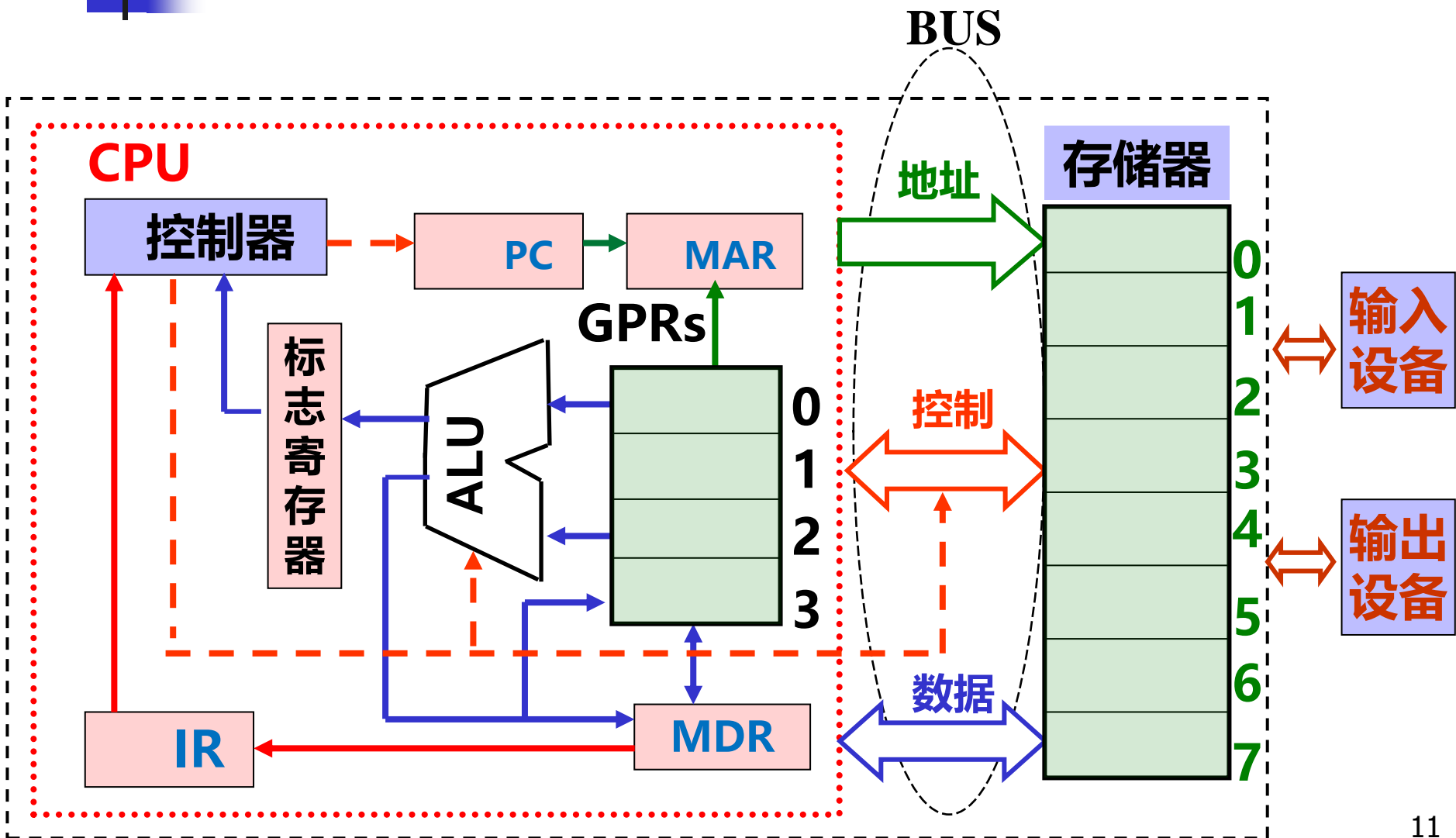
一、计算机的工作原理

1. 计算机中的指令执行过程

- 计算机的工作是逐条执行由指令构成的程序

取指令 → 指令译码 → 读取操作数 →
→ 执行指令 → 存放结果

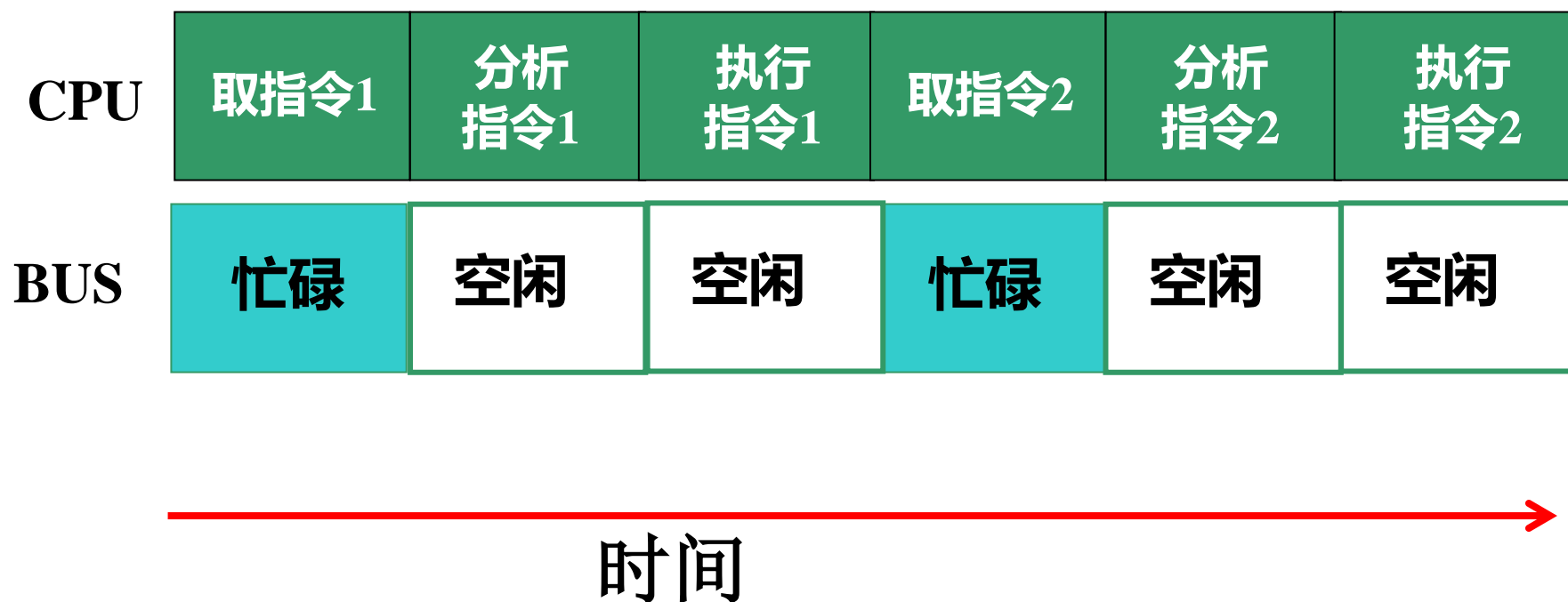
计算机基本组成结构回顾



指令的顺序执行和并行执行

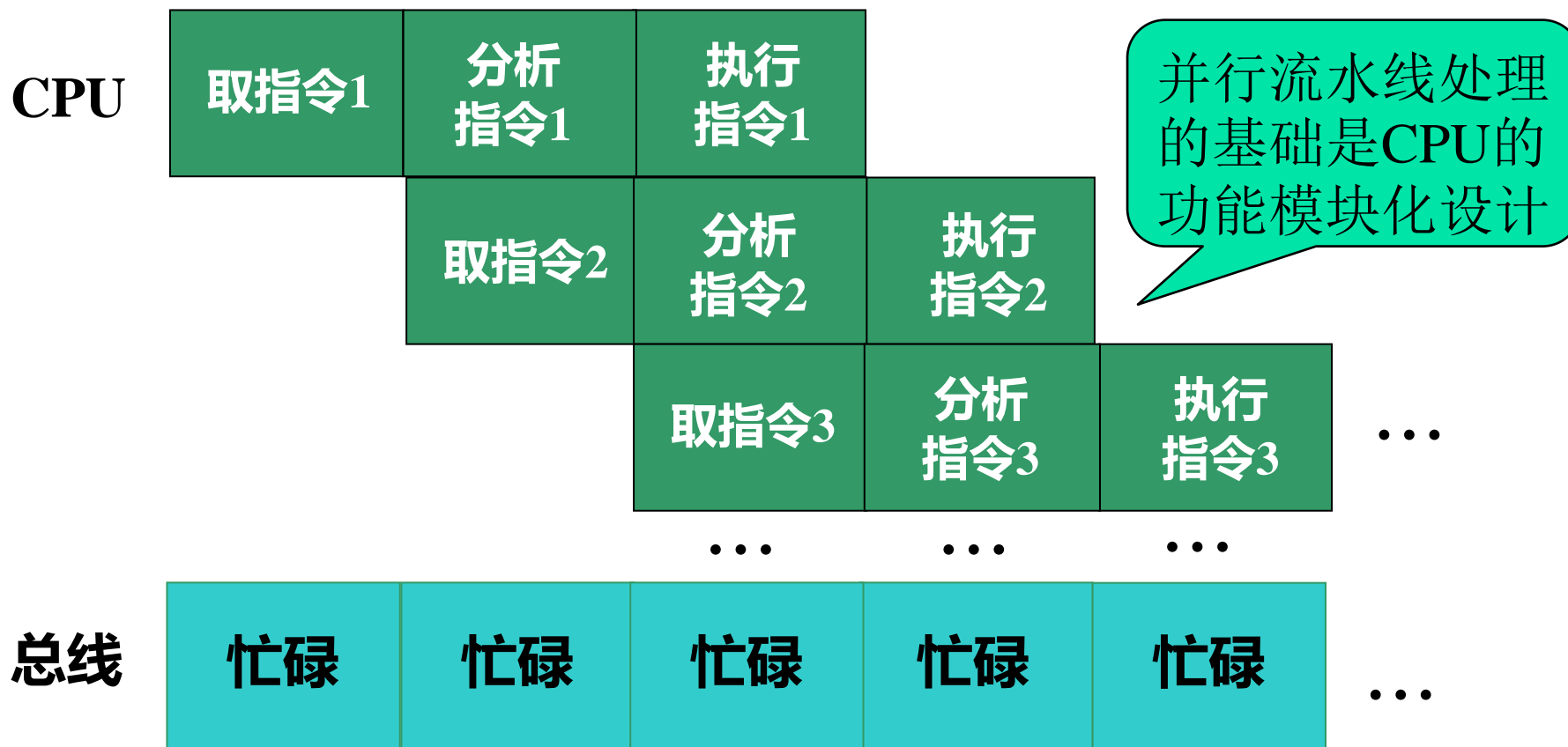
顺序工作方式

各功能部件交替工作，按顺序完成指令的执行过程。



并行流水线工作方式

将CPU划分为访问存储器、分析指令和执行指令等多个功能部件，各功能部件并行工作。

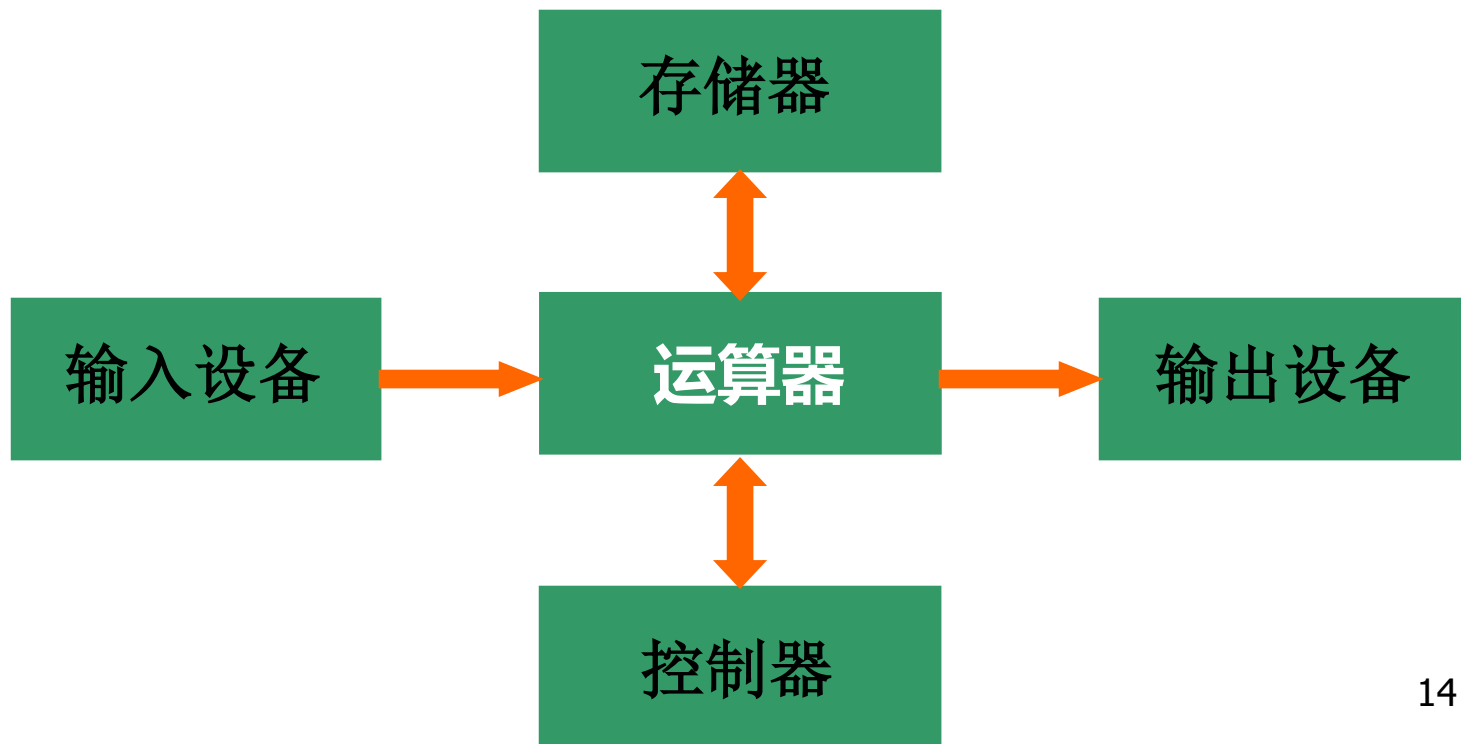


2. 冯 · 诺依曼计算机

■ 冯 · 诺依曼计算机的工作原理

- 存储程序工作方式
- 运算器为核心的结构特点

即使是非运算操作如寄存器间或寄存器与存储器之间的传送也要通过ALU





冯 • 诺依曼机的特点和不足



■ 特点:

- 存储程序，共享数据，顺序执行；
- 属于顺序处理机，适于确定的算法和数值处理。

■ 不足:

- 与存储器间有大量数据交互，对总线要求很高；
- 执行顺序由程序决定，对大型复杂任务较难处理；
- 以运算器为核心，处理效率较低；
- 由PC控制执行顺序，难以进行真正的并行处理。



3. 非冯 • 诺依曼计算机



- **主要特征**

- **并行性**

- **典型类型**

- **数据流计算机结构**

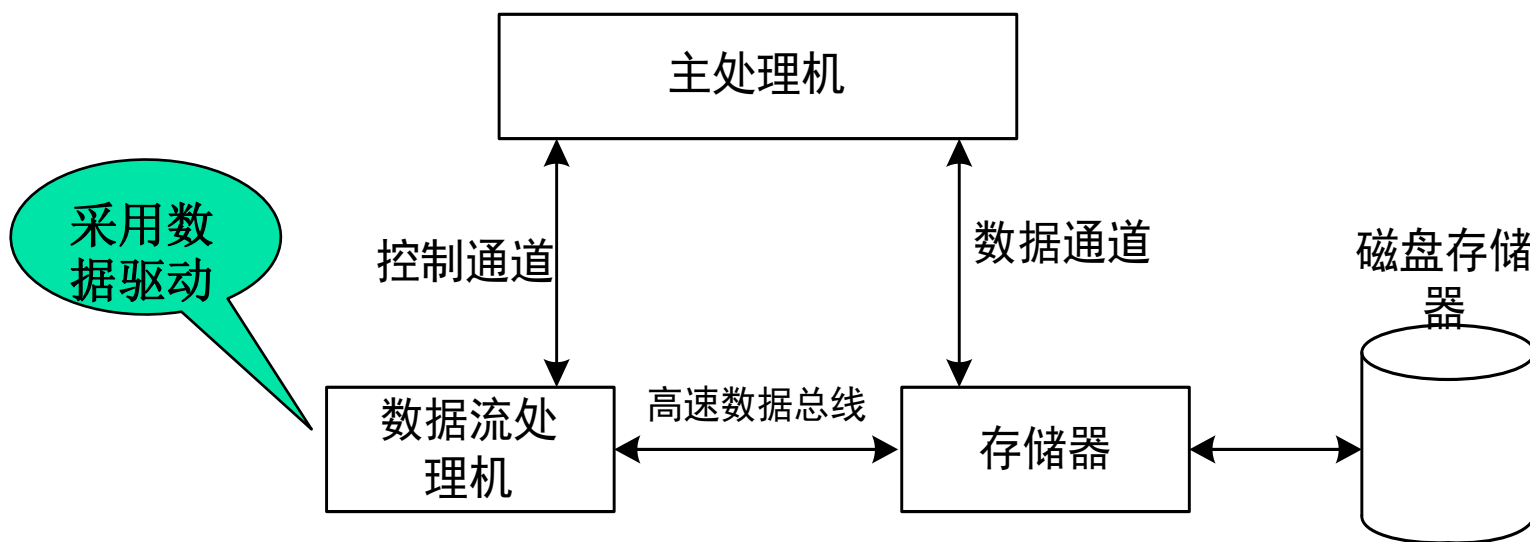
- (Dataflow Image Processing System)**

- **哈佛结构**

- (Harvard Architecture)**

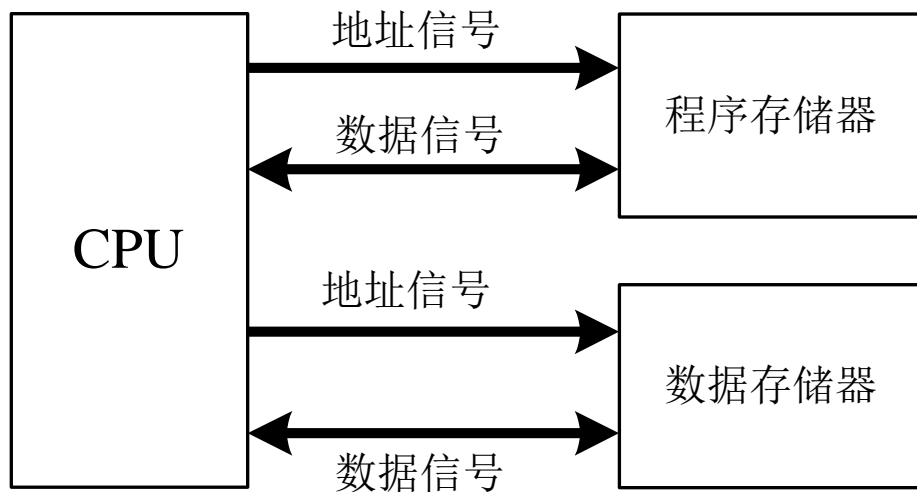
数据流计算机结构

- 数据流计算机采用数据驱动方式，是一种并行处理技术。
- 程序的执行顺序不是由程序计数器PC控制，而是由指令间的数据流控制



哈佛结构

- 指令和数据分别存放在两个独立的存储器模块中；
- CPU与存储器间的指令和数据的传送分别采用两组独立的总线；
- 可以在一个机器周期内同时获得指令操作码和操作数。





二、微机系统组成

1、微处理器

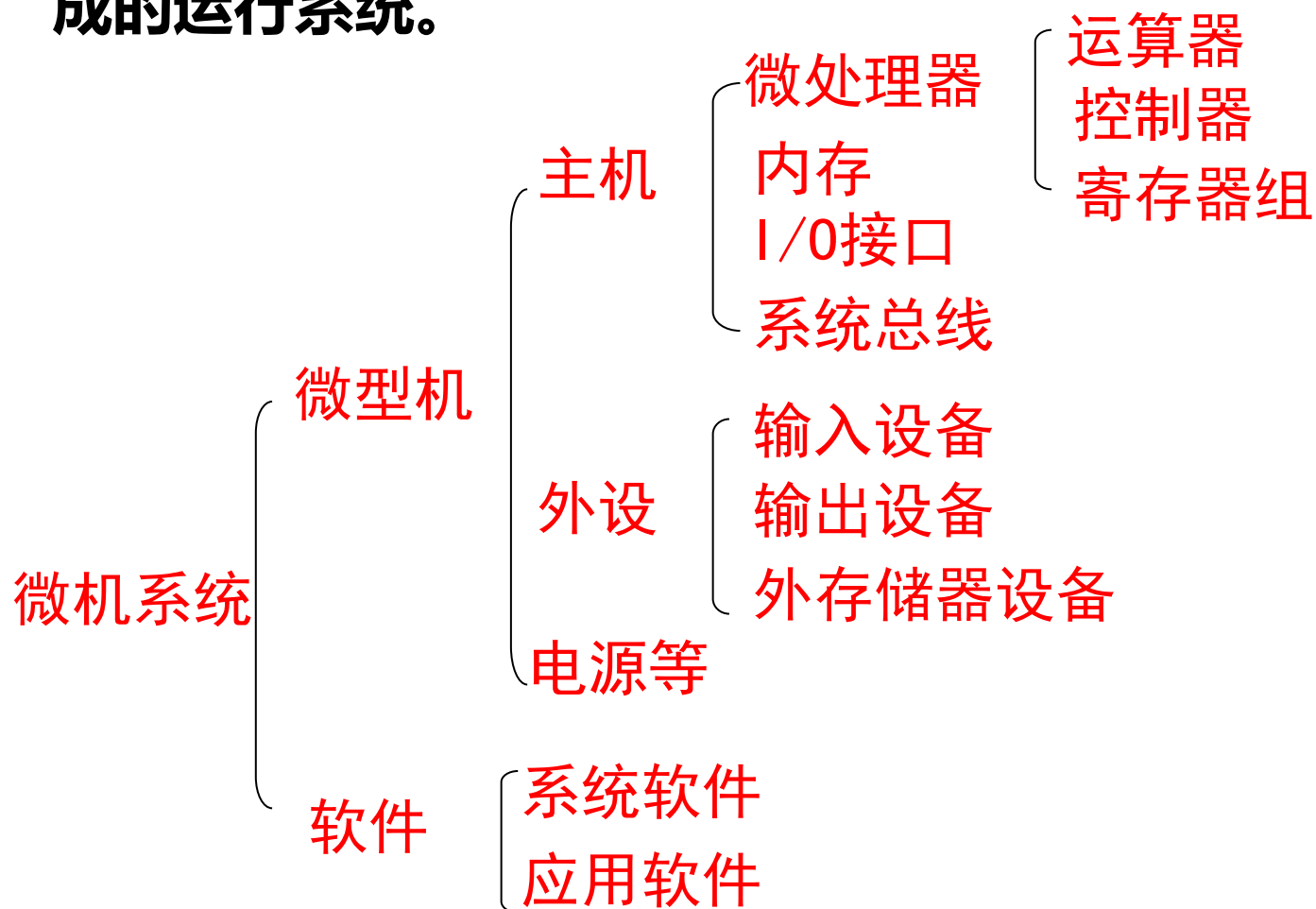
集成了运算器单元、控制器单元、寄存器单元及内部总线，并具有CPU全部功能的大规模集成电路芯片。

2、微型机

以微处理器为核心，配上内存、I/O接口、系统总线及电源等构成主机，连接输入/输出设备、外存设备等部件构成的硬件装置。

3、微机系统

为微型机硬件系统安装配置系统软件和应用软件后所构成的运行系统。



1.2 微型机分类

按微处理器字长

1位机
4位机
8位机
16位机
32位机
64位机

按结构

单片机
多片机

按组装

单板机
多板机

按外形

台式机
笔记本微机
掌上电脑



1、单片机

集成了CPU、部分内存、部分I/O接口及总线等单元，具有微机基本功能的超大规模集成电路芯片。

单片机特点：体积小、可靠性高、成本低。

应用领域：工业控制、智能仪器仪表、家用电器和其它各种嵌入式系统。



2、台式机（PC微机）



- 由系统主板、 I/O接口卡、磁盘、光驱、电源等组装在主机箱内
- 外接键盘、显示器、鼠标等设备
- 安装系统软件和应用软件

台式机特点：功能强、配置灵活、软件丰富、操作方便等。

应用领域：科学计算、事务处理、信息服务等许多领域。



1.3 微机系统发展概况



1) 第一阶段(1971-1973)

- 低档4位或8位微处理器与微机
- 系统结构与指令系统简单
- 集成度低、运行速度慢
- 机器语言或汇编语言编程

2) 第二阶段(1974-1978)

- 中档8位微处理器与微机
- 指令系统较丰富，具有典型计算机结构
- 集成度提高到5000-9000管/片
- 基本指令运行时间约1-2 μ s
- 出现高级语言编程与简单操作系统



3) 第三阶段(1978-1984)

- 16位微处理器和微型计算机
- 集成度和运算速度比第二代提高了一个数量级
- 指令系统更加丰富，系统结构增加了多级中断机制、多寻址机制、段式存储器结构、硬件乘除部件
- 支撑软件是操作系统
- 外部设备种类增多



4) 第四阶段(1985-1991)

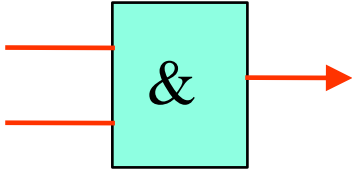
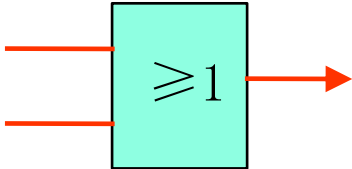
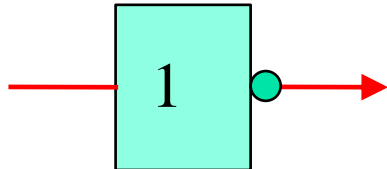
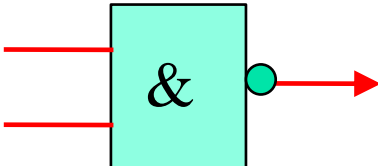
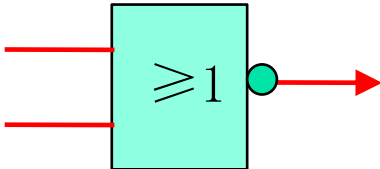
- 32位微处理器和微型计算机
- 微处理器芯片集成度达100万管/片
- 运行速度超过25MIPS
- 支持多用户多任务操作系统

5) 第五阶段(1992-目前)

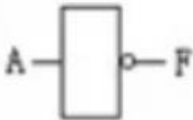
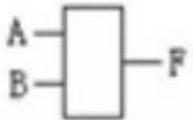
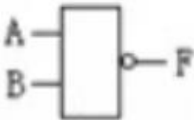
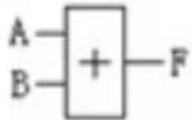
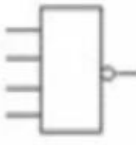
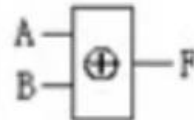




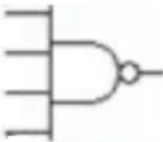

- 高档32位、64位微处理器和微型计算机
- 微处理器芯片集成度达2800万管/片以上
- 时钟主频高达3.0 GHz以上
- 支持多用户多任务操作系统

1.4 基本逻辑门及译码器

1. 教材中逻辑运算的图形符号表示

- “与” 运算: 
- “或” 运算: 
- “非” 运算: 
- “与非” 和 “或非” 运算:  

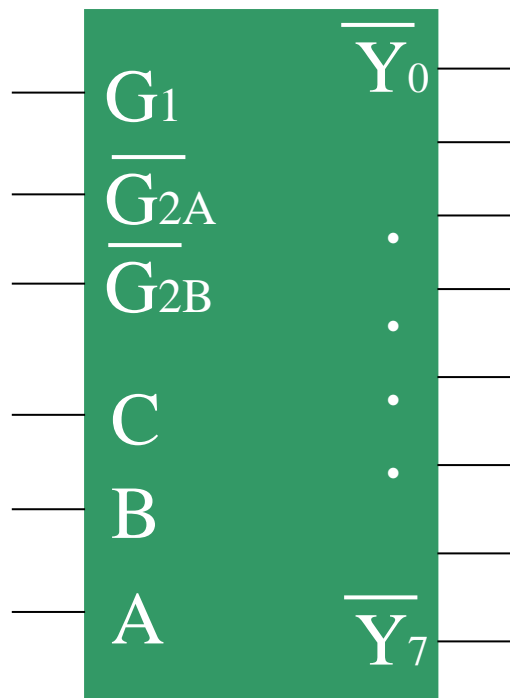
其他常见逻辑图形符号表示

名称	非门	二输入与门	二输入与非门	二输入或门	四输入与门	异或门
图形符号						
						
逻辑式	$F = \bar{A}$	$F = A \cdot B$	$F = \overline{A \cdot B}$	$F = A + B$	$F = A \odot B$	$F = A \oplus B$

2. 译码器--74LS138

■ 主要引脚及功能

输入端与输出端关系（真值表）



使能端			输入端			输 出 端							
G_1	$\#G_{2A}$	$\#G_{2B}$	C	B	A	$\#Y_0$	$\#Y_1$	$\#Y_2$	$\#Y_3$	$\#Y_4$	$\#Y_5$	$\#Y_6$	$\#Y_7$
0	x	x	x	x	x	1	1	1	1	1	1	1	1
x	1	x	x	x	x	1	1	1	1	1	1	1	1
x	x	1	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	0	0	1	1	1	1	1	1
1	0	0	0	1	0	0	1	0	1	1	1	1	1
1	0	0	0	1	1	0	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0