

# 微处理器与接口技术

## —微机原理与接口技术

# 教材及实验指导书

## ■ 教材：

- 《微机原理与接口技术》（第4版）。  
吴宁 乔亚男 主编。  
清华大学出版社



# 教材及实验指导书

## ■ 教材：

- 《单片机原理及接口技术》（第2版）  
李全利编，高等教育出版社



# 教材及实验指导书

## ■ 实验指导书

- 《微处理器与接口技术实验指导书》  
广东工业大学

授课学时：48（含实验学时8）

任课教师：李优新

联系方式：1297813518（QQ）

网站：<http://wjyl.gdut.edu.cn/>

# 学习目标

- 1、掌握8086/8088和8051等微处理器的系统组成、工作原理和总线架构，掌握微处理和接口的基本概念和开发方法。
- 2、运用微处理器的基本知识和设计方法，分析系统需求和接口资源，设计可行的微处理系统的软硬件方案。
- 3、掌握8086/8088和8051等微处理器系统，以及相关接口的设计和调试方法，搭建微处理器系统的实验平台并进行调试和验证。
- 4、掌握8086/8088、8051处理器系统的设计开发软件，能够在开发环境中对微处理系统进行软件开发和调试。

# 第1章

# 微型计算机基础概论

# 主要内容

- 微机系统的组成
- 计算机中的数制和编码
- 冯·诺依曼结构
- 二进制数的表示与运算
  - 机器数的表示及运算
  - 二进制数运算中的溢出
- 基本逻辑门及逻辑电路

# 1.1 微型计算机系统



# 冯·诺依曼计算机

现代计算机的基本工作结构：

——冯·诺依曼计算机结构

冯·诺依曼计算机原理的核心

——存储程序原理 P4

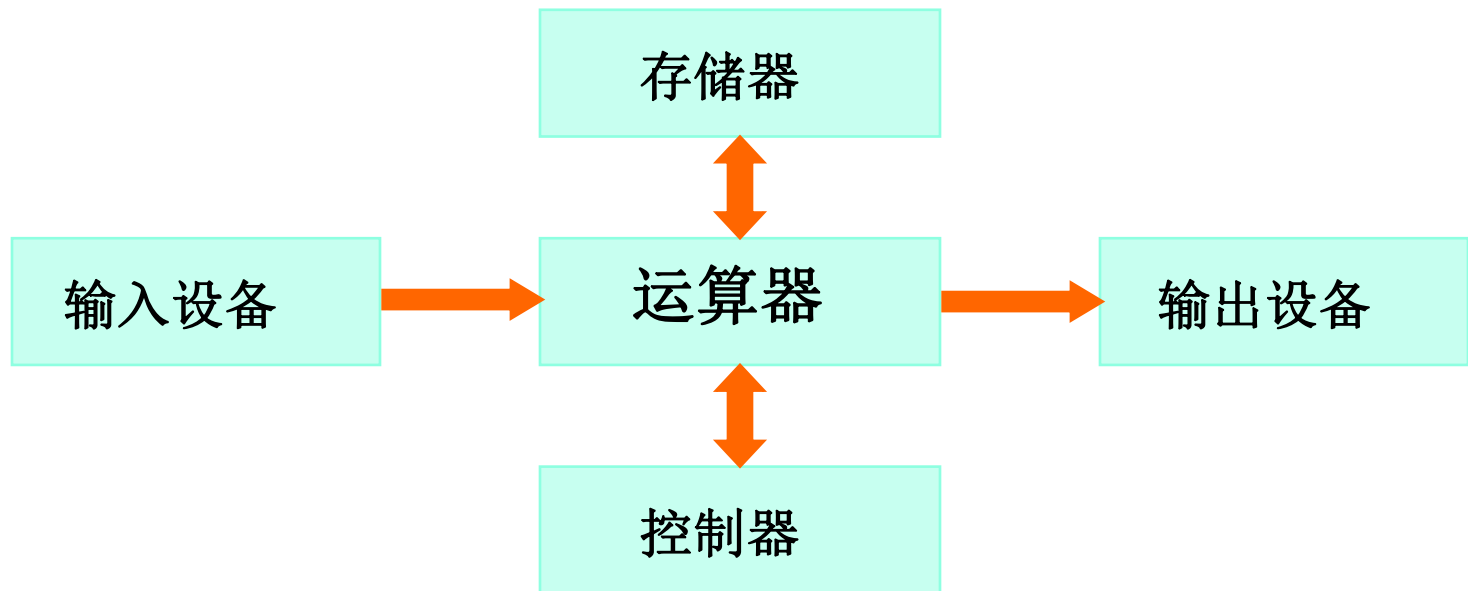
# 冯·诺依曼计算机

## ■ 特点:

- 将计算过程描述为由多条指令按一定顺序组成的程序，并放入存储器保存。
- 指令按其在存储器中存放的顺序执行；
- 由控制器控制整个程序和数据存取以及程序的执行；
- 以运算器为核心，所有的执行都经过运算器。

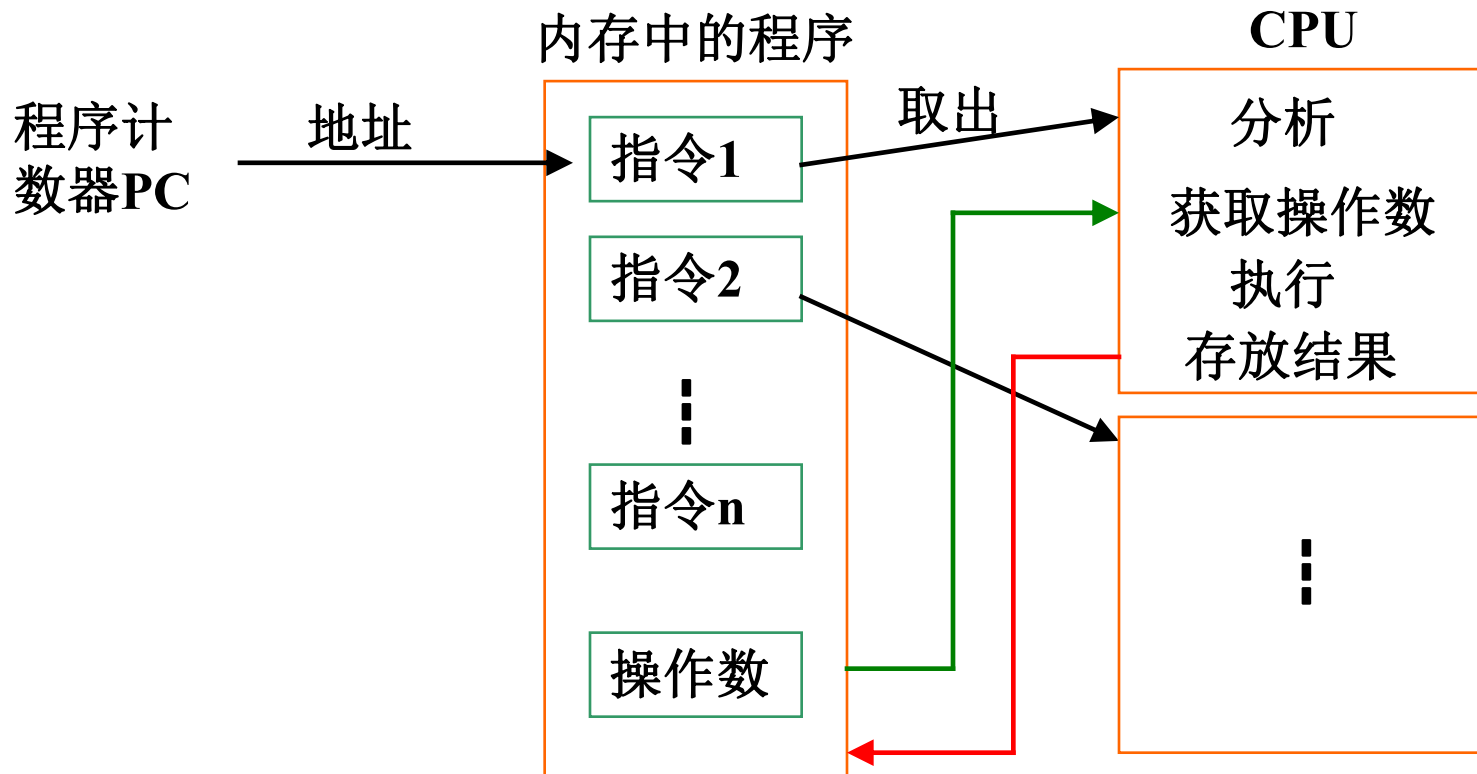
存储程序原理

# 冯·诺依曼计算机结构



# 冯·诺依曼机的工作过程

见P5



# 计算机工作过程：

假定程序已由输入设备存放内存中，当计算机要从停机状态进入运行状态时：

① 首先将第一条指令由内存中取出；

② 将取出的指令送指令译码器译码，以确定要进行的操作；

③ 读取相应的操作数（即执行的对象）；

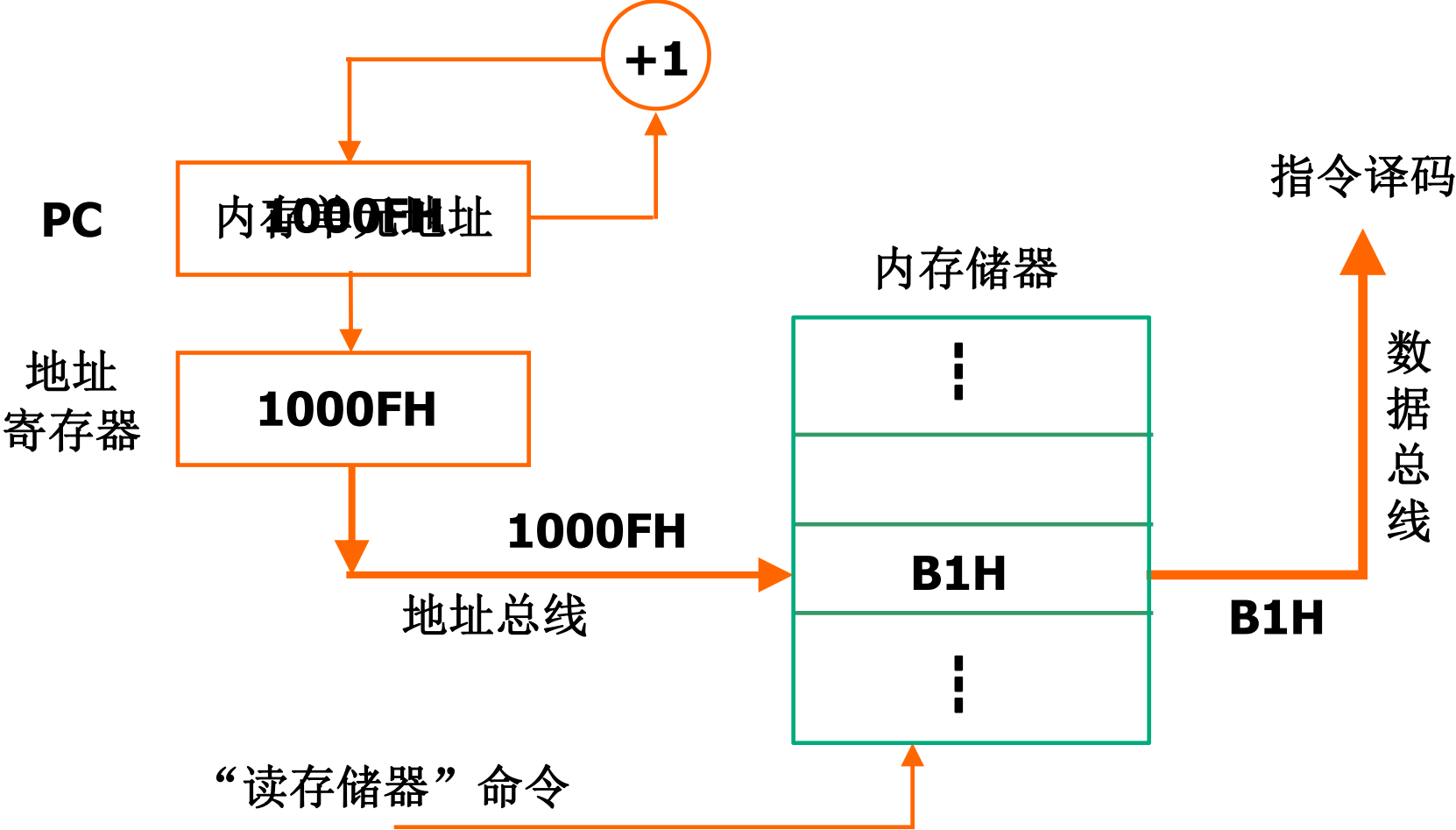
④ 执行指令；

⑤ 存放执行结果；

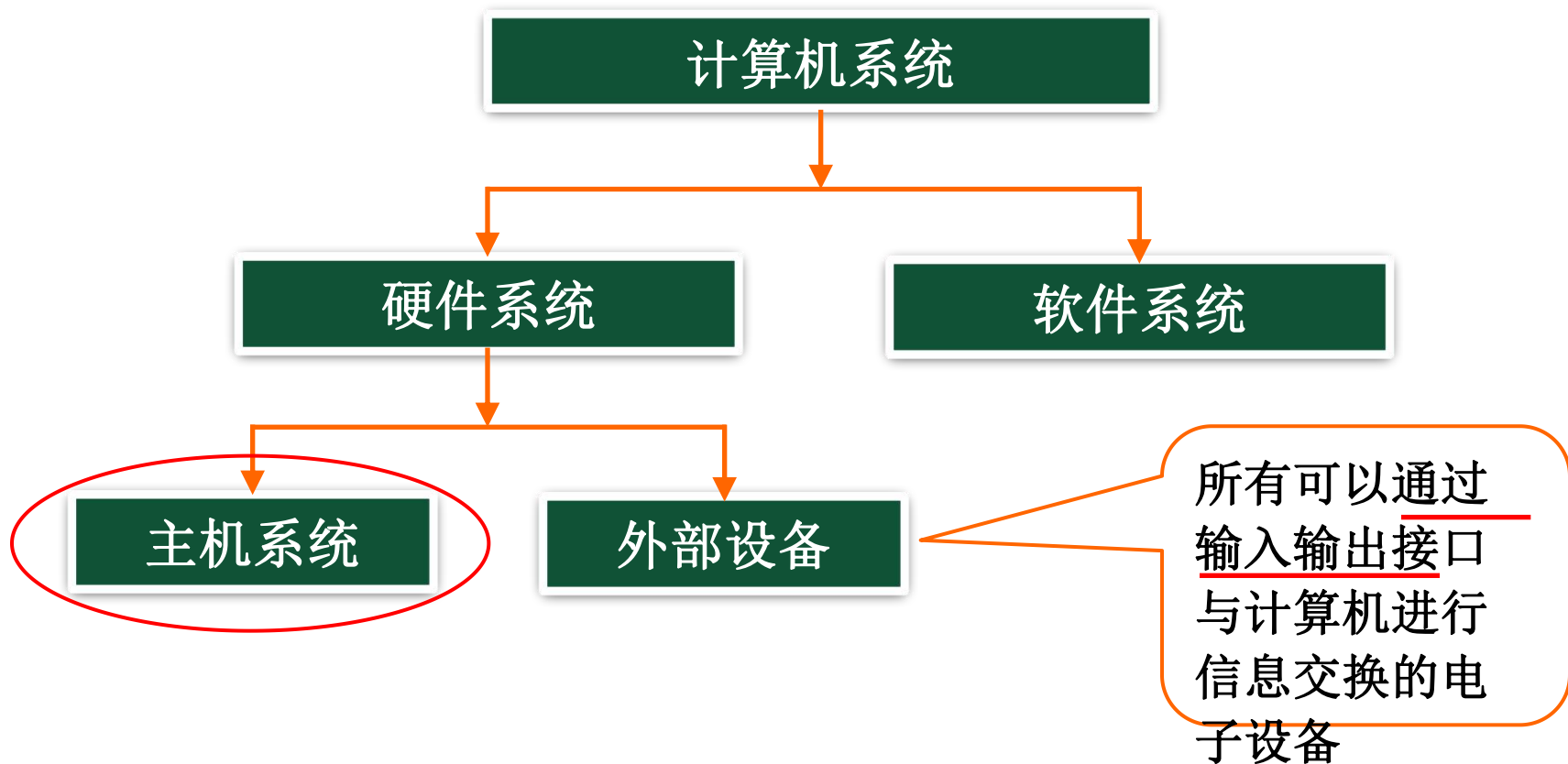
⑥ 一条指令执行完后，转入了下一条指令的取指令阶段，如此周而复始地循环，直到程序中遇到暂停指令方才结束。

取指令阶段都是由一系列相同的操作组成的，所以取指令阶段的时间总是相同的，称为公共操作。而执行阶段则由不同的事件顺序组成，它取决于被执行指令的类型。因此，指令不同，执行阶段所花费的时间也各不相同。

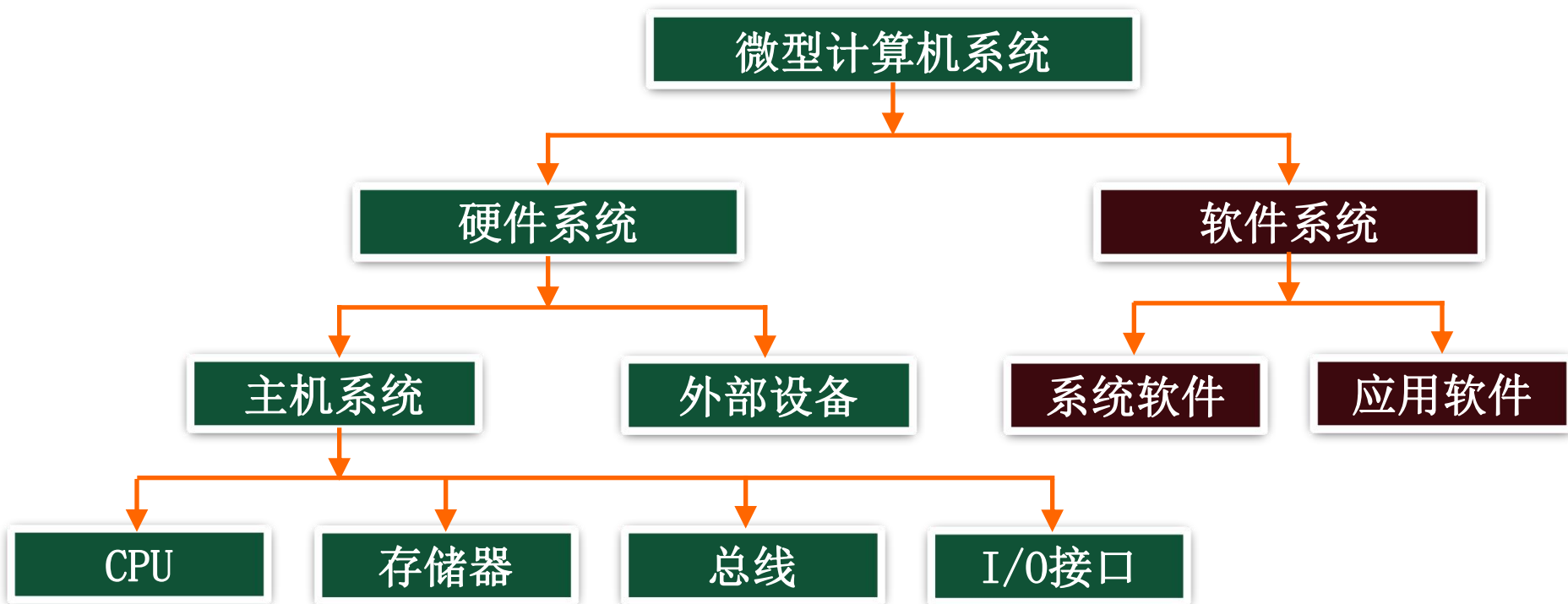
# 微机读取一条指令的工作过程:



# 微机系统组成



# 微机系统概念结构





# 1. 硬件系统      见教材P8

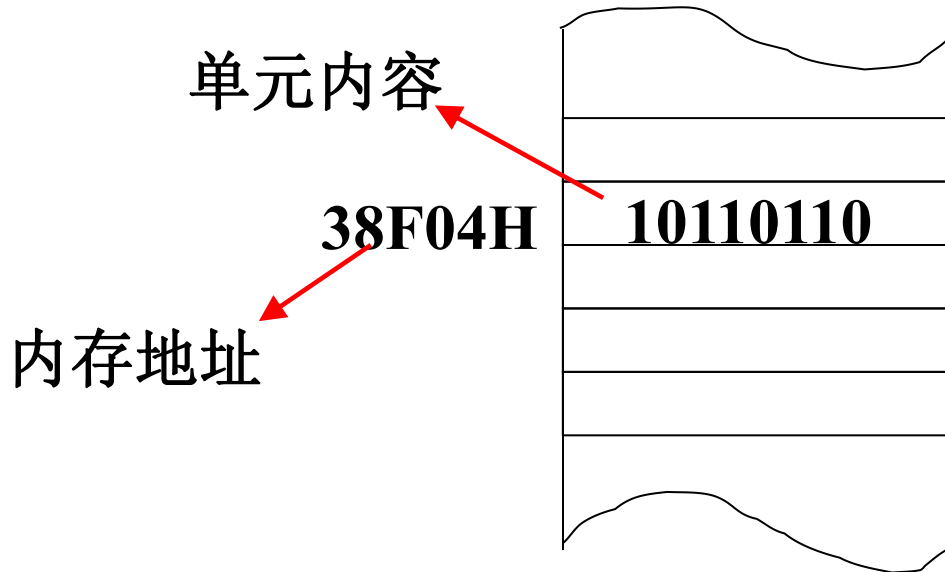
- 1) 微处理器
- 2) 存储器
- 3) 输入输出接口与输入输出设备
- 4) 总线

# 2. 软件系统      见教材P11

- 1) 系统软件
- 2) 应用软件

# 内存单元的地址和内容

- 内存按单元组织
- 每单元都对应一个地址，以方便对单元的寻址



# 1.2 计算机中的数制与编码

数制和编码

二进制

# 计算机中的数制与编码

- 计算机中信息的表示单位：

- **Bit, Byte, Word**

- 计算机中的常用计数制

- 二进制 (B)

- 十进制 (D)

- 十六进制 (H)

- 数制表示例：

- 234.98D或  $(234.98)_D$

- 1101.11B或  $(1101.11)_B$

- ABCD . BFH或  $(ABCD . BF)_H$

# 各种进制数间的转换 教材P14

## ■ 非十进制数到十进制数的转换

- 按相应的权值表达式展开

## ■ 十进制到非十进制数的转换

- 对整数：除 $n$ 取余；
- 对小数：乘 $n$ 取整。

## ■ 二进制与十六进制数之间的转换

- 每4位二进制码对应1位十六进制数
- 整数部分不够4位时在高位（左侧）补0；小数部分不够4位时在低位（右侧）补0

**熟练掌握  
二~十六  
进制转换**

## 1. 非十进制数到十进制数的转换

非十进制数转换为十进制数的方法比较简单,只要将它们按相应的权表达式展开,再按十进制运算规则求和,即可得到它们对应的十进制数。

**【例 1-4】** 将二进制数 1101.101 转换为十进制数。

**解:** 根据二进制数的权展开式,有

$$\begin{aligned}(1101.101)_2 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= (13.625)_{10}\end{aligned}$$

**【例 1-5】** 将十六进制数 64.CH 转换为十进制数。

**解:** 根据十六进制数的权展开式,有

$$\begin{aligned}(64.C)_{16} &= 6 \times 16^1 + 4 \times 16^0 + C \times 16^{-1} = 6 \times 16^1 + 4 \times 16^0 + 12 \times 16^{-1} \\ &= (100.75)_{10}\end{aligned}$$

## 2. 十进制数转换为非十进制数

### 1) 十进制数转换为二进制数

十进制数整数和小数部分应分别进行转换。整数部分转换为二进制数时采用“除 2 取余”的方法,即连续除 2 并取余数作为结果,直至商为 0,得到的余数从低位到高位依次排列即得到转换后二进制数的整数部分;对小数部分,则用“乘 2 取整”的方法,即对小数部分连续用 2 乘,以最先得到的乘积的整数部分为最高位,直至达到所要求的精度或小数部分为 0 为止(可以看出,转换的结果的整数和小数部分是从小数点开始分别向高位和低位逐步扩展)。

**【例 1-6】** 将十进制数 112.25 转换为等值的二进制数。

解：

整数部分	小数部分
$112/2=56\cdots\cdots\text{余数}=0(\text{最低位})$	$0.25\times 2=0.5\cdots\cdots\text{整数}=0(\text{最高位})$
$56/2=28\cdots\cdots\text{余数}=0$	$0.5\times 2=1.0\cdots\cdots\text{整数}=1$
$28/2=14\cdots\cdots\text{余数}=0$	
$14/2=7\cdots\cdots\text{余数}=0$	
$7/2=3\cdots\cdots\text{余数}=1$	
$3/2=1\cdots\cdots\text{余数}=1$	
$1/2=0\cdots\cdots\text{余数}=1$	

从而得到转换结果  $(112.25)_{10} = (1110000.01)_2$ 。

其余转换关系见教材



# 计算机中的编码

- 数值编码：
  - 二进制码
  - BCD码
- 西文字符编码

- ASCII码

使所有信息都以二进制码形式表示

# BCD码

- BCD (Binary Coded Decimal) 码
  - 用二进制表示的十进制数
  - 特点：
    - 保留十进制的权，数字用0和1表示。
- 分类：
  - 压缩BCD码
    - 用4位二进制码表示一位十进制数，每4位之间有一个空格
  - 扩展BCD码
    - 用8位二进制码表示一位十进制数，每8位之间有一个空格

# BCD码与二进制数之间的转换

- 先转换为十进制数，再转换二进制数；反之同样。

- 例：

- $(0001\ 0001.0010\ 0101)_{\text{BCD}}$   
= 11.25  
=  $(1011\ .01)_{\text{B}}$

# ASCII码

- 西文字符编码
- 要求:
  - 熟悉标准ASCII编码
    - 理解校验位的作用
    - 熟悉0---F的ASCII码

# ASCII码的奇偶校验

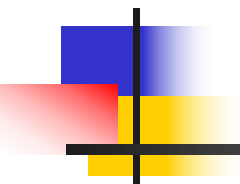
## ■ 奇校验

- 加上校验位后编码中“1”的个数为奇数。
- 例：A的ASCII码是41H（1000001B）
  - 以奇校验传送则为 C1H（11000001B）

## ■ 偶校验

- 加上校验位后编码中“1”的个数为偶数。
  - 上例若以偶校验传送，则为 41H。

## 1.3 无符号二进制数的算术运算与逻辑运算

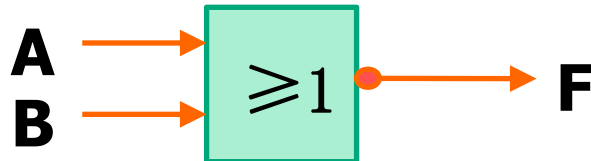
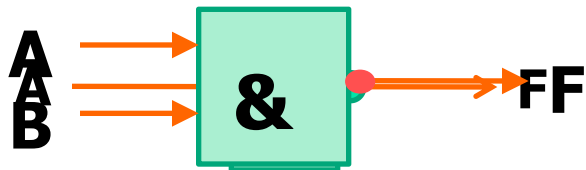
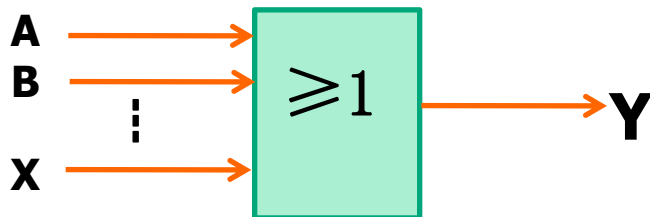
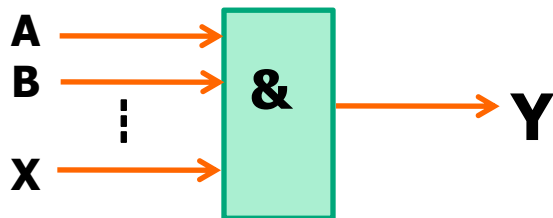


**无符号数的相关知识 见教材P20-22**

# 基本逻辑门 教材P24

## ■ 深入理解:

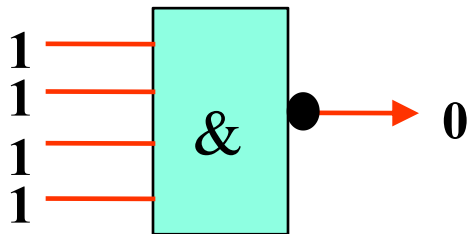
- 与、或、非、异或、与非、或非逻辑关系及逻辑门
- 逻辑图符号，真值表



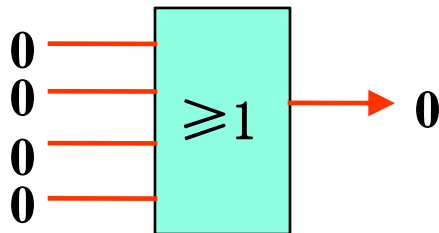


# 逻辑关系例

## ■ 例1



## ■ 例2:

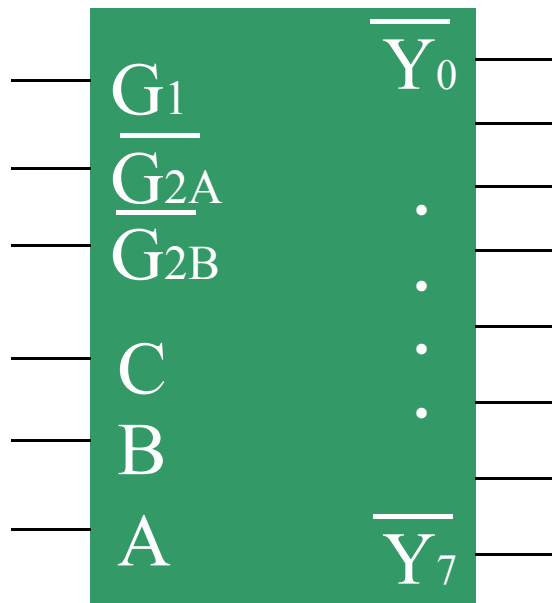


# 译码器

- 掌握74LS138译码器
  - 各引脚功能
  - 输入端与输出端关系（真值表）

# 74LS138译码器

- 主要引脚及功能



## 1.4 带符号二进制数的表示与运算

# 1. 计算机中的数

- 计算机中的数也称为机器数

- 无符号数

- 数中所有的0或1都是数值

- 有符号数

- 最高位是符号位，其余是数值部分

0——正数

1——负数

- 数的性质由设计者决定

- 程序设计时，若定义的操作数据为无符号数，则按照无符号数的处理方法；否则采用有符号数的处理方法。

## 2. 符号数的表示

- 有符号机器数的表示方法：
  - 原码
  - 反码
  - 补码

先搞清几个基本概念：机器数，真值      见教材P27

# 原码

- 最高位为符号位，其余为真值部分。
- 例：
  - $X = -1010110$ ,  $Y = +1010110$
  - $[X]_{\text{原}} = 11010110$ ,  $[Y]_{\text{原}} = 01010110$
- 优点：
  - 真值和其原码表示之间的对应关系简单，容易理解；
- 缺点：
  - 计算机中用原码进行加减运算比较困难
  - 0 的表示不唯一。

# 数0的原码

- 8位数0的原码:

- $+0=0$  0000000

- $-0=1$  0000000

**即：数0的原码不唯一。**



# 反码

对一个机器数X:

- 若 $X > 0$  , 则  $[X]_{\text{反}} = [X]_{\text{原}}$
- 若 $X < 0$ , 则  $[X]_{\text{反}}$  对应原码的符号位不变, 数值部分按位求反。
- 例:
  - $X = -52 = -0110100$   
 $[X]_{\text{原}} = \underline{1} \ 0110100$   
 $[X]_{\text{反}} = \underline{1} \ 1001011$

# 0的反码：

$$[+0]_{\text{反}} = 00000000$$

$$[-0]_{\text{反}} = 11111111$$

**即：数0的反码也不唯一。**

# 补码

定义:

- 若 $X > 0$ , 则 $[X]_{\text{补}} = [X]_{\text{反}} = [X]_{\text{原}}$
- 若 $X < 0$ , 则 $[X]_{\text{补}} = [X]_{\text{反}} + 1$

# 0的补码：

$$[+0]_{\text{补}} = [+0]_{\text{原}} = 00000000$$

$$[-0]_{\text{补}} = [-0]_{\text{反}} + 1 = 11111111 + 1 = \underline{1} \quad 00000000$$

对8位字长，进位被舍掉

# [例]

■  $X = -52 = -0110100$

$$[X]_{\text{原}} = 10110100$$

$$[X]_{\text{反}} = 11001011$$

$$[X]_{\text{补}} = [X]_{\text{反}} + 1 = 11001100$$

# 3. 符号数的算术运算

- 通过引进补码，可将减法运算转换为加法运算。
- 即：
  - $[X+Y]_{\text{补}} = [X]_{\text{补}} + [Y]_{\text{补}}$
  - $[X-Y]_{\text{补}} = [X+(-Y)]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}}$

**注：运算时符号位须对齐**

# 例1：

- $66 - 51 = 66 + (-51) = 15$

- 用二进制补码运算：

- $[+66]_{\text{补}} = [+66]_{\text{原}} = 0100\ 0010$

- $[-51]_{\text{原}} = 1011\ 0011$

- $[-51]_{\text{补}} = 1100\ 1101$

- $[+66]_{\text{补}} + [-51]_{\text{补}} = \boxed{1} \ 0000\ 1111$   
 $= 15$

超出字长  
表示范围

# 例2:

- $X=-52=-0110100$ ,  $Y=116=+1110100$ , 求 $X+Y=?$ 
  - $[X]_{\text{原}}=10110100$
  - $[X]_{\text{补}}=[X]_{\text{反}}+1=11001100$
  - $[Y]_{\text{补}}=[Y]_{\text{原}}=01110100$
  - $[X+Y]_{\text{补}}=[X]_{\text{补}}+[Y]_{\text{补}}$
  - $=11001100+01110100$
  - $=01000000$
  - $X+Y=+1000000$



- 现代计算机系统中，程序设计时，负数可用“-”表示，由编译系统将其转换为补码。
- 例：
  - 若输入数=-3
  - 程序编译后的值=FDH

# 特殊数10000000

- 对无符号数：
  - $(10000000)_B = 128$
- 在原码中定义为：
  - $(10000000)_B = -0$
- 在反码中定义为：
  - $(10000000)_B = -127$
- 在补码中定义为：
  - $(10000000)_B = -128$

# 4. 计算机能力的局限性

- 计算机的运算能力是有限的
  - 计算机无力解决无法设计出算法的问题
  - 无法处理无穷运算或连续变化的信息
- 计算机的表数范围是有限的
  - 计算机的表数范围受字长的限制
  - 例：对8位机：
    - 无符号数的最大值：1111 1111
    - 有符号正数的最大值：0111 1111

当运算结果超出计算机表数范围时，将产生溢出


# (1) 无符号数的表示范围：

$$0 \leq X \leq 2^n - 1$$

若运算结果超出这个范围，则产生溢出。

- 计算机判断无符号数加减运算溢出的方法：
  - 运算时，当最高位向更高位有进位（或借位）时则产生溢出。

**[例] :**

$$\begin{array}{r} 11111111 \\ + 00000001 \\ \hline 1\ 00000000 \end{array}$$


最高位向前有进位，产生溢出

## (2) 符号数的表示范围

- 原码和反码:

- $-(2^{n-1} - 1) \leq X \leq 2^{n-1} - 1$

- 补码:

- $-2^{n-1} \leq X \leq 2^{n-1} - 1$

- 对8位二进制数:

- 原码:  $-127 \sim +127$

- 反码:  $-127 \sim +127$

- 补码:  $-128 \sim +127$

# 符号数运算中的溢出判断

- 两个带符号二进制数相加或相减时，若运算结果超出可表达范围，则产生溢出
- 溢出的判断方法：
  - 最高位进位状态 $\oplus$ 次高位进位状态 $=1$ ，则结果溢出（“异或”运算相同则为0，相异则为1）

## [例] :

- 若:  $X=01111000$ ,  $Y=01101001$

则:  $X+Y=$

$$\begin{array}{r} 01111000 \\ + 01101001 \\ \hline 11100001 \end{array}$$

- 次高位向最高位有进位，而最高位向前无进位，产生溢出。  
(事实上，两正数相加得出负数，结果出错)



# 5. 符号二进制数与十进制的转换

- 转换方法：
  - 求出真值
  - 进行转换
- 计算机中的符号数默认以补码形式表示。

原码=符号位+绝对值

正数的补码=原码=符号位+绝对值

负数的补码 $\neq$ 原码。所以：负数的补码 $\neq$ 符号位+绝对值

# 例：补码数转换为十进制数

- $[X]_{\text{补}} = \underline{0} \ 0101110\text{B}$

↑  
正数

- 所以：

- $X\text{的真值} = 0101110\text{B} = +46$

- $[X]_{\text{补}} = \underline{1} \ 1010010\text{B}$

↑  
负数

- 所以：真值不等于  $-1010010\text{B}$

- 而是：

- $X = [[X]_{\text{补}}]_{\text{补}} = [11010010]_{\text{补}} = - \ 0101110 = - \ 46$

**只有原码的数值部分是真值**

**反码和补码的数值部分都不是真值**

**因正数的反码、补码与其对应的原码相同，  
故其数值部分亦为真值。**

# 结束语：

- 难点及要点：
  - 补码的概念及其运算
  - 基本逻辑门及其逻辑关系

