# 计算机组成原理

## 第1章 计算机系统概述

### 1.1计算机发展历程

自从1946年诞生人类第一台电子计算机以来，计算机的发展经历了4代：第1代（1946~1957）——电子管时代

第2代（1957~1964）——晶体管时代

第3代（1965~1971）——中小规模集成电路时代

第4代（1972至今）——大规模集成电路、超大规模集成电路时代

### 1.2计算机系统层次结构

#### 1.2.1计算机系统层次结构

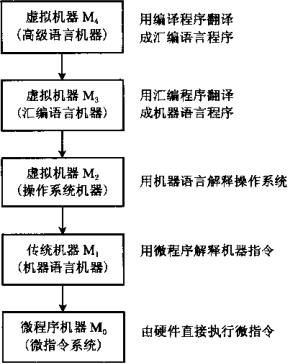


图1.3 多级层次结构的计算机系统

#### 1.2.2计算机硬件的基本组成

1. 冯●诺依曼体系结构

它是存储程序的计算机，其特点是：

（1）计算机由控制器、运算器、存储器、输入设备和输出设备五大部件组成。

（2）指令和数据以同等地位存放于存储器内，并可按地址寻访。

（3）指令和数据均用二进制表示。

（4）指令由操作码和地址码组成，操作码用来表示操作的性质，地址码用来表示操作数在存储器中的位置。

（5）指令在存储器内按顺序存放。通常，指令是顺序执行的，在特定条件下，可根据运算结果或根据设定的条件改变执行顺序。

（6）机器以运算器为中心，输入输出设备与存储器间的数据传送通过运算器完成。

2. 计算机硬件框图

典型的冯●诺依曼计算机是以运算器为中心的，如图1.1所示。

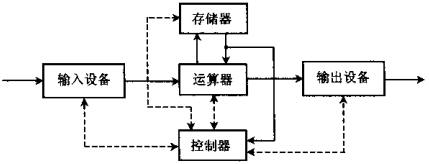


图1-1 典型的冯●诺依曼计算机结构框图

现代的计算机已转化为一存储器为中心，如图1.2所示。

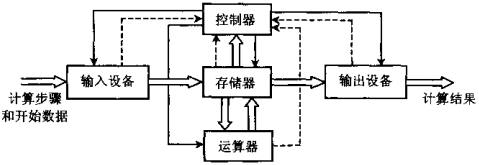


图1-2 以存储器为中心的计算机结构框图

通常把运算器和控制器集成在一个芯片上，称为**中央处理器**，简称**CPU**。CPU和主存储器在一起组成**主机**。除去主机以外的硬件（如输入输出设备、硬盘等）统称为**外部设备**。

#### 1.2.3计算机软件的分类

分为系统软件和应用软件

#### 1.2.4计算机的工作过程

上机前准备：

1.建立数学模型；2.确定计算方法；3.编制解题程序；

上机运行：

4.调试运行程序；5.输出运算结果。

### 1.3计算机硬件性能指标

1.机器字长

机器字长是指CPU一次能处理的数据的位数，通常与CPU的寄存器位数有关。

2.运算速度

有两种衡量指标，一种是CPU主频，用CPI（Cycle Per Instruction）即执行一条指令所需的时钟周期（CPU主频的倒数）数作计量单位。

另一种是单位时间内执行指令的条数，用MIPS（Million Instructions Per Second）或MFLOPS（Million Floating Point Operations Per Second）作为计量单位。

3.存储容量

包括主存容量和辅存容量。

## 第2章 数据的表示和运算

### 2.1数制及其转换

#### 2.1.1进位计数制



#### 2.1.2数制转换

1. r进制转换成十进制

按权展开

2. 十进制转换成r进制

整数部分：除r取余，余数倒排法

小数部分：乘r取整，整数顺排法

3. 二进制转换成8、16进制

将二进制数自小数点开始左右两边分为3位或4位一段，每段转换成相应的一位8进制或16进制数即可。

4. 8、16进制转换成二进制

将8进制或16进制数自小数点开始左右两边每位转换成3位或4位二进制数即可。

### 2.2非数值数据的表示

#### 2.2.1字符的表示

ASCII码

#### 2.2.2汉字的表示

输入码，如拼音输入法；

机内码，如GB18030标准，一个汉字用1B、2B或4B表示；

字模码，即汉字点阵。

#### 2.2.3 BCD码

用4位二进制数表示一个十进制数数位，每个数位内部满足二进制，二数位之间满足十进制，故称这种编码为“以二进制编码的十进制(Binary Coded Demical, **BCD**)”码。

常用的BCD码分为有权码和无权码。常用的**有权码**有8421码、2421码、5211码、4311码，常用的**无权码**有余3码、格雷码。

1. 8421码

4个二进制码的权从高到低为8、4、2、1。

如果两个一位8421码相加之和小于或等于(1001)2，即9，不需要修正；如果相加之和大于或等于(10)10，要进行加6修正，并向高位进位。

1. 余3码

余3码是在8421码的基础上，把每个编码都加上0011而形成的，其运算规律是：

当两个余3码相加不产生进位时，应从结果中减去0011；产生进位时，应向高位进位，本位加0011。

### 2.3数值数据的表示

#### 2.3.1真值与机器数

在计算机中参与运算的数有两大类：无符号数和有符号数。

对有符号数而言，符号的“正”、“负”机器是无法识别的，但由于“正”、“负”恰好是两种状态，如果用0表示正，1表示负，这样符号也被数字化了，并且规定将它放在有效数字的前面，即组成了有符号数。

把符号数字化的数称为**机器数**，把带“+”、“-”号的数称为**真值**，即机器数的实际值。

#### 2.3.2机器数的表示

为了书写方便以及区别整数和小数，约定整数的符号位和数值位之间用逗号隔开，小数的符号位和数值位之间用小数点隔开。

**1.原码**

机器数最高位为符号位，0表示正数，1表示负数，绝对值跟随其后。

整数原码的定义为

式中，x为真值，n为整数的位数，此时原码的表示范围是[-(2n-1), 2n-1]。

小数原码的定义为

式中，x为真值，若小数点后位数为n，则原码的表示范围是[-(1-2-n), 1-2-n]。

[0, 0000000]=+0，[1, 0000000]=-0，

[0. 0000000]=+0.0，[1. 0000000]=-0.0，

可见，原码中零有两种表示形式。

**2.补码**

机器数最高位为符号位，0表示正数，1表示负数。

整数补码的定义为

式中，x为真值，n为整数的位数，此时补码的表示范围是[-2n, 2n-1]。

小数补码的定义为

式中，x为真值，若小数点后位数为n，则补码的表示范围是[-1, 1-2-n]。

引入补码的概念是为了消除减法，但是根据补码的定义，在形成补码的过程中又出现了减法。

设负数x用-x1x2x3x4表示，于是

[x]补=25+x=11111+00001- x1x2x3x4=11111 - x1x2x3x4 + 00001 =1 + 00001

可见求整数的补码，可以看作对其原码除符号位外，每位取反，末位加1，简称“**取反加1**”。同理，对于小数也有同样的结论。

“取反加1”同样适用于已知补码求原码。

[x]补 ↔ [-x]补，符号位取反，数值位取反加1。

对于一个负数，若对其原码除符号位外，每位取反（简称“每位取反”），或是对其补码减去末位的1，就得到其反码。

**3.反码**

反码通常用来作为由原码求补码或者由补码求反码的中间过渡。

整数反码的定义为

式中，x为真值，n为整数的位数，此时反码的表示范围是[-(2n-1), 2n-1]。

小数反码的定义为

式中，x为真值，若小数点后位数为n，则反码的表示范围是[-(1-2-n), 1-2-n]。

**4.移码**

补码表示法，由于与习惯上的表示法不同，人们很难一眼直接判断出其真值大小，例如

[21]补=0, 10101 [21]补=1, 01011

如果对每个真值加上一个2n，情况就发生了变化，

[21]移=110101 [21]移=001011

很明显，110101>001011。

由此可得移码的定义[x]移=2n+x, -2n≤x<2n，式中，x为真值，n为整数的位数。

0在原码、反码中表示不唯一，在补码、移码中表示唯一。

移码和补码仅差一个符号位。

例3.1 设机器字长为8位，其中一位为符号位，当其分别表示无符号数、原码、补码、反码、移码时，对应的真值范围各是多少？

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 二进制代码 | 无符号数 | 原码 | 补码 | 反码 | 移码 |
| 00000000 | 0 | +0 | 0 | 0 | -128 |
| 00000001 | 1 | 1 | 1 | 1 | -127 |
| 00000010 | 2 | 2 | 2 | 2 | -126 |
| ... | … | … | … | … | … |
| 01111110 | 126 | 126 | 126 | 126 | -2 |
| 01111111 | 127 | 127 | 127 | 127 | -1 |
| 10000000 | 128 | -0 | -128 | -127 | 0 |
| 10000001 | 129 | -1 | -127 | -126 | 1 |
| 10000010 | 130 | -2 | -126 | -125 | 2 |
| ... | ... | ... | ... | ... | ... |
| 11111110 | 254 | -126 | -2 | -1 | 126 |
| 11111111 | 255 | -127 | -1 | -0 | 127 |

例3.2 设机器字长为8位，其中一位为符号位，当其在小数定点机中分别表示原码、补码、反码时，对应的真值范围各是多少？

|  |  |  |  |
| --- | --- | --- | --- |
| 二进制代码 | 原码 | 补码 | 反码 |
| 00000000 | +0.0 | 0.0 | +0.0 |
| 00000001 | 2-7 | 2-7 | 2-7 |
| 00000010 | 2-6 | 2-6 | 2-6 |
| ... | … | … | … |
| 01111110 | 1-2-6 | 1-2-6 | 1-2-6 |
| 01111111 | 1-2-7 | 1-2-7 | 1-2-7 |
| 10000000 | -0.0 | -1.0 | -(1-2-7) |
| 10000001 | -2-7 | -(1-2-7) | -(1-2-6) |
| 10000010 | -2-6 | -(1-2-6) | -(1-2-5+ 2-7) |
| ... | ... | ... | ... |
| 11111110 | -(1-2-6) | -2-6 | -2-7 |
| 11111111 | -(1-2-7) | -2-7 | -0.0 |

#### 2.3.3定点数和浮点数

**1.定点数**

定点数即小数点的位置固定的数。

当小数点位于数符和第一数值位之间是机器数时，机器数是纯小数；当小数点位于数值位之后时，机器数为纯整数。上一节已经给出了原码、补码、反码的定点数的表示范围。

**2.浮点数**

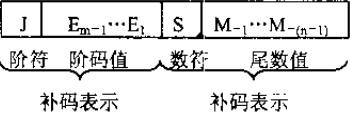
浮点数即小数点的位置不固定的数。

（1）浮点数的格式

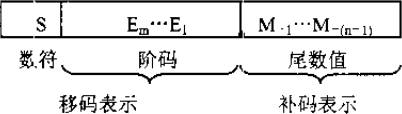
对任意一个二进制数N，总可以写成N=M ×rE，式中，M为尾数，可正可负，E为阶码，可正可负，r是基数。在计算机中基数一般取2。

计算机中的浮点数格式有好几种，常用的格式有以下两种。

①第一种浮点格式



②第二种浮点格式



（2）浮点数的规格化

为了提高运算精度，需要充分地利用尾数的数位，通常采用浮点数规格化形式，即规定尾数的最高数位必须是一个有效值。

当基数为2时，规格化时，尾数左移一位，阶码减1，称为左规；尾数右移一位，阶码加1，称为右规。

原码规格化后，正数为0.1xx...x的形式，负数为1.1xx...x的形式；补码规格化后，正数为0.1xx...x的形式，负数为1.0xx...x的形式。

**3.IEEE 754 标准**

现代计算机中，浮点数一般采用IEEE制定的国际标准，这种标准形式如下：

|  |  |  |
| --- | --- | --- |
| S | 阶码（含符号位） | 尾数 |

↑ ↑

数符 小数点位置

按IEEE标准，常用的浮点数有三种：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 符号位S | 阶码 | 尾数 | 总位数 |
| 短实数 | 1 | 8 | 23 | 32 |
| 长实数 | 1 | 11 | 52 | 64 |
| 临时实数 | 1 | 15 | 64 | 80 |

其中，S为数符，它表示浮点数的正负，但与其有效位（尾数）是分开的。阶码用移码表示，阶码的真值都被加上一个常数（偏移量），如短实数、长实数和临时实数的偏移量分别为7FH、3FFH和3FFFH。尾数部分通常都是规格化**原码**表示，即非“0”的有效位最高位总是“1”，但在IEEE标准中，有效位呈如下形式。

1▲xx...x

其中▲表示假想的二进制小数点。在实际表示中，对短实数和长实数，这个整数位的1省略，称为隐藏位；对于临时实数不采用隐藏位，表3.1列出了十进制数178.125的实数表示。

表3.1 实数178.125的IEEE 754 表示

|  |  |  |  |
| --- | --- | --- | --- |
| 实数表示 | 数值 | | |
| 原始十进制数 | 178.125 | | |
| 二进制数 | 10110010.001 | | |
| 二进制浮点表示 | 1.0110010001×2111 | | |
| 短实数表示 | 数符 | 阶码 | 尾数 |
| 0 | 00000111+01111111  =10000110 | 01100100010000000000000 |

### 2.4定点数运算

#### 2.4.1移位运算

无符号数的移位称为逻辑移位，有符号数的移位称为算术移位。

算数移位规则：

表3.2 不同码制机器数算数移位后的空位填补规则

|  |  |  |
| --- | --- | --- |
| 真值 | 码制 | 填补代码 |
| 正数 | 原码、补码、反码 | 0 |
| 负数 | 原码 | 0 |
| 补码 | 左移添0，右移添1 |
| 反码 | 1 |

由表3.2可得出如下结论。

1.机器数为正时，不论是左移还是右移，填补代码均为0。

2.由于负数的原码数值部分和真值相同，故在移位时只要使符号位不变，其空位均填0即可。

3.由于负数的反码各位除符号位外与负数的原码正好相反，故移位后所填的代码应与原码相反，即全部填1。

4.分析任意负数的补码可发现，当对其由低位向高位找到第一个“1”时，在此“1”的左边的各位均与对应的反码相同，而在此“1”右边的各位（包括此“1”在内）均与对应的原码相同。故负数的补码左移时，因空位出现在低位，故填补的代码与原码相同，即填0；右移时因空位出现在高位，则填补的代码与反码相同，即填1。

对于正数，左移时最高数位丢1，结果出错；右移时最低数位丢1，影响精度。

对于负数，负数的原码左移时，高位丢1，结果出错；右移时，低位丢1，影响精度。负数的反码左移时，高位丢0，结果出错；右移时，低位丢0，影响精度。负数的补码左移时，高位丢0，结果出错；低位丢1，影响精度。

#### 2.4.2加减法运算

**1.原码加减法运算**

加法规则：先判断符号位，若相同，绝对值相加，结果符号不变；若不同，则作减法，绝对值大的数减去绝对值小的数，结果符号与绝对值大的数相同。

减法规则：先将减数符号取反，然后按被减数与符号取反后的减数按原码加法进行运算。

**2.补码加减法运算**

补码加法的基本公式：

整数 [A]补+[B]补=[A+B]补(mod2n+1)

小数 [A]补+[B]补=[A+B]补(mod2)

补码减法的基本公式：

整数 [A-B]补=[A]补+[-B]补(mod2n+1)

小数 [A-B]补=[A]补+[-B]补(mod2)

**3.溢出判断**

补码加减法判断溢出有以下3个方法：

（1）单符号位法

由于减法运算在机器中是用加法器实现的，因此可得出如下结论：不论是做加法还是减法，只要实际参加操作的两个数符号相同，结果又与原操作数的符号不同，即为溢出。

（2）双符号位法

变形补码的定义[x]补=

变形补码判断溢出的原则是：当2位符号位不同时，表示溢出，否则，无溢出。不论是否发生溢出，高位符号位永远代表真正的符号。符号位为“01”表示上溢，符号位为“10”表示下溢。

（3）进位判断法

符号位产生进位与最高数值位产生进位进行异或，若结果为1，表示溢出，若为0，无溢出。若符号位无进位而最高数值位有进位，表示上溢；若符号位有进位而最高数值位无进位，表示下溢。

#### 2.4.3乘法运算

**1.原码一位乘**

以小数为例：

设 [x]原=x0.x1x2...xn

[y]原=y0.y1y2...yn

则[x]原●[y]原=x0⊕y0.(0.x1x2...xn)(0.y1y2...yn)

式中，x0.x1x2...xn为x的绝对值，y0.y1y2...yn为y的绝对值。

原码一位乘的运算规则如下：

（1）乘积的符号位由两原码符号位异或运算的结果决定。

（2）乘积的数值部分由两数绝对值相乘，其通式为



再令zi表示第i次部分积，上式可写成如下递推公式。

z0=0

z1=2-1(yn●|x|+z0)

z2=2-1(yn-1●|x|+z1)

...

zi=2-1(yn-i+1|x|+zi-1)

...

zn=2-1(y1●|x|+zn-1)

例3.3 已知x = - 0.1110, y = - 0.1101，求[x●y]原。

【解】：[x]原=1.1110，|x|=0.1110（绝对值），符号为负

[y]原=1.1101，|y|=0.1101（绝对值），符号为负

按原码一位乘的运算规则，[x●y]原的计算过程如表3.3所示。

表3.3原码一位乘计算过程

|  |  |  |
| --- | --- | --- |
| 部分积 | 乘数 | 说明 |
| 0. 0 0 0 0  + 0. 1 1 1 0 | 1 1 0 1 | 开始部分积z0=0  乘数为1，加上|x| |
| 0. 1 1 1 0  0. 0 1 1 1  + 0. 0 0 0 0 | 0 1 1 0 | →1位得z1，乘数同时→1位  乘数为0，加上0 |
| 0. 0 1 1 1  0. 0 0 1 1  + 0. 1 1 1 0 | 0  1 0 1 1 | →1位得z2，乘数同时→1位  乘数为1，加上|x| |
| 1. 0 0 0 1  0. 1 0 0 0  + 0. 1 1 1 0 | 1 0  1 1 0 1 | →1位得z3，乘数同时→1位  乘数为1，加上|x| |
| 1. 0 1 1 0  0. 1 0 1 1 | 1 1 0  0 1 1 0 | →1位得z4，乘数已经全部移出 |

故[x●y]原 = 0.10110110

**2.补码一位乘**

一般只会考Booth算法。

比较法是Booth夫妇首先提出来的，故又称Booth算法，它的运算规则可由校正法导出。

设 [x]补=x0.x1x2...xn

[y]补=y0.y1y2...yn

按补码乘法校正法规则，其基本算法可用一个统一的公式表示为

[x●y]补=[x]补( 0.y1y2...yn)-[x]补●y0 （6.14）

当y0=0时，表示乘数y为正，无需校正，即

[x●y]补=[x]补( 0.y1y2...yn) （6.15）

当y0=1时，表示乘数y为负，则

[x●y]补=[x]补( 0.y1y2...yn)-[x]补  （6.16）

式（6.14）可以改写为

[x●y]补=[x]补(y12-1+y22-2+...+yn2-n) -[x]补●y0

=[x]补(-y0+ y12-1+y22-2+...+yn2-n)

=[x]补[-y0+ (y1-y12-1)+ (y22-1-y22-2)+...+ (yn2-(n-1) -yn2-n)]

=[x]补[(y1-y0)+ (y2-y1) 2-1+...+ (yn- yn-1)2-(n-1) +(0-yn)2-n)]

=[x]补[(y1-y0)+ (y2-y1) 2-1+...+ (yn+1-yn)2-n)]

其中，yn+1=0。

这样，可得如下地推公式。

[z0]补=0

[z1]补=2-1{[z0]补+ (yn+1-yn)[x]补}

[z2]补=2-1{[z1]补+ (yn-yn-1)[x]补}

...

[zi]补=2-1{[zi-1]补+ (yn-i+2-yn-i+1)[x]补}

...

[zn]补=2-1{[zn-1]补+ (y2-y1)[x]补}

[x●y]补=[zn+1]补= [zn]补+ (y1-y0)[x]补

由此可见，开始时yn+1=0，部分积初值[z0]补=0，每一步乘法由(yi+1-yi)(i=1,2,...,n)决定原部分积加[x]补或加[-x]补或加0，再右移一位得新的部分积，由此重复n步。第n+1步由(y1-y0)决定原部分积加[x]补或加[-x]补或加0，但不移位，即得[x●y]补。

这里的(yi+1-yi)之差恰恰与乘数末两位yi及yi+1的状态对应，对应的操作如表3.4所示。当运算至最后一步时，乘积不再右移。这样的运算规则计算机很容易实现。

表3.4 yiyi+1的状态对应的操作

|  |  |  |
| --- | --- | --- |
| yiyi+1 | yi+1-yi | 操作 |
| 00  01  10  11 | 0  1  -1  0 | 部分积右移一位  部分积加[x]补，再右移一位  部分积加[-x]补，再右移一位  部分积右移一位 |

按Booth算法进行补码乘法时，像补码加、减法一样，符号位也参与运算。

例3.4 已知[x]补=0.1101，[y]补=0.1011，求[x\*y]补。

【解】：[-x]补=1.0011，表3.5列出了计算过程。

表3.5 补码一位乘(Booth算法)计算过程

|  |  |  |  |
| --- | --- | --- | --- |
| 部分积 | 乘数yn | 附加位yn+1 | 说明 |
| 0 0. 0 0 0 0  + 1 1. 0 0 1 1 | 0 1 0 1 1 | 0 | 初值[z0]补=0  ynyn+1=10，部分积加[-x]补 |
| 1 1. 0 0 1 1  1 1. 1 0 0 1  1 1. 1 1 0 0  + 0 0. 1 1 0 1 | 1 0 1 0 1  1 1 0 1 0 | 1  1 | →1位得[z1]补  ynyn+1=11，部分积→1位得[z2]补  ynyn+1=01，部分积加[x]补 |
| 0 0. 1 0 0 1  0 0. 0 1 0 0  + 1 1. 0 0 1 1 | 1 1  1 1 1 0 1 | 0 | →1位得[z3]补  ynyn+1=10，部分积加[-x]补 |
| 1 1. 0 1 1 1  1 1. 1 0 1 1  + 0 0. 1 1 0 1 | 1 1 1  1 1 1 1 0 | 1 | →1位得[z4]补  ynyn+1=01，部分积加[x]补 |
| 0 0. 1 0 0 0 | 1 1 1 1 |  | 最后一步不移位，得[x\*y]补 |

故[x\*y]补=0.10001111

例3.5 已知[x]补=1.0101，[y]补=1.0011，求[x\*y]补。

【解】：[-x]补=00.1011，表3.6列出了计算过程。

表3.6 补码一位乘(Booth算法)计算过程

|  |  |  |  |
| --- | --- | --- | --- |
| 部分积 | 乘数yn | 附加位yn+1 | 说明 |
| 0 0. 0 0 0 0  + 0 0. 1 0 1 1 | 1 0 0 1 1 | 0 | 初值[z0]补=0  ynyn+1=10，部分积加[-x]补 |
| 0 0. 1 0 1 1  0 0. 0 1 0 1  0 0. 0 0 1 0  + 1 1. 0 1 0 1 | 1 1 0 0 1  1 1 1 0 0 | 1  1 | →1位得[z1]补  ynyn+1=11，部分积→1位得[z2]补  ynyn+1=01，部分积加[x]补 |
| 1 1. 0 1 1 1  1 1. 1 0 1 1  1 1. 1 1 0 1  + 0 0. 1 0 1 1 | 1 1  1 1 1 1 0  1 1 1 1 1 | 0  0 | →1位得[z3]补  ynyn+1=00，部分积→1位得[z4]补  ynyn+1=10，部分积加[-x]补 |
| 0 0. 1 0 0 0 | 1 1 1 1 |  | 最后一步不移位，得[x\*y]补 |

故[x\*y]补=0.10001111

#### 2.4.4除法运算

**1.原码一位除（加减交替法）**

加减交替法比恢复余数法考的概率大。

加减交替法要点如下：

当Ri>0，商上“1”，做2Ri-|y|的运算。

当Ri<0，商上“0”，做2Ri+|y|的运算。

例3.6 已知x=-0.1011，y=0. 1101，求[x/y]原。

【解】：由已知条件得

[x]原=1.1011，|x|=0.1011，

[y]原=0. 1101，|y|=0. 1101，[-|y|]补=1.0011

表3.7列出了计算过程。

表3.7 加减交替法的计算过程

|  |  |  |
| --- | --- | --- |
| 被除数（余数） | 商 | 说明 |
| 0. 1 0 1 1  + 1. 0 0 1 1 | 0. 0 0 0 0 | +[-|y|]补（减除数） |
| 1. 1 1 1 0  1. 1 1 0 0  + 0. 1 1 0 1 | 0  0 | 余数为负，上商“0”  ←1位  +[|y|]补（加除数） |
| 0. 1 0 0 1  1. 0 0 1 0  + 1. 0 0 1 1 | 0 1  0 1 | 余数为正，上商“1”  ←1位  +[-|y|]补（减除数） |
| 0. 0 1 0 1  0. 1 0 1 0  + 1. 0 0 1 1 | 0 1 1  0 1 1 | 余数为正，上商“1”  ←1位  +[-|y|]补（减除数） |
| 1. 1 1 0 1  1. 1 0 1 0  + 0. 1 1 0 1 | 0 1 1 0  0 1 1 0 | 余数为负，上商“0”  ←1位  +[|y|]补（加除数） |
| 0. 0 1 1 1 | 0 1 1 0 1 | 余数为正，上商“1” |

商的符号为两数的符号异或，x0⊕y0=1，故[x/y]原=1.1101

**2.补码一位除（加减交替法）**

表3.8 新余数的算法

|  |  |  |
| --- | --- | --- |
| [Ri]补与[y]补 | 商 | 新余数[Ri+1]补 |
| 同号 | 1 | [Ri+1]补=2[Ri]补+[-y]补 |
| 异号 | 0 | [Ri+1]补=2[Ri]补+[y]补 |

如果对商的精度没有特殊要求，一般可采用“末位恒置1”的方法，这种方法操作简单，易于实现，而且最大误差仅为2-n。

例3.7 已知x=0.1001，y=0. 1101，求[x/y]补。

【解】：由已知条件得

[x]补=0.1001，[y]补=0.1101，[-y]补=1.0011

表3.9列出了计算过程。

表3.9补码一位除计算过程

|  |  |  |
| --- | --- | --- |
| 被除数（余数） | 商 | 说明 |
| 0. 1 0 0 1  + 1. 0 0 1 1 | 0. 0 0 0 0 | [x]补与[y]补同号，+[-y]补 |
| 1. 1 1 0 0  1. 1 0 0 0  + 0. 1 1 0 1 | 0  0 | [R]补与[y]补异号，上商“0”  ←1位  +[y]补 |
| 0. 0 1 0 1  0. 1 0 1 0  + 1. 0 0 1 1 | 0 1  0 1 | [R]补与[y]补同号，上商“1”  ←1位  +[-y]补 |
| 1. 1 1 0 1  1. 1 0 1 0  + 0. 1 1 0 1 | 0 1 0  0 1 0 | [R]补与[y]补异号，上商“0”  ←1位  +[y]补 |
| 0. 0 1 1 1  0. 1 1 1 0 | 0 1 0 1  0 1 0 1 1 | [R]补与[y]补同号，上商“1”  ←1位，末位恒置“1” |

故[x/y]补=0.1011

例3.8 已知x=-0.1001，y=0. 1101，求[x/y]补。

【解】：由已知条件得

[x]补=1.0111，[y]补=0.1101，[-y]补=1.0011

表3.10列出了计算过程。

表3.10补码一位除计算过程

|  |  |  |
| --- | --- | --- |
| 被除数（余数） | 商 | 说明 |
| 1. 0 1 1 1  + 0. 1 1 0 1 | 0. 0 0 0 0 | [x]补与[y]补异号，+[-y]补 |
| 0. 0 1 0 0  0. 1 0 0 0  + 1. 0 0 1 1 | 1  1 | [R]补与[y]补同号，上商“1”  ←1位  +[-y]补 |
| 1. 1 0 1 1  1. 0 1 1 0  + 0. 1 1 0 1 | 1 0  1 0 | [R]补与[y]补异号，上商“0”  ←1位  +[y]补 |
| 0. 0 0 1 1  0. 0 1 1 0  + 1. 0 0 1 1 | 1 0 1  1 0 1 | [R]补与[y]补同号，上商“1”  ←1位  +[-y]补 |
| 1. 1 0 0 1  1. 0 0 1 0 | 1 0 1 0  1 0 1 0 1 | [R]补与[y]补异号，上商“0”  ←1位，末位恒置“1” |

故[x/y]补=1.0101

### 2.5浮点数运算

#### 2.5.1加减法运算

浮点数加减法运算有以下几步：

1. 对阶，使两数的小数点位置对齐。一般按小阶向大阶看齐的原则。
2. 尾数运算，将对阶后的两尾数按定点加减法求和。
3. 规格化，为增加有效数字的位数，提高运算精度，必须将求和后的尾数规格化。
4. 舍入，为提高精度，要考虑尾数右移时丢失的数值位。

有“0舍1入”法和“末位恒置1”法。

1. 溢出判断，即判断结果是否溢出。一定要规格化后才能判断是否溢出。

**1.规格化**

由3.3.3节可知，当基数r=2时，尾数S的规格化形式为1/2≤|S|<1；当补码的最高数值位与符号位不同时，即为规格化形式。但对S<0时，有两种情况需特殊处理。

（1）S=-1/2，则[S]补=1.100...0。它满足规格化数的定义，但是形式上又与一般补码规格化数不同，为了便于硬件判断，特规定-1/2不是规格化的数（对补码而言）。

（2）S=-1，则[S]补=1.00...0。因小数补码允许表示-1，故-1视为规格化的数。

当尾数出现00.0xx...x或11.1xx...x时，需左规；当尾数出现01.xx...x或10.xx...x时，需右规，虽然在定点数这表示溢出，但是在浮点运算中不算溢出，可通过右规处理。

**2.溢出判断**

阶码[j]补=01,xx...x为上溢。

阶码[j]补=10,xx...x为下溢，按机器零处理。

当阶符为“01”时，需作溢出处理。

n位数值位的原码一位乘最多需要n次加法和n次移位；

n位数值位的补码一位乘Booth算法最多需要n+1次加法和n次移位；

n位数值位的原码一位除加减交替法最多需要n+1次加法和n次移位；

n位数值位的补码一位除加减交替法若采用末位恒置1最多需要n次加法和n次移位；

#### 2.5.2乘除法运算

**1.阶码运算**

若阶码用补码表示，乘积的阶码为[jx]补+[jy]补，商的阶码为[jx]补-[jy]补。

若阶码用移码表示，则因为

[jx]移+[jy]补=2n+jx+2n+1+jy=2n+1+[2n+(jx+jy)]= [jx+jy]移(mod2n+1)

因此，乘积的阶码为[jx]移+[jy]补，商的阶码为[jx]移+[-jy]补。

阶码采用移码后又如何判断溢出呢？如果在原有移码符号位的前面再增加一位符号位，并规定该位恒用“0”表示，便能方便的进行溢出判断。溢出的条件是运算结果最高符号位为1。此时若低位符号位为0，表示上溢；低位符号位为1，表示下溢。如果运算结果最高符号位为0，即表明没有溢出。此时若低位符号位为0，表示结果为正；低位符号位为1，表示结果为负。

**2.尾数运算**

（1）浮点数乘法尾数运算

① 检测两个尾数中是否有一个为0，若有，乘积必为0，不再作其他操作；若两个尾数均不为0，则可进行乘法运算。

② 尾数相乘可以采用任何一种定点小数乘法来完成，相乘结果可能需要进行左规。此外，尾数相乘会得到一个双倍字长的结果，若限定只取1倍字长，则乘积的若干低位将会丢失。如何处理丢失的各位值，通常由两种方法。一种是无条件的丢弃尾数最低位之后的全部数值，这种方法被称为截断处理。另一种是按浮点加减法运算讨论的两种舍入原则进行舍入处理，具体细节不再深究，考的可能性极低。

（2）浮点数除法尾数运算

① 检测被除数是否为0，若为0，则商为0；再检测除数是否为0，若为0，则商为无穷大，另作处理。若两数均不为0，则可进行除法运算。

② 尾数相除同样可以采用任何一种定点小数除法来完成。对已规格化的尾数，为了防止除法结果溢出，可先比较被除数和除数的绝对值，如果被除数的绝对值大于除数的绝对值，则先将被除数右移1位，其阶码加1，再做尾数相除。此时所得结果必然是规格化的定点小数。

### 2.6校验码

码距是任意两个合法码之间至少有几个不相同二进制位的位数。

数据校验的几个相关概念：

（1）海明重量：某码字非零元的个数。对于二进制码，即码字中1的个数。

（2）海明距离：两个码字之间对应位不相同的个数。

（3）最小距离：码字集合中，任意两个码字之间的距离的最小值称为最小距离，即码距。码距越大，抗干扰能力越强。

若码距大于等于e+1，则可以检测e个错误；

若码距大于等于2t+1，则可以纠正t个错误；

若码距大于等于e+t+1(e>t)，则可以纠正t个错误同时检测e个错误。

码距d为奇数时，如用来检错可发现d-1位错，如用来纠错，可纠正(d-1)/2位错；d为偶数时可发现d/2位错，并能纠正d/2-1位错。

（4）海明不等式：为了能纠正n位数据中所有可能的单错，就必须满足2r-1≥k+r，其中r是校验位的位数，k是数据位的位数。

#### 2.6.1奇偶校验码

偶校验：检验数据是否有偶数个1，若是，校验位置0，若不是，置1。

奇校验：检验数据是否有奇数个1，若是，校验位置0，若不是，置1。

1.校验位的形成

设有效信息位为D7D6D5D4D3D2D1D0，则

偶校验：在发送端求校验位P = D7⊕D6⊕D5⊕D4⊕D3⊕D2⊕D1⊕D0

奇校验：在发送端求校验位P =

2.校验原理

偶校验：在接收端求校验位

P’=P⊕D7⊕D6⊕D5⊕D4⊕D3⊕D2⊕D1⊕D0

奇校验：在接收端求校验位P’=

若P’=0，则无错；若P’=1，则有错。

奇偶校验码的码距为2。

奇偶你校验码能发现数据中奇数个位出错情况，但没有纠错能力。

#### 2.6.2海明校验码

海明校验码的基本思想是将有效信息按某种规律分成若干组，每组安排一个校验位进行奇偶测试。在一个数据位组中加入几个校验位，增加数据代码间的码距，当某一位发生变化时会引起校验结果发生变化，不同代码位上的错误会得出不同的校验结果。因此，海明码能检测出2位错误，并能纠正1位错误。

例3.9 求信息码01101110的海明校验码，画出能指出和纠正一位出错位的海明校验逻辑电路。

【解】：（1）求信息码01101110的海明校验码

①确定海明校验位的个数

设r为校验位的个数，则整个码字的位数应满足不等式：k+r≤2r-1，

可解得r最小取4。

②确定校验位的位置。20、21、22、23的位置作为校验位，记作P1、P2、P3、P4，余下的为数据位，即：

12 11 10 9 8 7 6 5 4 3 2 1

D7 D6 D5 D4 P4 D3 D2 D1 P3 D0 P2 P1

③分组。有4个校验位，将12位分成4组，第i位由校验位之和等于i的那些校验位所校验，如表2.2所示，如：第11位D6由P1（位号为1）、P2（位号为2）、P4（位号为8）所校验，因为1+2+8=11。

表3.11 4位校验位的分组

|  |  |
| --- | --- |
|  | 12 11 10 9 8 7 6 5 4 3 2 1  D7 D6 D5 D4 P4 D3 D2 D1 P3 D0 P2 P1  0 1 1 0 0 1 1 1 1 0 0 1 |
| 第一组  第二组  第三组  第四组 | √ √ √ √ √ √  √ √ √ √ √ √  √ √ √ √ √  √ √ √ √ √ |

④校验位的形成。

P1=第一组中所有位（P1）除外求异或= D6⊕D4⊕D3⊕D1⊕D0=1

P2=第一组中所有位（P2）除外求异或= D6⊕D5⊕D3⊕D2⊕D0=0

P3=第一组中所有位（P3）除外求异或= D7⊕D3⊕D2⊕D1=1

P4=第一组中所有位（P4）除外求异或= D7⊕D6⊕D5⊕D4=0

为了能检测两个错误，增加一位校验位P5，放在做高位，

P5= D7⊕D6⊕D5⊕D4⊕D3⊕D2⊕D1⊕D0⊕P⊕4P3⊕P⊕P1

信息码01101110的海明校验码为：1 0110 0 111 0 0 10

（2）校验原理

在接收端分别求G1、G2、G3、G4、G5。

G1= P1⊕D6⊕D4⊕D3⊕D1⊕D0

G2= P2⊕D6⊕D5⊕D3⊕D2⊕D0

G3= P3⊕D7⊕D3⊕D2⊕D1

G4= P4⊕D7⊕D6⊕D5⊕D4

G5= P5⊕D7⊕D6⊕D5⊕D4⊕D3⊕D2⊕D1⊕D0⊕P⊕4P3⊕P⊕P1

当G5=1时有一位错，由G4G2G3 G1的二进制编码指出出错位号。例如G4 G2G3 G1=1001说明第9位出错，将其取反，即可纠错。

当G5=0时无错或有两位错，当G4 G2G3 G1=0000时无错，否则有两位错。

能检测两位错并能纠正一位错的海明校验逻辑电路如图3.1所示。

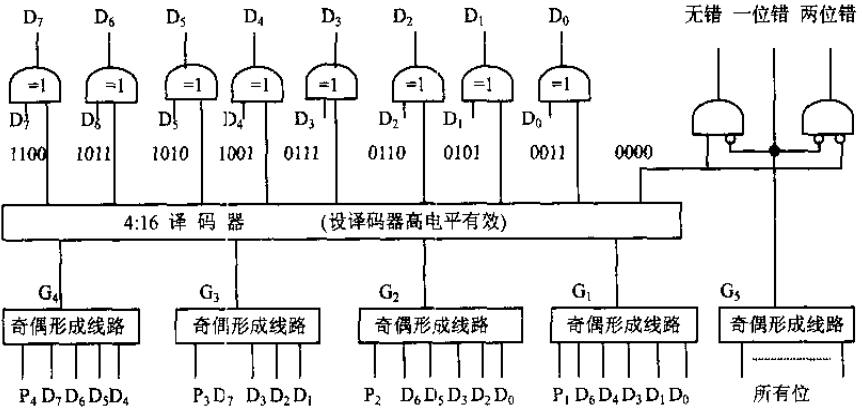


图3.1 海明校验逻辑电路

例3.10 一个8位字00111001，与它一起存储的校验位应该是0111。假定由存储器读出时计算出的校验位是1101，那么由存储器读出的数据字是什么。

【解】：由题可知该信息采用的是海明校验，由于读出时计算出的校验位是1101，所以G4和G2有错，G4 G2G3 G1=1010，海明码的位号排列如下：

12 11 10 9 8 7 6 5 4 3 2 1

D7 D6 D5 D4 P4 D3 D2 D1 P3 D0 P2 P1

由此可知D5有错，由存储器读出的数据字是00011001。

#### 2.6.3循环冗余校验(CRC)码

循环冗余校验（Cyclic Redundancy Check）码可以发现并纠正多位错误，因此在磁介质存储和计算机通信方面得到了广泛应用。

**1.CRC码的编码方法**

CRC码是用多项式M(x)●xr除以生成多项式G(x)所得的余数作为校验位。为了得到r位余数（校验位），G(x)必须是r+1位。

设所得余数表达式为R(x)，商为Q(x)。将余数拼接在信息位组左移r位空出的r位上，就构成了CRC码。CRC码可用多项式表达为

M(x)●xr+R(x)=[Q(x) ●G(x)+R(x)] +R(x)= Q(x) ●G(x)+[R(x) +R(x)]= Q(x) ●G(x)

因此所得CRC码可被G(x)表示的数码除尽。

例3.11 对4位有效信息位（1100）求循环冗余校验码，选择生成多项式(1011)。

【解】：M(x)=x3+x2=1100 (k=4)

M(x) ●x3= x6+x5=1100000 (左移r=3位)

G(x)=x3+x+1=1011 (r+1=4位)

M(x) ●x3/ G(x)=1100000/1011=1110+010/1011 (模2除)

M(x) ●x3+R(x)= 1100000+010=1100010 (拼接)

**2.CRC码的校验原理**

接收方将收到的CRC码与约定的G(x)相除，如果余数为0，则无错；如果余数不为0，则有错，余数即为出错位的位置。

**3.关于生成多项式**

并不是任何一个r+1位多项式都可以作为生成多项式的。从检错及纠错的要求出发，生成多项式应能满足下列要求：

1. 任意一位发生错误都应是余数部位0。
2. 不同位数发生错误应使余数不同。
3. 对余数继续模2除，应使余数循环。

将这些要求反应为数学关系是比较复杂的，对一个（n,k）码来说，可将(xn-1)分解为若干质因子，根据编码所要求的码距选取其中的因式或若干因式的乘积作为生成多项式。

例如x7-1=(x+1)(x3+x+1)(x3+x2+1) （模2运算）

选择G(x)= x3+x+1=1011或G(x)= x3+ x2+1=1101可构成(7,4)码，能纠一位错。

### 2.7算术逻辑部件（ALU）

#### 2.7.1加法器

1.半加器

不考虑进位输入，两数码Xn, Yn相加称为半加。由其功能表可得半加和Hn的表达式：



2.全加器

Xn, Yn及进位输入Cn-1相加称为全加。由其功能表可得全加和Fn和进位输出Cn的表达式：



3.串行进位加法器

将n个全加器串连可得n位串行进位加法器。

4.并行进位加法器

引入进位产生函数Gi和进位传递函数Pi的概念，它们的定义为



将Gi和Pi代入C1~C4，可得



当表达式只停留在第一步,就是串行进位加法器;当表达式往后展开,就是并行性进位加法器。也就是说，并行进位加法器是以增加线路为代价，实现并行进位的。

理想的并行进位加法器是n位全加器的n位进位同时产生，但这会大大增加线路的复杂度，实现起来有困难，这时可以采用折中方法。

5.分组并行进位

（1）单级分组并行进位

将n个全加器分成若干小组，小组内采用并行进位，小组间采用串行进位，因而这种进位有“组内并行、组间串行”之称。

（2）双重分组并行进位

将n个全加器分成若干大组，每个大组中又包含若干小组，小组内和小组间采用并行进位，大组间采用串行进位，因而这种进位有“组（小组）内并行、组（小组）间并行”之称。

#### 2.7.2 ALU的功能和结构

略，太难了，放弃 -\_-|||

## 第3章 存储器

### 3.1存储器的分类

#### 3.1.1存储器的分类

1.按存储介质分类：半导体存储器，磁表面存储器，光盘存储器，磁芯存储器（目前已被淘汰）。

2.按存取方式分类：随机存储器，顺序存储器，只读存储器。

3.按在计算机中的作用分类：高速缓存、主存、辅存。

#### 3.1.2存储器的性能指标

主存的主要技术指标是存储容量和存储速度。

存储速度是由存取时间和存取周期来表示的。存取时间（Memory Access Time）是指启动一次存储操作（读或写）到完成该操作所需的全部时间。存取周期（Memory Cycle Time）是指存储器进行连续的两次独立的存储操作所需的最小间隔时间。通常存取周期大于存取时间。

存储器带宽是指单位时间内存储器存取的数据量。

#### 3.2存储器的层次结构

高速缓存 — 主存 — 辅存

### 3.3半导体随机存取存储器

#### 3.3.1 SRAM

SRAM依靠双稳态电路的两个稳定状态来存储0和1。

#### 3.3.2 DRAM

DRAM依靠电容上暂存电荷来存储信息，有电荷表示1，无电荷表示0。

常见的DRAM基本单元电路有三管式和单管式。

DRAM的读操作是破坏性的，必须再生。

由于电容上的电荷只能维持1~2ms，即使电源不掉电信息也会慢慢消失，因此需要定时刷新。一般取2ms，这个时间称为刷新周期。刷新是一行行进行的，由专门的刷新电路来完成。

常见的刷新方式有三种：集中式、分散式（读一次刷一次）、异步式（读多次刷一次）。

刷新和再生是两个完全不同的概念。刷新是定时的，以一行为单位；再生是随机的，以一个存储单元为单位。

#### 3.3.3 存储器的读写周期

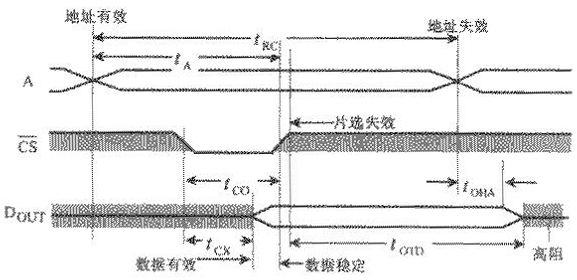
****

图3.1 2114 RAM的读周期时序

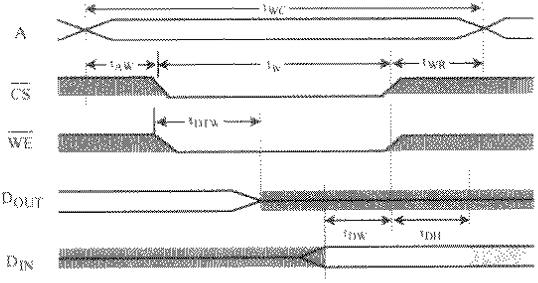
****

图3.2 2114 RAM的写周期时序

#### 3.3.4 SRAM和DRAM的比较

|  |  |  |
| --- | --- | --- |
|  | SRAM | DRAM |
| 存储信息  破坏性读出  需要刷新  送行列地址  速度  容量  功率  成本 | 触发器  否  否  一次  快  小  大  高 | 电容  是  是  分两次  稍慢  大  小  低 |

### 3.4 ROM

掩模ROM，不可编程；

PROM，一次性；

EPROM，可多次编程，分为紫外线擦除（UVEPROM）和电擦出（EEPROM）；

闪存

### 3.5 存储器与CPU的连接

有位扩展和字扩展。

主存和CPU的连接：

（1）数据线的连接：与CPU的数据线直接相连。

（2）地址线的连接：与CPU的地址线直接相连。

（3）控制线的连接：①，所有芯片的与CPU的直接相连；

②，位扩展时，多个芯片的连在一起，然后连接到译码器，字扩展时，单个芯片的直接与译码器相连；

③CPU的访存信号与译码器的使能端相连。

SRAM的引脚：①数据线，取决于字长；②地址线，与芯片容量有关；③控制线：片选信号和读写信号；④电源线，地线。

DRAM的引脚：①地址线，只有一半，行列地址分两次送，内部有两个锁存器暂存行列地址；②数据线，数据输入线和数据输出线是分开的；③控制线：，，，没有；④电源线，地线。

DRAM的扩展不会考，再者，DRAM的集成度高，不需要扩展。

例题请转至8.2节。

### 3.6 双口RAM和多模块存储器

#### 3.6.1 双口RAM

两个端口互不干扰，只有当两个端口试图同时访问同一个存储单元时，才会发生冲突。

解决方案：由判断逻辑决定暂时关闭另一个端口，即置其忙信号为“0”。

#### 3.6.2 单体多字

由于程序和数据是连续存放的，加上程序局部性原理，如果可以在一个存取周期内，从同一地址取出4条指令，然后再逐条将指令送至CPU执行，即每个1/4个存储周期，主存向CPU送一条指令，这样显然增大了存储器的带宽，提高了单体存储器的工作速度。

这种方法的前提是：指令和数据在主存内必须连续存放，一旦遇到转移指令，或者操作数不能连续存放，这种方法的效果就不明显。

#### 3.6.2多体单字

1.高位地址表示体号

这种编制方式即普通的顺序方式，但只要合理调动，使不同的请求源同时访问不同的体，便可实现并行工作。

优点：体内地址连续，有利于存储器的扩充。

2.地位地址表示体号

这种编制方式下，程序连续存放在相邻体中，故又有交叉存储之称。

设低位交叉的存储器模块数为n，存取周期为T，总线传输周期为τ，那么当采用流水线方式存取时，应满足T=nτ。连续读取n个字所需要的时间为T+(n-1) τ。

### 3.7 高速缓冲存储器(Cathe)

#### 3.7.1 程序访问的局部性原理

程序访问的局部性原理包括时间局部性和空间局部性。前者是指在最近的未来要用到的信息很可能是现在正在使用的信息，这是因为程序存在循环。后者是指在最近的未来要用到的信息很可能与现在正在使用的信息在存储空间上是临近的，这是因为指令通常是顺序存放、顺序执行的。

#### 3.7.2 Cache的基本工作原理

高速缓冲技术就是利用程序访问的局部性原理，把程序中正在使用的部分存放在一个高速的、容量较小的Cathe中，是CPU的访存操作大多数针对Cathe进行，从而使程序的执行速度大大提高。

在一个程序执行期间，设Nc为访问Cathe的总命中次数，Nm为访问主存的总次数，则命中率h为h=Nc/(Nc+Nm)。

设tc为访问一次Cathe的时间，tm为访问一次主存的时间，则Cathe -主存系统的平均访问时间ta为ta=htc+(1-h)tm。

用e表示访问效率，则有e=tc/ta。

Cathe主要由Cache存储体、地址映射变换机构、Cathe替换机构三大部件组成。

Cache的改进：（1）增加Cache的级数；（2）统一缓存变成分立缓存，如一个指令缓存，一个数据缓存（称为哈弗结构）。

#### 3.7.3 Cach-主存地址映射

**1.全相联映射**

全相联映射允许主存中的一块映射到Cache中的任何一块。在访问Cache时，需要和Cache的全部标记进行比较才能判断所访问主存地址的内容是否已在Cache中。

┌──────┐

Cathe的地址结构为：│块内地址 │

└──────┘

┌─────────┬──────┐

主存的地址结构为：│主存字块标记 │块内地址│

└─────────┴──────┘

**2.直接映射**

每个主存块只与一个Cache块相对应，如图4.2所示，映射关系为

i=j mod 2c

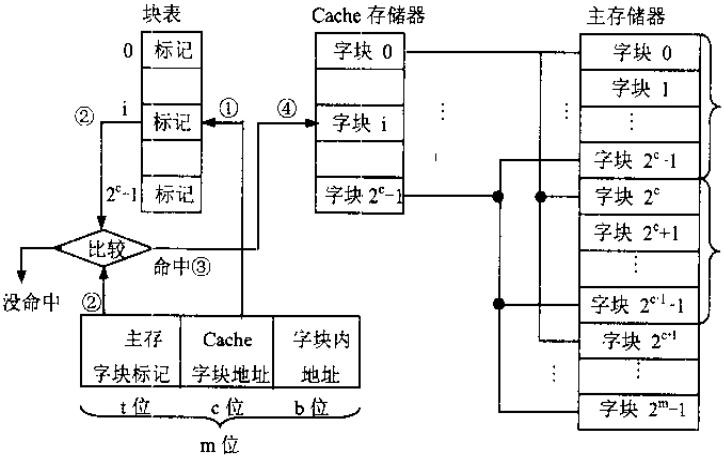


图4.2 直接映像Cache组织

其中，i为Cache块号，j为主存块号，2c为Cache块数。

┌───────┬──────┐

Cathe的地址结构为：│Cathe块号 │块内地址 │

└───────┴──────┘

主存的地址结构为：

┌─────────┬───────┬──────┐

│主存字块标记 │Cathe块号│块内地址 │

└─────────┴───────┴──────┘

**3.组相联映射**

组相联映射是直接映射和全相联映射的折中。它把Cache分为Q组，每组R块，每个主存块只与一组相对应，可以映射到组内任何一块上。如图4.3所示，映射关系为

i = j mod Q +k, 0≤k≤R-1

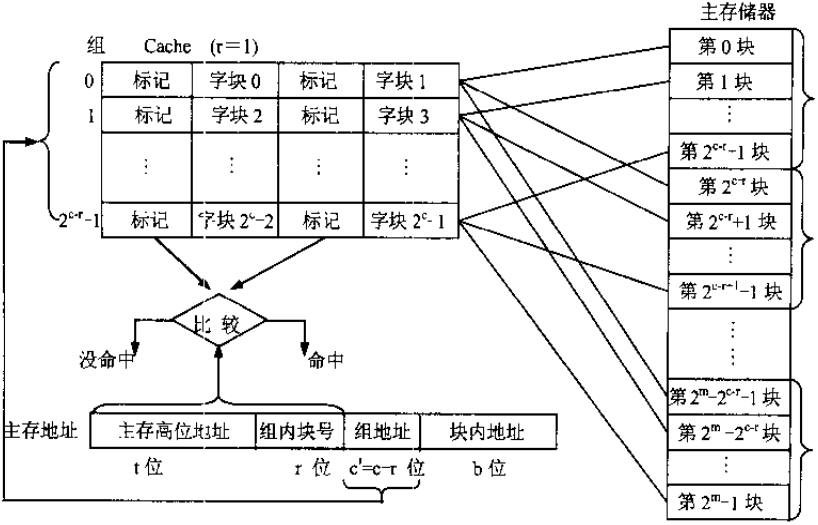


图4.3 组相联映像Cache组织

┌───────┬───┬──────┐

Cathe的地址结构为：│Cathe块号 │组号 │块内地址│

└───────┴────┴─────┘

┌─────────┬───┬──────┐

主存的地址结构为：│主存字块标记 │组号│块内地址│

└─────────┴───┴──────┘

#### 3.7.4 Cache 替换算法

FIFO, LRU

#### 3.7.5 Cache写策略

Cathe的写操作：（1）写直达法（Write Though），写入Cathe的同时也写入主存，速度慢但一致性好；（2）写回法（Write Back），修改标志位置1，当块被换出时写回主存。速度快但一致性稍低。

写直达法和写回法都是在Cathe写命中时的情况。如果Cathe写不命中，还需要考虑是否调块至Cathe的问题。非写分配法只写入主存，不进行调块；写分配法则除了写入主存外，还要将该块从主存调入Cathe。通常，写分配法与写回法合用，非写分配法与写直达法合用。

### 3.8 虚拟存储器

见《操作系统》中的“请求分页管理方式”，“请求分段管理方式”和“请求段页管理方式”。

虚拟存储器与Cathe的比较

相同之处如下：

(1) 最终目标都是为了提高系统性能，两者都有容量、速度、价格的梯度；

(2) 都把数据划分为小信息块，作为基本的传递单位，虚存系统的信息块更大；

(3) 都有地址的映射、替换算法、更新策略等问题；

(4) 依据程序法访问的局部性原理应用“快速缓存的思想”，将相对活跃的数据放在相对告诉的部件。

不同之处如下：

(1) Cathe主要解决系统速度，而虚存却是为了解决主存容量；

(2) Cathe全由硬件实现，对所有程序员透明；而虚存由OS和硬件共同实现，对系统程序员不透明，但对应用程序员透明；

(3) 不命中性能影响，因为速度Cathe/主存约为10倍，主存/硬盘为100倍以上，故虚存不命中时对系统性能影响更大；

(4) CPU与Cathe和主存都建立了直接访问的通路，而辅存与CPU没有直接通路。

## 第4章 指令系统

### 4.1指令格式

**1.指令格式**

指令由操作码和操作数组成。

操作码的长度可以是固定的，也可以是变长的。

根据指令中操作数的个数，可以将指令分为无操作数指令、单操作数指令、双操作数指令和多操作数指令。

操作数的来源和去向：CPU内部的通用寄存器，主存，外围设备的寄存器。

**2.操作码扩展**

最优化的编码方式是哈夫曼编码，但哈夫曼编码形成的操作码很不规整，每条指令的操作码字段的位数都可能不同，不便于译码，。一种实际可行的优化编码方法就是扩展操作码法，它是介于定长编码和哈夫曼编码之间的编码方式，操作码的位数既不是固定的，又不是任意的，而是有限的几种码长。这种方法仍然采用了哈夫曼编码的思想，即使用频度高的指令操作码字段短，使用频度低的指令操作码字段长。

例题请转至后面的9.3节。

### 4.2寻址方式

立即数寻址，寄存器寻址，寄存器间接寻址，直接寻址，间接寻址，相对寻址，基址寻址，变址寻址

基址寻址和变址寻址的区别：

（1）基址寻址中基址寄存器提供基准量而指令提供偏移量，而变址寻址中变址寄存器提供偏移量而指令提供基准量；

（2）基址寄存器的内容由操作系统给定，而且在程序的执行过程中不可变，变址寄存器的内容由用户给定，且在程序的执行过程中可变；

（3）基址寻址可用于程序定位，支持多道程序技术，变址寻址可用于处理数组，编制循环程序等。

### 4.3 RISC

RISC即精简指令集计算机（Reduced Instruction Set Computer），与复杂指令集计算机CISC（Complex Instruction Set Computer）相对应。

RISC的特点：

（1）优先选取使用频率很高的一些指令以及一些很有用但不复杂的指令，避免复杂指令。

（2）指令长度固定，指令格式少，寻址方式少。

（3）采用流水线技术，大部分指令在一个时钟周期内完成。

（4）只有取数/存数（LOAD/STOR）指令访问存储器。

（5）面向寄存器的结构，CPU内部设有大量的通用寄存器。

（6）硬布线控制器。

（7）注重编译的优化。

## 第5章 中央处理器（CPU）

### 5.1 CPU的功能和基本结构

CPU主要包括控制器和运算器两大部分。

**控制器**由程序计数器（PC）、指令寄存器（IR）、指令译码器、地址寄存器（MAR）、数据寄存器（MDR）、时序信号发生器和微操作信号发生器组成。主要功能有指令控制（取指、译码、执行）、操作控制（产生微操作信号）、时间控制（产生时序信号），中断处理。

**运算器**由算术逻辑单元（ALU）、通用寄存器、程序状态字（PSW）寄存器、数据暂存器、移位器等组成。主要功能是进行算术运算和逻辑运算。

### 5.2指令的执行过程

指令的执行过程包含取指、译码、执行三个阶段。

CPU取出并执行一条指令所需的全部时间称为**指令周期**。

由于不同指令的复杂程度不同，因此不同指令的指令周期是不同的。

完整的指令周期包括取值周期、间址周期（间接寻址时才有）、执行周期、中断周期（有中断时）。

指令周期可划分为几个不同的阶段，如取指、译码、执行，每个阶段所需的时间称为**机器周期**，又称为CPU周期。当存储字长等于指令字长时，取值周期可以看作机器周期,通常等于取指时间（访存时间）。

每个指令周期内的机器周期数可以不同，每个机器周期内的时钟周期数也可以不同。

### 5.3 数据通路的功能和基本结构

数据在主机的功能部件之间传送的路径称为**数据通路**。

### 5.4 硬布线控制器

略。

### 5.5 微程序控制器

#### 5.5.1 微程序控制器概述

1. 微程序控制的基本概念

一条机器指令由一段微程序来实现，即一条机器指令对应多条**微指令**，一条微指令由多个**微命令**组合而成。

由微命令控制实现的最基本的操作称为**微操作**，如开门、关门、打入等。

微操作和微命令是一一对应的。微命令是微操作的控制信号，微操作是微命令执行过程。

微命令可分为两类：

（1）电位型微命令： 如①各寄存器输出到总线的控制信号；②ALU运算控制信号；③主存读写信号。

（2）脉冲型微命令： 如①总线使用打入脉冲将数据打入各寄存器；②增量与减量信号。

2. 微程序控制器的组成

微程序控制器原理框图如图所示6.1所示。

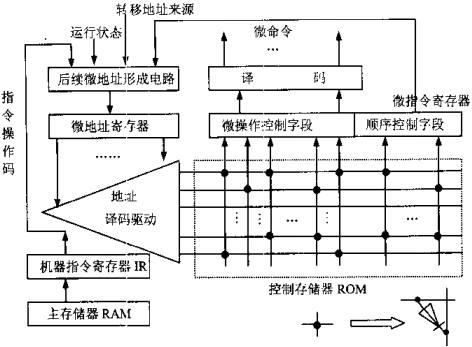


图6.1 微程序控制器原理框图

#### 5.5.2微指令的编码方法

1.直接控制编码

直接控制编码法无须进行译码，微指令的微操作码字段每一位都代表一个微命令。

2.字段直接编码

将微指令的微命令字段分成若干小段，把相斥性微命令组合在同一字段中，而把相容性微命令放在不同的字段中。每个字段独立编码，每种编码代表一个微命令且各字段编码含义单独定义，与其他字段无关，这就是字段直接编码。

一般每个小段还要留出一个状态位，表示本字段不作任何操作。例如当某字段的长度为3位时，最多只能表示7种互斥的微命令，通常用000表示不操作。

3.字段间接编码

字段间接编码是在字段直接编码的基础上进一步缩短微指令字长的一种编码方法。在这种编码方法中，一个字段的含义不仅由本字段编码决定，还兼由其他字段解释，以便用较少的位表示更多的信息。

#### 5.5.3微指令格式

1.水平型微指令

一次能定义并执行多个并行微操作的微指令称为水平型为指令。水平型微指令一般采用上述三种编码方法进行编码。

2. 垂直型微指令

在微指令中设置微操作码字段，由微操作码规定微指令的功能。

3.二者的比较

（1）水平型微指令并行操作能力强，效率高，垂直型微指令则较差。

（2）由水平型微指令组成的微程序，微指令较长而微程序短；垂直型微指令则相反，微指令较短而微程序长。

例题请转至后面的8.4节。

写微指令序列时，要把ALU看成是一个无状态的器件（况且ALU的确是组合逻辑电路），它不存储数据，向ALU输入数据，只要放入锁存器即可，不能直接放入通用寄存器，因为会破坏其原有内容，再者，通用寄存器是运算器的一部分，不是ALU的一部分，放入通用寄存器ALU还是够不着。

#### 5.5.4微地址的形成方式

**1.入口地址的形成**

由于每条机器指令都需要取指操作，所以将取指操作编织成一段公用微程序，通常安排在控制存储器0号或1号单元开始的一段CM空间。

每一条机器指令对应着一段微程序，其入口就是初始微地址。首先由“取指令”微程序取出一条机器指令到IR中，然后根据机器指令操作码转换成该指令对应的微程序入口地址。

**2.后续微地址的形成**

（1）微程序计数器（μPC）法

这种方式与用程序计数器产生机器指令地址相类似，在顺序执行微指令时，后续微指令地址由μPC+1产生；在非顺序执行微指令时，由转移微指令实行转移。转移微指令分成两部分：条件选择字段与转移地址字段。当转移条件满足时，将转移地址字段送入μPC；若转移条件不满足，则直接从μPC中取得下一条微指令地址。

用微程序计数器法产生微地址的缺点是微程序中出现大量的转移微指令，导致执行时间大大增加。另外，要区分一般微指令和转移微指令使得微程序控制电路复杂化。

（2）下址字段法

下址字段法（也称为断定方式）与计数器法不同，它不采用μPC，微指令地址由微地址寄存器μAR提供。在微指令格式中，设置一个下址字段，用于指明下一条要执行的微指令地址。当一条微指令被取出执行时，下一条微指令的地址（即下址字段）送入μAR。它相当于每条微指令都具有转移微指令的功能。采用这种方法就不必设置专门的转移微指令，但增加了微指令字的长度。

从上述也可以看到，在微程序控制器中，可以用μAR来代替μPC，那么可不可以用MAR来代替程序计数器PC呢？答案是不可以，因为微程序控制器中μAR只用于取微指令，在CPU中MAR不仅用于取指令，还用于取数。

（3）微程序计数器与下址字段相结合

在这种控制方式中微地址寄存器有计数功能（下址字段法中的微地址寄存器无计数功能），但在微指令中仍设置一个顺序控制字段，它分成两部分：条件选择字段与转移地址字段。由这两个字段结合，当转移条件满足时，将转移地址字段作下一个微地址；若无转移要求，则微地址寄存器增1得到下一条微指令的地址。

#### 5.5.5动态微程序设计与毫微程序设计

通常指令系统是固定的，对应每一条机器指令的微程序是计算机设计者事先编好的，因此微程序无需改变，其控制存储器一般采用ROM，这称为静态微程序设计。

如果采用EPROM作为控存，人们可以修改微指令和微程序来改变机器的指令系统，这称为动态微程序设计。

微程序可以看作是解释机器指令的，而毫微程序可以看作是解释微指令的。毫微程序存放在第二级控存中。

#### 5.5.6控制器的控制方式

1.同步控制方式

有统一的时钟信号，又分为两种方案：（1）采用定长的机器周期；（2）采用不定长的机器周期；（3）采用中央控制和局部控制相结合的方法，即把大部分指令安排在同一的、较短的机器周期内完成，称为中央控制，而把少数复杂指令安排在不统一的、较长的机器周期内完成，称为局部控制。

2.异步控制方式

没有统一的时钟信号，各部件按自身固有的速度工作，通过应答方式进行联络。

3.联合控制方式

同步控制和异步控制相结合。

CPU内部操作采用同步控制方式，CPU与内存和I/O设备的操作采用异步方式。CPU内部操作大部分指令采用中央控制方式，少数复杂指令采用局部控制方式。

4.人工控制方式

如复位键（reset），关机键。

CPU内部均采用同步控制。

### 5.6 指令流水线

#### 5.6.1指令流水线的基本概念

指令流水线就是把一条指令的执行拆分成多个步骤，不同指令的各步之间可以重叠执行。

#### 5.6.2影响流水线性能的因素

1. 结构相关

由于多条指令在同一时刻征用统一资源而产生的冲突称为结构相关。例如流水线既要取当前指令所需要的操作数，又要预取下一条指令，就会发生访存冲突。

有以下两种解决方案：

（1）让后续指令都暂停几个时钟周期；

（2）单独设置数据存储器和指令存储器，使两项操作各自在不同的存储器中进行，这属于资源重复配置。

2. 数据相关

后续指令需要使用前面指令的运算结果，而这一结果尚未产生或送到指定的位置，从而造成后续指令无法执行的局面称为数据相关。数据相关可分为RAW相关、WAR相关、WAW相关。

有以下两种解决方案：

（1）把遇到数据相关的指令及其后续指令都暂停一到几个时钟周期，直到数据相关问题消失后在继续执行；

（2）设置相关专用通路，即不等前一条指令把计算结果写回寄存器，下一条指令不再读寄存器，而是直接把前一条指令的ALU的计算结果作为自己的输入数据开始运算，使本来需要暂停的操作变得可以继续执行，这种技术称为数据定向或数据旁路技术。

3. 控制相关

控制相关主要是由转移指令引起，它会使流水线的连续流动受到破坏。

有以下四种解决方案：

（1）尽早判断转移是否发生，尽早生成转移目标地址；

（2）预取转移成功和不成功两个控制流方向上的指令；

（3）加快和提前形成条件码；

（4）提高转移方向猜准率。

#### 5.6.3流水线的性能指标

1.吞吐率（Throughput Rate）

指单位时间内流水线所完成的指令数量。

设k为流水线的段数，Δt为时钟周期，n为指令条数，得实际吞吐率为



当n→∞时，的最大吞吐率TPmax=1/Δt。

2.加速比（Speedup Ratio）

完成同样一批指令，不使用流水线与使用流水线所用的时间之比称为流水线的加速比。



当n→∞时，得最大加速比Smax=k。

3.效率（Efficiency）

流水线的设备利用率称为流水线的效率。即n个任务占用的时空区面积与n个任务所用的时间与k个流水段所围成的时空区总面积之比。



#### 5.6.4流水线中的多发技术

1.超标量技术

在一个时钟周期内同时（同一时刻）发出多条指令，如图6.2(b)所示。

2.超流水线技术

在一个时钟周期内分期（不同时刻）发出多条指令，好比将流水线再细分，如图6.2(c)所示。

3.超长指令字技术(VLIW)

超长指令字技术和超标量技术都是采用多条指令在多个处理机中并行执行的体系结构。VLIW是由编译程序在编译时挖掘出指令间潜在的并行性，把多条能并行执行的指令组合成一条具有多个操作码字段的超长指令（可长达几百位），由这条超长指令控制VLIW机中多个处理机并行工作。

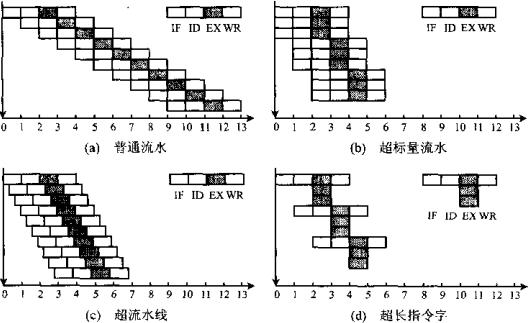


图6.2 四种流水技术的比较

## 第6章 总线

### 6.1总线概述

#### 6.1.1总线的基本概念

计算机中的五大部件之间的互连方式有两种，一种是各部件之间使用单独的连线，称为分散连接；另一种是将各部件连接到一组公共信息传输线上，称为总线连接。

计算机采用“面向总线”的结构有何优点？1.简化了系统结构；2.简化了硬件设计；3.便于系统扩展。

#### 6.1.2总线的分类

按连接部件不同，可分为片内总线、系统总线和I/O总线。

1. 片内总线

是指CPU内部的总线，如CPU内部寄存器与寄存器之间，寄存器与ALU之间都由内部总线连接。

2.系统总线

系统总线是指计算机中各高速部件之间的信息传输线。由于这些部件通常安放在主板或各个插板上，故又称板级总线或板间总线。系统总线可分为数据总线、地址总线和控制总线。

3.I/O总线

I/O总线是指CPU与中低速设备或计算机系统与其他系统（如测控仪表、移动通信）之间的信息传输线，也叫通信总线或外部总线。

#### 6.1.3总线的结构及性能指标

总线的结构：单总线结构，双总线结构，多总线结构

性能指标：总线宽度，总线频率，总线带宽

总线带宽=总线宽度×总线频率

### 6.2总线仲裁

总线上的各类设备，按其对总线有无控制权可分为主设备和从设备。主设备对总线有控制权，从设备只能响应从主设备发来的总线命令。

总线仲裁可分为集中式和分布式两种。

集中仲裁方式有三种：链式查询、计数器查询、独立请求方式。

1.链式查询

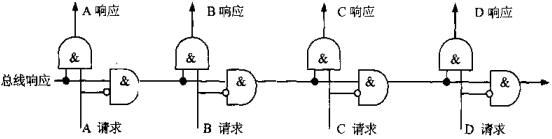


图7.1 链式排队电路

链式查询方式有3根控制线，BS总线忙，BR总线请求，BG总线同意。

其特点是控制线少，容易扩充设备，但对电路故障很敏感，且优先级低的设备可能很难获得总线。

2.计数器定时查询

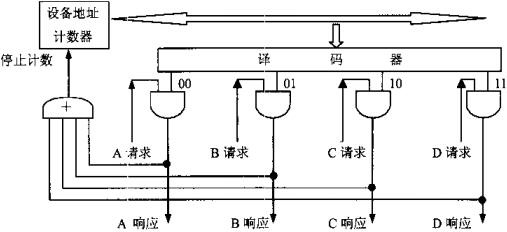


图7.2 计数器定时查询电路

计数器定时查询方式有1根BS线，1根BR线和log2N根设备地址线。

其特点是：对电路故障不如链式查询方式敏感，但增加了控制线，控制也较复杂。

3.独立请求方式

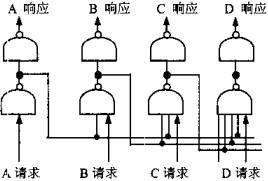


图7.3 独立请求方式排队电路

独立请求方式有1根BS线，N根BR线，N根BG线。

其特点是：响应速度快，优先次序控制灵活，但控制线数量多，总线控制更复杂。

### 6.3总线通信控制

总线完成一次完整的数据传输所需要的时间称为**总线周期**。它通常包含总线请求、总线仲裁、部件寻址、数据传输和总线释放5个阶段。

**总线通信控制**主要解决通信双方如何获知传输开始和传输结束，以及通信双方如何协调配合。通常有两种方式：同步通信和异步通信。

1.同步通信

通信双方依靠统一的时钟信号进行协调配合。

2.异步通信

双方采用应答方式进行协调配合。应答方式可分为不互锁、半互锁和全互锁。

（1）不互锁 “请求”和“应答”都有一定的时间宽度，“请求”信号的结束和“应答”信号的结束不互锁。

（2）半互锁 “请求”信号的撤销取决于接收到“应答”信号，而“应答”信号的撤销由从设备自己决定。

（3）全互锁 “请求”信号的撤销取决于 “应答”信号到来，而“请求”信号的撤销又导致“应答”信号的撤销。

#### 6.4总线标准

常见的总线标准：ISA、EISA、VESA、PCI、AGP、RS-232C、USB。

## 第7章 输入输出（I/O）系统

### 7.1 I/O系统基本概念

输入输出是以主机为中心而言的，将数据从外部设备传送到主机称为输入，反之称为输出。

I/O端口编址方式：统一编址和独立编址。端口是指I/O接口中的寄存器。

I/O控制方式：程序查询方式、中断方式、DMA方式和通道方式。

### 7.2 I/O设备

#### 7.2.1 输入设备

1.键盘

采用硬件确认哪个按键按下的方法称为键盘编码，采用软件判断则称为非键盘编码。

2.鼠标

分为机械式和光电式两类。

3.触摸屏

大致有5类：电阻式、电容式、表面超声波式、扫描红外式和压感式。

#### 7.2.3输出设备

1.显示器

以可见光的形式传递和处理信息的设备称为显示器。

按显示器件分类，有阴极射线管（Cathede Ray Tube,CRT）显示器，液晶显示器（Liquid Crystal Display, LCD），发光二极管（LED）显示器、等离子显示器（PD）等。按显示内容分类有字符显示器，图形显示器和图像显示器。

显示器有以下主要参数：

屏幕大小：以对角线长度表示，常用的有12～29英寸。

分辨率：所能表示的像素个数，以宽、高的像素的乘积表示。

点距：像素间的距离，常有0.28mm、0.25mm，甚至更小。

灰度级：在黑白显示器中指所显示的像素点的亮暗差别，在彩色显示器中则表现为颜色的不同，灰度级越多，图像层次越清楚逼真，典型的有256级（8位）等。

刷新频率：单位时间扫描整个屏幕内容的次数，安人的视觉生理，刷新频率大于30次/秒时才不会感到闪烁。

显示存储器（VRAM）：也称刷新存储器，为了不断提高刷新图像的信号，必须把一帧图像信息存储在刷新存储器中。其存储容量由图像分辨率和灰度级决定，分辨率越高，灰度级越多，刷新存储器容量越大。

VRAM容量=分辨率×灰度级位数

VRAM带宽=分辨率×灰度级位数×帧频

（1）字符显示器

显示存储器VRAM中存放要显示的字符的ASCII码，而屏幕上的字符由光点组成，因此必须有一个部件能将ASCII码转换为光点矩阵信息，这就是字符发生器，它实质上是一个ROM，里面存放字符点阵。

（2）图形显示器

显示图形有两种方法：随即扫描法和光栅扫描法。

（3）图像显示器

采用光栅扫描法

2.打印机

（1）打印机分类

按印字原理分类，有击打式和非击打式；按工作方式分类，有串行打印机和行式打印机，前者逐字打印，后者逐行打印。主要性能指标有：

打印分辨率：每英寸所包含的点阵数。

打印速度：每分钟输出的页数（Pages Per Minute，PPM）。

（2）针式打印机

针式打印机由打印头、横移结构，走纸机构，色带机构和控制电路组成。

特点：擅长“多层复写打印”，实现各种票据打印或蜡纸打印。

（3）喷墨打印机

喷墨打印机主要由喷头，充电电路，偏转电极，过滤回收系统，墨水泵及相应的控制电路组成。

特点：打印噪音小，可实现高质量彩色打印，但防水性差。

（4）激光打印机

打印过程：准备，照相，显影，转印，定影。

其核心部件是硒鼓。

几种打印机的比较：

针式打印机属于击打式打印机，可以逐字打印，也可以逐行打印；喷墨打印机只能逐字打印；激光打印机属于页式输出设备。后两种属于非击打式打印机。

#### 7.2.4外存储器

**1. 磁表面存储器的主要技术指标**

（1）记录密度

磁盘沿半径方向单位长度的磁道数称为道密度，单位死tpi（Tracker Per Inch，道每英寸）或tpm（道每毫米）。为了避免干扰，磁道与磁道之间需保持一定距离，相邻两条磁道中心线之间的距离称为道距，因此道密度Dt等于道距P的倒数，即Dt=1/P。

单位长度磁道能记录二进制信息的位数，称为位密度或线密度，单位是bpi（Bit Per Inch，道每英寸）或bpm（道每毫米）。对于磁盘，位密度Db=ft/πdmin，其中ft为每道总位数，dmin为同心圆中的最小直径。一般泛指磁盘位密度时，是指最内圈磁道上的位密度（最大位密度）。

（2）平均寻址时间

平均寻址时间=平均寻道时间+平均等待时间

平均访问时间=平均寻址时间+传输时间

**2. 磁记录原理和磁记录方式**

（1）磁记录原理

磁表面存储器通过磁头和磁性介质的相对运动完成读写操作。

（2）磁记录方式

磁记录方式又称为编码方式，它是按某种规律将一串二进制数字变换成磁表面相应的磁化状态。磁记录方式对记录密度和可靠性有很大影响，常用的记录方式有6种，如图8.1所示。

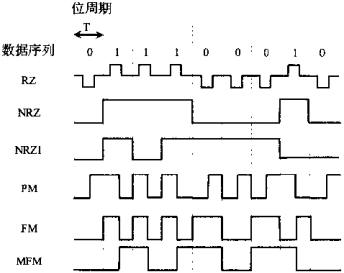


图8.1 6种磁记录方式的写入电流波形

①归零制（RZ）

正向脉冲表示1，负向脉冲表示0。由于两位信息之间驱动电流为零，故称为归零制。

②不归零制（NRZ）

正向电流表示1，负向电流表示0。由于线圈中始终有电流，故称为不归零制。NRZ只有当相邻两位不同时，电流才改变方向，故称为“见变就翻”的不归零制。

③“见1就翻”的不归零制（NRZ1）

和NRZ的不同之处在于，电流只有在记录1时改变方向，记录0时不方向不变。

④调相制（PM）

记录1时，电流先正后负；记录0时，电流先负后正。

⑤调频制（FM）

以驱动电流的变化的频率不同来区别记录1还是0。记录1时，在位周期中心电流改变一次方向，记录0时不改变。而且无论是记录1还是0，在位周期起始处电流均变化一次。

⑥改进调频制（MFM）

与FM的区别在于，只有在连续记录两个或两个以上0时，才在位周期起始处改变一次电流，不必在每个位周期起始处都改变电流方向。

评价磁记录方式的指标有编码效率，自同步能力，可靠性等。

**3.硬盘**

硬盘的分类：

（1）按磁头运动与否可分为固定磁头磁盘和可移动磁头磁盘。目前使用最多的是可移动磁头磁盘。

（2）根据磁头是否与盘面接触可分为接触式磁头和浮动式磁头。

接触式磁头多用于软盘，浮动式磁头主要用于硬盘。

（3）按盘片是否可更换可分为固定盘片磁盘和可换盘片磁盘。

温彻斯特盘是一种可移动磁头固定盘片的硬盘。

硬盘中的信息是按记录面、柱面、磁道、扇区的层次结构安排的。

硬盘由磁盘控制器、磁盘驱动器和盘片三大部分组成，如图8.2所示。

主机

磁盘控制器

磁盘驱动器

盘片

⇔ ⇔ ⇔

8.2 硬盘基本结构

**4.磁盘阵列**

廉价冗余磁盘阵列RAID（Redundant Array of In-expensive Disk）是用多个磁盘存储器组成的大容量外存储系统，其基础是数据分块技术，即在多个磁盘上交错存放数据，使之可并行存取。

**5.光盘**

光盘（Opital Disk）分类：

1.只读型光盘（CD-ROM）

2.只写一次型光盘（Write Once, Read Many, WORM）

3.可擦写型光盘

从原理上来看，目前仅有磁光型（热磁反转）和相变型（晶态-非晶态转）两种。

CD-ROM驱动器的速度是以数据传输率来标称的，习惯上把传输率为150KB/S（标准CD传输率）的光驱称为单倍速光驱，而把300KB/S的光驱称为双倍速光驱，依此类推。倍速一般以“X”表示，如48倍速写成“48X”。

### 7.3 I/O接口

#### 7.3.1 I/O接口的功能和基本结构

#### 7.3.2 I/O端口及其编址

I/O端口是指接口电路中可以被CPU直接访问的寄存器，主要数据端口、状态端口和控制端口，若干个端口加上相应的控制逻辑电路组成接口哦。通常，CPU能对数据端口执行读写操作，但对状态端口只能执行读操作，对控制端口只能执行写操作。

I/O端口的编址方式有与存储器统一编制和独立编址两种。

### 7.4 I/O控制方式

#### 7.4.1 程序查询方式

CPU一旦启动I/O，停止现行程序的运行。

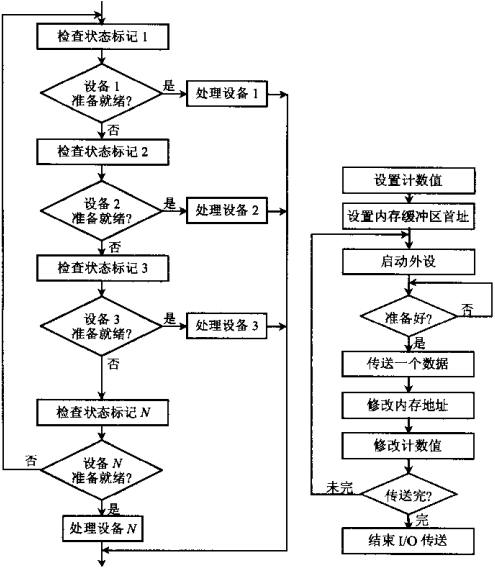


图8.3 多个I/O设备的查询流程 图8.4 单个设备的处理流程

#### 7.4.2 中断方式

1.中断的基本概念

CPU启动I/O后，不必停止现行程序的运行。

**中断**是指计算机系统内发生了某一急需处理的事件，使得CPU暂时中止当前正在执行的程序而去执行相应的事件处理程序，待处理完毕后又返回到原来被中断处继续执行。引起中断发生的事件称为**中断源**。中断源向CPU发出的请求中断处理信号称为**中断请求**，CPU收到中断请求后转向相应事件处理程序的过程称为**中断响应**。

发生中断时，刚执行完的那条指令所在的单元号称为**断点**，断点的逻辑后续指令的单元号称为**恢复点**。而**现场**是指中断的那一时刻能确保程序继续运行的有关信息。

在有些情况下，尽管产生了中断源和发出了中断请求，但CPU内部的处理机状态字PSW的中断允许位已被清除，从而不允许CPU响应中断，这种情况称为**禁止中断**（也称为关中断）。CPU禁止中断后只有等到PSW的中断允许位重新设置后才能接收中断。设置PSW的中断允许位称为开中断。开中断和关中断是为了保证某些程序执行的原子性。

**中断屏蔽**是指系统用软件的方式有选择地封锁部分中断而允许其余部分中断仍能得到响应。不过，有些中断请求是不能屏蔽甚至不能禁止的，也就是说，这些中断具有最高优先级，不管CPU是否关中断，只要这些中断请求一提出，CPU必须立即响应。例如电源掉电事件引起的中断就是不可禁止和屏蔽的中断。

2. 中断响应过程

（1）中断请求



（2）中断判优

（3）中断响应

CPU响应中断必须满足以下三个条件：

①中断源有中断请求；②CPU允许中断及开中断③一条指令执行完毕，且没有更紧迫的任务。

3. 中断处理过程

中断处理过程如下：（1）保护现场；（2）执行中断服务程序；（3）恢复现场，继续运行。如图7.5所示。

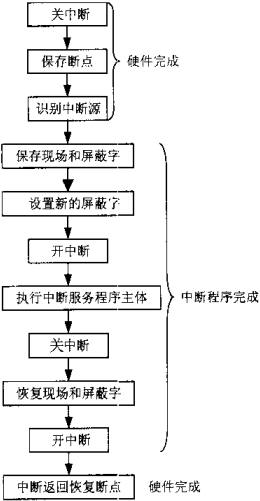


图7.5 中断的处理过程

中断服务程序入口地址的形成：

硬件向量法，利用硬件产生向量地址（又称为中断向量），再由向量地址在中断向量表中找到中断服务程序的入口地址。

4. 多重中断和中断屏蔽

CPU暂停现行的中断服务程序，转去处理新的中断请求，称为**多重中断**。

#### 7.4.3 DMA方式

1. DMA的工作原理

主存与DMA接口之间有一条直接数据通路，数据传送不再经过CPU，而是在内存与外设之间直接进行，因此称为直接存储器存取方式（Direct Memory Access, DMA）。

中断方式下一个字中断一次，DMA方式下一组中断一次。

2. DMA占用总线的方式

（1）独占总线：从传送首字开始直到成组数据传送完成，DMA都掌控总线不放。【霸道】

（2）周期挪用：趁存储器空闲DMA占用总线周期传送一个字。若与CPU发送总线争用时，此时CPU让出一个总线周期给DMA，之后才能使用总线。【见缝插针、礼让】

（3）交替方式：与CPU轮流使用一个时间片，不需要申请、建立和释放手续。【顾名思义】

无论是DMAC向CPU请求传送还是报告结束，都通过中断方式。【中介】

3. DMA控制器的组成

（1）设备地址寄存器DAR：用于存放设备号。

（2）内存地址计数器AR：用于存放在内存中要交换的数据的地址。在DMA传送之前，必须调用特定程序将数据在内存中的首地址送到内存地址计数器。当DMA进行时，每交换一次数据则将此计数器的值加1。

（3）字计数器WC：用于记录传送数据块的长度。在DMA传送之前，必须调用特定程序预置其值。当DMA进行时，每传送一个字，计数器就减1，当数据传送完毕时，此计数器溢出，使DMA控制器向CPU发中断信号。

（4）数据缓冲寄存器DBR：用于暂存每次的数据（一个字）。进行输入输出时，数据需要在此处中专后，才送往目的地。

（5）控制电路，中断机构等。

4.DMA传送过程

DMA的数据传送过程可分为3个阶段，即准备阶段、传送阶段和结束阶段。

（1）准备阶段

① 指明数据传送方向是输入（主存写）还是输出（主存读）；

② 设备号送入DMA接口中的设备地址寄存器DAR，启动设备；

③ 主存首地址送入DMA接口中的内存地址寄存器AR；

④ 数据字数送入DMA接口中的字计数器WC。

（2）传送阶段

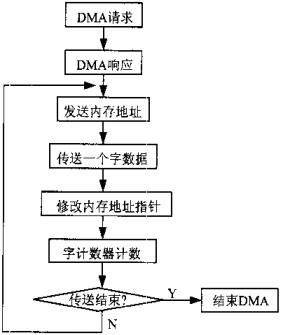


图7.6 DMA数据传送的流程

① 外设准备好发送数据（输入）或接收数据（输出）时，向主机发DMA请求（设备向DMA接口发请求，DMA接口向CPU申请总线控制权）；

② CPU在本机器周期执行完毕后响应该请求并使CPU的总线驱动器处于高阻状态，让出主存使用权；

③ DMA发送主存地址；

④ 挪用一个存储周期，传送一个数据；

⑤ 主存地址加1，字计数器减1；

⑥ 判断数据是否传送完毕，即字计数器是否为0，若字计数器不为0，返回①，若字计数器为0，则传送完毕，进入结束阶段。

（3）结束阶段

DMA向主机发出中断请求，报告结束。一旦DMA的中断请求得到响应CPU将停止主程序的执行，转去执行中断服务程序进行DMA操作后的处理，包括校验送入内存的数据是否正确；决定是否继续使用DMA方式传送数据还是结束传送；测试传送过程中是否发生错误。

#### 7.4.4通道方式

CPU通过执行I/O指令以及处理来自通道的中断实现对通道的管理。来自通道的中断有两种：数据传送正常结束或发生故障。

1.通道的类型

（1）字节多路通道

字节多路通道主要用于连接大量的低速设备，如键盘、打印机等。这些设备的数据传输率很低，而通道从设备接收或发送一个字节只需要几百纳秒，因此通道在传送两个字节之间有很多空闲时间，字节多路通道正是利用这个空闲时间为其他设备服务。

（2）数组选择通道

在物理上它可以连接多个设备，但是这些设备不能同时工作，在一段时间内只允许执行一个设备的通道程序。

（3）数组多路通道

数组多路通道是对选择通道的一种改进，它的基本思想是当某设备进行数据传送时，通道只为该设备服务；当设备在执行寻址等控制性动作时，通道暂时断开与这个设备的链接，挂起该设备的通道程序，去为其他设备服务，即执行其他设备的通道程序，所以数组多路通道很像一个多道程序的处理器。

字节多路通道和数组多路通道有很多共通之处，即它们都是多路通道，在一段时间内能交替执行多个设备的通道程序。

## 第8章 大题精选

### 8.1海明码、CRC码

### 8.2存储器扩展

### 8.3操作码扩展

### 8.4微操作序列、微指令设计

参考书籍：

[1] 计算机组成原理考研指导，徐爱萍，清华大学出版社，2003年1月第1版

[2] 计算机组成原理（第2版），唐朔飞，高等教育出版社，2008年1月第2版

[3] 2011计算机专业基础综合考试指导全书，王道论坛，2010年9月第1版

版本：2.0，最后更新日期：2010/11/29

官方地址：<http://www.gotothu.com> [未来清华人]

作者：戴方勤

个人博客：<http://www.yanjiuyanjiu.com/>

本手册遵循创作共享协议2.0，禁止一切商业用途。