# 第4篇

# 电子计算机

# 第4篇

# 电子计算机

**主编单位** 清华大学

**编写单位** 清华大学

**主　　编** 周远清

**副 主 编** 戴一奇

**编 写 人** 王爱英　潘孝梅　周之英　史美林

**主　　审** 杨天行

自从1946年第一台电子计算机问世以来，计算机的发展可分成四个阶段（即四代）。第一代计算机的基本器件是电子管，第二代是晶体管，第三代为中、小规模集成电路，第四代为大规模、超大规模集成电路。随着器件的更新，其速度、功能、可靠性的不断提高和价格的不断降低，促进了计算机体系结构的改进，推动了计算机的发展。与此同时，形成了计算机软件系统，随着应用范围的扩大，软件日臻完善。本篇主要涉及计算机硬件、软件的基础知识。第1章计算机组织与结构主要介绍计算机中数据运算的规则，计算机硬件的组成以及软、硬件人员都能见到的指令系统。由于广大机电工程技术人员广泛使用微型计算机，因此将微型计算机专门列为本篇的第2章。第3章软件基础介绍了操作系统、程序设计语言以及其他软件基础知识。第4章为计算机网络。

关于电子计算机应用方面的内容可参见自动化与通信卷第6篇。

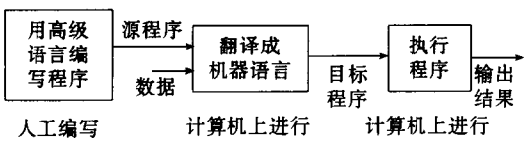
# 第1章　计算机组织与结构

## 1　计算机系统概述[1]

### 1·1　计算机系统构成

计算机系统由硬件和软件构成，硬件是计算机系统中的实际装置，是系统的基础和核心（称为硬核），一般由中央处理器（CPU）、存储器、输入/输出（I/O）设备构成，它以机器语言（即指令系统）提供给程序员。软件指的是操作系统、文本编辑程序、调试程序、汇编程序、编译程序、数据库管理系统、文字处理系统、视窗（window）软件、网络软件以及各种应用程序等。

现代计算机解题的一般过程如下：




计算机系统是一个具有多级层次结构的系统[2][3]，如图4·1-1所示，它的底层是由硬件组成的实际机器M1，配上操作系统后就成为虚拟机器M2，因为这台机器是依靠软件扩充后才构成的，所以称为虚拟机器。在M2的上面是用汇编语言或中间语言表示的虚拟机器M3，用户用高级语言编程时见到的是虚拟机器M4。

计算机软件和硬件在逻辑功能上是等效的，即某些操作既可以用软件，也可以用硬件实现，因此软、硬件之间的分界面是可变的，它取决于实际应用对性能的需求以及系统的性能价格比。回顾计算机的发展历史，在早期，由于硬件昂贵，所以计算机的硬件组织比较简单，尽量让软件完成更多的工作，但是器件价格在不断下降、性能在不断提高，使得硬件成本不断下降。而软件成本在计算机系统中所占比例不断上升，这就造成了软、硬件之间的分界面的推移，即将某些由软件完成的工作交给硬件去完成。

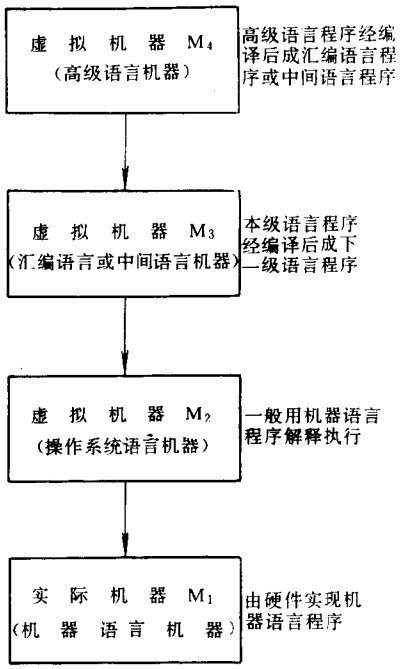



图4·1-1　计算机系统的多层次结构

随着计算机应用的日益普及，人机界面越来越友好，经常以菜单或更方便的形式来引导应用人员进行

操作。

## 1·2 计算机的性能价格比[4]和性能评测[5]

### 1·2·1 计算机的性能和评测

计算机的性能反映了计算机处理问题的速度，可用三种方法来衡量：

**1. 用"时间"来衡量计算机性能** 经常用每秒完成多少个事件（events）来表示。用户关心的是响应时间，即从提出任务开始，到计算机得出结果所需的时间，其中包括CPU操作、访问存储器、存取磁盘数据、I/O操作以及操作系统开销等时间。在计算机运行多道程序环境中，当某道程序因条件不具备而需挂起时，将CPU分配给其他程序运行，因此引入CPU时间的概念，它指的是完成这道程序所需的CPU时间，不包括I/O等待时间和运行其他程序时间。CPU时间又分为用户CPU时间（执行用户程序占用的CPU时间）和系统CPU时间（执行操作系统的CPU时间）。假如执行UNIX的time命令时返回以下数据：

90.7s 12.9s 2：39 65%

其意义为：此进程在用户态占用CPU时间为90.7s，进程在系统态占用CPU时间为12.9s，响应时间为2min39s，进程所用的CPU时间占响应时间的65%。在此例中有35%的时间用于等待I/O或（和）运行其他程序。

$$\text{CPU 时间} = I \ (CPI) \ T$$

式中 $I$——程序运行时所执行的指令数；

$CPI$——执行1条指令的平均周期数；

$T$——周期时间。

**2. 用MIPS来衡量计算机性能** MIPS（每秒执行100万条指令）以各类程序中执行指令的频度为依据，通过计算得到的。一般将VAX11/780计算机的运算速度定为1MIPS，即每秒执行100万条指令，实际上VAX11/780的运算速度小于1MIPS，称为1VAX MIPS。

用MIPS值来衡量机器速度并不完全确切，这是因为各条指令的繁简程度不一，而它们在程序执行时所占比例又不相等的缘故。

**3. 选择程序来评价性能** 基准程序法（Benchmark）是目前一致承认的测试性能的较好方法，有多种多样基准测试程序，如主要测试整数性能的基准程序、测试浮点性能的基准程序等。

（1）整数测试程序。Dhrystone是一个综合性基准测试程序。它本身没有现实的应用意义，仅仅是为了测试编译器及CPU处理整数指令和控制功能的有效性而人为地选择一些"典型指令"，将它们综合起来形成的测试程序。

（2）浮点测试程序。在计算机科学和工程应用领域内，浮点计算工作量占很大比例，因此在产品说明书中往往标出机器浮点性能。有些机器只标出单个浮点操作性能，如浮点加法时间、浮点乘法时间，而绝大部分工作站标出用Linpack和Whetstone基准程序测得的浮点性能。

Linpack基准程序是一个实际使用的子程序软件包，称为基本线性代数子程序包。它用FORTRAN语言写成，此程序主要进行浮点加法和浮点乘法操作，用它来测试机器处理向量的能力以及高速缓存的性能，测试结果用MFLOPS（每秒执行100万条浮点指令）来表示。

Whetstone基准程序是用FORTRAN语言编写的综合性测试程序，主要由执行浮点运算、整数算术运算、功能调用、数组变址、条件转移和超越函数的程序组成。Whetstone的测试结果用Kwips表示，1Kwips表示机器每秒钟执行1000条Whetstone指令。

其他还有Livermore Fortran内核基准程序、Spice基准程序等。

（3）计算机综合测试程序（System Performance Evaluation Coor Perative—SPEC）。SPEC基准程序能比较全面地反映机器性能，被各个公司用来测试工作站的性能。

SPEC是一套复杂的基准程序集，于1989年10月发布，主要用于测量与工程和科学应用有关的数字密集型的整数和浮点数方面的计算能力。它包含10个测试程序，其中有测试机器整数性能，用C语言编写的gcc、espresso、li和eqntott共4个程序；有测试浮点性能，用FORTRAN语言编写的Spice 296、doduc、nasa7、matrix300、fpppp和tomcatv共6个程序。SPEC基准程序测试结果一般有3个：SPEC mark（SPEC分数）、SPEC int（SPEC整数）和SPEC fp（SPEC浮点数）。其中SPEC分数是10个程序的几何平均值，SPEC整数是4个整数程序的几何平均值，SPEC浮点数是6个浮点程序的几何平均值。

1992年发布的SPEC int 92由4个程序扩充到6个程序，SPEC fp 92由6个程序增加到14个程序。

### 1·2·2 计算机的成本与价格[4]

由于电子器件产品市场价格变化很大，因此考虑

计算机成本时，不应按设计时的成本估算，而是要考虑该产品投放市场时的成本。图 4·1-2为构成计算机存储器的主要芯片 DRAM（动态随机存储器）的价格变化情况，图中曲线分别表示 16KB、64KB、256KB 和 1MB4 种芯片投放市场的时间及其与价格的关系。计算机的价格，除了由成本决定外，还受市场供需关系以及是否有多渠道货源可供选择等因素的影响。图 4·1-3从左到右列出从零部件成本到最后标价的增值情况。

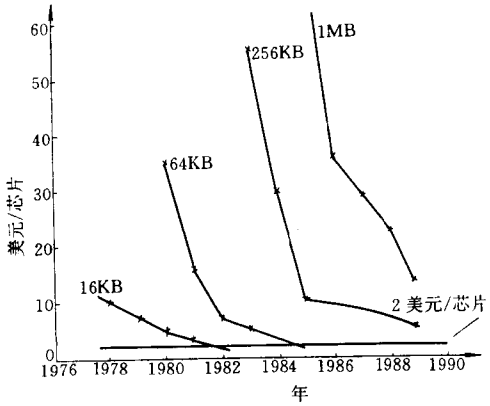



图 4·1-2　存储器芯片价格变化的趋势

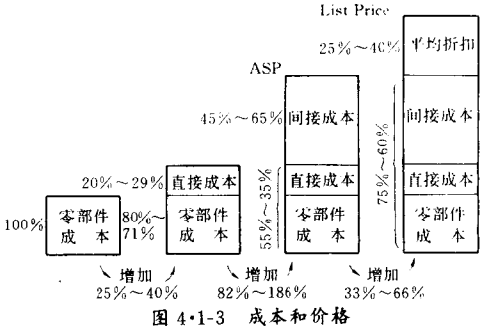



图 4·1-3　成本和价格

（1）零部件成本。购买计算机器件和部件的费用。

（2）直接成本。与产品直接有关的成本，例如劳动成本、采购费用、合理消耗等，直接成本约为零部件成本的 25%～40%。

（3）间接成本。公司的研究开发费用、市场销售费用、生产设备维修费、房租、利润和税收等。

以上三项之和为平均出售价格 ASP（Avevage Selling Price）

（4）标价（List Price）。公司出售计算机系统时要打折扣，因此给用户开的价格高于 ASP，称为 List Price，平均折扣（discont）约占 List Price 的 25%～40%。PC 机经常由代理商出售，他们能支配 List Price 的 40%。

### 1·2·3　开放式计算机系统[6]

推广开放系统是计算机工业最重要的发展趋势之一。当前计算机工业已从“厂商推动”的市场转向“用户推动”的市场，用户愿意购买基于开放系统的计算机，但是究竟什么是开放系统还没有确切定义，现介绍一些比较一致的看法。

**1. 专有系统与开放系统**

（1）专有系统。一些公司不肯向外提供自己的技术专利，因此硬件与软件都由一家公司提供。

（2）半开放系统或基本开放系统。如 IBM PC 机，有很多兼容机，硬件可由多家公司提供，系统软件则由 Microsoft 一家公司提供，而 Microsoft 独立于所有生产硬件的公司。又如 80 年代兴起的一些公司，一开始就考虑采用国际标准，采用开放技术政策，但在某些关键处，例如图形接口上搞了一个专用系统不对外开放，因此不能称为完全的开放系统。

（3）开放系统。硬件和软件都可由多家公司提供。计算机系统的所有部分，如计算机体系结构、系统总线、操作系统、窗口系统等都是开放的，并符合与制造商无关的国际标准。这样硬件、软件公司很容易进行分工，产品能集成在一起工作。用户可任选市场上的软件、硬件组成计算机系统或信息系统。

**2. 开放系统的特点及优势**

（1）遵循标准接口，使得计算机之间有“可移植性”和“互操作性。”

可移植性是指操作系统或应用软件很容易移植到不同厂家的不同型号计算机上使用。互操作性是指不同厂家的各类计算机可以相互交换信息。为达到互操作性，要求各厂家的计算机采用相同的通信协议、数据格式等，具有高度的互连性。

（2）开放系统可由制造商、增值商或用户从不同公司购买设备进行装配、扩充或升级，而且原有软件仍能运行，可以保护用户在软件上的投资。

（3）有大量第三方软件公司或用户的软件可在系统上运行。

（4）开放接口应有一个公开发表的技术规格说明，应以合理的费用提供，并由多家厂商提供产品。

## 1·3　计算机的组成[1]

计算机的硬件由运算器、控制器、存储器和输入输出设备组成，通常由运算器和控制器组成中央处理器 CPU。

输入设备用于输入原始数据和处理这些数据的程序。输入的信息由数字、字母和控制符等组成，经常用 8 位二进制码来表示一个数字（0～9）、一个字母（A、

B，C，…，X，Y，Z）或其他符号。

输出设备用来输出计算机的处理结果，可以是数字、字母、表格和图形等。最常用的输入输出设备是显示终端和打印机。显示终端设备采用键盘作为输入工具，处理结果显示在屏幕上，为了监视人工输入的正确性，在用键盘输入信息时，将刚输入的信息显示在屏幕上，如有错误可及时纠正。

运算器是对信息或数据进行处理和运算的部件，经常进行的是算术运算和逻辑运算，所以在其内部有一个算术及逻辑运算部件ALU。算术运算是按照算术规则进行的运算，例如加、减、乘、除、求绝对值、取负值等。逻辑运算指的是非算术性质的运算，例如比较大小、移位、逻辑乘、逻辑加等。应用于科学计算的计算机，经常设置有浮点运算部件。在计算机中，一些复杂的运算往往被分解成一系列算术运算和逻辑运算。

控制器主要用来控制程序的自动执行。在控制器控制之下，从输入设备输入程序和数据，并自动存放在存储器中，然后指挥各部件（运算器、存储器等）协同工作以执行程序，最后将结果打印输出。作为控制用的计算机则直接控制对象。

存储器用来存放程序和数据，是计算机各种信息的存储和交流中心。存储器可与输入输出设备、运算器、控制器交换信息，起存储、缓冲、传递信息的作用。

在计算机中，各部件间来往的信号可分成3种类型，它们是地址、数据和控制信号，图4·1-4是用总线将计算机各部件连接起来的框图。

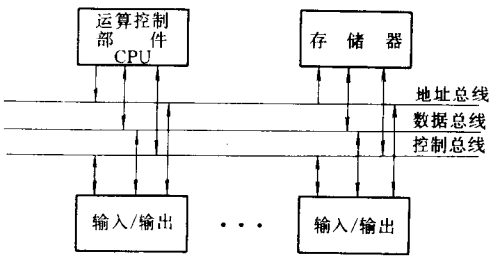



图4·1-4　以总线连接的计算机框图

## 2　数据的表示、转换和运算[1][7]

### 2·1　数制及其转换（参见第2篇第7章1节）

### 2·2　码制（原码、补码、反码）

**1. 正数与负数**　数的符号也用数码表示，一般用“0”表示正数的符号，“1”表示负数的符号。

**2. 原码、补码、反码**　正数的原码、补码、反码是相同的，而负数则不同。设 $x$ 为 $n$ 位小数，用小数点左面一位表示数的符号，则

(1) $[x]_{原}=\begin{cases}x & (0\leqslant x<1)\\ 1-x & (-1<x\leqslant 0)\end{cases}$

数的范围为 $(1-2^{-n})\sim-(1-2^{-n})$

零有两种表示，正零为0.00…0，负零为1.00…0。

(2) $[x]_{补}=\begin{cases}x & (0\leqslant x<1)\\ 2+x & (-1\leqslant x<0)\ (\mathrm{mod}2)\end{cases}$

数的范围为 $(1-2^{-n})\sim-1$

零的表示是唯一的，即0.00…0。

(3) $[x]_{反}=\begin{cases}x & (0\leqslant x<1)\\ (2-2^{-n})+x & (-1<x\leqslant 0)\end{cases}$

数的范围为 $(1-2^{-n})\sim-(1-2^{-n})$

零有两种表示，正零为0.00…0，负零为1.11…1。

将 $[x]_{原}$ 的符号位保持不变，数值部分按位取反（0→1，1→0），可得 $[x]_{反}$。$[x]_{反}$ 的最低位加1，得 $[x]_{补}$。补码的运算规则：

$$[x+y]_{补}=[x]_{补}+[y]_{补}$$

$$[x-y]_{补}=[x]_{补}+[-y]_{补}$$

将加、减法运算统一成加法运算，补码运算的结果仍为补码。

### 2·3　数据的表示及运算

#### 2·3·1　定点数、浮点数及其运算

**1. 定点数表示法**　小数点位置固定的数通常有两种表示法：小数点位于最高位与符号位之间的小数表示法，以及小数点位于最低位后面的整数表示法。

**2. 定点数运算**　算术运算：加、减、乘、除法运算。加减法通常采用补码运算，乘除法用原码运算比较简单，但也可采用补码运算，例如采用布斯补码乘法。

溢出：两个数运算的结果超出机器所能表示的范围，称为溢出。补码加减法运算时判溢出的方法通常有3种。

(1) 两符号相同的数相加，结果的符号与加数不同；或两符号不同的数相减，结果的符号与减数相同。上述情况为溢出。

(2) 双符号位法：每个操作数有两个符号位（正数为00，负数为11），与数值部分一起参加运算，如运算结果两符号位不同，则产生了溢出。

(3) 在数运算过程中，数的最高位和符号位有且仅有一个产生进位信号，为溢出。

当溢出时，发故障中断。

定点数的乘法运算采取逐次加与移位的方法实现，一次可进行一位、两位或多位相乘。

一位原码乘法运算规则：每次根据乘数（数值部分）的每个数位（自低位到高位）是“1”还是“0”，决定加上被乘数还是加上“0”，获得部分积后右移一位，直到做完为止。两个 $n$ 位数相乘，得 $2n$ 位结果。小数乘法运算不会溢出。

定点数的除法运算采取逐次减与余数左移相结合的方法。当每次进行减法运算时，若够减，上商 1；不够减，上商 0。通常采取加减交替法。当除数为 0 或结果溢出发故障中断。

**3. 浮点数表示法**　浮点数由尾数与阶码组成，尾数通常用小数表示，它的长度决定了数据的精度；阶码用于扩大数的范围。数 $N$ 的浮点表示是：$N=MR^e$。式中 $M$ 表示尾数，$e$ 表示指数（即阶码），$R$ 为基数，一般 $R$ 值取 2，8，16，在机器设计时已约定。阶码用补码或移码（也叫增码）表示，移码等于阶码的真值加 $2^{n-1}$（设阶码有 $n$ 位），所以用移码表示的正数，它的最高位为 1，负数的最高位为 0。32 位浮点数为单精度浮点数，64 位浮点数为双精度浮点数。

用相同的位数表示浮点数，采用的基数不同，所能表示的数的范围也不同，如 32 位浮点数（阶码 8 位，尾数 24 位），当基数为 2 时，数的范围为 $-1\times2^{127}\sim(1-2^{-23})\times2^{127}$；当基数为 16 时，数的范围为 $-1\times16^{127}\sim(1-2^{-23})\times16^{127}$。

为了提高数的有效位数，浮点数以规格化的形式出现。当基数为 2 且尾数为补码时，规格化浮点数的特征是尾数最高位与符号位相反，即尾数为 0.1××…×或 1.0××…×。

当一个浮点数的尾数为 0，或阶码小于机器所能表示的最小值时，当作零看待，称为机器零。这时把浮点数的阶码和尾数全置成零。

**4. 浮点数的运算**　参加运算的数是规格化的数，其运算步骤如下：

（1）浮点加减法运算

1）对阶。小阶的数向大阶取齐，先求出两数的阶差，小阶的尾数按阶差右移若干位，使两个数的阶码相等。

2）尾数相加或相减。其方法同定点数运算。

3）结果规格化。如尾数加减后出现溢出，将尾数右移 1 位，阶码加 1，称为右规。如加 1 后阶码溢出，称为上溢。如尾数加减后不溢出，也不是规格化的数，则进行向左规格化，每左移 1 位，阶码减 1，直到结果为规格化数为止。在左规过程中如出现阶码为机器所能表示的最小值，则将结果置成“机器零”，这种情况称为下溢。如机器进行舍入操作，有可能使结果成为不规格化数，需再右规。当发生上溢情况时将引起故障中断。注意仅当阶码溢出时数据才溢出。

（2）浮点乘法运算。阶码相加，尾数相乘。尾数乘法运算规则与定点乘法同，结果规格化。根据阶码判断运算结果是否溢出。

（3）浮点除法运算。阶码相减，尾数相除，结果规格化。根据阶码判断运算结果是否溢出。

### 2·3·2　十进制数及其运算

用 4 位二进制码来表示 1 位十进制数，有 6 种编码是多余的。最常用的 8421 码是一种有权码，它所表示的十进制数 0～9 与二进制表示的代码完全相同，所以又称为以二进制编码的十进制码，简称 BCD 码或二-十进制码。在多位情况下它与二进制码不同，例如十进制数 39，用 BCD 码表示是 00111001，其中前 4 位表示 3，后 4 位表示 9；用二进制码表示则是 00100111。另外有一种称为余 3 码，是由 8421 码加 0011 得来的无权码。表 4·1-1列出几种常见的十进制编码。

**表 4·1-1　常见的十进制编码**

| 十进制数 | 二进制码 | BCD 码（8421 码） | 余 3 码 |
|---|---|---|---|
| 0 | 0000 | 0000 | 0011 |
| 1 | 0001 | 0001 | 0100 |
| 2 | 0010 | 0010 | 0101 |
| 3 | 0011 | 0011 | 0110 |
| 4 | 0100 | 0100 | 0111 |
| 5 | 0101 | 0101 | 1000 |
| 6 | 0110 | 0110 | 1001 |
| 7 | 0111 | 0111 | 1010 |
| 8 | 1000 | 1000 | 1011 |
| 9 | 1001 | 1001 | 1100 |

十进制数（8421 码）的运算规则：

当某位相加结果小于等于 9 时，所得结果即为 BCD 码，否则本位加 6，并向高位送 1（进位）。

某些计算机可直接对二-十进制数据运算，此时按上述规则对结果进行修正。

### 2·3·3　ASCII 码[1]和 EBCDIC 码

ASCII 码是美国信息交换标准委员会制定的，它

用7位二进制码来表示128个符号，包括数字0～9，大、小写英文字母，运算符和控制符等。其中数字0～9的低4位与对应的二进制编码相同。一般在ASCII码的最高位附加1位校验位，用8位二进制码（一个字节）来表示一个字符。

EBCDIC码是扩展的二-十进制交换码，用8位二进制码来表示信息。数字的低4位与BCD码相同，因此进入机器后可压缩成4位，一个字节可放两个十进制数字。

### 2·4　逻辑运算

最基本的逻辑运算为逻辑加（∨，+）、逻辑乘（∧，·）与逻辑非，由构成机器的基本电路（或门、与门、非门）来完成运算，参见第2篇第7章。表4·1-2列出逻辑加与逻辑乘的真值表，A、B为两个输入，逻辑非由反相器实现。

表　4·1-2

| 输　入 | | 输　出　结　果 | |
|---|---|---|---|
| A | B | 逻辑加 | 逻辑乘 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |

### 2·5　数据校验码

数据在传送、存储过程中有可能出错，随着计算机科学的发展和应用范围的扩大，要求计算机能自行检错，甚至纠错。

**1. 码制的距离**　两个码字的距离：两个码字逐位比较，其不同代码的个数称为这两个码字的距离。例如001110与101011有3个代码不同，距离为3。

一个码制的距离：在此码制中各码字之间的最小距离。二进制数码的距离为1，因为改变任意一个码字中的任意1位（0变1或1变0），仍然是这码制中的一个码字，也就是说不存在冗余码。假如产生错误，计算机不能检测出来。但如果码制的距离为2，当其中1位传送出错时，将不再是这码制中的码字，即为冗余码，此时可判断出是个错字，但不知哪一位出错。当距离再增大时有可能纠错。

**2. 奇偶校验**　在每个字组（即码字）中增加一个校验位，使整个字组中的"1"的个数凑成奇数（奇校验）或偶数（偶校验），统称奇偶校验。

距离为1的二进制码加上奇偶校验位，就成为2距离码，这种编码能发现1个错误或奇数个错误，但不能纠正。

**3. 海明校验**　海明校验以奇偶校验为基础，但校验位不是一个，而是一组。假设有$k$个校验位，则它能表示$2^k$个信息，用其中的一个信息指出"没有错误"，其余的$2^k-1$个信息指出错误发生在哪一位。由于也可能是校验位出错，所以只有$n=2^k-1-k$个信息能用于纠正被传送的数据，即数据的长度不能超过$n$位。假如校验位有4位，则$n\leqslant 11$位，假如存储一个32位数，用上式推算需要设置6个校验位。

**4. 循环冗余校验**　磁表面上的缺陷，如氧化点、涂层不匀，划伤、灰尘等引起存储器出错，通常它是在一磁道上连续集中发生的，叫做猝发性错误(Burst Error)。循环冗余校验码（CRC）可校验并纠正这些错误。

## 3　指令系统[1][7]

指令系统是计算机所有指令的集合，是硬件设计人员和程序员都能见到的机器的主要属性。它决定了计算机的基本功能及其复杂程度，并直接影响到操作系统及编译程序的编写与程序运行效率。

### 3·1　指令类型

一台计算机最基本的、必不可少的指令是不多的，因为很多指令可以用其他指令组合起来实现。例如乘法/除法运算指令、浮点运算指令既可以用硬件实现，也可以用由简单的指令编成子程序来实现，但这两种方法所需时间差别很大，因此在指令系统中，有相当一部分指令是为了提高程序的执行速度和便于程序员编写程序而设置的。

一台计算机通常有几十条至几百条指令，按它所完成的功能可分为算术逻辑运算指令、移位操作指令、浮点运算指令、十进制运算指令、字符串处理指令、向量运算指令、数据传送指令、转移指令、堆栈操作指令、输入/输出指令、特权指令等。下面简述各类指令的功能。

**1. 算术逻辑运算指令**　此处的算术运算是指定点数运算，即相当于高级语言中对整数（Integer）的处理。有些低档微型机要求价格便宜，因此硬件结构比较简单，支持的算术运算指令就较少，一般只支持二进制加、减法，比较和取负数等最基本的算术指令；而其他计算机由于要兼顾性能和价格两方面因素，还设置乘

除法运算指令。通常根据算术运算的结果置状态位(或称条件码)，一般有 Z（结果为 0）、N（结果为负）、V（结果溢出）、C（产生进位或借位）等状态位。当满足括弧内所提出的条件时，相应的条件位置成“1”，否则为“0”。例如运算结果为 0 时，Z=1，否则 Z=0。

逻辑运算指的是对两个数进行逻辑操作，通常计算机具有逻辑乘（与）、逻辑加（或）、按位加（异或）、求反（非）等逻辑运算指令。有些计算机还设置位操作指令，如位测试（测试指定位的值）、位清除（把指定位清零）、位求反（取某位的反码）指令等。

**2. 移位指令**　移位指令分为算术移位、逻辑移位和循环移位 3 种，将操作数左移或右移若干位，其过程见图 4·1-5。算术移位与逻辑移位的操作对象不同（前者的操作数为带符号数，后者的操作数为无符号数），所以移位的结果有所不同。它们的主要差别在于右移时移入最高位的内容不同：算术右移保持最高位(符号位）不变，而逻辑右移最高位补零。循环右移将最低位移出的数送入最高位，循环左移将最高位的数送入最低位。

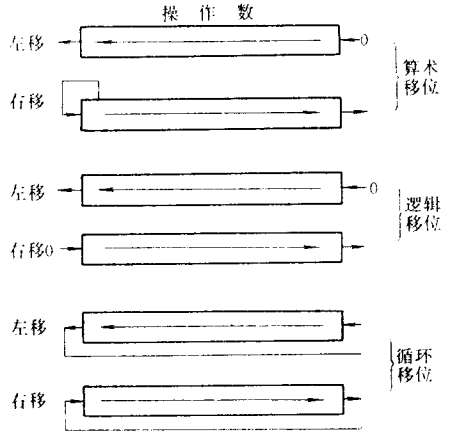



图 4·1-5　移位操作过程

算术逻辑移位指令还可以实现简单的乘除法运算，算术左移或右移 $n$ 位，分别实现带符号数据乘以或整除以 $2^n$ 的运算。移位指令的执行时间比乘除运算短，因此用它来实现简单乘除法运算可取得较高的速度。

**3. 浮点运算指令**　高级语言中的实数（Real）经常先转换成浮点数形式而后再进行运算。有些计算机没有设置浮点运算指令而用子程序实现，其速度很低，因此主要用于科学计算的计算机应该设置浮点运算指令。浮点数通常有单精度（32 位）、双精度（64 位）两种数据格式。一般计算机的浮点数及其操作符合 IEEE754 浮点标准。

**4. 十进制运算指令**　从外界输入计算机的数据通常是以十进制形式表示的。在某些数据处理系统中，输入输出的数据很多，但对数据本身的计算却很简单。在不具有十进制运算指令的计算机中，首先将十进制数据转换成二进制数，然后在机器内运算，最后又转换成十进制数据输出，会将大量 CPU 时间消耗在数据转换上，如设置十进制运算指令能提高数据处理速度。

**5. 字符串处理指令**　早期的计算机主要用于科学计算或军事上，侧重于数值运算，只有少数计算机才有非数值处理指令。而随着计算机应用领域的扩大，更多的计算机用于信息管理、数据处理、办公室自动化等领域，这就需要有很强的非数值处理能力，因此现代计算机越来越重视非数值指令的设置。

字符串指令就是一种非数值处理指令，一般包括字符串传送、字符串比较、字符串查询、字符串转换等指令。

**6. 数据传送指令**　这类指令用以实现寄存器与寄存器、寄存器与存储器单元、存储器单元与存储器单元之间的数据传送。有些机器设置了数据交换指令，完成源操作数与目的操作数的互换。

**7. 转移类指令**　这类指令用以控制程序流的转移。在大多数情况下计算机是按顺序方式执行程序的，但也会遇到离开原来的顺序转移到另一段程序或循环执行某段程序的情况。

(1) 无条件转移与条件转移。无条件转移指令不受任何条件约束，转移到指令所规定的目的地。条件转移指令则根据计算机处理结果来决定程序如何执行，它先测试根据处理结果设置的条件码，然后根据条件是否满足来决定转移与否，如条件为“真”则转移，如条件为“假”则顺序执行下一条指令。通常用算术指令建立的条件码 N、Z、V、C 来控制程序的执行方向，实现程序的分支。

(2) 调用指令与返回指令。在编写程序过程中，一般预先编写一些常用的、能独立完成某一功能的程序段，在需要时能随时调用，以免多次重复编写，这种程序段称为“子程序”或“过程”。

通常使用“调用指令”(Call 指令）从一个程序转移到另一个程序，并将调用指令的地址保存起来。当执行完被调用程序段后，执行一条返回指令，使程序回到原调用程序继续执行下去。所以调用指令（Call）和返回指令（Return）是一对配合使用的指令。

(3) 陷阱(Trap)和陷阱指令。当计算机运行出现故障(如电源电压不稳、执行非法指令等)时，自动发出陷阱信号中止当前程序的执行，转入故障处理程序进行相应的故障处理，有的计算机设置可供程序员使用的陷阱指令，利用它来实现系统调用和程序请求。

**8. 堆栈及堆栈操作指令**　堆栈(Stack)是由若干个连续存储单元组成的先进后出(FILO)存储区。第一个送入堆栈的数据存放在栈底，最后送入堆栈中的数据放在栈顶，用一个称为堆栈指针(SP)的寄存器指出栈顶地址。在一般计算机中，堆栈主要用来暂存中断或子程序调用时的现场数据及返回地址，用于访问堆栈的指令只有压入(即数据进栈)和弹出(即数据从堆栈中取出)两种。在堆栈结构计算机中，堆栈还用来提供操作数和保存运算结果。

**9. 输入/输出(I/O)指令**　输入指令将指定的I/O设备寄存器的内容读入CPU，输出指令则将寄存器或存储器单元中的内容送到输出设备。

有些计算机采用I/O设备与存储器统一编址的方法，把I/O设备中的寄存器看成是存储器的某些单元，任何访问存储器的指令均可访问I/O设备，因此不再设置I/O指令。

**10. 特权指令**　用户对某些指令使用不当会破坏系统或其它用户程序，为了安全起见，这类指令只能用于操作系统或系统软件，而不提供给用户使用，称为特权指令。

**11. 多处理机指令**　在多处理机系统中，为了管理共享的公共资源和相互通信，一般设置“测试与设定”或“数据交换”指令。

**12. 向量指令**　参见本章第7·3节。

## 3·2　指令格式

通常1条指令包含下列信息：

**1. 操作码**　说明指令所规定的操作，例如加法、减法或转移等，每一条指令有一个相应的操作码，7位操作码最多可表示128条指令，有的计算机操作码位置不固定，位数也是可变的。

**2. 源操作数的来源**　可以是寄存器地址或存储器地址，根据此地址可取得源操作数；也可以是数据本身，此时称为立即数。

**3. 目的地址**　运算结果保存在目的地址中，可以是寄存器地址或存储器地址。

**4. 下一条指令地址**　当顺序执行程序时，下一条指令默认由程序计数器(PC)给出，仅当改变程序的运行顺序时(如执行转移指令或调用指令)，下一条指令的地址与本条指令的地址码有关。

上述2、3、4统称为地址码，根据地址码部分所给出的地址个数来划分指令，可以有零地址、一地址、二地址、三地址等多种格式，如图4·1-6所示。

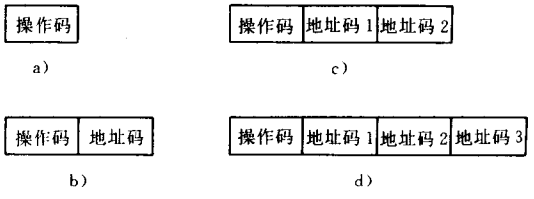



图4·1-6　指令格式

a) 零地址　b) 一地址　c) 二地址　d) 三地址

目前计算机数据字长一般为32位，存储器的地址一般按字节表示。通常计算机的指令系统可支持对字节、半字、字或双字的运算。为便于硬件处理，一般要求多字节数据对准边界，当以二进制表示地址时，半字地址的最低位恒为零，字地址的最低两位为零，双字地址的最低三位为零。某些机器不要求数据对准边界，则不受上述限制。

## 3·3　寻址方式(编址方式)

确定操作数或下一条指令地址的方式。

**1. 寄存器直接寻址或存储器直接寻址**(图4·1-7a、b)　指令的地址码部分给出操作数地址，图4·1-7中仅给出一个操作数地址；当有多个地址时情况类似。

**2. 基址寻址**(图4·1-8a、b)　在计算机中设置一个专用的基址寄存器，或由指令指定一个通用寄存器作为基址寄存器，操作数的地址由基址寄存器的内容与指令的地址码相加，此时地址码A称为位移量Disp。

基址寄存器主要用于为程序或数据分配存储器，其值由系统程序通过特权指令设置。

**3. 变址寻址**(图4·1-8c)　操作数的地址由变址寄存器的内容与指令的地址码A相加得出。

**4. 间接寻址**(图4·1-7c、d)　根据指令的地址码所取出的内容既不是操作数，也不是下一条要执行的指令，而是操作数的地址或指令的地址，称为间接寻址或间址。

**5. 相对寻址**　将程序计数器PC的内容(即当前执行指令的地址)与位移量Disp之和作为操作数地址或程序的转移地址，称为相对寻址。主要用于转移指

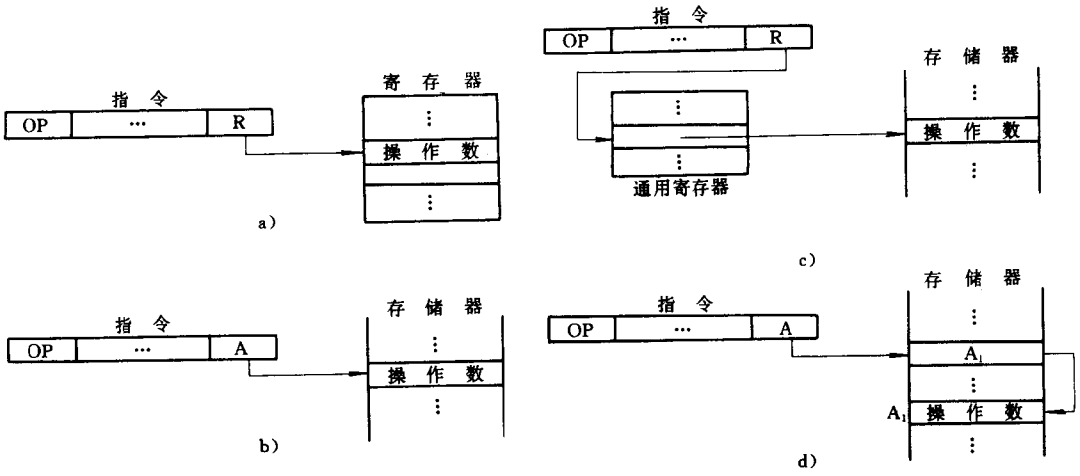



图 4·1-7 直接寻址和间接寻址

a) 寄存器直接寻址 b) 存储器直接寻址 c) 寄存器间接寻址 d) 存储器间接寻址

注：OP 为操作码

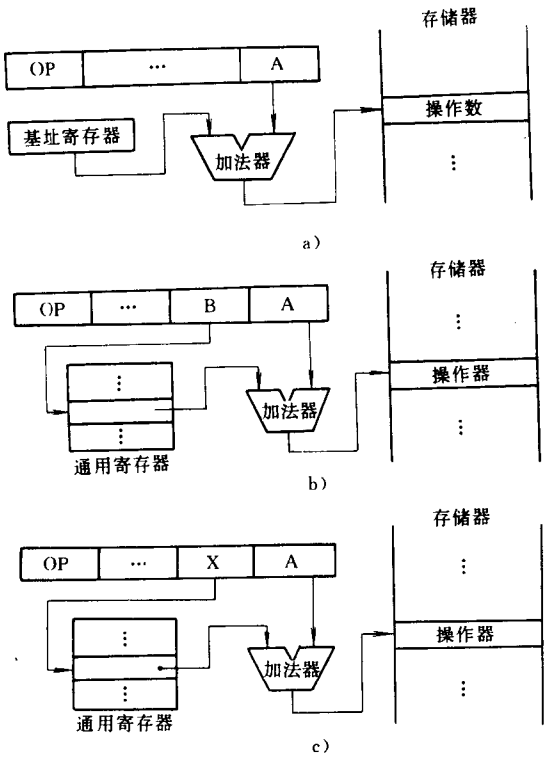



图 4·1-8 基址寻址和变址寻址

a) 基址寻址（专用基址寄存器） b) 基址寻址（通用寄存器作基址寄存器） c) 变址寻址

令，此时转移地址随着 PC 值变化，无论程序装入存储器的任何地方，均能正确运行，适用于浮动程序。

**6. 立即数** 指令的地址码部分直接给出操作数，称为立即数。

## 3·4 指令系统的兼容性

各计算机公司生产的计算机，其指令数量、指令功能、指令格式、寻址方式和数据格式等都有较大差别，因此尽管各种型号计算机的高级语言基本相同，但将它翻译成机器语言后差别很大，所以将机器语言程序移植到其他机器上去几乎是不可能的。由于计算机硬件发展很快，新机型不断出现，因此存在软件如何跟上的问题。通常新机器推出时，仅有少量系统软件（如操作系统、编译程序）可提供给用户，大量软件在其后不断充实，应用程序中有相当一部分是用户在使用机器时不断开发的，这就是所谓第三方提供的软件。为了缓解新机器的推出与原有应用程序继续使用的矛盾，1964 年 IBM 公司在设计 IBM360 系列计算机时采用了系列机思想。从此以后各个计算机公司生产的同一系列的计算机尽管其硬件实现方法可以不同，但指令系统、数据格式、I/O 系统等保持一致，因而软件完全兼容。当研制该系列计算机的新型号或高档产品时，允许指令系统进行扩充，但仍保留原有的全部指令，保持软件向上兼容的特点，即低档机或旧型机上的软件不加修改即可在新机器上运行，可以保护用户在软件上的投资。

## 3·5 精简指令系统计算机(RISC)和复杂指令系统计算机（CISC）

随着 VLSI 技术的发展，计算机的硬件成本不断

下降，软件成本不断提高。为了提高操作系统效率，便于编译和降低软件成本，同时为了缩短指令系统与高级语言的语义差别，在计算机中增加了不少复杂指令，有的计算机还具有多种寻址方式。人们称这些计算机为“复杂指令系统计算机”，简称CISC。这种日趋庞大的指令系统增加了计算机的设计难度。1975年IBM公司通过对指令系统合理性的研究提出了精简指令系统的想法。

对CISC进行测试表明，各种指令的使用频率相差甚远，最常使用的是一些比较简单的指令，仅占指令总数的20%，但在程序中出现的频率却占80%，某些复杂指令实际上用得很少。经编译后的程序，使用的寻址方式种类也不多，因此在80年代中后期，RISC工作站/服务器/小型机得到很快发展，到90年代初期已成为这一领域中的主流计算机。

精简指令系统计算机（Reduced Instruction System Computer-RISC）的着眼点并不是简单地放在简化指令系统上，而是通过简化指令使计算机的结构更加简单合理，通过减少指令的执行周期数提高运算速度。

RISC的指令系统具有以下特点：

(1) 选取使用频率最高的一些简单指令以及很有用但不复杂的指令，从而使大部分指令在采用流水线组织的情况下能在一个机器周期内实现。

(2) 指令长度固定，指令格式种类少，寻址方式种类少。

(3) 只有Load/Store（取数/存数）指令访问存储器，其余指令的操作都在寄存器之间进行，因此CPU中通用寄存器的数量相当多。

目前，世界上主要的计算机公司和半导体公司都已生产或应用RISC处理器构成计算机系统。有关RISC芯片的性能及其生产厂家参见本篇第2章2·6节。

## 4 中央处理器（CPU）[1]

CPU中包括算术逻辑运算部件、寄存器组（或称为寄存器堆）、控制器、时钟以及一些专用寄存器。算术逻辑运算部件、寄存器组以及数据传送电路统称为数据通路。CPU的主要功能是控制程序的执行以及完成对数据的处理。

### 4·1 计算机执行程序的过程

计算机硬件执行的是机器语言程序，它是由指令组成的。计算机进行信息处理的过程分成两个步骤：

(1) 将数据和程序调入计算机主存储器中。

(2) 从“程序入口”开始执行此程序，得到处理结果后，结束运行。程序入口指的是此程序开始执行的第一条指令地址。

当机器刚加电时，假如不采取措施，那么半导体存储器以及寄存器的内容将处于随意状态，可能会执行一些不该执行的操作。通常机器加电时，由硬件产生一个复位信号（Reset）使计算机处于初始状态：将某些控制寄存器状态置“0”，并将加电后要执行的第一条指令的地址送程序计数器PC，然后从PC指示的程序地址开始执行程序。首先要把外存储器中的程序调入主存，或从键盘输入程序。为保证正常工作，在机器内一般设置有存放“引导程序”的只读存储器，利用加电时产生的Reset信号，使计算机从“引导程序”入口开始运行，从外部设备输入操作系统或用户程序等。有的机器接着对计算机各部件进行测试，然后进入操作系统环境。因此计算机的工作过程可描述如下：

加电→产生Reset信号→执行程序→停机→停电

执行程序的过程就是不断执行指令的过程。

一条指令的执行过程如下：

**1. 取指** 将当前要执行的指令地址送存储器，从存储器取出指令，送指令寄存器IR。

**2. 指令译码** 产生实现本条指令操作所需的控制信号。

**3. 执行** 完成本条指令的操作。除此以外，还要确定下一条指令地址，当实现程序转移时，将转移地址送程序计数器，否则将PC+1送PC，并判断本条指令是否是停机指令，若是，则执行4.，否则返回1.，继续执行下一条指令。

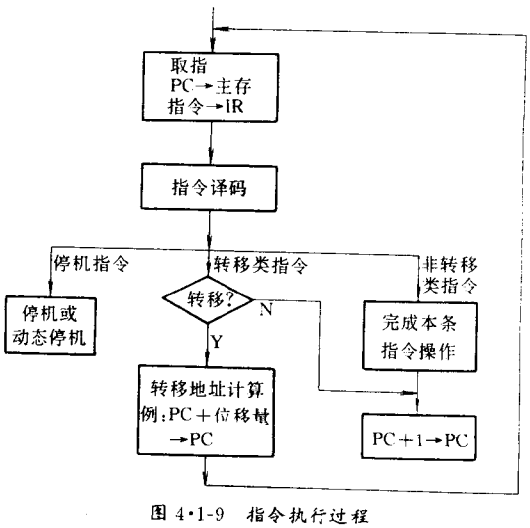



图4·1-9 指令执行过程

**4. 停机或动态停机**　停机是使 CPU 暂停工作；动态停机，则让机器进入无休止的循环程序中(例如执行一条转移到自身的无条件转移指令)，待有外来事件引起中断后脱离动态停机状态。在多道程序环境中一般采取动态停机方式。

上述过程可用图 4·1-9表示。

## 4·2　中央处理器的组成

图 4·1-10 是经过简化后的 CPU 简框图，图的右半部为运算器，左半部为控制器。

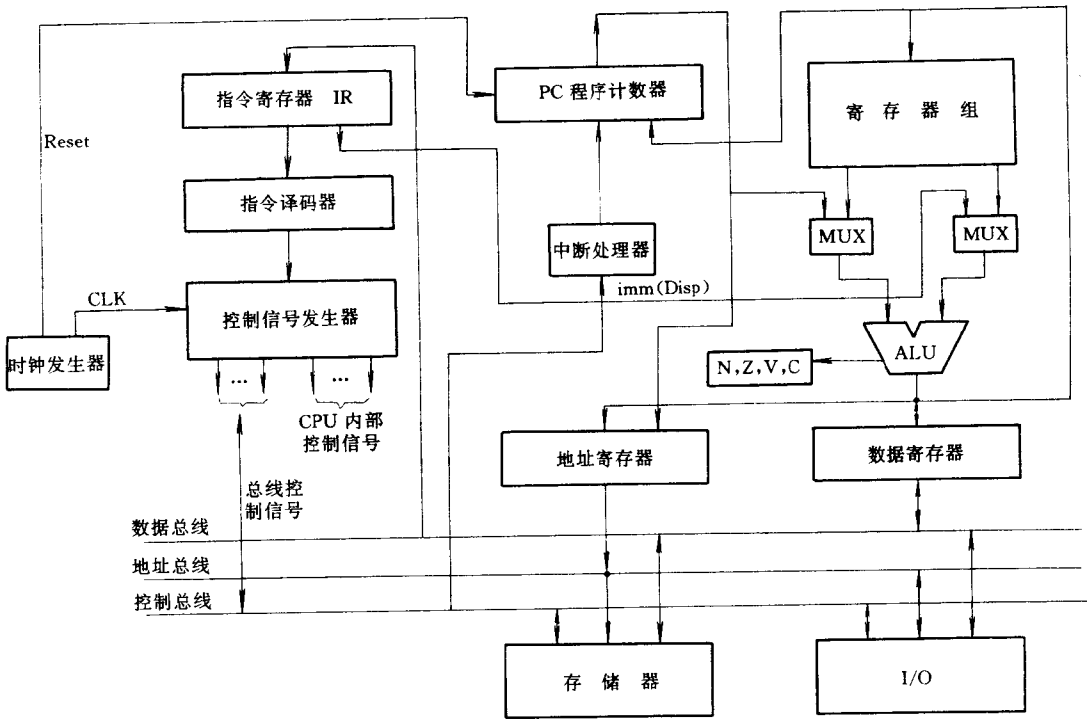



图 4·1-10　CPU 简框图

### 4·2·1　运算器的组成

**1. 算术逻辑运算单元**(ALU)　完成定点数运算。

**2. 寄存器组**　保存参加运算的操作数和运算结果等。运算时，寄存器的内容通过多路转接器 MUX 送到 ALU。

有效地址的计算可在上述 ALU 中进行，也可另设一个 ALU 完成。

**3. 地址寄存器和数据寄存器**　暂存访问存储器的地址及读写数据。

**4. 状态寄存器**　记录运算结果的状态标志，如 N、Z、V、C 等。

### 4·2·2　控制器的组成

**1. 程序计数器 PC**　保存当前正在执行的指令地址。

**2. 指令寄存器 IR**　保存当前正在执行的指令。

**3. 指令译码器和控制信号发生器**　对指令进行译码，并发出完成本条指令功能所需的控制信号。

**4. 中断处理器**　在机器运行时，当发生诸如出错等异常情况或 I/O 设备请求处理时，向中央处理器发出中断请求信号，要求 CPU 暂时中止当前正在执行的程序，转而进行异常处理或 I/O 处理。

**5. 时钟**　CPU 按一定规则进行工作，在机器内必须有一时钟发生器，产生周期性变化的时钟信号。由一个或若干个时钟组成一个机器周期，完成某一项工作，例如完成加法运算，或从存储器取数等。CPU 完成不同操作所需的机器周期数可以不相等。不同的 CPU，组成机器周期的时钟数也是不同的。

## 4·3　控制信号形成逻辑

产生控制信号有两种方法，一种是利用逻辑电路实现，主要由门电路及触发器来实现控制，称为硬布线逻辑或组合逻辑。另一种称为微程序控制。

### 4·3·1　微程序控制的概念

一条指令的执行过程分成若干个机器周期，每个机器周期完成一些规定的操作，假如将一个机器周期完成的操作定义为一条微指令的操作，那么按一定次序执行若干条微指令就可以实现一条指令的功能。这些微指令的集合就叫做微程序。所以微程序的概念与程序设计中的子程序概念相当。每条指令都有相应的一段微程序（或称为微子程序），执行一条指令实际上就成为执行一段微程序。微程序存放在控制存储器中如图 4·1-11a。当指令送入 IR 后，在指令译码器中将操作码翻译成该条指令的微程序入口地址，从控制存储器中取出第一条微指令，保存在微指令寄存器中。微指令由两部分组成，其一为控制部分，发出完成本条微指令所需的控制信号，并有连接线与需要这些信号的电路相连接。另一部分为下址，指出下一条微指令的地址，称为微地址。假如微程序顺序执行，也可将当前微程序计数器加 1 后形成下址。

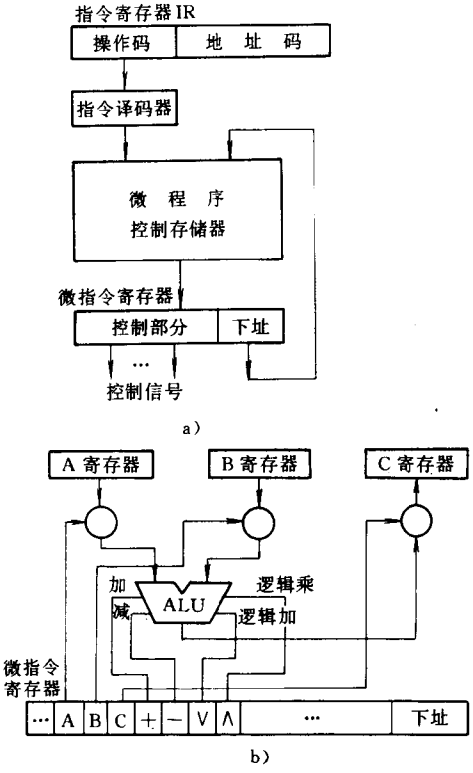



图 4·1-11　微程序控制器与微指令

a）微程序控制器　b）微指令控制示意图

图 4·1-11b 为微程序控制示意图。假设有 A、B、C 三个寄存器，并有一个 ALU 部件可完成加（+）、减（−）、逻辑乘（Λ）和逻辑加（V）四种运算，图示微指令可实现（A）OP（B）→C 运算，OP 表示上述四种运算中的任意一种。在微指令中设置 A、B、C 三个控制位分别控制 A 寄存器与 B 寄存器的输出门以及 C 寄存器的输入门，当相应的控制位为“1”时，将门打开，允许数据传送。另外设置有+、−、Λ、V 四个控制位，当需要进行加法运算时，控制位“+”为“1”，其他三个控制位为“0”。由于在同一时刻，ALU 只能进行一种运算，所以上述四种运算是互斥的，而 A、B、C 的控制信号则不是互斥的。

图 4·1-12 是微程序流程图，每一方框表示一条微指令，框的右上角的数字表示本条微指令的地址，框内右下角的数字表示下一条要执行的微指令的地址。

微程序控制器一般采用只读存储器（固存），那是因为机器的指令系统是固定的，假如用可读、可写的存储器代替固存，那就允许用户或程序员设计新的指令，只要编写相应的微程序送入控制存储器就行了，这就叫做动态微程序设计。

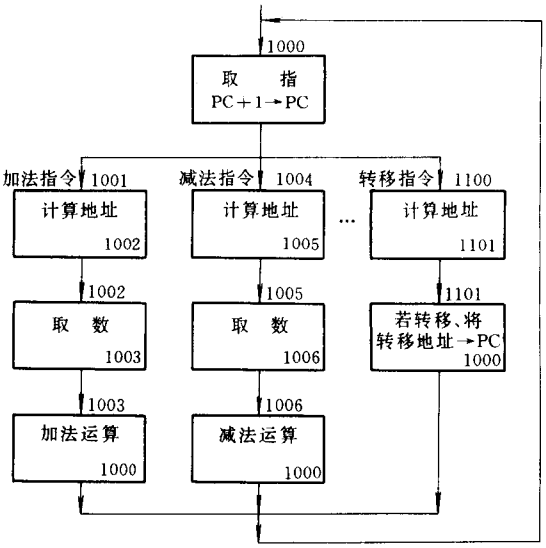



图 4·1-12　微程序流程图

微程序控制速度一般比组合逻辑低，因为一条指令由若干条微指令实现，要多次访问控制存储器。

微程序设计具有软件的特点，当设计存在某些错误时，可通过修改微程序内容予以纠正，所以 CISC 都采用微程序设计技术。

### 4·3·2　微指令编译法

微指令由控制字段与下址字段组成。下面讨论几

种常用的控制字段编译法。

**1. 直接控制法** 在控制字段中，一位代表一个微命令如图 4·1-11b 所示。

**2. 字段直接编译法** 将互斥的微命令编成一组，成为微指令的一个字段，用二进制编码来表示，例如将 7 个互斥的微命令编成一组，用 3 位二进制码来表示，这样这一字段就从 7 位缩减成 3 位，缩短了微指令的长度，但在微指令相应字段的输出端增加一个译码器，在译码器的输出可以有 8 个微命令，其中 7 个是互斥微命令，另外一个表示不操作。

**3. 字段间接编译法** 如果在字段直接编译法中，还规定一个字段的某些微命令，要由另一字段中的某些微命令来解释，这就是字段间接编译法，它可以进一步缩短微指令字的长度。

**4. 常数源字段** 在微指令中，还可设置一个常数字段，就像指令中的立即数一样。一般用来给 CPU 中的某些部件发送常数，所以有时称为发射字段，其位数通常在 4 位到 8 位之间。

下面讨论微指令的下一地址（下址）问题。

在顺序执行微指令时，后继微指令的地址是由现行微地址加上一个增量而产生的，通常由微地址计数器实现。而在非顺序执行微指令时，必须通过转移方式，使现行微指令执行完后能够按既定方式转移到指定的微指令地址，此时下址字段就要发挥作用了。此外，为了便于微程序循环，在微程序控制器中通常设置有一个计数器，用来累计循环次数，以控制循环是否结束，同时为了便于实现微子程序或微程序嵌套，还可以设置微程序返回寄存器或微堆栈（寄存器）。

### 4·3·3 水平型微指令和垂直型微指令

根据控制方式不同，微指令可分为水平型微指令和垂直型微指令。

**1. 水平型微指令** 微指令控制部分的长度是根据控制全机所需的信号而定的，每个信号 1 位，整机所需的信号可能有几十个到几百个。除了上面提到的控制读写寄存器的信号以及 ALU 的信号外，还有控制存储器读写、程序转移、I/O 操作…等控制信号。为缩短微程序长度（字数），提高指令的执行速度，在设计微指令时，应尽量提高操作的并行度，即能同时执行的操作尽量安排在一条微指令中。

**2. 垂直型微指令** 垂直型微指令的格式与指令相类似，一条微指令分成微操作码和微地址码两部分，它与指令的区别是一条微指令只能完成一个操作。

由此可见，垂直型微指令与水平型微指令比较，微指令字的宽度短，但微程序的长度长（即字数多），执行指令的速度慢。

### 4·3·4 硬布线控制逻辑

通过逻辑电路直接连线产生控制计算机各部分操作所需的控制信号的方式称为硬布线控制方式或组合逻辑控制方式。

**1. 机器周期** 一条指令的实现需要几个机器周期，例如 Load 指令一般需要取指、计算地址、访问存储器（取数）、写入寄存器 4 个机器周期，图 4·1-13 用两位计数器来表示一条指令的 4 个机器周期，在初始化时（Reset）令机器处于取指周期，这样可保证机器从取指周期开始工作，此时译码器输出端 0 为高电位，其他 3 个为低电位。

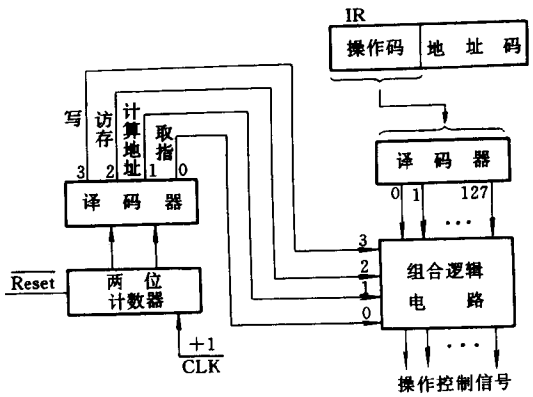



图 4·1-13 硬布线控制逻辑控制信号的形成

由于每条指令的功能不同，所以所需的机器周期数可能不同，例如某些指令可能缺少某个周期，而有些复杂指令的某个周期则需要延长或需增加若干个机器周期，因此实际电路要复杂得多。

**2. 操作码译码器** 指令的操作码部分表示当前正在执行的是什么指令，假如操作码有 7 位，则最多可表示 128 条指令，可在机器内设置一个指令译码器，其输入为操作码（7 位），输出有 128 根线，在任何时候，有且仅有一根线为高电位，其余均为低电位（或反之），反映出当前正在执行的指令。译码器的实际电路与上述的差别很大。

**3. 操作控制信号的形成** 由两个译码器（周期和指令）送来的信息，通过图 4·1-13 所示的组合逻辑电路，产生执行当前指令所需的控制信号。组合逻辑电路通常由门电路组成，设计工作量大，容易出错。

## 4·4　流水线实现原理

### 4·4·1　基本原理

前面讲到“指令执行过程”的基本出发点是一条指令执行完毕后再执行下一条指令。而当今不少计算机采取流水线组织，在一条指令没有执行完时就进行下一条指令的处理，即相邻指令重叠执行(在时间上)。图4·1-14为SUN Microsystems公司的SPARC芯片指令重叠执行情况，它将一条指令（如ADD加法指令）的执行过程分成四个阶段，分别是：(1) 取指；(2) 译码与取数；(3) ALU（加法）运算；(4) 写结果。当计算机取得第$n$条指令后，立即取第$n+1$条指令。图中在$t_{n+3}$周期以后，有4条指令在同时执行。假如每一个机器周期的时间为$t$，那么一条指令的执行时间为$4t$，当第一条指令处理完后，每隔$t$时间就能得到一条指令的处理结果，平均速度提高了4倍，其过程相当于现代工业生产装配线上的流水作业，因此称之为流水线处理机。在程序开始执行时，由于流水线未装满，有的部件没有工作，所以效率较低。要求流水线每一阶段的处理时间基本相同，否则取4段中时间最长的一段作为$t$值，此时有些部件会经常处于空闲状态，而影响流水线作用的发挥。为了解决这一问题，可采用将几个时间较短的功能段合并成一个功能段，或将时间较长的功能段分成几段，最终使各段所需时间相差不大。当执行某些指令或遇到一些特殊情况而在一个机器周期中完不成指定的操作时，有时会采取临时延长周期或冻结时钟的办法。

以上讨论的是指令级流水线。其他还有运算操作流水线（见图4·1-15），例如执行“浮点加”运算可分成“对阶”、“尾数相加”及“结果规格化”3段，每段设置专用逻辑电路完成指定操作，并将其输出保存在锁存器中，作为下一段的输入，如图4·1-15b所示。

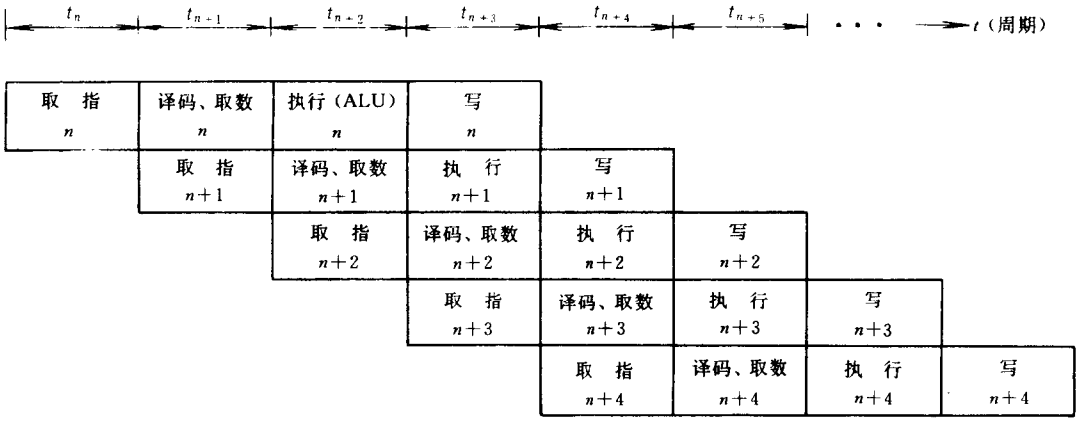



图4·1-14　SPARC指令流水线

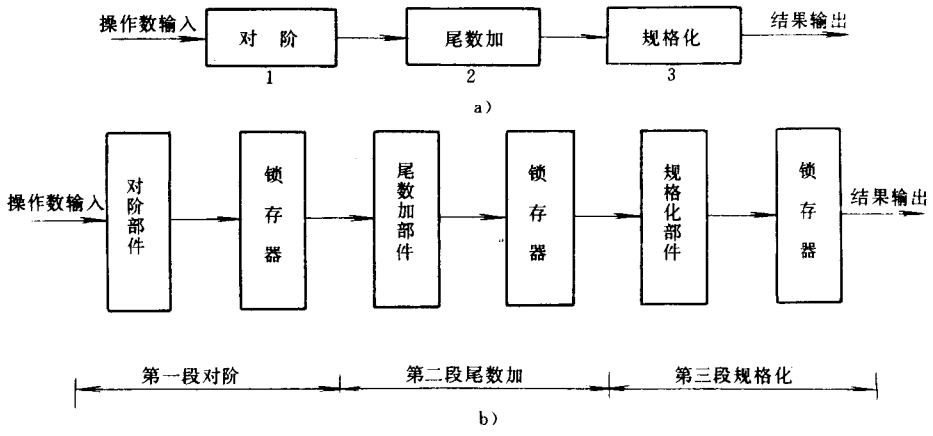



图4·1-15　运算操作流水线