单片微型计算机原理及其应用

陈伟人 编著

DAN DAN

清华大学出版社

单片微型计算机原理及其应用

陈伟人　编著

清华大学出版社

内 容 简 介

本书介绍美国 Intel 公司 MCS-48 和 MCS-51 系列单片微型计算机的基本原理和应用方法。全书共分五章，前四章介绍微型计算机的基础知识、MCS-48 和 MCS-51 系列单片微型计算机硬件结构、指令系统、扩展方法以及常用外围芯片；最后一章详细讲述开发应用问题，并列举了大量线路、程序和应用实例。内容通俗易懂，注重实用。

可供各行各业计算机应用人员、广大具有高中以上文化程度的青年电工、一般科技人员和大中专学校师生参考，也可作为单片微型计算机应用短训班的教材。

单片微型计算机原理及其应用

陈伟人 编著

☆

清华大学出版社出版

北京 清华园

北京京辉印刷厂印刷

新华书店北京发行所发行

☆

开本：787×1092 1/16 印张：21 字数：540千字

1989年5月第1版 1989年5月第1次印刷

印数：00001—15000

ISBN 7-302-00431-5/TP·143

定价：8.30元

序

当今世界正面临着一场以计算机技术为标志的新技术革命，计算机的作用引起人们的极大关注，计算机的应用得到了各方面的普遍重视。可以预见，在我国未来的社会主义建设事业中，计算机必将发挥它越来越巨大的作用。

综观计算机的发展趋势，它可以归纳为两大方面：一是向着快速、实时、智能方向发展；一是向着微型、简便方向发展。超速、并行的巨型机属于前者；而单片微型计算机则属于后者。

单片微型计算机具有体积小、价格便宜、功能还相当强等优点，它带着计算机技术的触角伸向工农业生产和社会生活的各个角落。因此，从具有普遍应用意义上来讲，单片微型计算机无疑将显得十分重要。

为了促进单片微型计算机在各行各业中的广泛应用，需要有一本能适应我国读者情况的单片微型计算机实用书籍。“单片微型计算机及其应用”一书，较系统地讲述了美国Intel公司MCS-48和MCS-51系列单片微型计算机的硬件与软件的基本原理，尤其是根据作者近年来丰富的实践经验，以实际例子较详细地介绍了应用单片微型计算机的一些基本的具体的方法，列举了大量实用的线路、程序和应用实例。书中概念清晰、层次清楚、深入浅出、通俗易懂。我相信，这本很有实用价值的书定会有助于单片微型计算机的普遍推广应用。

科技工作者对于推广计算机的应用所做的任何努力都应得到充分的肯定和鼓励。所以，在这本书出版的时候，我愿讲这几句话，表示我对本书的一点看法以及对作者的勉励。

常　迥

1987年9月于清华大学

前　言

在近四十年的时间里，电子计算机的发展经历了从电子管、晶体管、中小规模集成电路到大规模集成电路这样四个阶段，尤其是随着半导体集成技术的飞跃发展，七十年代初诞生了一代新型的电子计算机——微型计算机，使得计算机应用日益广泛；而单片微型计算机的问世，则更进一步推动了这一发展趋势，使计算机应用渗透到各行各业，达到了前所未有的普及程度。一个由微电子技术为先导，计算机技术为标志，包括新材料、宇航、生物工程、海洋工程等多种学科在内的新技术革命正在兴起。

按照传统的概念，构成一台完整微型计算机的几大部件，根据其逻辑功能分别集成在几块芯片上，然后组装起来，如果我们把这称作为“分立式”微型计算机的话，那么，单片微型计算机则在一块集成芯片上就集成了CPU、RAM、ROM、I/O口、振荡器和时钟等各种电路，它已经包括了一台完整微型计算机的几乎全部功能，因而，我们可以把它称作为“集成式”微型计算机。

由于单片微型计算机具有功能强、体积小、价格低廉等许多独特的优点，因此，在智能仪器仪表、工业自动控制、计算机智能终端、家用电器、儿童玩具等许多方面，都已得到了很好的应用，而且应用的领域正越来越广泛。

随着计算机应用的推广普及，单片微型计算机已开始在国内引起人们的兴趣和重视。为了适应计算机应用中这一新趋势的需要，编者在多次“单片微型计算机应用短训班”讲稿的基础上，经过整理编写了本书。在编写过程中，力求做到内容深入浅出、简短明了、注重实用。作为一本微型计算机入门书，本书适合于广大具有高中以上文化程度的青年电工和一般科技人员阅读，即只要有一定基础电工和数字电路知识的同志，虽然对微型计算机还不甚了解，都可读懂本书。作为一本单片微型计算机应用方面较详细的参考书、工具书，本书同样适合于对微型计算机基础知识已经有所了解而希望把单片微型计算机引入、应用到他们各自工程领域中去的同志。

本书第一章较详细地叙述了微型计算机的基础知识，为不了解微型计算机的初学者提供了入门的捷径，也作为学习后续各章的基础；第二、三、四章分别介绍了MCS-48和MCS-51系列单片微型计算机的硬件结构特点、指令系统、扩展方法以及常用的一些外围芯片；最后一章着重讲述开发应用中的问题，包括推荐二种应用单片微型计算机所必须的典型的简易在线开发编程工具，并以一些已经开发成功的典型单片微型计算机应用项目，来阐明如何将单片微型计算机应用到实际的工程领域中去。

本书编写过程中，得到中国科学院学部委员、清华大学常迥教授的热情关怀和热心指导，提出了大量极为有益的宝贵意见，并为本书写了“序”；另外，王能权同志为书稿的整理和抄写付出了大量辛勤的劳动，在此编者对他们表示衷心的感谢。

由于编者水平有限，书中难免存在不少缺点和不足之处，恳请广大读者批评指正。

编者　1987年8月

目　录

第一章　微型计算机基础

1.1 预 备 知 识

1.1.1 微型计算机中的数和代码

一、为什么要用二进制

微型计算机的最基本功能是进行数的计算和处理加工，而数在计算机中是以电子元件的物理状态来表示的。电子元件通常只有两种稳定的状态，例如，导通与阻塞、饱和与截止、开与关、高电平与低电平、脉冲有与无等。而要找出一种具有十个不同稳定状态的电子元件则是相当困难的。在二进制中，只有二个数 0 和 1，这正好和电子元件的两个不同稳定状态相对应。例如，以 1 代表高电平，则 0 代表低电平，这样，采用二进制后，就可以利用二值电路来进行计数，容易实现，容易运算，方便可靠。所以，在计算机中，数据和其它字母、符号等都是以二进制的形式来表示并进行运算处理的，或者说，计算机只认识二进制的数。

二、进位计数制

数制是人们利用符号来计数的科学方法。人们最经常使用的是十进制，逢十进一。人们所以喜欢采用十进制，其主要原因也许是由于人们有十个指头，比较方便直观。在日常生活中，我们还遇到其它一些进位计数制，例如，六十进制（一分钟等于60秒），十六进制（一市斤等于16老两），十二进制（一打等于12个）等。但在计算机的设计与使用中，最常使用的则为二进制、八进制、十进制和十六进制。

1. 数制的基和权

所谓基数 J，就是在一个计数系统里，用来表示数时可以选用的不同数字的个数。因此，每种计数制都有一个固定的基数 J，它的每一位都可能取 J 个不同的数字，每一位都是逢 J 进一的。

所谓权 J^i，就是在一个计数系统里，每个数位 i 所具有的固定值 J^i。因此，小数点左面各位的权依次是基数 J 的正次幂 J^0、J^1、J^2…；而小数点右面各位的权则依次是基数 J 的负次幂 J^{-1}、J^{-2}、J^{-3}…。由于在每种计数制中，都是逢 J 进一的，所以，小数点左移一位等于减小 J 倍；小数点右移一位则等于增大 J 倍。

2. 二进制

二进制的基数 J 为“2”，即它所使用的数码为 0、1 共二个，二进制各位的权是以基数 2 为底的幂。例如

$$(1011.011)_2 = 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} + 1\times 2^{-3}$$

一般地说，任意一个二进制数B，都可以表示为

$$B = B_{n-1}\times 2^{n-1} + \cdots + B_1\times 2^1 + B_0\times 2^0 + B_{-1}\times 2^{-1} + \cdots + B_{-m}\times 2^{-m}$$

$$= \sum_{i=n-1}^{-m} B_i \times 2^i$$

3．八进制

八进制的基数J为“8”，即它所使用的数码为0、1、2、3、4、5、6、7共八个，八进制各位的权是以基数8为底的幂。例如

$$(73.45)_8 = 7\times 8^1 + 3\times 8^0 + 4\times 8^{-1} + 5\times 8^{-2}$$

一般地说，任意一个八进制数Q，都可以表示为

$$Q = Q_{n-1}\times 8^{n-1} + \cdots + Q_1\times 8^1 + Q_0\times 8^0 + Q_{-1}\times 8^{-1} + \cdots + Q_{-m}\times 8^{-m}$$

$$= \sum_{i=n-1}^{-m} Q_i \times 8^i$$

4．十进制

十进制的基数J为“10”，即它所使用的数码为0、1、2、3、4、5、6、7、8、9共十个，十进制各位的权是以基数10为底的幂。例如

$$(8691.75)_{10} = 8\times 10^3 + 6\times 10^2 + 9\times 10^1 + 1\times 10^0 + 7\times 10^{-1} + 5\times 10^{-2}$$

一般地说，任意一个十进制数D，都可以表示为

$$D = D_{n-1}\times 10^{n-1} + \cdots + D_1\times 10^1 + D_0\times 10^0 + D_{-1}\times 10^{-1} + \cdots + D_{-m}\times 10^{-m}$$

$$= \sum_{i=n-1}^{-m} D_i \times 10^i$$

5．十六进制

十六进制的基数J为“16”，即它所使用的数码为0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F共十六个，十六进制各位的权是以基数16为底的幂。例如

$$(A87.B79)_{16} = A\times 16^2 + 8\times 16^1 + 7\times 16^0 + B\times 16^{-1} + 7\times 16^{-2} + 9\times 16^{-3}$$

一般地说，任意一个十六进制数H，都可以表示为

$$H = H_{n-1}\times 16^{n-1} + \cdots + H_1\times 16^1 + H_0\times 16^0 + H_{-1}\times 16^{-1} + \cdots + H_{-m}\times 16^{-m}$$

$$= \sum_{i=n-1}^{-m} H_i \times 16^i$$

三、不同进位数之间的转换方法

1．二进制与八进制之间的相互转换

由于$2^3=8$，所以3位二进制数相当于1位八进制数，它们之间是完全对应的。因此，只要把1位八进制数字化成3位二进制数字，或者反之，把3位二进制数字化成1位八进制数字，即可实现八进制到二进制，或者二进制到八进制的转换。例如

八进制到二进制

```
  1  5  .  6
  ↓  ↓  ↓  ↓
001 101 . 110
```

即 $(15.6)_8 = (1101.11)_2$

二进制到八进制

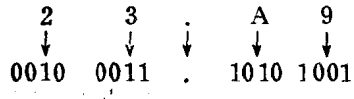
```
100 010 . 100 100
 ↓   ↓   ↓   ↓   ↓
 4   2     4   4
```

即 $(100010.1001)_2 = (42.44)_8$

2．二进制与十六进制之间的相互转换

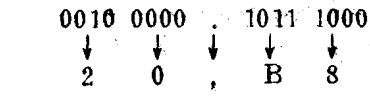
由于$2^4=16$，所以4位二进制数相当于1位十六进制数，它们之间是完全对应的。因此，只要把1位十六进制数字化成4位二进制数字，或者反之，把4位二进制数字化成1位十六进制数字，即可实现十六进制到二进制，或者二进制到十六进制的转换。例如

十六进制到二进制



即 $(23.A9)_{16}=(100011.10101001)_2$

二进制到十六进制



即 $(100000.10111)_2=(20.B8)_{16}$

3．二进制、八进制、十六进制转换成十进制

根据它们各自的定义表示式，按权展开相加即可将二进制数、八进制数、十六进制数转换成十进制数。例如

二进制到十进制

$$(1011.1)_2=1\times2^3+0\times2^2+1\times2^1+1\times2^0+1\times2^{-1}=(11.5)_{10}$$

八进制到十进制

$$(30.7)_8=3\times8^1+0\times8^0+7\times8^{-1}=(24.875)_{10}$$

十六进制到十进制

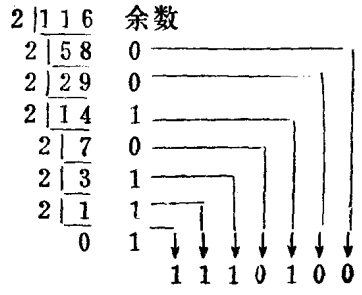
$$(2C.C)_{16}=2\times16^1+C\times16^0+C\times16^{-1}=(44.875)_{10}$$

4．十进制转换成二进制、八进制、十六进制

（1）整数部分

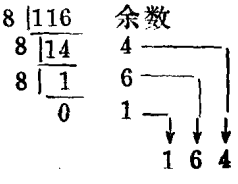
分别用基数2、8、16不断地去除待转换的十进制整数，直到商为0为止，每次除所得的余数相应为二进制、八进制、十六进制数码，最初得到的为最低有效数字，最后得到的为最高有效数字。例如：

十进制到进二制：



即 $(116)_{10}=(1110100)_2$

十进制到八进制：



即 $(116)_{10}=(164)_8$

十进制到十六进制：

```
16 | 116    余数
16 |  7     4
   |  0     7
                7 4
```

即　　$(116)_{10}=(74)_{16}$

（2）小数部分

分别用基数2、8、16不断地去乘待转换的十进制小数，直到积的小数部分为0（或直到所要求的位数）为止，每次乘所得的整数即相应为二进制、八进制、十六进制数码，最初得到的为最高有效数字，最后得到的为最低有效数字。例如：

十进制到二进制：

```
0.84375
×      2
1.68750
0.6875
×      2
1.3750
0.375
×      2
0.750
0.75
×      2
1.50
0.5
×      2
1.0
1  1  0  1  1
```

即　　$(0.84375)_{10}=(0.11011)_2$

十进制到八进制：

```
0.84375
×      8
6.75000
0.75
×      8
6.00
6  6
```

即　　$(0.84375)_{10}=(0.66)_8$

十进制到十六进制：

```
0.84375
×     16
13.50000
0.5
×     16
8.0
D  8
```

即　　$(0.84375)_{10}=(0.D8)_{16}$

因此，对于一个同时具有整数和小数部分的十进制数，在转换为二进制、八进制、十六进制数时，应将它的整数部分和小数部分分别转换，然后用小数点将这两部分连起来即可。

对于不同数制的数，为在书写上能够加以区别，常在数后用字母符号来表示不同的数制。例如

B——二进制
Q——八进制
D——十进制
H——十六进制

四、二进制代码

1．BCD码

二进制数在计算机中容易实现，其运算规律也十分简便，因此，在计算机中一般都采用二进制数字系统。但对于人们的习惯来讲，二进制数毕竟不怎么直观方便。因而，对于计算机的输入和输出，通常还采用一种二进制编码的十进制数——BCD码，它是十进制数，遵守逢十进一的规则，但它的十个不同的数字符号不是通常的0、1、2…9，而是采用4位二进制编码来表示，即分别用0000、0001、0010…1001来表示。这样，一方面适应了数字电路表示0和1比较方便的特点，另方面又符合于人们逢十进一的十进制数的习惯。但是，4位二进制是逢十六进一的，这与我们要求逢十进一不相符合，因此需要加6修正，即当大于9时，应自动地加6，以便进行调整。例如

BCD码　$(0110\ 1001\ 0001\ 1001.0011\ 0011)_{BCD}$

其表示的十进制数为　6919.33

算式

$$\begin{array}{r} 87 \\ +79 \\ \hline 166 \end{array}$$

其BCD码运算过程如下

$$\begin{array}{r} 1000\ \ 0111 \\ +\ \ 0111\ \ 1001 \\ \hline \text{半进位}\rightarrow 10000 \\ +\ \ 110 \quad \text{(加6修正)} \\ \hline 0110 \end{array}$$

$$\begin{array}{r} \text{进位}\rightarrow\ 10000 \\ +\ 110 \quad \text{(加6修正)} \\ \hline 0110 \end{array}$$

其结果为 $(0001\ 0110\ 0110)_{BCD}$

即　　$87+79=166$

2．ASCII码

在微型计算机中，机器只处理二进制数，因此，字母和各种符号也必须按照某种特定的规则用二进制代码来表示。目前，世界上最普遍采用的是ASCII码（美国标准信息交换码）。它用7位二进制代码来表示，故可表示128个不同的字符，其中包括：

（1）26个大写英文字母；

（2）26个小写英文字母；

（3）10个十进制数字；

（4）7个标点符号；

（5）9个运算符号；

（6）50个其它符号（例如打印格式符号、控制符号等）。

如要确定一个数字、字母或符号的ASCII码，可以先在ASCII码表中找到这个字符，然后将字符所在的列与行所对应的3位、4位二进制数连起来（列对应的3位在前，行对应的4位在后），所得到的7位二进制代码，即为该字符所对应的ASCII码。例如：

大写英文字母C的ASCII码为100 0011

小写英文字母w的ASCII码为111 0111

在计算机中传送ASCII码时，通常采用8位二进制数码，因此，最高有效位用作奇偶校验位，以便用于检查代码在传送过程中是否发生差错。

五、带符号数的表示

1．机器数与真值

计算机在运行数的运算中，不可避免地会遇到正数和负数，那么，在计算机中，正负符号是怎样表示的呢？通常我们将一个二进制数的最高位用作为符号位，来表示这个数的正负。规定符号位用0表示正，用1表示负。这样，一个二进制数，连同符号位在内作为一个数，称作为机器数，而不包括符号位的数值，称作为该机器数的真值。例如

机器数　$(01010110)_2 = +86$

$(11010110)_2 = -86$

它们的真值均为

$$(1010110)_2 = 86$$

2．原码

当正数的符号位用0表示，负数的符号位用1表示时，这种表示法称为原码表示法。例如

$$(+55)_{原} = 00110111$$

$$(-55)_{原} = 10110111$$

注意：

（1）8位二进制原码能表示数的范围为－127～＋127。

（2）在原码中，＋0与－0的表示法不同。

3．反码

在反码表示法中，正数的反码与正数的原码相同，负数的反码由它的正数的原码按位取反形成。例如

$$(+103)_{反} = (+103)_{原} = 01100111$$

$$(-103)_{反} = \overline{(+103)_{原}} = \overline{01100111} = 10011000$$

注意：

（1）8位二进制反码能表示数的范围为－127～＋127。

（2）在反码中，＋0与－0的表示法不同。

（3）当符号位为0时，其余位为数的真值；当符号位为1时，其余位按位取反后才是

数的真值。

4. 补码

在补码表示法中，正数的补码与正数的原码相同，负数的补码由它的反码加1形成。例如：

$$(+30)_{补}=(+30)_{原}=00011110$$

$$(-30)_{补}=(-30)_{反}+1=\overline{(+30)_{原}}+1=11100001+1=11100010$$

注意：

（1）8位二进制补码能表示数的范围为－128～＋127。

（2）在补码中，＋0与－0的表示法相同。

（3）当符号位为0时，其余位为数的真值；当符号位为1时，其余位按位取反再加1后才是数的真值。

六、二进制数的算术运算

1. 二进制运算规则

（1）加法 0＋0＝0
1＋0＝1
1＋1＝0 （进位1）
1＋1＋1＝1（进位1）

（2）减法 0－0＝0
1－0＝1
0－1＝1（借位1）
1－1＝0

（3）乘法 0×0＝0
1×0＝0
1×1＝1

2. 不带符号数的运算

（1）加法

按照加法运算规则，从最低位开始逐位相加。例如

```
  10111010          10000111
+ 00101000        +  01111001
----------        -----------
  11100010  进位→  100000000
```

所以，两个8位二进制数相加，其“和”可能仍为8位，也可能超过8位，这时便产生进位。

（2）减法

按照减法运算规则，从最低位开始逐位相减。例如

```
  10110100          01101001
-  10100010       -  01110011
----------        -----------
  00010010  借位→  111110110
```

所以，两个8位二进制数相减，当被减数小于减数时，产生借位，其“差”为负数，但在这里，计算机只处理不带符号的数，故无法表示，运算结果出错。

（3）乘法

按照乘法运算规则和移位相加的办法实现。例如：

被乘数左移的方法

```
    1010
  × 1011
  ------
    1010
  + 1010
  ------
   11110
+  0000
  ------
  011110
+ 1010
  ------
 1101110
```

部分积右移的方法

```
      1010
×     1011
------------
      1010   (部分积)
       1010  (右移一位)
+     1010
------------
      11110  (部分积)
       11110 (右移一位)
+     0000
------------
      011110 (部分积)
        011110(右移一位)
+     1010
------------
      1101110
```

（4）除法

从被除数的最高位开始，定出超过除数的位数，当找到这位时商为1，然后把选定的被除数值减去除数得余数，将被除数的下一位移到余数上，当小于除数时商为0，再移下一位，当大于除数时商为1，减去除数又得余数，再移下一位，直至所有的位均移下为止。例如

```
        11000 (商)
110) 10010011
   - 110
   -----
     110
   - 110
   -----
      0011 (余数)
```

3．带符号数的运算

（1）加法

带符号数进行加法运算时，加数与被加数均用补码形式表示，其结果仍为补码。只要结果不超过规定的数表示的范围，也就是只要不发生出溢出，则结果总是正确的。而当发生溢出时，使符号位遭到破坏，则结果出错。例如：

正数加正数

```
  01000101
+ 00110011
----------
  01111000
```

因为69＋51＝120＜127，所以未发生溢出，符号位未受破坏，结果正确。

```
  01110011
+ 00100001
----------
  10010100
```

因为115＋32＝147＞127，所以发生溢出，符号位受破坏，结果不正确。

负数加负数

```
  11010100
+ 11010110
----------
进位→ 110101010
```

因为(－44)＋(－42)＝－86＞－128，所以未发生溢出，符号位没有遭到破坏，结果正确。

```
  10100011
+ 10110000
----------
进位→ 101010011
```

因为(－93)＋(－80)＝－173＜－128，所以发生溢出，符号位受破坏，结果不正确。

正数加负数