微型计算机及其接口

〔英〕R.C.赫尔兰德　著
王启智　单和平　译
刘颖孙　审校

人民邮电出版社

微型计算机及其接口

〔英〕R.C.赫尔兰德著
王启智　单和平译
刘　颖　孙　校

人民邮电出版社

内 容 提 要

本书在介绍微型计算机结构和数字、模拟接口技术以及专用外设接口的基础上，重点讲解Intel 8085、Zilog Z80、Texas Instruments 9980A微处理器和个人计算机的接口技术，并列举了一些基本的、有实用价值的程序，内容丰富，可读性强，是一本实用性的科普读物。

本书可供大专院校学生和从事计算机工作的一般科技人员参考。

微型计算机及其接口

wei xing ji suan ji ji qi jiekou

〔英〕R.C.赫尔兰德 著

王启智 单和平 译 刘颖孙审校

责任编辑 高丕武

人民邮电出版社出版

北京东长安街27号

北京兴华印刷厂印刷

新华书店北京发行所发行

各地新华书店经售

开本：787×1092 1/32 1988年11月 第一版

印张：6.20/32 页数：106 1988年11月北京第1次印刷

字数：150千字 印数：1—5 000册

ISBN 7-115—03707—8/TP·017

定价：2.65元

前　言

一系列的书籍向读者介绍了微处理器技术的入门知识。它们对微电子电路的基本原理、微处理器的操作和编程已给予了充分的描述。然而，它们对微型计算机如何与外部装置和外设进行连接，却未能作出精确的分析。本书详细地研究了接口技术和接口电路，包括主要接口部件，并探讨了这些部件是如何用电路形式和编制的程序来完成接口任务的；具体地介绍了微型计算机与打印机、直观显示设备、电视监视器、显示器、键盘、电控制设备、其它计算机、仪器仪表等的连接方法，还包括一些专用设备的明细表。详细讨论了来自 Intel 、Zilog 、Texas Instruments、MOS Technology、BBC、Apple等厂家的接口部件和系统。

微型计算机的原理作为附加内容也在本书内。因此，这本书为读者了解微型计算机技术的基本原理，特别是接口技术，提供了一个完整的学习指南。例如，第1、2和7章叙述的是一般技术（逻辑、电路结构框图和编程），而其余各章的大部分内容向读者介绍的是接口技术的原理和编程。

在本书的最后，对最常用的术语作了解释。

对在本书写作当中给予大力支持和协助的同行和同事表示感谢。

译 者 的 话

在微型计算机的应用中，接口技术是一个引人注目的课题和关键环节。英国R.C.赫尔兰德所著《微型计算机及其接口》一书，通俗地介绍了微型计算机与其外部设备的连接方法，系统地叙述了数字和模拟输入/输出接口电路，并从实用的角度对常见微型计算机的接口做了详尽的说明。该书内容丰富，层次分明，同一些从设计角度论述接口技术的书相比，具有浅显易懂、言简意赅的特点。为了进一步普及微型计算机接口技术方面的知识，我们翻译了这本书，供广大从事计算机及其应用的同志参考。

本书在翻译过程中，得到不少同志的支持和协助，译稿完成后，承北方交通大学刘颖孙副教授审核定稿，在此表示衷心的感谢。

由于我们的水平有限，加之时间仓促，译文中的错误之处在所难免，恳请广大读者批评指正。

译者

1987年6月于北京

目　　录

第一章　微型计算机基础

假定大多数读者对计算机工作的基本原理，包括二进制数、逻辑、电压电平和硅片器件的物理性能是熟悉的。然而，对于初次接触微型计算机及其接口电路的读者来说，在第一章中对这些问题作简短的介绍还是必要的。

1·1　二进制数和十六进制数

长期以来人类使用基数为10的数制。这种数制中的数称十进制（或十进位）数。一个简单的数285可以展开成下式：

$$\text{十进制数}285=(285)_{10}=2\times10^2+8\times10^1+5\times10^0$$
$$=200+80+5$$

计算机电路是很难处理十个不同的信号电平的，但要设计出能处理两个信号电平（例如有电压和无电压）的电路则很容易。因此在计算机中采用的是以2为基数的二进制计数法。在当今信息技术时代，微电子电路和微型计算机占有统治地位，为了完全掌握这项新技术，迫使人们不得不学习二进制计数法。例如：

$$\text{二进制数}1100=(1100)_2=1\times2^3+1\times2^2+0\times2^1+0\times2^0$$
$$=8+4+0+0$$

因此，二进制数1100＝十进制数12。

每个二进制数或“位”，可以是0或1。高一位相当于低一位的两倍。

二进制数转换成十进制数的方法如下：

$$(1001)_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 8 + 0 + 0 + 1$$
$$= (9)_{10}$$

十进制数转换成二进制数有如下两种常用方法：

方法A—连续被2除（即除2取余法—译者注）

例如转换$(14)_{10}$为二进制数：

```
    1    (1←余数)  (3)
2 | 3    (1←余数)  (2)
2 | 7    (0←余数)  (1)
2 | 14
```

所以$(14)_{10} = (1110)_2$

方法B—直接用2的幂来表示（即按权展开法—译者注）

例如转换$(14)_{10}$为二进制数：

$$8(2^3)\ 4(2^2)\ 2(2^1)\ 1(2^0)$$

$$(14)_{10} = 1 + 1 + 1 + 0 = (1110)_2$$

这种方法需要灵活的心算或用碎纸头写出十进制数对应的二进制位权值（1、2、4、8、16、32、64等）的组合。

例如：

```
128 64 32 16 8 4 2 1
```

$(163)_{10} = 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1 = (10100011)_2$

在下面所举的例子中，注意在两组4位数之间留一个空隙。通常4位为一组，它有助于防止表示大的二进制数时产生错误。另外，用二进制数表示出来往往很长，如能用一个数码表示每组四位二进制数就简明多了。例如：

```
1001  0111  1000  0100←——二进制数
 9     7     8     4 ←—— “十六进制”码
```

每四位一组数可以用一个0到15的十进制数来表示。但是为了避免在同个地方使用两个字符，而用字母A、B、C、D、E和F来表示9以上的数。这就引出了“十六进制”码。

十进制、二进制和十六进制数之间的关系如表1·1所示。

表 1·1　　数　　制

十进制	二进制	十六进制
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	2
3	0 0 1 1	3
4	0 1 0 0	4
5	0 1 0 1	5
6	0 1 1 0	6
7	0 1 1 1	7
8	1 0 0 0	8
9	1 0 0 1	9
10	1 0 1 0	A
11	1 0 1 1	B
12	1 1 0 0	C
13	1 1 0 1	D
14	1 1 1 0	E
15	1 1 1 1	F

请看下面的例子：

例1：

将下面二进制数用十六进制数表示。

0010　1100　0110　1010

2　C　6　A　　(=2C6A$_{16}$)

例2：

用二进制数表示十六进制数32D5。

3　2　D　5

0011　0010　1101　0101

所以hex.32D5＝（0011　0010　1101　0101）。

（hex＝十六进制的缩写符号）

例3：

将$(B\ 9)_{16}$变换为十进制数。

$$B(11_{10})\times 16^1+9\times 16^0$$

$$=176+9=185_{10}$$

例4：

将138_{10}变换为十六进制数（首先变成二进制，然后再按四位一组变成十六进制）

	128	64	32	16	8	4	2	1
$(138)_{10}=$	1	0	0	0	1	0	1	0

$=10001010_2$

$=8A_{16}$

微型计算机往往用八位一组或十六位一组的形式表示数据和处理的单元。

1·2　二进制算术

进行二进制算术运算通常使用与十进制相同的运算规则。我们讨论下面两个例子：

例1：

$$\begin{array}{r} 0011\ 0100_2 \\ +\ 0111\ 0110_2 \\ \hline 1010\ 1010 \end{array}$$

例2：　0101 0101$_2$ − 0011 1000$_2$

$$\begin{array}{r} 0101\ 0101 \\ -\ 0011\ 1000 \\ \hline 0001\ 1101 \end{array}$$

乘法和除法仅仅是这些规则的扩展。但要注意，所有性能比较低的微型计算机（八位机）如今还不能用硬件完成乘法和除法运算。这些算术功能必须用精巧的加法和减法程序来实现。然而，所有较高性能的计算机（16位机）都具有这些功能。

负数在微型计算机中采用“补码”形式。在这种表示方法里数的最左位（或最高有效位）为符号位，即0 = 正，1 = 负。如果此位为1，则其它位以补码形式表示该负数的大小，如下所示：

符号位⟶ 1 1 1 1　0 1 1 1

↓　所有位求反

0 0 0 0　1 0 0 0

↓　加1

负数的大小→ 0 0 0 0　1 0 0 1

因此，该数为 -9_{10}

用相同的转换规则（逐位取反加1），可求出任一个数的补码。例如

$+12_{10}$ = 0 0 0 0　1 1 0 0

逐位取反：　1 1 1 1　0 0 1 1

加1：　1 1 1 1　0 1 0 0

因此，-12_{10} = 1 1 1 1　0 1 0 0$_2$

有经验的计算机程序员常会碰到一个非常特殊的补码，这就是全1：

1 1 1 1　1 1 1 1$_2$

它等于 -1_{10}。

读者可以用上面的规则检验一下它们之间的关系。

1·3 逻　辑

许多信号是以“模拟”或连续的形式（例如室内温度的读数）从外部送入计算机的。自然，这样的信号必须转换成一种数字信号形式。实际上在对信号处理之前，已经在计算机内部把它转换成仅仅由0和1组成的二进制数字形式了。另外，“逻辑”功能常用二进制位（比特）来表示外部事件或读数和其它信号。这些逻辑功能可用“硬件”（电子电路）或“软件”（计算机程序）来实现。

主要逻辑功能是：

（a）与

两个一位信号可以一起“选通”而产生一个合成的输出“与”信号，如图1·1所示。“真值表”也表示在该图中，它给出了信号的所有不同组合的结果。

（a）电路符号

A	B	X = A.B
0	0	0
0	1	0
1	0	0
1	1	1

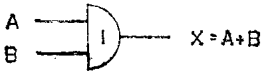
（b）真值表

图 1·1　与门

从与门的工作可以看出，只有当输入全是1的时候输出才为1。

(b) 或

图1·2表示的是一个或门的电路符号和真值表。只要它的输入中有一个或两个是1则输出就为1。



(a) 电路符号

A	B	X=A+B
0	0	0
0	1	1
1	0	1
1	1	1

(b) 真值表

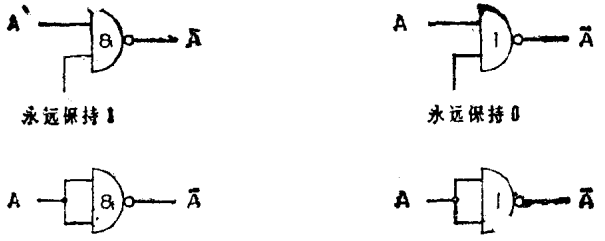
图 1·2 或门

(c) 反相器（或补码器）

图1·3说明了一个反相器的作用，也就是说一个门电路使一个一位信号反相或求补，即从0变为1或从1变为0。信号上面的横线表示该信号反相。实现反相功能可供选择的电路也示于图中；这有助于减少一些微电子系统中门电路的类型。注意这里使用的门的类型是与非门和或非门，即它们完成反相（如图中所表示的门输出端处的小圆圈）以及与和或的逻辑功能。实际上使用电路元件是很容易实现反相作用的。标志与非门和或非门功能的真值表如图1·4所示。

当外部信号输入到计算机的时候经常要用到这些类型的门，有时也作为必需的部分包含在全部微型计算机电路中。在这种情况下，信号0或1，一般地分别用0V和5V的电平来表示。更重要的是微型计算机的中央处理单元（称CPU或“微

（a）电路符号



（b）可选用的四种不同电路

A	Ā
0	1
1	0

（c）.真值表

图 1·3 反相器（补码器）

A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

（a）NAND门

A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

（b）NOR门

图 1·4 与非门和或非门的真值表

处理器”）能完成多位数或位模式的运算。例如，一个八位机能完成两个八位数的与运算。如下所示：

$$\begin{array}{rl} A = & 0\,1\,0\,1 \quad 1\,0\,0\,1 \\ \underline{B = } & \underline{0\,0\,0\,0 \quad 1\,1\,1\,1} \\ A \cdot B = & 0\,0\,0\,0 \quad 1\,0\,0\,1 \end{array}$$

“与”运算是在各个对应位上分别进行的。在这里，“与”运算的作用可看作屏蔽了高四位，即在A·B之后使其高四位全部为0。

由“或”运算的作用可以看出它产生出多个“1”，如下所示：

$$\begin{array}{r} A=0\,0\,1\,0 \quad 0\,1\,1\,1 \\ B=1\,1\,1\,1 \quad 0\,0\,0\,0 \\ \hline A+B=1\,1\,1\,1 \quad 0\,1\,1\,1 \end{array}$$

微处理器将这些多位数据存放在“寄存器”中以待处理。例如，一个十六位寄存器保存一个十六位数据值。微处理器只占用一小部分寄存器，而把处理后的数据送入存储器的存储单元。微计算机一般拥有几千个或更多的存储单元，用它们来保存这些数据，同时也保存一组将要执行的“程序指令”。因此一条指令是多位的位组合形式（或“字”）。通过对一个微计算机存储器的粗略检查，往往很难分辨它所存储的内容究竟是计算机依次取出并执行的指令，还是将要进行处理（例如使用算术运算和逻辑运算）的数据。

1·4 集成电路

直到70年代，计算机的门电路、寄存器和较强功能组件还是用分立元件晶体管、电阻和其他器件构成的。这样的电路体积大而且功耗高。现在由几万个晶体管组成的复杂电路已被封装在一个硅片上，称之为“集成电路”。

标准的组件封装是双列直插式(DIL)的，如图1·5所示。在这种器件的硅片上制成了上万个晶体管电路，并利用细金属导线连接到外部接点的引脚上。