普通高等教育“十一五”国家级规划教材

高等学校计算机硬件技术课程系列教材

微型计算机硬件技术基础

（第2版）

冯博琴　吴　宁　主编

吴　宁　陈文革　张　建　编

高等教育出版社
Higher Education Press

普通高等教育“十一五”国家级规划教材
高等学校计算机硬件技术课程系列教材

微型计算机硬件技术基础

Weixing Jisuanji Yingjian Jishu Jichu

（第2版）

冯博琴　吴　宁　主编
吴　宁　陈文革　张　建　编

高等教育出版社·北京
HIGHER EDUCATION PRESS　BEIJING

内容提要

本书是普通高等教育“十一五”国家级规划教材，是《微型计算机硬件技术基础》一书的修订版。作者根据微型计算机技术的发展、课程体系的变化以及教学过程中的体会，对原书内容做了一定的修订，结构进行了一定的调整。修订后的第2版保持了第1版中关于计算机中的数制、微机系统组成及其工作原理、指令系统等部分的叙述风格。对包括微处理器、总线技术、存储器系统、接口电路等章节的内容进行了调整和修改，使其更能反映当前微机领域的新进展和新技术。特别地，增加了汇编语言程序设计方面的内容，以方便读者体会微型计算机的工作原理，同时也增加了教材的实用性。删除了常用外部设备、设备驱动程序及计算机中的多媒体技术等已在新设立的“大学计算机基础”课程中介绍的内容。

经修订、调整后的本书，在强调基本概念的基础上，更强调与实践应用相结合，引入了大量的实例来阐明各种应用问题，实用性进一步增强。

本书可作为普通高等院校非计算机专业本科学生的“微型计算机硬件技术”或“微机原理与接口技术”类课程的教材，也可作为成人高等教育的培训教材及广大科技工作者的自学参考书。

图书在版编目(CIP)数据

微型计算机硬件技术基础/冯博琴，吴宁主编. —2版.
—北京：高等教育出版社，2010.2
ISBN 978-7-04-028829-2

Ⅰ.①微… Ⅱ.①冯… ②吴… Ⅲ.①微型计算机-硬件-高等学校-教材 Ⅳ.①TP360.3

中国版本图书馆CIP数据核字(2010)第020610号

策划编辑	孙惠丽	**责任编辑**	郭福生	**封面设计**	于文燕
责任绘图	杜晓丹	**版式设计**	王艳红	**责任校对**	杨凤玲
责任印制	陈伟光				

出版发行	高等教育出版社	**购书热线**	010-58581118
社　　址	北京市西城区德外大街4号	**咨询电话**	400-810-0598
邮政编码	100120	**网　　址**	http://www.hep.edu.cn
总　　机	010-58581000		http://www.hep.com.cn
		网上订购	http://www.landraco.com
经　　销	蓝色畅想图书发行有限公司		http://www.landraco.com.cn
印　　刷	北京七色印务有限公司	**畅想教育**	http://www.widedu.com
		版　　次	2003年5月第1版
开　　本	787×1092　1/16		2010年3月第2版
印　　张	23.5	**印　　次**	2010年3月第1次印刷
字　　数	570000	**定　　价**	36.40元

物料号　28829-00

第2版前言

这是一本面向非计算机专业本科“微型计算机硬件技术”课程的通用教材，其第1版为国家“十五”规划教材，自2003年出版以来，经6年多的使用，获得较好评价。作为普通高等教育国家“十一五”国家级规划教材，作者在汲取近6年的教学改革成果并适应新技术发展的基础上，对原教材进行了修订。本次修订主要涉及以下几个方面：

（1）考虑到“大学计算机基础”课程内容的改革及非计算机专业学生对计算机基础知识学习的要求，本次修订删去了第1版书中的第9章“常用外部设备及设备驱动程序”和第6章中的“新一代内存条的硬件技术发展”及“外存储器简介”部分，将第1章中的内容简化后合并到第3章中，形成第2版的第2章。

（2）为了帮助读者更进一步理解微型计算机的工作原理，同时也增加教材在使用中的实用性，第2版增加了“汇编语言程序设计”一章。

（3）将第1版第5章“指令系统”中的“汇编语言源程序结构”内容放到了第2版第5章“汇编语言程序设计”中。

（4）考虑到Cache技术在今天的微机系统中的重要性，第2版适当增加了有关Cache存储器的地址映射和地址变换的描述。

（5）在“存储系统”一章中，加强了有关内存储器设计的内容。同时，在接口技术中，也增添了更多的与实际相结合的应用实例，以便读者能够具备微型计算机硬件系统的初步设计能力。

本书不仅可作为课堂用教材，也能对学生以后的工作有一定指导作用。

本书由冯博琴教授策划并任主编，第1、5章由张建编写，第2、3章由陈文革编写，第4、6、7、8章由吴宁编写，并最后统稿。

第2版虽然是在第1版的基础上修订而成，但由于计算机新技术层出不穷，编者水平有限，且时间也较仓促，书中错误和不当之处在所难免，敬请各位读者和专家批评指正，以便及时修正。

编　者

2009年7月于西安交通大学

目 录

第1章 数 制

本章提要

应用计算机编程解决一个实际问题，首先要熟知计算机能够处理哪些数据类型。本章介绍计算机中常用记数制及编码的表示方法，它们相互间的转换，二进制数的算术运算和逻辑运算方法，以及定点数和浮点数的表示等计算机基础知识。其中，补码的概念及运算是本章中读者需要深入理解的部分。

1.1 计算机中的数制

在生活中，人们常用十进制数来进行记数和计算。众所周知，计算机中的数是用二进制表示的，因为计算机只能识别“0”和“1”两种代码。但二进制数位数冗长难于记忆，表示较大的数字时尤为如此。有时为了阅读和书写方便，在计算机中也采用十六进制数和十进制数。所以，在学习计算机原理之前，首先需要了解和掌握这3种常用记数制及其相互间的转换。

1.1.1 常用记数制

1. 十进制数

十进制数的表示方法可概括为“位权表达式”。十进制数中有0～9十个数码，任意十进制数 D，都可用这十个数码的组合来表示。写成位权展开式为：

$$(D)_{10}=D_{n-1}\times10^{n-1}+D_{n-2}\times10^{n-2}+\cdots+D_1\times10^1+D_0\times10^0+D_{-1}\times10^{-1}+\cdots+D_{-m}\times10^{-m}$$

$$=\sum_{i=-m}^{n-1}D_i\times10^i \tag{1.1}$$

其中，D_i是 D 的第 i 位的数码，可以是0～9十个数码中的任何一个数码；n 和 m 为正整数，n 表示小数点左边的位数，m 表示小数点右边的位数；10为基数，基数是指十进制允许使用的基本数码的个数；10^i称为十进制的位权，简称“权”。

【例1-1】十进制数8756.23可表示为

$$(8756.23)_{10}=8\times10^3+7\times10^2+5\times10^1+6\times10^0+2\times10^{-1}+3\times10^{-2}$$

2. 二进制数

二进制数的每一位只有0和1两个数码，且记数法则为逢二进一。一个二进制数 B 按权展开表示为

$$\begin{aligned}(B)_2&=B_{n-1}\times2^{n-1}+B_{n-2}\times2^{n-2}+\cdots+B_0\times2^0+B_{-1}\times2^{-1}+\cdots+B_{-m}\times2^{-m}\\&=\sum_{i=-m}^{n-1}B_i\times2^i\end{aligned}\tag{1.2}$$

其中，B_i 只能取1或0，2为基数，2^i 为二进制的权，m、n 的含义与十进制表达式相同。一个二进制数通常用下标2表示，以区别于其他进位记数制。

【例1-2】二进制数1110.01可表示为

$$(1110.01)_2=1\times2^3+1\times2^2+1\times2^1+0\times2^0+0\times2^{-1}+1\times2^{-2}$$

3. 十六进制数

十六进制数共需要16个数码，即0～0FH，其记数法则为逢十六进一。一个十六进制数 H 按权展开为

$$\begin{aligned}(H)_{16}&=H_{n-1}\times16^{n-1}+H_{n-2}\times16^{n-2}+\cdots+H_0\times16^0+H_{-1}\times16^{-1}+\cdots+H_{-m}\times16^{-m}\\&=\sum_{i=-m}^{n-1}H_i\times16^i\end{aligned}\tag{1.3}$$

其中，H_i 的取值范围为0～9及A～F，16为基数，16^i 为十六进制数的权，m、n 的含义与上述相同。十六进制数通常用下标16表示。

【例1-3】十六进制数34A.5可表示为

$$(34A.5)_{16}=3\times16^2+4\times16^1+A\times16^0+5\times16^{-1}$$

由于 $2^4=16$，也就是说一位十六进制数恰好可用4位二进制数来表示，所以，在计算机应用中，常采用二-十六缩写方式，虽然机器只能识别二进制数，但在数字的表达上更广泛地采用十六进制数。

计算机中常用的二进制数、十六进制数和十进制数之间的关系如表1-1所示。

表1-1 数制对照表

十进制数	二进制数	十六进制数	十进制数	二进制数	十六进制数
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

4. 其他非十进制记数制

除以上介绍的二进制、十进制和十六进制这3种常用的进位记数制外，计算机中还可能用到八进制数，这里不再详细介绍了。下面给出任一进位制数的权展开式的一般形式。

一般地，对任意一个 K 进制数 S 都可表示为

$$(S)_K = S_{n-1} \times K^{n-1} + S_{n-2} \times K^{n-2} + \cdots + S_0 \times K^0 + S_{-1} \times K^{-1} + \cdots + S_{-m} \times K^{-m}$$
$$= \sum_{i=-m}^{n-1} S_i \times K^i \tag{1.4}$$

其中，S_i 是 S 的第 i 位的数码，可以是所选定的 K 个符号中的任何一个；n 和 m 的含义同上；K 为基数；K^i 称为 K 进制数的权。

例1-3中以基数作为下标来表示数的进制，还可以在数的后面加上字母B、H、D来分别表示二进制数、十六进制数和十进制数，如11000101B、2C0FH、1300D等。十进制数后面的D可以省略。

1.1.2 各种数制之间的转换

计算机内部采用的是二进制数，而编写程序又多采用十六进制数，不符合人们的日常习惯，那么计算机在运算和处理时必然涉及不同数制之间进行转换的问题。

1. 非十进制数转换为十进制数

非十进制数转换为十进制数，只要将它们按上述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1011.011转换为十进制数。

解：根据二进制数的权展开式，有

$$(1011.011)_2 = 1\times2^3+0\times2^2+1\times2^1+1\times2^0+0\times2^{-1}+1\times2^{-2}+1\times2^{-3}$$
$$= (11.375)_{10}$$

【例1-5】将十六进制数85.CH转换为十进制数。

解：根据十六进制数的权展开式，有

$$(85.C)_{16} = 8\times16^1+5\times16^0+C\times16^{-1}$$
$$= 8\times16^1+5\times16^0+12\times16^{-1}$$
$$= (133.75)_{10}$$

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“除基取余”法，二进制数基数为2，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“乘基取整”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零为止。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向低位逐步扩展。

【例1-6】将十进制数124.25转换为等值的二进制数。

解：

整数部分	小数部分
124/2=62……余数=0(最低位)	0.25×2=0.5……整数=0(最高位)
62/2=31…… 余数=0	0.5×2=1.0…… 整数=1
31/2=15…… 余数=1	
15/2=7 …… 余数=1	
7/2=3 …… 余数=1	
3/2=1 …… 余数=1	
1/2=0 …… 余数=1	

从而得到转换结果:$(124.25)_{10}=(1111100.01)_2$

2) 十进制数转换为十六进制数

同样,十进制数转换为十六进制数时,整数转换采用"除16取余"的方法,而小数转换采用"乘16取整"的方法。

【例1-7】将十进制数455.6875转换为等值的十六进制数。

解:

整数部分	小数部分
455/16=28……余数=7	0.6875×16=11.0000……整数=$(11)_{10}=(B)_{16}$
28/16=1 ……余数=C	
1/16=0 ……余数=1	

所以,$(455.6875)_{10}=(1C7.B)_{16}$。

也可将十进制数先转换为二进制数,再转换为十六进制数。下边将会看到,后者的转换非常简便。

3. 二进制数与十六进制数之间的转换

由于$2^4=16$,所以二进制数与十六进制数之间的相互转换非常容易。将二进制数转换为十六进制数的方法是:从小数点开始分别向左和向右把整数和小数部分每四位分为一组。若整数最高位的一组不足4位,在其左边补零;若小数最低位的一组不足4位,则在其右边补零。然后将每组二进制数用对应的十六进制数代替,即得到转换结果。

【例1-8】将二进制数110100110.101101B转换为十六进制数。

解:

二进制数	0001	1010	0110.	1011	0100
	↓	↓	↓	↓	↓
十六进制数	1	A	6.	B	4

所以,$(110100110.101101)_2=(1A6.B4)_{16}$。

十六进制数转换为二进制数的方法与上述过程相反,即用四位二进制代码取代对应的一位十六进制数。

【例1-9】将十六进制数7AD4.6DH转换为二进制数。

解：

十六进制数	7	A	D	4.	6	D
	↓	↓	↓	↓	↓	↓
二进制数	0111	1010	1101	0100.	0110	1101

所以，$(7AD4.6D)_{16}=(0111101011010100.01101101)_2$

1.2 无符号二进制数的算术运算和逻辑运算

1.2.1 二进制的算术运算

1. 加法运算

二进制数只有0和1两个数码，其运算规则比十进制数简单得多。二进制的加法运算“逢二进一”，遵循如下法则：

0+0=0　　0+1=1　　1+0=1　　1+1=0(有进位)

【例1-10】计算 10100110B+01011101B = (?)B

进位	1 11111000
被加数	10100110
加数	+ 01011101
	1 00000011

解：

可得 10100110B+01011101B=100000011B

2. 减法运算

二进制数的减法遵循如下法则：

0-0=0　　1-0=1　　1-1=0　　0-1=1（有借位）

【例1-11】计算 11000100B-00100101B=(?)B

解：

借　位	01111110
被减数	11000100
减数	- 00100101
	10011111

可得 11000100B-00100101B = 10011111B

3. 乘法运算

二进制数的乘法遵循如下法则：

0×0=0　　0×1=0　　1×0=0　　1×1=1

即仅当两个1相乘时结果为1，否则结果为0。所以，二进制数的乘法也是非常简单的。若乘数位为1，就将被乘数加于中间结果，若乘数位为0，则加0于中间结果，只是在相加时要将每次中间结果的最后一位与相应的乘数位对齐。

【例1-12】求两个二进制数1100B与1010B的乘积。

解法一：按照十进制的乘法过程有

```
   1100          被乘数
  ×1010          乘数
  ------
   0000          部分积
  1100
 0000
1100
------
1111000          乘积
```

可得 1100B×1010B = 1111000B。

解法二：采用移位加的方法，则有

```
乘数                                          被乘数    部分积
1 0 1 0                                       1100      0000
  |  |  |
  |  |  └→乘数位为0，不加被乘数，被乘数左移1位   11000     0000
  |  └→乘数位为1，加被乘数到部分积上，并将被     110000    11000
  |     乘数左移一位
  └→乘数位为0，不加被乘数，被乘数左移1位       1100000
                                                        11000
  └→乘数位为1，左移后的被乘数加到部分积上              +1100000
                                                        1111000
```

可得 1100B×1010B = 1111100B。

由方法二可看出，二进制的乘法运算可以转换为加法和移位的运算。事实上，计算机中乘法运算就是这样做的。每左移一位，相当于乘以2，而左移 n 位就相当于乘以 2^n。

4. 除法运算

除法是乘法的逆运算。所以二进制数的除法运算也可转换为减法和右移运算。每右移一位相当于除以2，右移 n 位就相当于除以 2^n。

1.2.2 无符号数的表示范围

1. 无符号二进制数的表示范围

一个 n 位的无符号二进制数 X，它可表示的数的范围为

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

对于一个8位的二进制数，即 $n=8$，其表示范围为 $0 \sim 2^8-1$，即 00H ~ FFH(0 ~ 255)。若运算结果超出数的可表示范围，则会产生溢出，结果将不正确。

【例1-13】计算 11011011B+01001101B=(?)B

解：

$$\begin{array}{r} 11011011 \\ +\ 01001101 \\ \hline 100101000 \end{array}$$

由上式可得，上面两个 8 位二进制数相加的结果为 9 位，超出了 8 位数的表示范围。若仅取 8 位字长（00101000B），结果显然不正确，这种情况就称为溢出。事实上，$(11011011)_2=(219)_{10}$，$(01001101)_2=(77)_{10}$，则 219+77=296，大于 8 位二进制数所能表示的最大值 255，所以最高位的进位（代表 256）给丢失了，这样最后的结果就是 296−256=40，即 00101000B。

2. 无符号二进制数的溢出判断

设无符号二进制数加法（或减法）中最高有效位 D_i 的进（借）位为 C_i，则两个无符号二进制数相加（或相减）时，若最高有效位 D_i 产生进位（或相减有借位），即 $C_i=1$，则产生溢出。比如在例 1-13 中，两个 8 位无符号二进制数相加，最高有效位（即 D_7 位）产生了进位 C_7，结果就出现溢出。

1.2.3 二进制数的逻辑运算

逻辑运算与算术运算不同，不是将一个二进制数的所有位作为一个数值整体来考虑，而是对二进制数按位独立进行操作，因此逻辑运算没有进位和借位。基本逻辑运算包括“与”、“或”、“非”及“异或”四种。

1. “与”运算

“与”运算的规则是按位相“与”。与运算符一般用符号“∧”表示。其规则如下：

$$1\wedge1=1 \qquad 1\wedge0=0 \qquad 0\wedge1=0 \qquad 0\wedge0=0$$

即参加“与”操作的两位中只要有一位为 0，则“与”的结果就为 0，仅当两位均为 1 时，其结果才为 1。相当于按位相乘（但不进位），又叫做“逻辑乘”。

【例 1-14】计算 11011010B∧10010110B=(?)B

解：

$$\begin{array}{r} 11011010 \\ \wedge\ 10010110 \\ \hline 10010010 \end{array}$$

即 11011010B∧10010110B = 10010010B。

2. “或”运算

“或”运算又叫做“逻辑加”，其规则为按位相“或”。“或”运算符一般用符号“∨”表示。其规则如下：

$$0\vee0=0 \qquad 0\vee1=1 \qquad 1\vee0=1 \qquad 1\vee1=1$$

即参加“或”操作的两位中仅当两位均为 0 时，其结果才为 0，只要有一位为 1，则“或”的结果就为 1（比较二进制数的“或”运算和加法运算的规则有什么不同）。

【例 1-15】计算 11011001B∨10010110B=(?)B

解：

$$\begin{array}{r} 11011001 \\ \vee\ 11111111 \\ \hline 11111111 \end{array}$$

即 11011001B ∨ 11111111B = 11111111B。

3. “非”运算

“非”运算的规则为按位取反，即 1 的“非”为 0，而 0 的“非”为 1。“非”属于单边运算，即只有一个运算对象，其运算符为一上横线（¯）。

$$\overline{1}=0 \qquad \overline{0}=1$$

【例 1-16】计算$\overline{11011001B}$=（?）B

解：对 11011001 按位求反，即可得

$$\overline{11011001B}=00100110B$$

4. “异或”运算

“异或”运算相当“按位相加”（不进位），进行“异或”操作的两个二进制位不相同时，结果就为 1；两位相同时，结果为 0。“异或”运算符用符号⊕表示。

$$0\oplus 0=0 \qquad 1\oplus 1=0 \qquad 0\oplus 1=1 \qquad 1\oplus 0=1$$

【例 1-17】计算 11010011B ⊕ 10100110B =（?）B

解：

$$\begin{array}{r} 11010011 \\ \oplus 10100110 \\ \hline 01110101 \end{array}$$

即 11010011B⊕ 10100110B = 01110101B。

比较有趣的是：二进制数的“异或”运算可以看做不进位的“按位加”，也可以看做不借位的“按位减”。

1.2.4 基本逻辑门及常用逻辑部件

本小节从应用的角度出发介绍几种常用的计算机基本逻辑部件。只介绍它们的逻辑功能和外部引线连接，而不涉及其具体的内部电路。

1. 与门(AND gate)

与门是对多个逻辑变量（取值范围只有“0”和“1”）进行“与”运算的门电路。对两个逻辑变量 A 和 B 的“与”操作，其结果 Y 可表示为

$$Y=A \wedge B$$

它们的关系也可从表 1-2 所示的真值表中清楚地了解到。即仅在输入 A 和 B 均为 1 时，输出 Y 才为 1，A 和 B 中只要一个为 0，则 Y 就等于 0。以电路变化来表示，若采用正逻辑，则仅当与门的输入 A 和 B 都是高电平时，输出 Y 才是高电平，否则 Y 就输出低电平。

在电路连接上，与门常用图 1-1 所示的逻辑符号表示。

表 1-2 与门的真值表

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

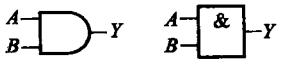

图 1-1 与门的逻辑符号

2. 或门(OR gate)

或门是对多个逻辑变量进行“或”运算的门电路。对两个逻辑变量 A 和 B 的“或”操作,可用下边的逻辑表达式表示:

$$Y=A\vee B$$

即两个输入变量 A 和 B 中任意一个为 1,输出 Y 就为 1;仅当 A 和 B 都为 0 时 Y 才为 0。从电路的角度来说,当或门的输入 A 和 B 只要有一个是高电平,输出 Y 就为高电平,否则 Y 就输出低电平。

或门的逻辑符号见图 1-2,其真值表见表 1-3。

表 1-3 或门的真值表

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

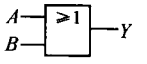

图 1-2 或门的逻辑符号

3. 非门(NOT gate)

非门又称为反相器,是对单一逻辑变量进行“非”运算的门电路,其输入变量 A 与输出变量 Y 之间的关系可用下式表示:

$$Y=\overline{A}$$

非运算也称求反运算,变量 A 上的上划线“¯”在数字电路中表示反相之意。非门的逻辑符号见图 1-3,其真值表见表 1-4。

表 1-4 非门的真值表

A	Y
0	1
1	0

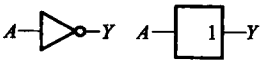

图 1-3 非门的逻辑符号

4. 与非门(NAND gate)

与门与非门结合形成“与非门”。若输入变量为 A 和 B,则先对 A 和 B 进行“与”运算,再对结果进行“非”运算,运算表达式为

$$Y=\overline{A\wedge B}$$

与非门的逻辑符号见图 1-4,图中逻辑符号图中的小圆圈表示“非”(本书将始终采用这种表示方法)。与非门的真值表见表 1-5。

表 1-5 与非门的真值表

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

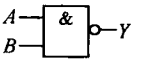

图 1-4 与非门的逻辑符号

5. 或非门(NOR gate)

和与非门类似,或非门是“或”门与“非”门的结合。即先对输入 A 和 B 进行“或”运算,再对其结果进行“非”运算,其运算表达式为

$$Y=\overline{A \vee B}$$

或非门的逻辑符号如图 1-5 所示,真值表见表 1-6。

表 1-6 或非门的真值表

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

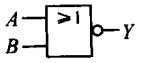

图 1-5 或非门的逻辑符号

6. 译码器

在计算机硬件系统中,常常需要通过一定的控制电路从一组地址信号中选出对某一芯片的片选信号,这个控制电路称为译码电路,组成该电路的逻辑部件就称为译码器。

这里介绍一种常用的 3-8 线译码器 74LS138。74LS138 的引脚如图 1-6 所示。图中 G_1、$\overline{G_{2A}}$、$\overline{G_{2B}}$ 为译码器的三个使能输入端,它们共同决定了译码器当前是否被允许工作:当 $G_1=1$,$\overline{G_{2A}}=\overline{G_{2B}}=0$ 时,译码器处于使能状态(enable),否则就被禁止(disable)。C、B、A 为译码器的三条输入线(输入的 3 位二进制代码分别代表了 8 种不同的状态),它们的不同的状态组合,决定了 8 个输出端 $Y_0 \sim Y_7$ 的状态。74LS138 的功能表见表1-7,表中电平为正逻辑,即高电平表示逻辑“1”,低电平表示逻辑“0”,“×”表示不定,“#”表示该信号低电平有效(与上横线标注“¯”含义相同)。

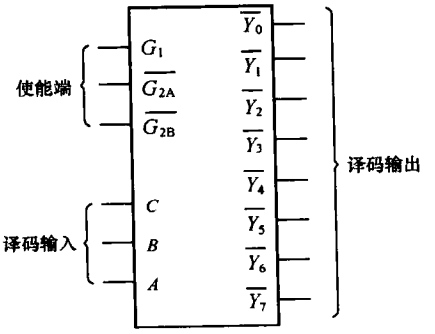

图 1-6 74LS138 的引脚功能图

表 1-7 74LS138 功能表

使能端		输入端			输出端							
G_1	#G_2	C	B	A	#Y_0	#Y_1	#Y_2	#Y_3	#Y_4	#Y_5	#Y_6	#Y_7
×	1	×	×	×	1	1	1	1	1	1	1	1
0	×	×	×	×	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1	1	1
1	0	1	0	0	1	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	1	1	0	1	1
1	0	1	1	0	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	1	0