第 1 章　微型计算机基础

1.1　计算机中的数和编码

1.1.1　计算机中的数制

计算机最早是作为一种计算工具出现的，所以它的最基本的功能是对数进行加工和处理。数在机器中是以器件的物理状态来表示的。一个具有两种不同的稳定状态且能相互转换的器件就可以用来表示 1 位（bit）二进制数。二进制数有运算简单，便于物理实现，节省设备等优点，所以目前在计算机中数几乎全是采用二进制表示。但是二进制数书写起来太长，且不便于阅读和记忆；而 4 位二进制数有 16 个不同的状态 0000～1111，即是 1 位十六进制数；所以微型计算机中的二进制数都采用十六进制数来缩写。十六进制数用 0～9 和 A～F 等 16 个数码表示 4 位二进制数 0000～1111，这 16 个二进制数 0000～1111 的大小就是十进制数 0～15。1 个 8 位的二进制数用 2 位十六进制数表示，1 个 16 位的二进制数用 4 位十六进制数表示等。这样书写方便，又便于阅读和记忆，且转换方便，因此常用十六进制数来缩写二进制数。然而人们最熟悉、最常用的是十进制数。为此，要熟练地掌握十进制数、二进制数和十六进制数间的相互转换。它们之间的关系如表 1-1 所列。

表 1-1　十进制数、二进制数及十六进制数对照表

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

为了区别十进制数、二进制数及十六进制数 3 种数制，可在数的右下角注明数制，或者在数的后面加一字母。如 B（binary）表示二进制数制；D（decimal）或不加字母表示十进制数制；H（hexadecimal）表示十六进制数制。

1．二进制数和十六进制数整数间的相互转换

根据表 1-1 所示的对应关系即可实现它们之间的转换。

二进制整数转换为十六进制数，其方法是从右（最低位）向左将二进制数分组:每 4 位为 1 组，最后一组若不足 4 位则在其左边添加 0，以凑成 4 位 1 组，每组用 1 位十六进制数表示。如：

1111111000111B →1 1111 1100 0111B → 0001 1111 1100 0111B=1FC7H

十六进制数转换为二进制数，只需用 4 位二进制数代替 1 位十六进制数即可。如：

3AB9H=0011 1010 1011 1001B

2．十六进制数和十进制数间的相互转换

十六进制数转换为十进制数十分简单，只需将十六进制数按权展开相加即可。如：

1F3DH=$16^3\times1+16^2\times15+16^1\times3+16^0\times13$=4096×1＋256×15＋16×3＋1×13=4096＋3840＋48＋13=7997

十进制整数转换为十六进制数可用除 16 取余法，即用 16 不断地去除待转换的十进制数，直至商等于 0 为止。将所得的各次余数，依倒序排列，即可得到所转换的十六进制数。如将 38947 转换为十六进制数，其方法及算式如下：

```
16 | 38947   3
16 | 2434    2
16 | 152     8
16 | 9       9
     0
```

即 38947=9823H

3．二进制数和十进制数整数间的相互转换

把一个十进制数转换为二进制数，可以先把该数转换为十六进制数，然后再转换为二进制数，这样可以减少计算次数；反之，要把一个二进制数转换为十进制数，也可以采用同样的办法。若使用 2^n（2^n 的二进制数等于 1 后跟 n 个 0）和十六进制数、十进制数的对应关系（如表 1-2 所示）以及个别十进制整数和十六进制数的对应关系（如 50=32H，80=50H，100=64H 等），则转换起来更为方便。如:

38947=32768+4096+2048+32+3=8000H+1000H+800H+20H+3H=9823H

1F3DH=2000H−(80H+40H+3H)=8192−(128＋64+3)=7997

表 1-2　部分二进制数与十进制数的对应关系

2^n	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{20}	2^{30}	2^{40}	2^{50}	2^{60}
十六进制数	20	40	80	100	200	400	800	1000	2000	4000	8000	10000					
十进制数	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536					
常用的缩写						1K	2K	4K	8K	16K	32K	64K	1M	1G	1T	1P	1E

1.1.2　符号数的表示法

1．机器数与真值

二进制数与十进制数一样有正负之分。在计算机中，常用数的符号和数值部分一起编码的方法表示符号数。常用的有原码、反码和补码表示法。这几种表示法都将数的符号数码化。通常正号用“0”表示，负号用“1”表示。为了区分一般书写时表示的数和机器中编码表示的数，我们称前者为真值，后者为机器数，即数值连同符号数码“0”或“1”一起作为一个数就称为机器数，而它的数值连同符号“＋”或“-”称为机器数的真值。把机器数的符号位也当作数值的数，就是无符号数。

为了表示方便，常把 8 位二进制数称为字节，把 16 位二进制数称为字，把 32 位二进制数称为双字。对于机器数应将其用字节、字或双字表示，所以只有 8 位、16 位或 32 位机器数的最高位才是符号位。

2．原码

按上所述，数值用其绝对值，正数的符号位用 0 表示，负数的符号位用 1 表示，这样表示的数就称为原码。如：

105 的真值是 +01101001B　　105 的原码是 01101001B

−105 的真值是 −01101001B　　−105 的原码是 11101001B

其中最高位为符号，后面 7 位是数值。用原码表示时，＋105 和−105 的数值部分相同而符号位相反。

原码表示简单易懂，而且与真值的转换方便。但若是两个异号数相加，或两个同号数相减，就要做减法。为了把减运算转换为加运算，从而简化计算机的结构，就引进了反码和补码。

3．反码

正数的反码与原码一样，符号位为 0，其余位为其数值；负数的反码为它的绝对值（即与其绝对值相等的正数）按位 （连同符号位）取反。如：

105 的真值是 +01101001B　　105 的反码是 01101001B

−105 的真值是 −01101001B　　−105 的反码是 10010110B

4．补码

正数的补码，其符号位为 0，其余位为其数值；负数的补码为它的绝对值（即与该负数的绝对值相等的正数）的补数。把一个数连同符号位按位取反再加 1，可以得到该数的补数。如：

105 的真值是 +01101001B　　105 的补码是 01101001B

−105 的真值是 −01101001B　　−105 的补码是 10010111B

求补数还可以直接求，方法是从最低位向最高位扫描，保留直至第一个“1”的所有位，以后各位按位取反。负数的补码可以由与其绝对值相等的正数求补得到。根据两数互为补数的原理，对补码表示的负数求补就可以得到该负数的绝对值。

例如，对−105 的补码 10010111B（97H）求补，从右向左扫描，第一位就是 1，故只保留该位，对其左面的七位均求反得其绝对值 01101001B(69H)，即补码数 97H 的真值是−69H（−105）。

一个用补码表示的机器数，若最高位为 0，则其余几位即为此数的绝对值；若最高位为 1，则其余几位不是此数的绝对值，把该数（连同符号位）求补，才得到它的绝对值。

当数采用补码表示时，就可以将其符号位作为一位数值与其他数值位一样参与数值计算，还把减法运算转换为加法运算，运算的结果也是一补码数。例如，64−10 的补码运算如下：

64−10=64+(−10)

$[64]_{补}$=40H=0100 0000B

$[10]_{补}$=0AH=0000 1010B

$[-10]_{补}$=1111 0110B

做减法运算过程如下：

$$\begin{array}{r} 0100\ 0000 \\ -\ 0000\ 1010 \\ \hline 0011\ 0110 \end{array}$$

用补码相加过程如下：

```
  0100  0000
+ 1111  0110
-----------
1 0011  0110
↑
└进位自然丢失
```

结果相同，其真值为：54（36H=48+6）。

最高位的进位是自然丢失的，故做减法与用补码相加的结果是相同的。因此，在微型机中，凡是符号数一律是用补码表示的。一定要记住运算的结果也是用补码表示的。如：

34−68=34+(−68)

34=22H=0010 0010B

68=44H=0100 0100B

−68=1011 1100B

做减运算过程如下：

```
  0010  0010
+ 0100  0100
-----------
1 1101  1110
↑
└借位自然丢失
```

用补码相加过程如下：

```
  0010  0010
+ 1011  1100
-----------
  1101  1110
```

结果相同。因为符号位为 1，所以结果为负数。对其求补，得其真值：−00100010B，即为−34（−22H）。

由上面两个例子还可以看出，当数采用补码表示后，两个正数相减，若无借位，化为补码相加就会有进位；若有借位，化作补码相加就不会有进位。

5．8 位二进制数的范围

8 位二进制数，将其看成无符号数和符号数，它所表示的数的大小是不同的。为加深读者的印象，将其列于表 1-3 中。

表 1-3　8 位二进制数（2 位十六进制数）的大小

8 位二进制数	2 位十六进制数	无符号数	原码数	反码数	补码数
0000 0000	00	0	0	0	0
0000 0001	01	1	1	1	1
0000 0010	02	2	2	2	2
⋮	⋮	⋮	⋮	⋮	⋮
0111 1101	7D	125	125	125	125
0111 1110	7E	126	126	126	126
0111 1111	7F	127	127	127	127
1000 0000	80	128	−0	−127	−128
1000 0001	81	129	−1	−126	−127
1000 0010	82	130	−2	−125	−126
⋮	⋮	⋮	⋮	⋮	⋮
1111 1101	FD	253	−125	−2	−3
1111 1110	FE	254	−126	−1	−2
1111 1111	FF	255	−127	−0	−1

由表 1-3 可知，8 位二进制无符号数的数值范围为 00H～FFH（0～255）。8 位二进制反码数的数值范围为 80H～7FH（-127～127）。8 位二进制原码数的数值范围为 FFH～7FH（-127～127）。8 位二进制补码数的数值范围为 80H～7FH（-128～127）。原码数 80H 和 00H 的数值部分相同、符号位相反，它们分别为-0 和+0；补码数 80H 的最高位既代表了符号为负又代表了数值为 1，80H 的真值是-128(-80H)。

对于一个 16 位的二进制数，若把它看成无符号数，则其数值范围为 0000H～FFFFH（0～65535）；若把它看成反码数，则其数值范围为 8000H～7FFFH（-32767～32767）；若把它看成原码数，则其数值范围为 FFFFH～7FFFH（-32767～32767）；若把它看成补码数，则其数值范围为 8000H～7FFFH（-32768～32767）。

上述分析表明，若 8 位二进制补码数运算结果超出-128～127，16 位二进制补码数运算结果超出-32768～32767，则产生溢出。小于-128 或小于-32768 的运算结果称为下溢出，大于 127 或大于 32767 的运算结果称为上溢出。产生溢出的原因是数据的位数少了，使得结果的数值部分挤占了符号位的位置，为了避免产生溢出，应该将数位扩展。

6．二进制数的扩展

二进制数的扩展是指一个数据从位数较少扩展到位数较多，如从 8 位（字节）扩展到 16 位（字），或从 16 位扩展到 32 位（双字）。一个二进制数扩展后，其数的符号和大小应保持不变。

无符号数的扩展是将其左边添加 0。如 8 位无符号二进制数 F8H 扩展为 16 位无符号二进制数，则为 00F8H。

对于用原码表示的二进制数，它的正数和负数仅 1 位符号位相反，数值位都相同。所以，原码二进制数的扩展是将其符号位向左移至最高位，符号位即最高位与原来的数值位间的所有位都填入 0。例如：68 用 8 位二进制数表示的原码为 44H，用 16 位二进制数表示的原码为 0044H；-68 用 8 位二进制数表示的原码为 C4H，用 16 位二进制数表示的原码为 8044H。

补码表示的二进制数的符号位向左扩展若干位后，所得到的补码数的真值不变。所以，对于用补码表示的二进制数，正数的扩展应该在其前面补 0，而负数的扩展，则应该在前面补 1。例如：68 用 8 位二进制数表示的补码为 44H，用 16 位二进制数表示的补码为 0044H；-68 用 8 位二进制数表示的补码为 BCH，用 16 位二进制数的补码表示为 FFBCH。

反码表示的二进制数的扩展与补码相同。

1.1.3　二进制数的加减运算

计算机把机器数均当作无符号数进行运算，即符号位也参与运算。运算的结果要根据运算结果的符号，运算有无进（借）位和溢出等来判别。计算机中设置有这些标志位，标志位的值由运算结果自动设定。

1．无符号数的运算

无符号数实际上是指参加运算的数均为正数，且整个数位全部用于表示数值。n 位无符号二进制数的范围为 $0\sim(2^n-1)$。

（1）两个无符号数相加，由于两个加数均为正数，因此其和也是正数。当和超过其位数所允许的范围时，就向更高位进位。如：

127＋160 = 7FH＋A0H

```
  0111  1111
+ 1010  0000
-------------
1 0001  1111=11FH=256+16+15=287
↑
└─进位
```

（2）两个无符号数相减，被减数大于或等于减数，无借位，结果为正；被减数小于减数，有借位，结果为负。如：

192−10=C0H−0AH

```
  1100  0000
- 0000  1010
-------------
  1011  0110=B6H=176+6=182
```

反过来相减，即 10−192，运算过程如下：

```
  0000  1010
- 1100  0000
-------------
1 0100  1010=−10110110B=−B6H=−182
↑
└─借位
```

由此可见，对无符号数进行减法运算，其结果的符号用进位来判别：CF=0（无借位），结果为正；CF=1（有借位）结果为负（对 8 位数值位求补得到它的绝对值）。

2．补码数的运算

n 位二进制补码数，除去一位符号位，还有 *n*−1 位表示数值，所能表示的补码的范围为：-2^{n-1}～$(2^{n-1}-1)$。如果运算结果超过此范围就会产生溢出。如：

105＋50=69H+32H

```
  0110  1001
+ 0011  0010
-------------
  1001  1011=9BH=155 或 =−65H=−101
```

若把结果视为无符号数，为 155，结果是正确的。若将此结果视为符号数，其符号位为 1，结果为−101，这显然是错误的。其原因是和数 155 大于 8 位符号数所能表示的补码数的最大值 127，使数值部分占据了符号位的位置，产生了溢出，从而导致结果错误。又如：

−105−50=−155

```
  1001  0111
+ 1100  1110
-------------
1 0110  0101
↑
└─进位
```

两个负数相加，和应为负数，而结果 01100101B 却为正数，这显然是错误的。其原因是和数−155 小于 8 位符号数所能表示的补码数的最小值−128，也产生了溢出。若不将第 7 位（第 7 位 ~ 第 0 位）0 看成符号，也看成数值而将进位看成数的符号，结果为−0 1001 1011B=−155，结果就是正确的。

因此，应当注意溢出与进位及补码运算中的进位或借位丢失间的区别：

（1）进位或借位是指无符号数运算结果的最高位向更高位进位或借位。通常多位二进制数将其拆成二部分或三部分或更多部分进行运算时，数的低位部分均无符号位，只有最高部分的最高位才为符号位。运算时，低位部分向高位部分进位或借位。由此可知，进位主要用于无符号数的运算，这与溢出主要用于符号数的运算是有区别的。

（2）溢出与补码运算中的进位丢失也应加以区别，如：

−50−5=−55

```
  1100  1110
+ 1111  1011
-------------
1 1100  1001=−00110111B=−55
↑
└进位丢失
```

两个负数相加，结果为负数是正确的。这里虽然出现了补码运算中产生的进位，但由于和数并未超出8位二进制补码数−128～127的范围，因此无溢出。那么如何来判别有无溢出呢?

设符号位向进位位的进位为 C_Y，数值部分向符号位的进位为 C_S，则溢出

$OF=C_Y \oplus C_S$

OF=1，有溢出；OF=0，无溢出。

下面用 M、N 两数相加来证明。设 M_S 和 N_S 为两个加数的符号位，R_S 为结果的符号位，则有表 1-4 所列的真值表。由真值表得逻辑表达式：

$OF=\overline{C}_S C_y + C_S \overline{C}_\tau = C_S \oplus C_y$

表 1-4　符号、进位、溢出的真值表

M_S	N_S	R_S	C_S	C_Y	OF
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	1	0
0	1	1	0	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	1	0	0	1	1
1	1	1	1	1	0

再来看 105+50、−105−50 和−50−5 3 个运算有无溢出：

```
  0110  1001          1001  0111          1100  1110
+ 0011  0010        + 1100  1110        + 1111  1011
-------------       -------------       -------------
  1001  1011        1 0110  0101        1 1100  1001
```

$C_y=0, C_s=1$　　$C_y=1, C_s=0$　　$C_y=1, C_s=1$

OF=0⊕1=1,有溢出　　OF=1⊕0=1,有溢出　　OF=1⊕1=0,无溢出

1.1.4　二进制数的逻辑运算与逻辑电路

计算机除了可进行基本的算术运算外，还可对两个或一个无符号二进制数进行逻辑运算。计算机中的逻辑运算，主要是“逻辑非”、“逻辑乘”、“逻辑加”和“逻辑异或”等 4 种基本运算。下面介绍这 4 种基本逻辑运算及实现这些运算的逻辑电路。

1．逻辑非

逻辑非也称“求反”。对二进制数进行逻辑非运算，就是按位求它的反，常用变量上方加一横来表示。例如，A=01100001B，B=11001011B，$\overline{A}$=10011110B，$\overline{B}$=00110100B

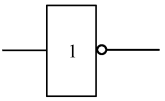

图 1-1　非门的符号表示

实现逻辑非运算的电路称为非门，又称反相器。它只有一个输入和一个输出。它的国标符号如图 1-1 所示。

2．逻辑乘

对两个二进制数进行逻辑乘，就是按位求它们的“与”，所以逻辑乘又称“逻辑与”，常用记号“∧”或“•　”来表示。1 位二进制数逻辑乘的规则为：

0∧0=0，0∧1=0,1∧0=0,1∧1=1

例如，01100001B∧11001011B，逻辑乘算式如下：

$$\begin{array}{r} 0110\ \ 0001 \\ \wedge\ 1100\ \ 1011 \\ \hline 0100\ \ 0001 \end{array}$$

即 01100001B∧11001011B=0100 0001B

实现逻辑乘运算的电路称为与门，2 输入与门的国标符号如图 1-2 所示。

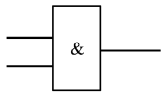

图 1-2　与门的符号表示

3．逻辑加

对两个二进制数进行逻辑加，就是按位求它们的“或”，所以逻辑加又称“逻辑或”，常用记号“∨”或“＋”来表示。1 位二进制数逻辑加的规则为：

0∨0=0,0∨1=1,1∨0=1,1∨1=1

例如，01100001B∨11001011B，逻辑加算式如下：

$$\begin{array}{r} 0110\ \ 0001 \\ \vee\ 1100\ \ 1011 \\ \hline 1110\ \ 1011 \end{array}$$

即 01100001B∨11001011B=11101011B

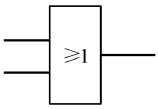

图 1-3　或门的符号表示

实现逻辑加运算的电路称为或门，2 输入或门的国标符号如图 1-3 所示。

4．逻辑异或

对两个二进制数进行逻辑异或，就是按位求它们的模 2 和，所以逻辑异或又称“按位加”，常用符号“⊕”来表示。1 位二进制数的逻辑异或运算规则为：

0⊕0=0,0⊕1=1,1⊕0=1,1⊕1=0

例如，01100001B⊕11001011B，逻辑异或算式如下：

$$\begin{array}{r} 0110\ 0001 \\ \oplus\ 1100\ 1011 \\ \hline 1010\ 1010 \end{array}$$

即 01100001B⊕11001011B=10101010B

注意：按位加与普通整数加法的区别是它仅按位相加，不产生进位。

实现逻辑异或运算的电路称为异或门，2 输入异或门的国标符号如图 1-4 所示。

异或门的特点是，只有当输入的两个变量相异时，输出为高（“1”），否则输出为低（“0”）。

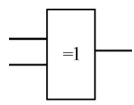

图 1-4　异或门的符号表示

5．正逻辑与负逻辑

逻辑电路实现的逻辑关系，可用高电平表示逻辑 1，用低电平表示逻辑 0，在这种规定下的逻辑关系称为正逻辑。事实上，还有另一种规定：用低电平表示逻辑 1，用高电平表示逻辑 0，在这种规定之下的逻辑关系称为负逻辑。对于同一个逻辑电路，由于采用的逻辑制度不同，使它实现的逻辑关系也就不同。

逻辑门的正逻辑和负逻辑形式如图 1-5 所示。在图中我们看到，每一种逻辑门都可以用两种逻辑符号来表示。在电路的输入和输出端上的小园圈可以看成是逻辑运算中的“非”，所以称作反相圈。在逻辑电路中使用反相圈是为了强调此处的逻辑关系是负逻辑。只要我们认真地分析一下“正与门”和“正或门”的输入输出关系，就可以看出正与门就是负或门，正或门就是负与门。计算机中常使用负逻辑，即信号多数是低电平有效，所以常把正逻辑的“或门”表示成负逻辑的“与门”形式，这就是为了表示这里是低电平的“与”操作，即输入同时为低电平时，输出为低电平；而正逻辑的“与门”又常表示成负逻辑的“或门”形式，这就是为了表示这里是低电平的“或”操作，即只要输入有一个为低电平时，输出就为低电平。还常使用负逻辑的“与非门”（正逻辑的“或非门”）和负逻辑的“或非门”（正逻辑的“与非门”），表示输出为高电平有效。

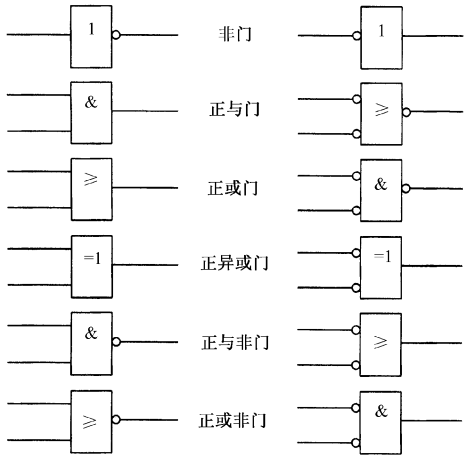

图 1-5　正负逻辑门电路的符号表示

1.1.5　二进制编码

如上所述，计算机中数是用二进制表示的，而计算机又应能识别和处理各种字符，如大小写英文字母、标点符号、运算符号等，这些又如何表示呢?在计算机里，字母、各种符号以及指挥计算机执行操作的指令，都是用二进制数的组合来表示的，这种组合的二进制数称为二进制编码。

1．ASCII（American standard code for information interchange）码

计算机里也是用 8 位二进制数表示一个字符，普遍采用 ASCII 码，常用字符的 ASCII 码如表 1-5 所列。

表 1-5　常用字符的 ASCII 码）

字　　符	ASCII 码（H）
0～9	30～39
A～Z	41～5A
a～z	61～7A
换行 LF	0A
回车 CR	0D

将十进制数的 ASCII 码转换为二进制数，要先将其转换为它所表示的 ASCII BCD 数，然后写出该 ASCII BCD 数的十进制数，最后再将十进制数转换为二进制数。例如，将十进制数的 ASCII 码 31393934H 转换为二进制数，其方法是：

（1）31393934H→01090904H

（2）01090904H→1994

（3）1994=7CAH

将二进制数转换为十进制数的 ASCII 码的过程与上述过程相反。

可以根据十进制数的 ASCII 码直接写出十进制数，但难以根据十六进制数的 ASCII 码直接写出十六进制数。十进制数的 ASCII 码与十进制数之间有一固定差值 30H，这是因为十进制数 0～9 以及十进制数的 ASCII 码 30H～39H 都是连续的。而十六进制数的 ASCII 码却是不连续的，十六进制数的 16 个 ASCII 码为 30H～39H 和 41H～46H，它们分段连续，在 39H

和 41H 之间还有一差值 7。因此，将十六进制数的 ASCII 码转换为十六进制数或将十六进制数转换为十六进制数的 ASCII 码，就要分段相减或相加。即先判别 ASCII 码是在哪个区段内，然后再加或减 30H 或 37H。例如，将 3 位十六进制数的 ASCII 码 374341H 转换为十六进制数，其方法是：37H−30H=07H、43H−37H=0CH、41H−37H=0AH，即 ASCII 码 374341H 的十六进制数为 7CAH。

2．二进制编码的十进制数

十进制数有 0～9 10 个数码。要表示这 10 个数码，需要用 4 位二进制数，这称为二进制编码的十进制数，简称为 BCD 数（binary coded decimal）。用 4 位二进制数编码表示 1 位十进制数的方法很多，较常用的是 8421 BCD 码，因组成它的 4 位二进制数位的权为 8、4、2、1 而得名。用二进制数或十六进制数表示的十进制数 0～9 10 个数码的 8421 BCD 数如表 1−6 所列。

表 1−6　8421 BCD 编码表

十进制数	压缩 BCD 数	非压缩 BCD 数（ASCII BCD 数）
0	0H（0000B）	00H（0000 0000B）
1	1H（0001B）	01H（0000 0001B）
2	2H（0010B）	02H（0000 0010B）
3	3H（0011B）	03H（0000 0011B）
4	4H（0100B）	04H（0000 0100B）
5	5H（0101B）	05H（0000 0101B）
6	6H（0110B）	06H（0000 0110B）
7	7H（0111B）	07H（0000 0111B）
8	8H（1000B）	08H（0000 1000B）
9	9H（1001B）	09H（0000 1001B）

8 位二进制数可以放 2 个十进制数位，这种表示的 BCD 数称为压缩的 BCD 数。而把用 8 位二进制数表示 1 个十进制数位的数称为非压缩的 BCD 数。例如，将十进制数 1994 用压缩的 BCD 数表示为：

0001 1001 1001 0100B 或 1994H，

而用非压缩的 BCD 数表示为：

00000001 00001001 00001001 00000100B 或 01090904H。

十进制数的 10 个数码 0～9 的 ASCII 码是 30H～39H，它们的低 4 位与其 BCD 码相同，且又是用 8 位二进制数表示 1 个十进制数，因此也称非压缩 BCD 数为 ASCII BCD 数。

十进制数与 BCD 数的转换是比较直观的，但是 BCD 数与二进制数之间的转换却是不直接的。将 BCD 数转换为二进制数，要先写出 BCD 数的十进制数，然后再按十进制数转换为二进制数的方法将十进制数转换为二进制数。例如，将压缩的 BCD 数 1994H 转换为二进制数，其方法是：

（1）压缩的 BCD 数 1994H 即是十进制数 1994。

（2）1994=2048−54=2^{11}−32H−4=1000 0000 0000B−32H−4H=800H–36H=7CAH。

同样，将二进制数转换为 BCD 数，要先按二进制数转换为十进制数的方法将二进制数转换为十进制数，然后再根据十进制数写出 BCD 数。例如，将二进制数 0111 1100 1010B（即 7CAH）转换为非压缩的 BCD 数，其方法是：

（1）7CAH=1994。

（2）十进制数 1994 的非压缩 BCD 数为 01090904H。

要注意用十六进制数表示的 BCD 数与十六进制数的差别。十六进制数 1994H 的真值为

1994H=1000H+800H+100H+80H+14H=4096+2048+256+128+20=6548

而压缩的 BCD 数 1994H 的真值为 1994。

1.1.6 BCD 数的加减运算

每位 8421 BCD 数的 4 位二进制数之内是二进制关系，BCD 数低位的 4 位二进制数与高位的 4 位二进制数之间是逢“10”进 1 的，而 4 位二进制数之间是逢“16”进 1 的。用二进制数的运算器进行 BCD 数的运算就会导致：两个 BCD 数的数位之和大于 9、小于 16 时不产生进位，其和为一非 BCD 数；大于或等于 16 时产生进位，进位值是 16 而不是 10。因此，BCD 数进行运算后必须进行调整处理。

1．BCD 数加法

两个 BCD 数相加，若相加各位的结果都在 0～9 之间，则其加法运算规则完全同二进制数的加法规则；若大于 9，则应对其进行加 6 调整。如：48＋59，因为低 4 位相加，和为 17 大于 9，高 4 位相加并与低 4 位的进位相加，和为 10 也大于 9，故都应加 6 调整，其运算和调整过程如下：

```
  0100 1000
+ 0101 1001
  1010 0001
+ 0110 0110
 10000 0111
```

和为 107。

2．BCD 数减法

两个 BCD 数相减，若本位的被减数大于或等于减数，则减法规则完全同二进制数；反之，就会向高位借位，十进制数向高位借 1 作 10，而按二进制运算规则，借 1 作 16，因此应进行减 6 调整。如 28−19，低位 8 减 9，向高位借位故应减 6 调整。其运算与调整过程如下：

```
  0010 1000
− 0001 1001
  0000 1111
− 0000 0110
  0000 1001
```

差为 9。

通常计算机中都设置有二、十进制数的调整电路，BCD 数的运算也把数当作二进制数做二进制数的运算，运算后再调整。

1.2 逻辑单元与逻辑部件

逻辑部件是用来对二进制数进行寄存、传送和变换的数字部件，其种类繁多，本书只简单地介绍微型计算机中常用的几种逻辑部件。构成逻辑部件的基本单元电路是触发器，所以首先介绍触发器。

1.2.1 触发器

触发器是具有记忆功能的基本逻辑单元电路。它能接收、保存和输出逻辑信号“0”和“1”。各类触发器都可以由逻辑门电路组成。

1．RS 触发器

RS 触发器是最简单的触发器，它是将两个与非门的输入与输出交叉连接构成，如图 1-6 所示。触发器的两个输入端是 $\overline{R}$ 和 $\overline{S}$，其中 $\overline{S}$ 称为置 1 或置位（Set）端，$\overline{R}$ 称为置 0 或复位（Reset）端。触发器有两个输出端 Q 和 $\overline{Q}$，在正常工作时，它们总是处于互补的状态。

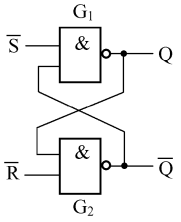

图 1-6 与逻辑组成的 RS 触发器

我们用 Q 端的状态来表示触发器的状态。由与非门的逻辑功能决定，要使触发器为 1 状态，可使 $\overline{S}=0, \overline{R}=1$。同样要使触发器为 0 状态，需令 $\overline{R}=0$，$\overline{S}=1$。触发器一旦为 1 状态（或 0 状态），$\overline{S}$（或 $\overline{R}$）端从 0 变成 1，触发器将保持 1 状态（或 0 状态）不变。即 $\overline{R}=1$，$\overline{S}=1$ 时触发器的状态不变。$\overline{R}$ 和 $\overline{S}$ 不能同时为 0，因为同时为 0 时，Q 和 $\overline{Q}$ 都为 1。当这种输入状态消失时，触发器的 Q 端可能为 0，也可能为 1，到底是 0 还是 1，是不确定的。因此不允许 $\overline{R}=0$、$\overline{S}=0$ 的情况出现，为使这种情况不出现，特给 RS 触发器加一个约束条件：$\overline{R}+\overline{S}=1$。

2．D 触发器

D 触发器有 2 个互补输出端 Q、$\overline{Q}$ 和两个输入信号，一个输入信号是时钟信号 CP，另一个是激励信号 D。

D 触发器的逻辑电路和逻辑符号如图 1-7 所示。逻辑电路中时钟信号 CP 是下降沿有效，即下降沿触发；逻辑符号中 CP 端有小圆圈表示下降沿触发，若无小圆圈表示上升沿触发。图中 G_1、G_2、G_3 和 G_4 四个逻辑门组成的电路与 G_5、G_6、G_7 和 G_8 四个逻辑门组成的电路完全相同。G_5、G_6、G_7 和 G_8 四个逻辑门组成的电路受时钟信号 CP 控制；G_1、G_2、G_3 和 G_4 四个逻辑门组成的电路受 $\overline{CP}$ 控制。G_1 和 G_2 组成 RS 触发器，G_3 和 G_4 的输出总是反相，G_5 和 G_6 组成 RS 触发器，G_7 和 G_8 的输出总是反相，所以上述 2 个 RS 触发器的输入总是满足其约束条件：$\overline{R}+\overline{S}=1$。

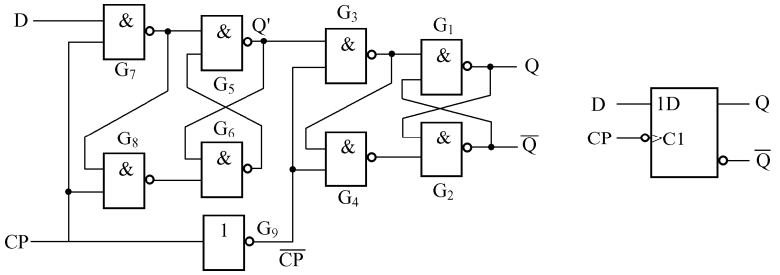

图 1-7 D 触发器逻辑电路和逻辑符号

在 CP=0 时，G_7 和 G_8 两个逻辑门被封锁，它们的输出 G_{7OUT}=1，G_{8OUT}=1，所以无论数据输入端 D 怎样变化，G_5 和 G_6 组成的 RS 触发器的输出状态保持不变，但 G_3 和 G_4 两个逻辑门被打开，它们的输出 $G_{3OUT}=\overline{Q}'$、$G_{4OUT}=Q'$，即 G_1 和 G_2 组成的 RS 触发器的 $\overline{S}=\overline{Q}'$、$\overline{R}=Q'$，所以 $Q=Q'$

在 CP=1 时，G_3 和 G_4 两个逻辑门被封锁，它们的输出 G_{3OUT}=1，G_{4OUT}=1，所以无论 G_{5OUT} 的输出 Q′ 怎样变化，G_1 和 G_2 组成的 RS 触发器的输出状态保持不变，而 G_7 和 G_8 两个逻辑门被打开，它们的输出 $G_{7OUT}=\overline{D}$、G_{8OUT}=D，即 $\overline{S}=\overline{D}$、$\overline{R}$=D，所以得到 Q′=D。

在 CP 下降沿时刻即 CP 由 1 变为=0 时，G_7 和 G_8 两个逻辑门被封锁，G_3 和 G_4 两个逻辑门被打开，此时 Q′ 锁存 CP 下降沿时刻的 D 值而不再变化，随后将该值送入 G_1 和 G_2 组成的 RS 触发器，使得 Q=D。

CP 下降沿过后 G_7 和 G_8 两个逻辑门被封锁，Q′ 锁存的 CP 下降沿时刻的 D 值保持不变，G_3 和 G_4 两个逻辑门被打开，D 触发器的状态 Q 当然也保持不变。

综上所述可得 D 触发器的特性方程：

$$Q^{n+1}=D$$

根据 D 触发器的特性方程可以列出 D 触发器的特性表，如表 1-7 所示。

表 1-7　D 触发器特性表

D	Q^n	Q^{n+1}	说明
0	0	0	置 0
0	1	0	
1	0	1	置 1
1	1	1	

有些 D 触发器还有异步复位端 $\overline{R_D}$ 和异步置位端 $\overline{S_D}$，利用它们也能实现置数的功能。

3．JK 触发器

JK 触发器有 2 个互补输出端 Q、$\overline{Q}$ 和 3 个输入信号，一个输入信号是时钟信号 CP，另两个是激励信号 J 和 K。

JK 触发器的逻辑电路和逻辑符号如图 1-8 所示。JK 触发器的逻辑电路是在 D 触发器的基础上增加 3 个逻辑门，并将输出 Q 馈送回两个与逻辑，同激励输入 J、K 相与后再相或组成的。JK 触发器与 D 触发器一样，逻辑电路中时钟信号 CP 也是下降沿有效；逻辑符号中 CP 端有小圆圈表示下降沿触发，无小圆圈表示上升沿触发。

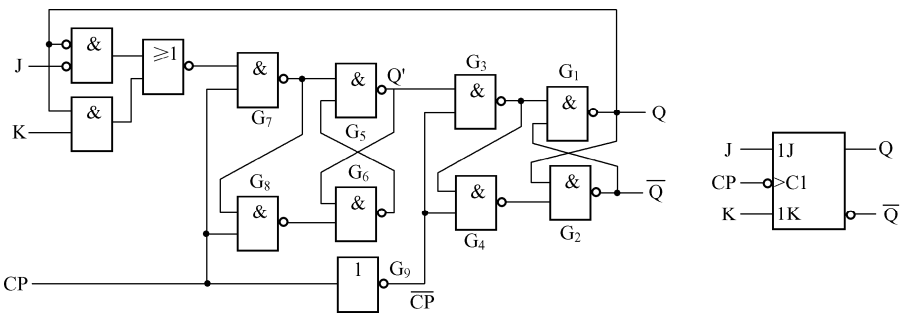

图 1-8　JK 触发器的逻辑电路和逻辑符号

将 JK 触发器的逻辑电路同 D 触发器的逻辑电路相比照可知，JK 触发器新增加的或逻辑的输出就是 D 触发器的 D 端，因此，由 JK 触发器的逻辑电路和 D 触发器的特性方程可以很容易地得到 JK 触发器的特性方程（同或的非等于异或）：

$$Q^{n+1}=\overline{J\overline{Q^n}+KQ^n}=(J+Q^n)(\overline{K}+\overline{Q^n})=J\overline{Q^n}+\overline{K}Q^n+J\overline{K}=J\overline{Q^n}+\overline{K}Q^n$$

根据 JK 触发器的特性方程可以列出 JK 触发器的特性表，如表 1-8 所列。

表 1-8　JK 触发器的特性表

J	K	Q^n	Q^{n+1}	说明
0 0	0 0	0 1	0 1	保持
0 0	1 1	0 1	0 0	置 0
1 1	0 0	0 1	1 1	置 1
1 1	1 1	0 1	1 0	翻转

根据 JK 触发器的特性表可以总结出 JK 触发器的特性是：J=K=0、$Q^{n+1}=Q^n$，J=K=1、$Q^{n+1}=\overline{Q}^n$，J、K 相异、Q^{n+1}=J。

4．T 触发器

将 JK 触发器的两个输入端连接在一起变成一个输入端 T，便构成 T 触发器。T 触发器的逻辑电路和逻辑符号如图 1-9 所示，上部的可控翻转是下降沿触发，下部的可控翻转是上升沿触发。

若取用 JK 触发器两个输入信号相等时的状态，即 J=K=T，则触发器的状态为：

$$Q^{n+1}=T\overline{Q}^n+\overline{T}Q^n=T\oplus Q^n$$

这就是 T 触发器的特性方程。由特性方程可知，D 触发器也可以构成 T 触发器，由 D 触发器构成 T 触发器如图 1-10 所示，左边的可控翻转是下降沿触发，右边的可控翻转是上升沿触发。

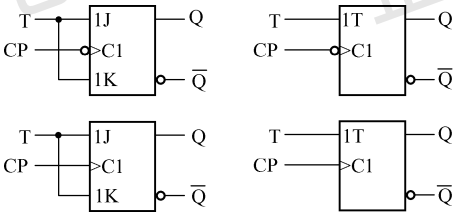

图 1-9　T 触发器的逻辑电路和逻辑符号

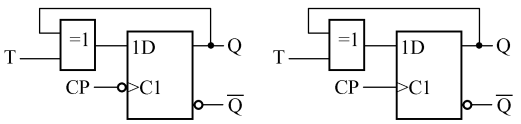

图 1-10　D 触发器构成的 T 触发器

1.2.2　寄存器

寄存器是计算机中用得最多的逻辑部件之一，它用来存放二进制信息，它具有接收二进制数码和寄存二进制数码的功能。寄存器由触发器组成。触发器具有两个稳定状态，每一个触发器可以存放 1 位二进制数，*N* 个触发器可以构成存放 *N* 位二进制数的寄存器。图 1-11 为由 4 个具有异步复位端的 D 触发器构成的寄存器的逻辑图。当 $\overline{CR}$=1 时（$\overline{CR}$ 为清 0 端，$\overline{CR}$=0 时，寄存器的 4 个 Q 端都为 0），时钟脉冲将待送的数码 $D_4D_3D_2D_1$ 送到寄存器的 $Q_4Q_3Q_2Q_1$ 保存起来。

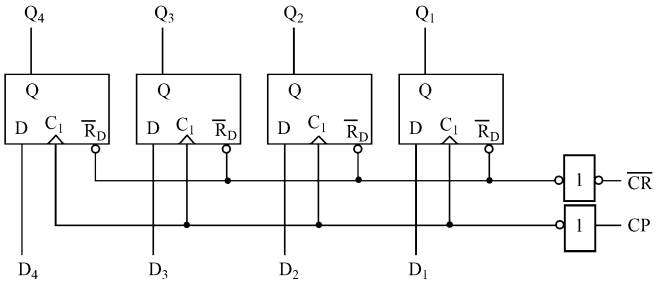

图 1-11　寄存器逻辑图

1.2.3　移位寄存器

具有移位逻辑功能的寄存器称为移位寄存器。移位寄存器一般由 D 触发器构成。图 1-12 为由 4 个 D 触发器构成的移位寄存器的逻辑图，它的第 4 级触发器的 D 端接输入信号，其余各触发器的 D 端接前一级触发器的 Q 端，所有触发器的 CP 端连在一起接收时钟脉冲信号。每来一个时钟脉冲，来自外部的输入数码（即第 4 级触发器的 D 端的输入信号）便输入一位，已被寄存的数码右移一位。

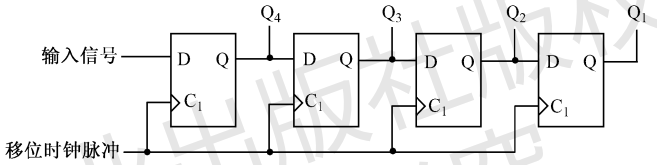

图 1-12　移位寄存器逻辑图

1.2.4　计数器

计数器是计算机中又一种常用的逻辑部件，它不仅能存储数据，而且还能记录输入脉冲的个数，计数器的种类繁多，可以从不同角度来分类。按工作方式，可分为同步计数器和异步计数器；按加减计数顺序，可分为加法计数器和减法计数器；按进位制，可分为二进制计数器、十进制计数器和任意进制计数器等。

1．异步二进制加法计数器

由 J K 触发器构成的 3 位异步二进制加法计数器的逻辑图如图 1-13 所示。其工作波形如图 1-14 所示。初始时，将计数器置为全 0 状态（即 $Q_3Q_2Q_1$ 为 000）。第 1 个计数脉冲来到后，第 1 级触发器翻转，Q_1 由 0 变 1，第 2、3 级触发器因时钟端无触发脉冲，它们维持原状态不变，故计数器的状态 $Q_3Q_2Q_1$ 为 001。第 2 个计数脉冲来到后，第一级触发器又翻转，Q_1 由 1 变 0，第 2 级触发器因其时钟输入端有脉冲下降沿的作用，也进行翻转，Q_2 由 0 变 1，Q_3 仍保持原状态，计数器的状态 $Q_3Q_2Q_1$ 为 010。按照这样的顺序工作下去，直至第 7 个计数脉冲来到后，计数器的状态 $Q_3Q_2Q_1$ 为 111。此时再来一个计数脉冲，计数器又回到初始时的全 0 状态。图 1-13 所示就是这样周而复始地工作的。

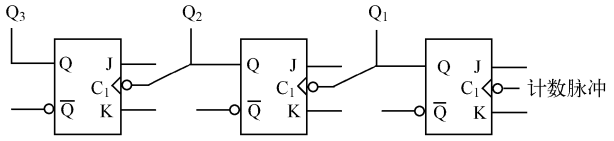

图 1-13　异步二进制加法计数器的逻辑图

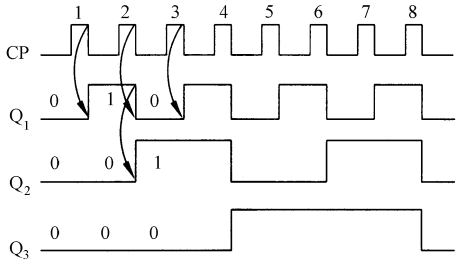

图 1-14　异步二进制加法计数器的工作波形

2．同步二进制加法计数器

由 JK 触发器构成的 3 位同步二进制加法计数器的逻辑图如图 1-15 所示。其工作波形如图 1-16 所示。初始时，将计数器置为全 0 状态（即 $Q_3Q_2Q_1$ 为 000）。第 1 个 CP 脉冲来到后，由于第 1 级的 JK 端为 1，第 2 级和第 3 级的 JK 端为 0，所以第 1 级触发器翻转，Q_1 由 0 变 1，第 2 级和第 3 级触发器维持原状态不变，计数器的状态 $Q_3Q_2Q_1$ 为 001。第 2 个 CP 脉冲来到后，由于第 1 级和第 2 级的 JK 端为 1，第 3 级的 JK 端为 0，故第 1 级和第 2 级触发器翻转，第 3 级触发器维持原状态不变，计数器的状态 $Q_3Q_2Q_1$ 为 010。第 3 个 CP 脉冲来到后，第 1 级触发器翻转，第 2 级和第 3 级触发器维持原状态不变，计数器的状态 $Q_3Q_2Q_1$ 为 011。第 4 个 CP 脉冲来到后，3 级触发器的 JK 端都为 1，故 3 个触发器均翻转，计数器的状态 $Q_3Q_2Q_1$ 为 100。按照这样的顺序工作下去，直至第 7 个 CP 脉冲来到后，计数器的状态 $Q_3Q_2Q_1$ 为 111。此时再来 1 个 CP 脉冲，由于 3 个触发器的 JK 都为 1，故 3 级触发器均翻转，计数器的状态 $Q_3Q_2Q_1$ 又回到初始的全 0 状态。图 1-15 所示就是这样周而复始地工作的。

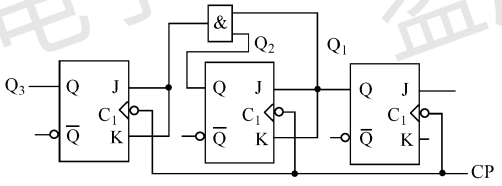

图 1-15　同步二进制加法计数器的逻辑图

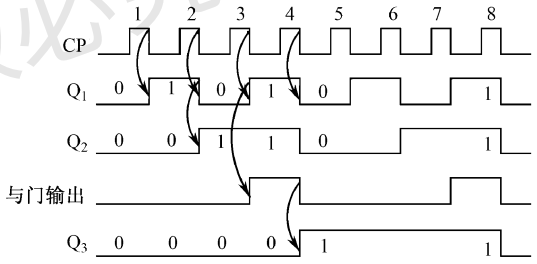

图 1-16　同步二进制加法计数器工作波形

1.2.5　三态输出门与缓冲放大器

在逻辑电路中，逻辑值有 1 和 0，它们分别对应于高电平和低电平这两种状态。三态输出门除去通常的那两种状态之外，还有被称作“高阻抗”的第三种状态。可以把高阻抗状态理解为输出与输入之间近于开路的状态。决定三态输出门是否进入高阻态，是由一条辅助控制线来控制的：当这条线的控制电平为允许态时（1 或者 0），三态输出门与一般的两态输出门一样；当这条线的控制电平成为禁止态时（0 或者 1），三态门就进入高阻态，这种三态输出门电路的符号如图 1-17 所示。三态输出门也可以称为三态缓冲器。

三态输出门电路可以加到寄存器的输出端上，这样的寄存器就称为三态（缓冲）寄存器。使用三态输出门电路计算机可以通过一组信息传输线与一个寄存器接通，也可以与其断开而

与另外一个寄存器接通，即一组信息传输线可以传输任意多个寄存器的信息，这组传输线就是计算机的总线（BUS）。

三态输出门电路还可以使一组总线实现双向信号传输。双向信号传输线如图 1-18 所示，当 E=0 时，数据 D_i 传向 D_j；当 E=1 时，数据 D_j 传向 D_i。

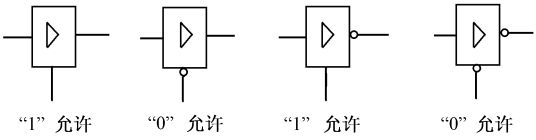

图 1-17　4 种类型的三态缓冲器

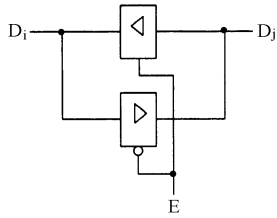

图 1-18　由三态缓冲器组成的双向传输线

1.2.6　译码器

在计算机中常常需要将一种代码翻译成控制信号，或在一组信息中取出所需要的一部分信息，能完成这种功能的逻辑部件称为译码器。2-4 译码器如图 1-19 所示。当 E=0 时，$\overline{Y_0}$ ～ $\overline{Y_3}$ 均为 1，即译码器没有工作。当 E=1 时，译码器进行译码输出。如果 A_1A_0=00，则 $\overline{Y_0}$ =0，其余为 1；同样 A_1A_0=01 时，只有 $\overline{Y_1}$ =0；A_1A_0=10 时，只有 $\overline{Y_2}$ =0；A_1A_0=11 时，只有 $\overline{Y_3}$ =0。由此可见，输入的代码不同，译码器的输出状态也就不同，从而完成了把输入代码翻译成对应输出线上的控制信号。

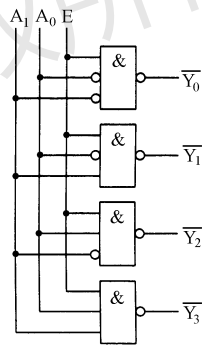

图 1-19　2-4 译码器逻辑图

1.3　微型计算机的结构和工作原理

1.3.1　微型计算机常用的术语

1．位（bit）

位是计算机所能表示的最基本、最小的数据单元。因为计算机采用二进制数，所以位就是 1 个二进制位，它有两种状态“0”和“1”。由若干个二进制位的组合就可以表示各种数据、字符等。

2．字（word）和字长

字是计算机内部进行数据处理的基本单位，通常它与计算机内部的寄存器、算术逻辑单元、数据总线宽度相一致。计算机的每一个字所包含的二进制位数称为字长。

3．字节（byte）

把相邻的 8 位二进制数称为字节。字节长度是固定的，但不同计算机的字长是不同的。8 位微机的字长等于 1 个字节，而 16 位微机的字长等于 2 个字节，32 位微机的字长等于 4 个字节。

目前为了表示方便，常把一个字节定为 8 位，把一个字定为 16 位，把一个双字定为 32 位。

4．指令（instruction）

指令是规定计算机进行某种操作的命令。它是计算机自动运行的依据。计算机只能直接识别 0 和 1 数字组合的编码，这就是指令的机器码。微型计算机的机器码指令有 1 字节、2 字节，也有多字节，如 4 字节、6 字节等。指令除了规定执行的操作外，它还要规定参与该操作的操作数据或者是操作数据的位置。

5．程序（program）

程序是指令的有序集合；是一组为完成某种任务而编制的指令的序列。

6．指令系统（instruction set）

指令系统指一台计算机所能执行的全部指令。

1.3.2 微型计算机的基本结构

微型计算机主要由微处理器或中央处理单元（CPU）、存储器（RAM 和 ROM）、I/O 接口、I/O 设备及总线组成，如图 1-20 所示。

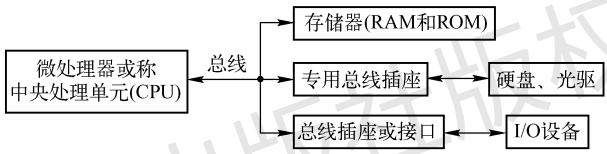

图 1-20　微型计算机的基本结构

1．中央处理器 CPU（central processor unit）或称微处理器（microprocessor unit）

中央处理器具有算术运算、逻辑运算和控制操作的功能，是微型计算机的核心部分。它主要由 3 个基本部分组成：

（1）算术逻辑单元 ALU（arithmetic logic unit）。用来执行基本的算术运算和逻辑运算。

（2）寄存器（register）。CPU 中有多个寄存器，用来存放操作数、中间结果以及反映运算结果的状态标志位等。

（3）控制器（control unit）。控制器具有指挥整个系统操作的功能。它按一定的顺序从存储器中读取指令，进行译码，在时钟信号的控制下，发出一系列的操作命令，控制 CPU 以及整个系统有条不紊地工作。

2．总线

总线是把计算机各个部分有机地连接起来的一组并行的导线，是各个部分之间进行信息交换的公共通道。微型计算机中，连接 CPU、存储器和各种 I/O 设备并使它们之间能够相互传送信息的信号线及其控制信号线称之为总线。一个功能部件只要符合总线标准，就可以连接到采用这种总线标准的微型计算机系统中，使系统的功能得到扩展。在微型计算机的不同层次结构中，有不同的总线，它们是 CPU 总线、局部总线和外部总线。从传输信息的类型上，这 3 类总线按功能分为 3 组，这 3 组总线是地址总线 AB（address bus）、数据总线 DB（data bus）和控制总线 CB（control bus）。

（1）地址总线。负责传输数据的存储位置或 I/O 接口中的寄存器的一组信号线称之为地址总线。它传送 CPU 发出的地址，以便选中 CPU 所寻址的存储单元或 I/O 端口（一个接口有 1 个或几个端口）。MCS-51 单片机对外部扩展的地址总线为 16 位，用 A_{15}～A_0 表示，可寻址的内存储器单元或 I/O 端口为 2^{16}=64K（1K 为 1024）。80x86 的地址总线为 20 位或 32 位，用 A_{19}～A_0 或 A_{31}～A_0 表示，所以可寻址的内存储器单元为 2^{20}=1M 或 2^{32}=4G；对 I/O 端口是通过地址总线的低 16 位来寻址的，故可寻址 I/O 端口 64K。

（2）数据总线。负责传输数据的一组信号线称之为数据总线。数据在 CPU 与存储器和 CPU 与 I/O 接口之间的传送是双向的，故数据总线为双向总线。MCS-51 单片机对外部扩展的数据总线为 8 位，用 D_7～D_0 表示，即字长为 8 位。8086 和 80286 的数据总线为 16 位，用 D_{15}～D_0 表示。8088 的数据总线为 8 位，用 D_7～D_0 表示，8088 为准 16 位微处理器，这是 8086 和 8088 的唯一区别。80386 和 80486 的数据总线为 32 位，Pentium 的数据总线为 64 位。数据的含义是广义的，数据总线上传送的不一定是真正的数据，有可能是从存储器取出的指令代码，还可能是 I/O 设备的状态量和控制量。

（3）控制总线。在传输与交换数据时起管理控制作用的一组信号线称之为控制总线。它传送各种信息，有的是 CPU 到存储器或 I/O 接口的控制信号，如读信号 $\overline{RD}$、写信号 $\overline{WR}$、地址锁存允许信号 ALE（address latch enable）、中断响应信号 $\overline{INTA}$（interrupt acknowledge）等；有的是 I/O 接口到 CPU 的信号，如可屏蔽中断请求信号 INTR、准备就绪信号 READY 等。控制信号线有的是高电平有效，如：ALE、INTR、READY 等；有的是低电平有效，如：$\overline{RD}$、$\overline{WR}$、$\overline{INTA}$ 等。

3．存储器（memory）

（1）存储器的分类。存储器分为内部存储器和外部存储器两大类，分别简称为内存和外存，外存也叫辅存，内存也叫主存。程序和数据以文件的形式保存在外存中，要执行的程序和要使用的数据必须事先调入内存。为了加快 CPU 访问内存的速度，从而提高程序的运行速度，在内存和 CPU 之间或者 CPU 内部增加了存取速度较高的高速缓冲存储器，即 cache。为了扩充内存容量，还将外存作为内存的辅助，给用户提供比内存大得多的逻辑存储容量，这就是所谓的“虚拟存储器”。

① 虚拟存储器。人们通常所指的内存是由“内存条”组成的物理存储器，物理存储器是由地址总线直接访问的存储空间，其地址称为物理地址。显然，地址总线的条数决定了物理存储器即内存的最大容量。虚拟存储器是相对物理存储器而言的，虚拟存储器是指程序使用的逻辑存储空间，它可以比物理存储空间大得多。虚拟存储器由内存，辅存和管理部件共同组建。通过管理软件达到内存和辅存密切配合，使整个存储系统的速度接近内存，容量接近辅存。当应用程序访问虚拟存储器时，必须给出虚拟地址即逻辑地址，在此过程中，先通过硬件和软件找出逻辑地址到物理地址之间的对应关系，判断要访问的是否已装入内存，如已装入则直接访问内存；否则，相应的硬件和管理软件会将要访问的有关数据块从辅存调入内存，与此同时将内存中原有的暂时不使用的某数据块调回辅存，并且将虚拟地址转变为物理地址。有了虚拟存储器，用户程序就不必考虑内存的容量大小了。用户程序运行时，存储管理部件会把要用到的程序和数据从辅存一块一块调入内存，好像内存的容量变得足够大，从而程序不再受内存容量的限制，因此，可运行大型的程序。块是内存和辅存之间数据传送的基本单位。根据对虚拟存储器不同的管理方式，块可以具体化为段、页和段页 3 种形式，

相应地形成分段存储管理、分页存储管理和段页存储管理。

② 高速缓冲存储器。高速缓冲存储器即高速缓存 Cache 是使用速度与 CPU 相当的静态随机读/写存储器芯片组成的小容量存储器，用来存放微处理器最近要用的指令和数据。Cache 中的内容其实是内存中一小部分内容的复制品，内存中经常被微处理器使用到的一部分内容要烤贝到 Cache 中，并不断地更新 Cache 中的内容，使得 Cache 中总是保存有最近经常被微处理器使用的一部分内容。Cache 中存放的内容除了内存中的指令和数据外，还要存放这些指令和数据在内存中的对应地址。当微处理器存取指令和数据时，Cache 截取微处理器送出的地址，并判别这个地址与 Cache 中保存的地址是否相同。若相同，则从 Cache 中存取该地址中的指令或数据；否则就从内存中存取。Cache 的存储容量比内存的容量小得多，但不能太小，太小会使命中率太低。所谓命中率是在 CPU 访问 Cache 时，所需信息恰好在 Cache 中的概率。Cache 的存储容量也没有必要过大，过大不仅会增加成本，而且当 Cache 的容量超过一定值后，命中率随容量的增加将不会有明显的增长。

（2）存储器单元的地址和内容。存储器的主要功能是存放程序和数据，程序是计算机操作的依据，数据是计算机操作的对象。不管是程序还是数据，在存储器中都是用二进制的“1”或“0”表示，统称为信息。为实现自动计算，这些信息必须预先放在存储器中。存储器被划分成许多小单元，称为存储单元。每个存储单元相当于一个缓冲寄存器。为了便于存入和取出，每个存储单元必须有一个固定的地址，称为单元地址。单元地址用二进制编码表示，如图 1-21 所示。每个存储单元的地址只有一个，固定不变，而存储在其中的信息是可以更换的。存储器的地址是数以千计的。为了减少存储器向外引出的地址线，在存储器内部都自带有地址译码器。

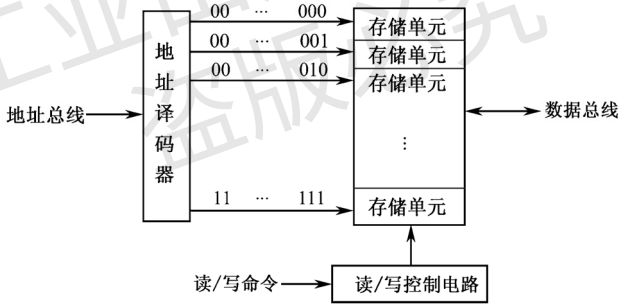

图 1-21　存储器的单元地址

（3）存储器的操作。CPU 对存储器的操作有读和写两种，读操作是 CPU 将存储单元的信息取到 CPU 内部，而写操作是 CPU 将其内部的信息传送到存储单元保存。写操作要改变被写存储单元的内容，而读操作则不改变被读存储单元的内容。向存储单元存放或取出信息，都称为访问存储器。访问存储器时，先由地址译码器将送来的单元地址进行译码，找到相应的存储单元；再由读写控制电路，根据送来的读或写命令确定访问存储器的方式，完成读出（读）或写入（写）操作。

4．总线插座和接口

外部设备与 CPU 之间通过总线和接口连接。设置接口主要有以下几个方面的原因，一是外部设备大多数都是机电设备，传送数据的速度远远低于计算机，因而需要接口作

数据缓存。二是外部设备表示信息的格式与计算机不同。例如，由键盘输入的数字、字母，先由键盘接口转换成 8 位二进制码（ASCII 码），然后再送入计算机，因此需用接口进行信息格式的转换。三是接口还可以向计算机报告设备运行的状态，传达计算机的命令等。

通过外部总线与微型计算机相连的外部设备，其自身就带有接口，所以只需将其连接它们的电缆直接插入相应的总线插座；通过局部总线与微型计算机相连的外部设备，就需要用电缆将其连接到相应的接口卡，再将该接口卡插入相应的总线插座。

5．I/O 设备

I/O 设备又称为外部设备，它通过 I/O 接口与微型计算机连接。

输入设备是变换输入信息形式的部件。它将人们熟悉的信息形式变换成计算机能接收并识别的信息形式。输入的信息形式有数字、字母、文字、图形、图像等多种形式，送入计算机的只有一种形式，就是二进制数据。一般的输入设备只用于原始数据和程序的输入。常用的输入设备有键盘、模数转换器、扫描仪等。

输出设备是变换计算机的输出信息形式的部件。它将计算机处理结果的二进制信息转换成人们或其他设备能接收和识别的形式，如字符、文字、图形等。常用的输出设备有显示器、打印机、绘图机等。

磁盘和光盘等大容量存储器也是计算机的重要的外部设备，它们既是输入设备，也是输出设备。它们有存储信息的功能，因此，常常作为辅助存储器使用。

1.3.3　计算机的工作原理

CPU、存储器、总线和接口、外部设备构成了计算机的硬件（hardware），光有这样的硬件还只是具有了计算的可能。计算机要真正能够进行计算还必须有多种程序的配合。那么什么是程序呢？当我们要用计算机完成某项任务时，例如，要解算一道数学题时，就要先把题目的解算方法分成计算机能识别并能执行的基本操作命令，这些基本操作命令按一定顺序排列起来，组成了程序，而其中每一条基本操作命令就是一条指令，指令是对计算机发出的一条条工作命令，命令计算机执行规定的操作。因此，程序是实现既定任务的指令序列，其中的每条指令都规定了计算机执行的一种基本操作，计算机按程序安排的顺序执行指令，就可以完成既定任务。

指令必须满足两个条件：一是指令的形式是计算机能够理解的，因此指令也采用和数据一样的二进制数字编码形式表示；二是指令规定的操作必须是计算机能够执行的，即每条指令的操作均有相应的电子线路实现。各种类型的计算机的指令都有自己的格式和具体的含义，但必须指明操作性质（如加、减、乘、除，比较大小等）和参加操作的有关信息（如数据或数据的存放地址等）。

指令的不同组合方式，可以构成完成不同任务的程序，一台计算机的指令种类是有限的，但在人们的精心设计下，实现信息处理任务的程序可以无限多，计算机严格忠实地按照程序安排的指令顺序，有条不紊地执行规定的操作，完成预定任务。为实现自动连续地执行程序，必须先把程序和数据送到具有记忆功能的存储器中保存起来，然后由控制器和 ALU 依据程序中指令的顺序周而复始地取出指令，分析指令，执行指令，直到完成全部指令操作为止。存储程序和程序控制体现了现代计算机的基本特性，是计算

机的基本工作原理。

1.4 微处理器

微型机是由具有不同功能的一些部件组成的。微处理器或称中央处理单元（CPU）是微型机的心脏，它决定了微型机的结构。要构成一台微型计算机，必须了解微处理器的内部结构。Intel 公司的 8086/8088、80286、80386、80486、Pentium 等新一代微处理器统称为 80x86 微处理器。32 位的 Pentium 微处理器是从 16 位的 8086 微处理器经由 80286、80386 和 80486 逐渐发展而来的，为了保持与已开发的软件的兼容性，又能够充分发挥 32 位微处理器的特有优势，80x86 采用多种工作方式来解决这一矛盾。Pentium 微处理器的工作方式有 4 种，它们是实地址方式、虚地址保护方式、虚拟 8086 方式和系统管理方式。本节将根据 80x86 芯片发展和演变过程介绍 80x86 系列微处理器，着重介绍 8086 和 Pentium 微处理器及其内部结构，它是掌握以 80x86 微处理器为 CPU 的微型计算机的基础。

1.4.1 8086/8088

8086/8088 是 Intel 公司 1981 年推出的 16 位的微处理器。8086 微处理器的数据总线（8088 内部是 16 位、外部是 8 位）为 16 位，地址总线为 20 位，可寻址 2^{20} 字节即 1MB 内存。著名的 PC XT 微型计算机就是 IBM 公司用 8088 作为 CPU 的最早的 PC 机。

8086/8088 由两个独立的工作单元组成，如图 1-22 所示，即执行单元（execution unit）和总线接口单元（bus interface unit）。图的左半部分为执行单元 EU，右半部分为总线接口单元 BIU。EU 不与外部总线（或称外部世界）相连，它只负责执行指令。而 BIU 则负责从存储器或外部设备中读取指令和读/写数据，即完成所有的总线操作。这两个单元处于并行工作状态，可以同时进行读/写操作和执行指令的操作。这样就可以充分利用各部分电路和总线，提高微处理器执行指令的速度。

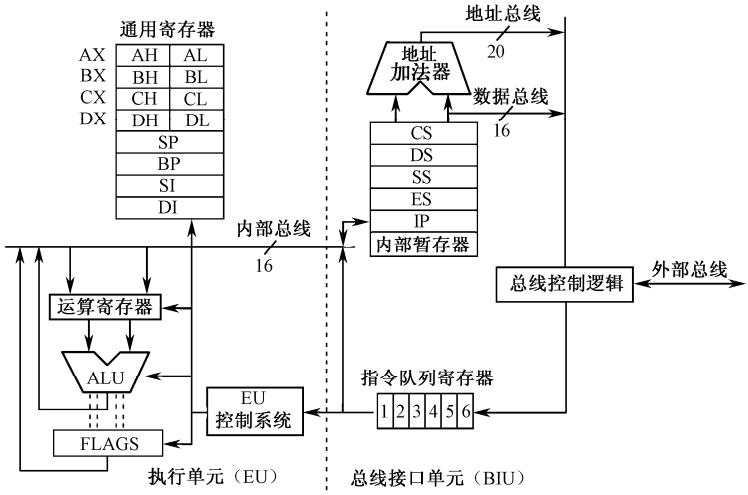

图 1-22　8086/8088 的内部结构

1．执行单元 EU

执行单元 EU 包括一个 16 位的算术逻辑单元 ALU，一个反映 CPU 状态和控制标志的状态标志寄存器 FLAGS，一组通用寄存器，运算寄存器和 EU 控制系统。所有的寄存器和数据传输通路都是 16 位的，它们之间进行快速的内部数据传输。EU 从 BIU 中的指令队列寄存器中取得指令和数据，执行指令要求的操作。该操作有两种类型：一是进行算术逻辑运算，二是计算存储器操作数的偏移地址。当指令要求执行存储器或 I/O 设备的数据存取操作时，EU 向 BIU 发出请求。BIU 根据 EU 的请求，完成 8086/8088 与存储器或外部设备之间的数据传送。

2．总线接口单元 BIU

总线接口单元 BIU 包括一组段寄存器（CS、DS、SS、ES）、一个指令指示器 IP、6 个（8088 是 4 个）字节的指令队列、地址加法器和总线控制逻辑。段寄存器提供的段地址与偏移地址在地址加法器中相加，并将其结果存放在物理地址锁存器中。指令队列寄存器为一个能存放 6 个字节的存储器，在 EU 执行指令的过程中，BIU 始终根据指令指示器提供的偏移地址，从存放指令的存储器中预先取出一些指令存放在指令队列中。取来的指令在指令队列中是按字节顺序存放的，如同排队购物一样，取来的指令在指令队列中排队。在大多数情况下，指令队列中至少应有一个字节的指令，这样 EU 不必等待 BIU 去取指令。BIU 在下面两种情况下执行取指操作：一是当指令队列中出现两个以上字节空的时候，BIU 自动地执行取指操作，将所取指令补充到指令队列中。二是当程序发生转移时，BIU 执行取指操作，BIU 将所取得的第一条指令直接送到 EU 中去执行，将随后取来的指令重新填入指令队列，冲掉转移前放入指令队列中的指令。

1.4.2 80286

80286 微处理器是 Intel 公司 1982 年推出的产品。80286 内部和外部数据总线都是 16 位，地址总线为 24 位，可寻址 2^{24} 字节即 16MB 内存，支持虚拟存储器管理技术，虚拟存储空间可达 1GB(2^{30})。PC AT 机就是 IBM 公司用 80286 作为 CPU 的 286 PC 机。

80286 微处理器由地址单元 AU（Address Unit）、总线单元 BU（Bus Unit）、指令单元 IU（Instruction Unit）和执行单元 EU 等 4 个单元组成，80286 将 8086 中的总线接口单元 BIU 分成了地址单元 AU、指令单元 IU 和总线单元 BU 等 3 部分。地址单元 AU 中的存储器管理机构使用段式管理方式，首次实现了虚拟存储器管理。这样，就提高了这些单元操作的并行性，从而提高了吞吐率，加快了 CPU 的处理速度。

1.4.3 80386

80386 是 Intel 公司 1985 年推出的一种高性能 32 位微处理器。80386 内部和外部数据总线都是 32 位的，地址总线为 32 位，可寻址 4GB，虚拟存储空间可达 64TB(2^{46})。

80386 微处理器由总线接口单元 BIU、指令译码单元 IDU（Instruction Decode Unit）、指令预取单元 IPU（Instruction Prefetch Unit）、执行单元 EU、分段单元 SU（Segmentation Unit）和分页单元 PU（Paging Unit）等 6 个单元组成。80386 的结构和 80286 基本相同，主要的区别是 80386 的存储器管理机构由分段单元和由分页单元组成。段单元用来把逻辑地址变换成

线性地址。页单元的功能是把线性地址换算成物理地址。

1.4.4 80486

80486 是 Intel 公司于 1989 年推出的新型 32 位微处理器。80486 的内部数据总线为 64 位，外部数据总线为 32 位，地址总线为 32 位。

80486 微处理器内部由总线接口单元、指令译码单元、指令预取单元、执行单元、分段单元、分页单元以及浮点处理单元（FPU）和高速缓存（cache memory）等 8 个单元组成，比 80386 新增加了相当于 80387 功能的 FPU 和 Cache 两个单元。8086/8088、80286 和 80386 的字长为 16 位或 32 位，能表达的数据范围不大，对于数值计算不太适宜。为此，在 8086／8088、80286 和 80386 微处理器的基础上设计了与之配合的专门用于数值计算的协处理器 8087、80287 和 80387。这些协处理器分别与 8086、80286 和 80386 密切配合，可以使数值运算，特别是浮点运算的速度提高约 100 倍。而 80486 将 FPU 集成在其内部，其处理速度显著提高，比 80387 约快 3～5 倍。为了进一步提高处理速度，在 80486 内部又集成了 8KB Cache。

从应用角度看，80486 相当于以 80386 的 CPU 为核心，内含 FPU 和 Cache 的微处理器。再加上 80486 采用了 RISC（Reduced Instruction Set Computer 精减指令系统计算机）技术，时钟倍频技术和新的内部总线结构，所以 80486 的处理速度有极大的提高。

1.4.5 Pentium

Intel 公司对 80x86 系列微处理器的性能不断地创新与改造，继 80486 之后，1993 年推出新一代名为 Pentium 的微处理器。1995 年又推出名为 Pentium Pro 的微处理器。1997 年、1999 年和 2000 年又相继推出 Pentium Ⅱ、Pentium Ⅲ和 Pentium Ⅳ微处理器。Pentium 是希腊字 Pente（意思为 5）演变来的。Pentium 有 64 位数据线和 32 位地址线。Pentium Pro/Ⅱ/Ⅲ/Ⅳ具有 64 位数据线和 36 位地址线。

除了将控制寄存器和测试寄存器均增加到 5 个外，Pentium 与 80486 的最大区别是：Pentium 内部具有 8KB 指令 Cache 和 8KB 数据 Cache，而 Pentium Pro 内部具有 8KB 指令 Cache 和 8KB 数据 Cache 外，还有 256KB 二级 Cache。PentiumⅡ/Ⅲ/Ⅳ的指令 Cache 和数据 Cache 均增加到 16KB，二级 Cache 也增加到 512KB。Pentium 和 Pentium Pro/Ⅱ/Ⅲ/Ⅳ还采用了一些其他的最新技术，在体系结构上还有一些新的特点。因而它们的性能明显高于 80486。

1．Pentium 微处理器的内部结构

Pentium 微处理器内部由总线接口单元、指令译码单元、指令预取单元、分段单元和分页单元、浮点处理单元、控制 ROM、控制单元、指令 Cache 和数据 Cache、U 流水线和 V 流水线、分支转移目标缓冲器（branch target buffer）以及整数寄存器组和浮点数寄存器组等 11 个单元组成，如图 1-23 所示。

（1）总线接口单元。总线接口单元实现微处理器与微型计算机系统总线的连接，其中包括 64 位数据线、32 位地址线和众多的信号线，以此实现微处理器与外部的信息交换。Pentium 微处理器与外部交换数据可以是 64 位，还可以是 32 位、16 位或者 8 位，即数据总线的宽度是可以控制的，总线接口单元通过控制信号实现对总线宽度的控制。

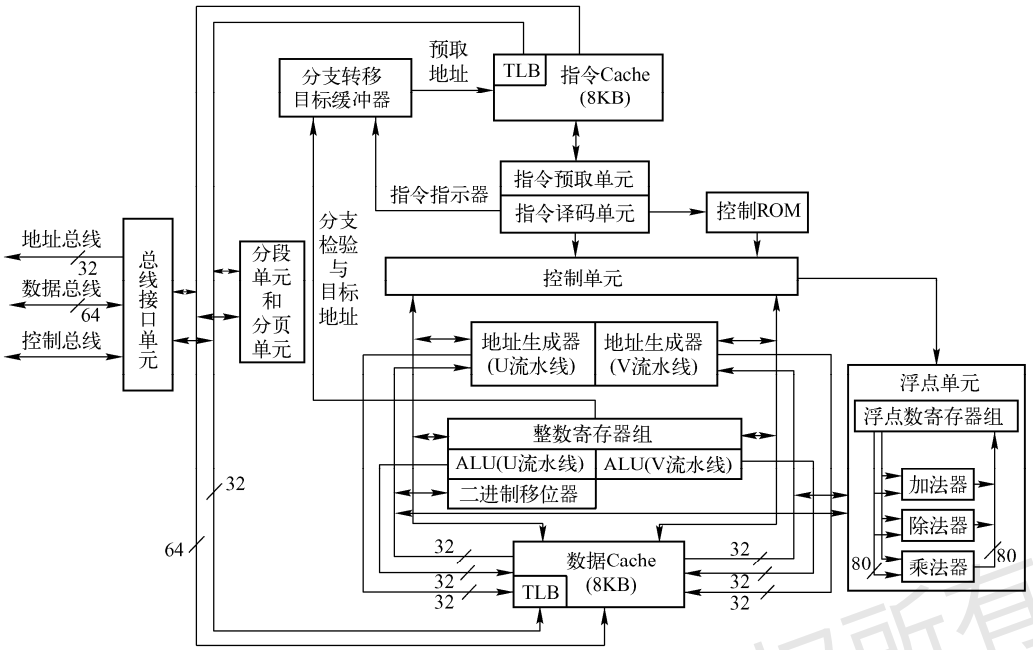

图 1-23　Pentium 微处理器内部结构框图

（2）分段单元和分页单元。分段单元将程序提供的逻辑地址转换为线性地址，分页单元将线性地址转换为物理地址。分页是将段分为多个固定大小的页面（页面大小通常为 4KB），分页支持虚拟存储器环境。在虚拟存储器环境中，用一个小容量的物理内存和一些磁盘空间来模拟一个非常大的线性地址空间。内存中只保留程序访问的页面，而众多的页面被存储在磁盘中。当程序要访问线性地址空间中的某个地址时，分页单元先将线性地址转换为存储器的物理地址，然后执行对该地址的读操作或写操作。如果所访问的页面不在物理内存中，微处理器就会暂时中断该程序的执行，由操作系统将所需的页面从磁盘读入物理内存中，然后接着执行被中断的程序。

（3）U 流水线和 V 流水线。Pentium 采用两条流水线，这两条流水线都拥有自己的算术逻辑单元 ALU、地址生成电路和数据 Cache 的接口。这种双流水线结构允许 Pentium 可以一次执行两条指令，每条流水线中执行一条。

（4）指令 Cache 和数据 Cache。在 Pentium 中，指令 Cache 和数据 Cache 两者分开，从而减少了指令预取和数据存取操作之间可能发生的冲突，并可提高命中率。Pentium 的数据 Cache 有两个接口，分别与 U 和 V 两条流水线相连，以便同时和两条流水线交换数据。

（5）指令预取单元、指令译码单元和控制 ROM。指令预取单元从指令 Cache 中预先取指令，每次取两条指令。如果是简单指令，并且后一条指令不依赖前一条指令的执行结果，那么，便通过指令译码单元译码后，将两条指令分别送到 U 流水线和 V 流水线执行。如果是复杂指令，微处理器就通过控制 ROM 将其转换成对应的一系列微指令，再送到 U 流水线和 V 流水线执行。复杂指令对应的微指令存放在控制 ROM 中。

微指令是微处理器能够直接执行的指令，微指令不同于 80x86 的变长指令，它的长度是固定的，因此很容易在流水线中进行处理。指令译码单元会将指令转换成一条或多条微指令，

如果是复杂的指令则会分解为一系列的微指令。大多数 80x86 的指令会被编译成 1～2 条微指令，一些很简单的指令如加指令、逻辑与指令、逻辑或指令、逻辑异或指令等仅被编译成 1 条微指令，而乘、除以及非直接存储器寻址的指令则会生成较多条微指令，极为复杂的指令则可能会生成上百条微指令。

（6）控制单元。控制单元的功能是通过对来自指令译码单元和和控制 ROM 中微程序的解析，控制 U 流水线、V 流水线和浮点处理单元的正常运行。

（7）分支转移目标缓冲器。分支转移目标缓冲器在遇到分支转移指令时用来预测转移是否发生，并据此为分支指令处的指令提供预取地址。

（8）浮点处理单元。浮点处理单元主要用于浮点运算，使得浮点运算的速度得到提高。由于浮点处理单元内含专用的加法器、乘法器和除法器，对加法、乘法和除法这些常用浮点指令用专门的硬件电路来实现，所以，大多数浮点运算指令仅一个时钟周期即可执行完毕。

2．Pentium 微处理器的先进技术

（1）超标量流水线。80486 微处理器只有一条指令流水线，而 Pentium 微处理器具有 U、V 两条指令流水线。所谓超标量流水线是一个处理器中有多条指令流水线。Pentium 微处理器是超标量为 2 的首款 Intel 微处理器，每个时钟周期能同时执行两条整型指令。

Pentium 的指令流水线由总线接口单元、指令预取单元、指令译码单元、控制单元和控制 ROM 构成。总线接口单元连接 CPU 和其他单元，它控制数据总线和地址总线的信息传输。指令预取单元在总线空闲时从存储器读取指令放入指令预取队列，每当队列有一部分空字节或者产生一次控制转移后，指令预取单元就会读取指令使指令预取队列得到补充。指令译码单元对指令进行译码，它从指令队列中取出指令并将其译为内部代码，再将这些代码送入译码指令队列中等待执行处理。两条流水线分别由运算器 ALU、一系列寄存器、地址生成电路和连接数据 Cache 的接口组成，每条流水线通过各自的接口对 Cache 存取数据，使 Pentium 具有更高的速度。控制单元控制两条流水线的运行。

Pentium 的整数运算流水线含有 5 个流水线级，分别为指令预取级、首次译码级、二次译码级、执行级和回写级。在指令预取级，通过指令预取单元取得指令；在首次译码级，对指令进行译码；在二次译码级，对指令操作数的地址进行计算；在执行级，执行指令规定的操作；而在回写级，则将指令运行的结果写入寄存器或存储器，同时根据指令运行的结果修改状态标志位。Pentium 的浮点运算流水线含有 8 个流水线级，分别为指令预取级、首次译码级、二次译码级、执行级、一次浮点操作级、二次浮点操作级、浮点运算结果写入级和出错级。

流水线运行时，一条接一条的指令连续不断地送到流水线，于是，在同一个时钟周期内，多个单元部件分别对多条指令的不同步骤进行操作。Pentium 在整数运算流水线中，每个步骤用一个时钟周期，所以一个时钟周期即可完成一条指令的全部操作。而非流水线方式，指令的执行过程包括取指令、指令译码和执行指令 3 个步骤。即先将指令从存储器取出送到指令寄存器；然后指令译码器对指令进行译码，分析指令要进行什么操作并指明操作数的地址；最后取出操作数，完成指令规定的操作，再把结果送到指定的地方。在非流水线方式下，执行一条指令所需要的时间就是这 3 个步骤所用时间的总和。在一条指令执行时，后面的指令不能有任何操作，只能处于排队等待的状态。流水线技术使得 Pentium 微处理器实现了指令

的高速执行。

（2）分支转移的动态预测。在程序设计中，循环程序和分支程序使用十分普遍，循环和分支是由分支转移指令实现的。通常，分支转移指令在执行前不能确定分支转移是否发生。而指令预取单元是顺序取指令的，如果产生转移，那么指令预取单元中取得的后续指令全部白取，且前几个流水线阶段的工作都要“作废”，必须清除并重新到转移地址处取指令，从而造成流水线断流，影响了流水线操作的效率。

Pentium 微处理器借助分支转移目标缓冲器 BTB（branch target buffer）等逻辑单元部件实现了分支转移的动态预测。BTB 含有 1KB 容量的 Cache，其中可以存放 256 条转移指令的目标地址和历史状态。历史状态用 2 位二进制数表示 4 种可能情况，即一定转移、可能转移、可能不转移和一定不转移。被预取的指令送到指令寄存器，同时将指令的地址送入 BTB 中进行查找。如果在 BTB 中没有该指令的地址，就不进行预测，继续预取指令。倘若在 BTB 中找到了这个地址，那么微处理器就将根据 BTB 中对应记录的历史状态来预测执行这条指令时是否发生转移，然后根据预测来预取指令。当 BTB 判断正确时，循环程序或分支程序会如同分支转移未发生一样，维持流水线的照常运行；当 BTB 判断错误时，则修改历史记录并重新取指令，清除流水线中的内容，重新建立流水线。总的来说，Pentium 微处理器的分支转移的动态预测功能大大加速了程序的执行。

（3）独立的指令 Cache 和数据 Cache。80486 微处理器片内只有 8KB Cache，Pentium 微处理器片内则有 16KB Cache，且将指令 Cache 与数据 Cache 完全分开，各为 8KB。这样就完全避免了预取指令与存取数据两者之间的冲突。而且指令 Cache 与数据 Cache 都有各自的转换检测缓冲器 TLB（translation lookaside buffer），由于有 TLB 的协助，分页单元部件就能迅速地将指令或数据的线性地址转换为物理地址。

（4）重新设计的浮点单元。Pentium 微处理器的浮点单元在 80486 的基础上做了重新设计，浮点单元的 U 流水线有 8 级流水，前 5 级与 V 流水线共享，后 3 级则有自己独立的浮点流水线，使每个时钟周期可以完成一个浮点操作。浮点单元内含专用的加法器、乘法器和除法器，加法器和乘法器均能在 3 个时钟周期内完成相应的运算，除法器则在每个时钟周期内产生 2 位二进制商。

1.5 80x86 的寄存器

1.5.1 8086/8088 的寄存器

Intel 8086/8088 的寄存器如图 1-24 所示。8086/8088 的寄存器有 8 个通用寄存器、1 个指令指示器、1 个标志寄存器和 4 个段寄存器。

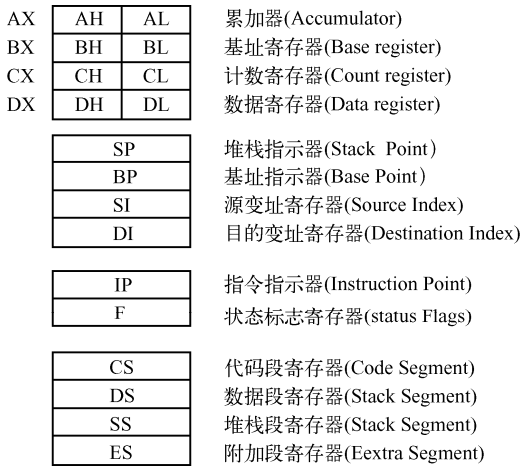

图 1-24 8086/8088 的寄存器

1．通用寄存器

通用寄存器是微处理器内部的存储器，如果一个微处理器中没有通用寄存器，那么在指

令执行过程中要用到操作数时，必须到存储器中去取，运算的结果（不是最后结果）也必须立即送到存储器中保存起来，而访问存储器的操作是比较费时间的。如果在微处理器中设置一些寄存器用来暂时存放参加运算的操作数和运算的结果（中间结果），则在程序执行过程中不必每时每刻都到存储器中去存取数据，就可以提高程序执行的速度。一般来说，微处理器中包含的通用寄存器越多，编程就越灵活，程序执行的速度就越快。通用寄存器就是这样一些快速的访问单元。

通用寄存器是 16 位的寄存器 AX、BX、CX、DX、SP、BP、SI 和 DI。在大多数情况下，这些通用寄存器都可以参与算术和逻辑操作。操作的结果存入参与操作的两个寄存器的一个之中。其中 AX、BX、CX、DX 均可以分成高 8 位和低 8 位两部分，可以分别作为独立的 8 位寄存器使用。所以 8086/8088 既可以处理 16 位二进制数，又可以处理 8 位二进制数。每个通用寄存器又各有某种专门的用途，所以对它们分别又有不同的称呼。称 AX 为累加器，BX 为基址寄存器，CX 为计数寄存器，DX 为数据寄存器，SP 为堆栈指示器，BP 为基址指示器，SI 和 DI 分别为源变址和目的变址寄存器。表 1-9 归纳了这些寄存器的专门用途。

表 1-9　通用寄存器的专门用途

寄　存　器	专 门 用　途
AX、AL	在乘法、除法指令中，作累加器；在输入、输出指令中，作数据寄存器
AH	在非压缩 BCD 数的调整指令中，作目的寄存器；在 LAHF（SAHF）指令中，作目的（源）寄存器
AL	在 BCD 数运算指令和调整指令中，作累加器；在 XLAT 指令中，作数据表的位移量
BX	作间址和基址寄存器
CX	在循环控制指令和串操作指令中，作计数器
CL	在移位指令中，作移位位数计数器
DX	在输入、输出指令中，作间址寄存器；在乘法、除法指令中，作辅助累加器
BP	作间址和基址寄存器
SP	作堆栈指示器
SI	作间址和变址寄存器；在串操作指令中，作源字符串的间址或变址寄存器
DI	作间址和变址寄存器；在串操作指令中，作目的字符串的间址或变址寄存器

2．指令指示器 IP（instruction point）

我们知道计算机所以能脱离人的直接干预，自动地进行计算或控制，这是由人把实现这个计算或控制的一步一步操作用命令的形式，即一条一条指令预先输入到存储器中，在执行时 CPU 把这些指令一条条地取出来，加以译码和执行。计算机所以能自动地一条一条地取出并执行指令，是因为 CPU 中有一个跟踪指令地址的电路，该电路就是指令指示器 IP。在开始执行程序时，给 IP 赋以第 1 条指令的地址；然后，每取一条指令 IP 的值就自动增量指向下一条指令的地址。

3．状态标志寄存器（status flags）

8086/8088 的状态标志寄存器有 9 个标志位，如图 1-25 所示。其中 6 个是状态标志，3 个是控制标志。

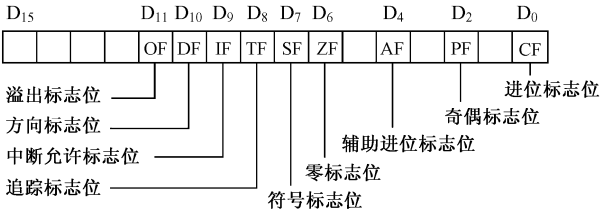

图 1-25　8086/8088 的标志寄存器

状态标志位反映了 EU 执行算术或逻辑运算以后的结果，执行有些指令可以改变某些状态标志的状态。6 个状态标志位如下：

（1）进位标志位 CF（carry flag）。加减算术指令执行后，最高位有进位或借位，CF=1；无进位或借位，CF=0。该标志主要用于多字节或多字数的加减运算指令。指令 STC 将其置 1，CLC 将其清 0，CMC 将其取反。

（2）辅助进位标志位 AF（auxiliary carry flag）。最低 4 位 D_3～D_0 位有进位或借位，AF=1；无进位或借位，AF=0。该标志用于 BCD 数的算术运算（调整）指令。

（3）溢出标志位 OF（overflow flag）。计算机所进行的运算均是无符号数运算，即把符号数的符号位也当数值进行运算，又把所有数的运算结果当符号数来影响标志位。即若指令执行后结果超出了机器数所能表示的数的范围（字节运算为：-128～127，字运算为：-32768～32767），OF=1；反之则 OF=0。该标志表示运算结果是否产生了溢出。

（4）符号标志位 SF（sign flag）。该标志表示结果的符号，其值与结果的符号位相同，即若结果为负数，SF=1；结果为正数，SF=0。

（5）零标志位 ZF（zero flag）。指令执行后，结果为 0，ZF=1；结果不为 0，ZF=0。

（6）奇偶标志位 PF（parity flag）。指令执行后，结果的低八位中 1 的个数为偶数，PF=1；若为奇数，PF=0。该标志可用来检查数据在传送过程中是否发生错误。

控制标志用于控制微处理器的操作，它们是：

（1）方向标志位 DF（direction flag）。该标志用来控制数据串操作指令的步进方向。若 DF=0，则数据串中操作指令自动增量地从低地址向高地址方向进行；若 DF=1，串操作的方向是从高地址向低地址方向进行。指令 CLD 将其清 0，指令 STD 将其置 1。

（2）中断允许标志位 IF（interrupt enable flag）。IF=1，允许 CPU 响应外部可屏蔽中断；IF=0，不允许 CPU 响应外部可屏蔽中断。允许中断又称为开中断，不允许中断又称关中断。指令 STI 将其置 1，指令 CLI 将其清 0。

（3）追踪标志位 TF（trap flag）。TF=1，微处理器每执行一条指令就自动地发生一个内部中断，微处理器转去执行一个中断程序，因而微处理器单步执行程序，常用于程序的调试，故又称其为陷阱标志位；TF=0，CPU 正常执行程序。

4．段寄存器

8086/8088 有 20 条地址线，存储器的物理地址必须用 20 位二进制数表示。可是它的 ALU 只能处理 16 位的地址运算，而且与地址有关的寄存器：指令指示器，堆栈指示器，间接寻址的寄存器 BX，BP，SI，DI 等都只有 16 位。因此 8086/8088 将存储器分为若干个段，把 20 位的地址分成 16 位的段基址和 16 位的偏移地址来表示。段基址就是段首址（即段的起始地址）的高 16 位地址，偏移地址即是存储单元所在段的起始地址到该存储单元的字节距离。

所以一个存储单元的地址由段基址和偏移地址两部分组成，用冒号连接段基址和偏移地址，即“段基址：偏移地址”。像这样表示的地址称为逻辑地址。

段寄存器就是用来存放段基址的寄存器。所以逻辑地址可以表示为：段寄存器名:偏移地址。8086/8088 微处理器有 4 个段寄存器，它们是代码段寄存器 CS（code segment），数据段寄存器 DS（data segment），堆栈段寄存器 SS（stack segment）和附加段寄存器 ES（extra segment）。它们分别用来存放代码段、数据段、堆栈段和附加段的段基址。4 个段寄存器的使用，使得在任意时刻，程序都可以仅通过偏移地址立即访问 4 个段中的存储器。8086/8088 微处理器自动根据偏移地址安排到代码段中去存取指令代码，到数据段中去存取数据，到堆栈段中进行进栈和出栈操作。

1.5.2 80286 的寄存器

80286 的通用寄存器、段寄存器和指令指示器与 8086/8088 完全一样，它们是 AX、BX、CX、DX、SP、BP、SI、DI、CS、DS、SS、ES 和 IP。不同之处在于新增加了 1 个机器状态字 MSW（machine status word）寄存器，标志寄存器 EFLAGS 新增加了两个标志位。MSW 是一个 16 位寄存器，只定义了它的低 4 位。标志寄存器中新增加的两个标志位，任务嵌套标志位 NT（nested task）和 I/O 特权级标志位 IOPL（I/O privilege level field）分别位于 D_{14} 和 D_{13}～D_{12} 3 位，其他 9 个位标志位的定义与位置均和 8086/8088 相同。

1.5.3 80386 的寄存器

80386 共有 2 类寄存器，它们是基本寄存器和系统寄存器。

1．基本寄存器

（1）通用寄存器。80386 将 8086 的 8 个 16 位的通用寄存器扩展为 32 位的通用寄存器，它们是累加器 EAX、基址寄存器 EBX，计数器 ECX，数据寄存器 EDX，堆栈指示器 ESP，基址指示器 EBP，源变址寄存器 ESI 和目的变址寄存器 EDI。它们的低 16 位就是 8086 的通用寄存器 AX、BX、CX、DX、SP、BP、SI、DI，其用法也与 8086 完全相同。

（2）指令指示器 EIP 和标志寄存器 EFLAGS。80386 的指令指示器 EIP 和标志寄存器 EFLAGS 都是 32 位的寄存器，它们的低 16 位即是 8086 的 IP 和 FLAGS，并可单独使用。在微处理器工作于保护方式下时，EIP 是 32 位的寄存器；在微处理器工作于实地址方式下时，指令指示器 EIP 就是 16 位的寄存器即 IP。80386 除了保留 80286 的所有标志外，在高位字的最低两位 D_{17} 和 D_{16} 增加了虚拟 8086 方式标志 VM（virtual 8086 mode）和恢复标志 RF（resume flag）两个标志位。

（3）段寄存器。80386 有 6 个 16 位段寄存器，它们是 CS、SS、DS、ES、FS 和 GS。其中 CS、SS、DS 和 ES 的作用与 8086 相同，而新增加的两个附加段寄存器 FS 和 GS 的作用与 ES 相同，都可以用来表示相应的数据段。在 80386 中存储单元的地址仍由段寄存器和偏移地址两部分组成，只是此时段基址与段寄存器的关系要由微处理器 80386 的工作方式确定。

2．系统寄存器

（1）系统地址寄存器。80386 的 4 个系统地址寄存器是全局描述符表寄存器 GDTR（Global

Descriptor Table Register）、中断描述符表寄存器 IDTR（Interrupt Descriptor Table Register）、局部描述符表寄存器 LDTR（Local Descriptor Table Register）和任务寄存器 TR（Task Register）。它们主要用来在保护模式下管理用于生成线性地址和物理地址的 4 个系统表。

（2）控制寄存器。80386 的 4 个控制寄存器是 CR_0～CR_3，CR_1 为备用。CR_0 的低位字节是机器状态字寄存器（MSW），与 80286 中的 MSW 寄存器相同。控制寄存器用来进行分页处理。

（3）调试寄存器。80386 的 8 个调试寄存器是 DR_0～DR_7，主要用来设置程序的断点。

（4）测试寄存器。80386 的 2 个测试寄存器是 TR_6 和 TR_7，也是用来进行页处理的寄存器。

1.5.4 80486 的寄存器

80486 的寄存器除了新增加 3 个测试寄存器 TR_3～TR_5 用于片内 Cache 测试以及浮点处理单元的浮点寄存器外，和 80386 的寄存器相同，不同之处是 80486 对标志寄存器的标志位进行了扩充。80486 除了保留 80386 的所有标志外，在 D_{18} 位增加了对准检查标志 AC（alignment check）。

1.5.5 Pentium 的寄存器

Pentium 的寄存器除了将控制寄存器增加到 5 个、测试寄存器增加到 18 个以及 FPU 部件外，其他寄存器和 80486 的寄存器相同，不同之处是 Pentium 对标志寄存器的标志位又进行了扩充。Pentium 共有 3 类寄存器，它们是基本寄存器、系统寄存器和浮点寄存器。

1．基本寄存器

Pentium 类似于 8086 的寄存器即基本寄存器包括通用寄存器、指令指示器、标志寄存器和段寄存器，如图 1-26 所示。

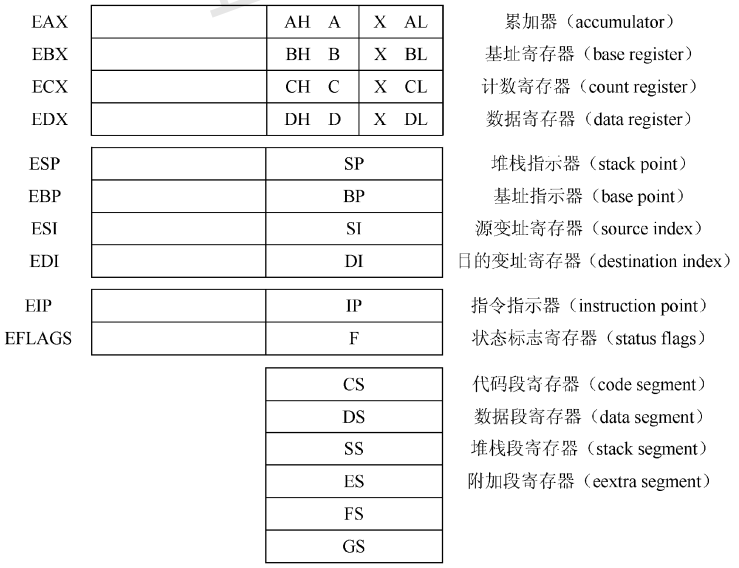

图 1-26　Pentium 的基本寄存器

（1）通用寄存器。Pentium 有 8 个 32 位的通用寄存器，它们是 8086 16 位通用寄存器的扩展，故命名为累加器 EAX。基址寄存器 EBX，计数寄存器 ECX，数据寄存器 EDX，堆栈指示器 ESP，基址指示器 EBP，源变址寄存器 ESI 和目的变址寄存器 EDI，它们的低 16 位可以作为 16 位寄存器使用，就是 8086 的 16 位通用寄存器 AX、BX、CX、DX、SP、BP、SI、DI，而 AX、BX、CX 和 DX 的低位字节（0～7 位）和高位字节（8～15 位），也同 8086 一样，都可以作为 8 位的寄存器单独使用，其命名仍为 AH、AL、BH、BL、CH、CL、DH 和 DL。

（2）指令指示器 EIP。Pentium 的指令指示器 EIP 是 32 位的寄存器，它用来存放下一条要执行指令的偏移地址。EIP 的低 16 位即是 8086 的指令指示器 IP，并可单独使用。在微处理器工作于保护方式下时，EIP 是 32 位的寄存器；在微处理器工作于实地址方式下时，EIP 就是 16 位的指令指示器 IP。

（3）标志寄存器 EFLAGS。Pentium 的标志寄存器 EFLAGS 也是 32 位的寄存器，Pentium 除了保留 80486 的所有标志外，在 D_{21}～D_{19} 增加了识别标志 ID（identifition）、虚拟中断挂起标志 VIP（virtual interrupt pending）、虚拟中断标志 VIF（virtual interrupt flag）3 个标志位。Pentium 微处理器的标志寄存器 EFLAGS 在 8086 微处理器的标志寄存器 FLAGS 基础上新增加了 8 个标志位，如图 1-27 所示。

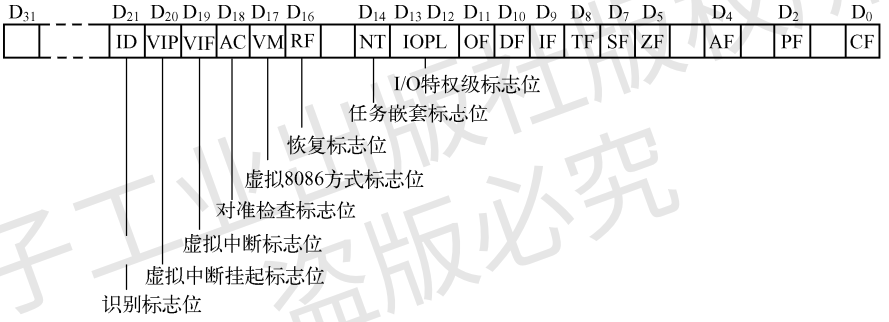

图 1-27　Pentium 的标志寄存器 EFLAGS

Pentium 微处理器的标志寄存器中新增加的 8 个标志位如下：

① 识别标志位 ID（identifition）。此位为 1，允许使用取 CPU 标识指令 CPUID 来读取 Pentium 微处理器的标识和相关信息。

② 虚拟中断挂起标志位 VIP（virtual interrupt pending）。此位为 1 表示在虚拟 8086 方式下有被挂起的中断未处理。

③ 虚拟中断标志位 VIF（virtual interrupt flag）。此位是在虚拟 8086 方式下，中断标志 IF 的虚拟映像。

④ 对准检查标志位 AC（alignment check）。在对字、双字或 4 字数据访问时，此位用来指出是否检查它们的起始字节的地址是否对准。字、双字和 4 字的起始字节的地址分别是 2、4 和 8 的倍数时，则是对准的。该位被置 1 且 CR0 寄存器的 AM 位也为 1 时，就进行对准检查，若进行非对准的字、双字或 4 字数据访问，则产生异常中断。

⑤ 虚拟 8086 方式标志位 VM（virtual 8086 mode）。此位被置 1 时，微处理器由虚地址保护方式切换到虚拟 8086 方式；该位被置 0 时，微处理器工作在虚地址保护方式。

⑥ 恢复标志位 RF（resume flag）。该标志位与调试寄存器配合使用，用来控制是否接受调试故障：该位为 1 忽略调试故障；该位为 0 接受调试故障。每执行完一条指令，此位自动清 0。当接受调试故障而调试失败后，微处理器将此位置 1，强迫程序恢复执行。

⑦ 任务嵌套标志位 NT（nested task）。该标志位用来控制中断返回指令的执行。该位被置 1 时，表示当前的任务被嵌套在另一个任务内，执行中断返回指令返回到前一个任务；该位被置 0 时，执行常规中断返回。

⑧ I/O 特权级标志位 IOPL（I/O privilege level field）。在虚地址保护方式下，用于控制对 I/O 地址空间的访问。该标志位用 2 位二进制数来定义允许执行的 I/O 指令的 I/O 特权级，分为 0～3 四级，0 级最高，3 级最低。如果当前任务的 I/O 特权级高于或等于 IOPL 中规定的级别，则可以执行该任务中的 I/O 指令，否则产生异常中断，执行任务的程序被挂起。

（4）段寄存器。Pentium 微处理器访问存储单元的逻辑地址仍由段寄存器和偏移地址两部分组成，只是此时段基址与段寄存器的关系要由微处理器 Pentium 的工作方式确定。在实地址方式，段寄存器用来存放段的段基址即段的起始地址高 16 位地址；在虚地址保护方式，段寄存器中存放的是选择字，选择字包含 13 位的描述符索引 DI（descriptor index）、1 位描述符表指示标志 TI（table indicator）和 2 位请求特权级 RPL（request privilege level），如图 1-28 所示。请求特权级 RPL 也称为选择字特权级，RPL 用来定义选择字的特权级，用于特权检查。如果段寄存器是 CS，则其中的 RPL 也叫当前特权级 CPL（current privilege level），表示当前任务的特权级别。特权级分为 0～3 级，0 级最高，3 级最低。TI 为 0 时，将从全局描述符表中检索描述符；TI 为 1 时，将从局部描述符表 LDT 中检索描述符。每个描述符包括 32 位的段首址、20 位的界限即段长度和 12 位的段属性，共 64 位即 8 个字节。段属性包括可读/写性、段的类型（系统段或非系统段，代码段、数据段还是堆栈段）、是否被访问过等，对于不同的段，其段属性的各数位定义有所不同。描述符索引 DI 用于检索描述符，从描述符表中检索描述符时，将描述符索引的值乘以 8 就得到描述符在描述符表中的偏移量，据此即可获得对应段的段首址。

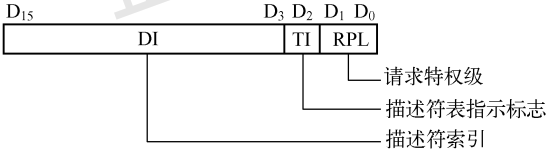

图 1-28　Pentium 在虚地址保护方式的段寄存器

Pentium 有 6 个 16 位段寄存器，它们是 CS、SS、DS、ES、FS 和 GS。在实地址方式，CS、SS、DS 和 ES 的作用与 8086 相同，分别用来存放代码段、堆栈段、数据段和附加数据段的段首址的高 16 位地址，而新增加的两个附加段寄存器 FS 和 GS 的作用与 ES 相同，都可以用来存放附加数据段的段首址的高 16 位地址；在虚地址保护方式，CS 中的描述符索引指向当前代码段对应的段描述符，SS 中的描述符索引指向当前堆栈段对应的段描述符，DS 中的描述符索引指向当前数据段对应的段描述符，ES、FS 和 GS 中的描述符索引指向当前 3 个附加数据段对应的段描述符，由此可以找到当前各个段的段首址。

2．系统寄存器

Pentium 的系统寄存器包括 4 个系统地址寄存器、5 个控制寄存器、8 个调试寄存器和 18

个测试寄存器。

（1）系统地址寄存器。Pentium 的 4 个系统地址寄存器是全局描述符表寄存器 GDTR（Global Descriptor Table Register）、中断描述符表寄存器 IDTR（Interrupt Descriptor Table Register）、局部描述符表寄存器 LDTR（Local Descriptor Table Register）和任务寄存器 TR（Task Register）。它们和段寄存器一起完成对存储器的管理。

Pentium 中有 3 种描述符表，它们是全局描述符表 GDT、中断描述符表 IDT 和局部描述符表 LDT。每个描述符表中，最多可含有 8192 个描述符，每个描述符对应一个存储段。一个系统中，全局描述符表 GDT 和中断描述符表 IDT 都只有一个，局部描述符表 LDT 可以有多个，每个 LDT 对应一个任务。IDT 和每个 LDT 各对应一个存储段，所以也各对应一个描述符存放在全局描述符表 GDT 中。

① GDTR。GDTR 是 48 位寄存器，其中高 32 位是全局描述符表 GDT 的线性基地址，低 16 位是 GDT 的界限（尺寸）。

② IDTR。IDTR 也是 48 位寄存器，其中高 32 位是中断描述符表 IDT 的线性基地址，低 16 位是 IDT 的界限（尺寸）。

③ LDTR。LDTR 是 16 位寄存器，用来存放描述符索引，据此可在全局描述符表 GDT 中检索到局部描述符表 LDT 对应的描述符。另外还有一个隐含的描述符高速缓冲器，用来存放检索到的 LDT 表的描述符。

④ TR。TR 是 16 位寄存器，用来存放描述符索引，据此可在全局描述符表 GDT 中检索到任务状态段 TSS 对应的描述符。微处理器为每个任务配置一个任务状态段 TSS，用来表示此任务的运行状态。与之相应，也有一个描述符高速缓冲器，用来存放检索到的任务状态段 TSS 的描述符。

（2）控制寄存器。Pentium 的 5 个控制寄存器是 CR_0～CR_4，CR_1 为备用。它们都是 32 位的寄存器，主要供操作系统使用，其内容可以被应用程序读取，但多数不允许应用程序改写。

① CR_0。CR0 用来保存系统的标志，这些标志表示微处理器的状态或者控制微处理器的工作方式，如图 1-29 所示。CR_0 的低位字是机器状态字 MSW（machine status word）。

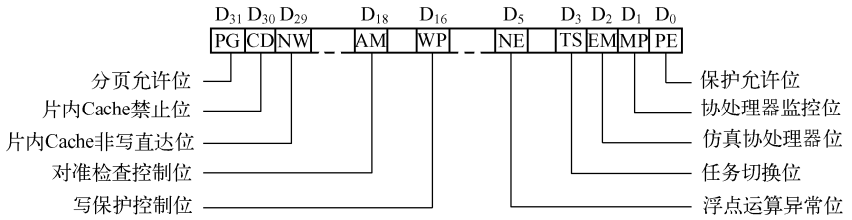

图 1-29　Pentium 的控制寄存器 CR_0

PE——保护允许位，当 PE=0 时，微处理器处在实地址方式；当 PE=1 时，微处理器处在虚地址保护方式。在微处理器复位时，PE=0，启动系统进入实地址方式。

MP——是协处理器监控位，EM 是仿真协处理器位，TS 是任务切换位，NE 是浮点运算异常位。这 4 位都与浮点单元的控制有关，本书不讲述浮点运算，故不介绍它们的具体定义。

WP——写保护控制位。该位为 1，对用户使用的页面进行写保护，即不允许系统程序对其进行修改。

AM——对准检查控制位。当标志寄存器的 AC 位为 1 此位也为 1 时，在访问存储器时则进行对准检查。

NW——片内 Cache 非写直达位。当 NW=1 时，数据仅写入片内 Cache；当 NW=0 时，数据既写入片内 Cache 同时还写入内存。

CD——片内 Cache 禁止位。该位为 0 时，允许访问片内 Cache；否则禁止访问。

PG——分页允许位。当 PG=1 时，微处理器工作在虚地址保护方式下允许分页；当 PG=0 时，禁止分页。

② CR_2 和 CR_3。CR_2 和 CR_3 是两个用于存储器管理的地址寄存器。在分页操作时，如果出现异常，CR_2 中则会保存异常处的 32 位线性地址。CR_3 的前 20 位保存着页目录表的基地址，CR_3 的 D_3 位和 D_4 位用来对外部 Cache 进行控制。

（3）CR_4：CR_4 只用了最低 7 位，其他位都为 0，如图 1-30 所示。所用位的定义如下。

VME——虚拟 8086 方式允许位。该位为 1，允许虚拟 8086 方式；为 0，则禁止虚拟 8086 方式。

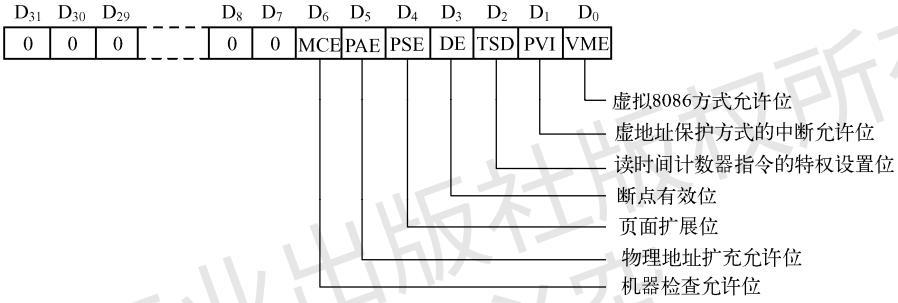

图 1-30　Pentium 的控制寄存器 CR4

PVI——虚地址保护方式的中断允许位。在虚地址保护方式下，该位为 1，允许中断；为 0，则禁止中断。

TSD——读时间计数器指令的特权设置位。仅当此位为 1 时，读时间计数器指令 RDTSC 才能作为特权指令在任何时候执行；否则该指令只能在系统级执行。

DE——断点有效位。该位为 1，允许设置断点；否则禁止设置断点。

PSE——页面扩展位。该位为 1，页面尺寸为 4MB；为 0，页面尺寸为 4KB。

PAE——物理地址扩充允许位。该位为 1，允许按 36 位物理地址运行分页机制；为 0，则按 32 位物理地址运行分页机制.

MCE——机器检查允许位。该位为 1，允许机器检查异常。

（4）调试寄存器。Pentium 的 8 个调试寄存器 DR_0～DR_7 主要用来设置程序的断点和程序调试。

DR_0～DR_3 是用来保存 4 个断点的线性地址的寄存器。在程序的调试过程中，可一次性设置 4 个断点。程序运行时，当程序执行到与断点的线性地址一致处，便会停下来，并显示当前各个寄存器的状态，以便程序调试人员进行分析。

DR_4 和 DR_5 是 Intel 公司保留的。

DR_6 是调试状态寄存器，在调试过程中用来报告断点处的状况。其中 4 位分别表示 4 个断点的调试状态，进入调试状态时为 1，退出调试状态时为 0。还有 3 位分别表示各种操作

状态。

DR_7 是配合断点设置的断点控制寄存器，用来设置控制标志，控制断点的设置、设置条件、断点地址的有效范围以及是否进入异常中断等。

（5）测试寄存器。Pentium 有 18 个测试寄存器，用寄存器号 00H～14H 来表示，其中有 3 个号未使用。每个测试寄存器有一个特定的测试功能。Pentium 有专用的读指令 RDMSR 写指令 WRMSR 来访问这些测试寄存器。要先将寄存器号放入 ECX 中，然后执行指令访问。00H 和 01H 号是 64 位的寄存器，读或写的内容在 EDX 和 EAX 中，其余都是 32 位的寄存器，读或写的内容在 EAX 中。

3．浮点寄存器

Pentium 的浮点寄存器包括数据寄存器、标记字寄存器、状态寄存器、控制字寄存器、指令指示器和数据指示器。

（1）数据寄存器。数据寄存器有 8 个，它们是 R_0～R_7，每个寄存器有 80 位，80 位的浮点数中 1 位为符号位、15 位为阶码、64 位为尾数。

（2）标记字寄存器。标记字寄存器是 1 个 16 位的寄存器，每 2 位为 1 个标记，共 8 个标记，分别指示 8 个数据寄存器的状态。最高 2 位是数据寄存器 R_7 的标记，……，最低 2 位是数据寄存器 R_0 的标记。通过标记字来表示相应的数据寄存器是否为空，这种功能可以使浮点处理单元更加简洁地对数据寄存器做检测。

（3）状态寄存器。状态寄存器是 1 个 16 位的寄存器，用来指示浮点处理单元的当前状态，如图 1-31 所示。

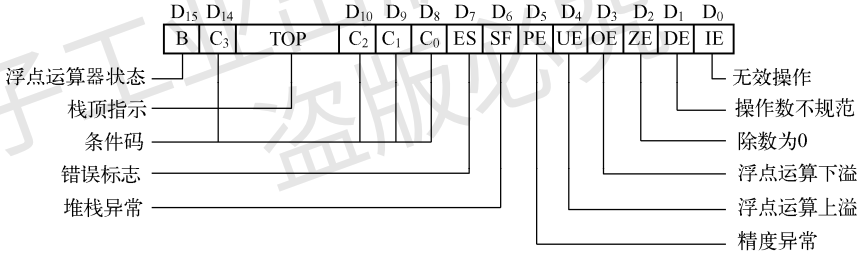

图 1-31　浮点处理单元的状态寄存器

① IE 为 1，表示无效操作，这是非法操作引起的故障。

② DE 为 1，表示操作数不符合规范引起的故障。

③ ZE 为 1，表示除数为 0 引起的故障。

④ OE 和 UE 为 1，分别表示为浮点运算下溢和上溢引起的故障。

⑤ PE 为 1，表示运算结果不符合精度要求。

⑥ SF 是堆栈故障标志。当 IE=1 且 SF=1 时，若条件码 C1=1，则表示堆栈上溢引起无效操作；若 C1=0，则表示堆栈下溢引起无效操作。

⑦ ES 为错误标志。上述 6 个故障中任何一个故障都会使 ES=1，且使 Pentium 的浮点运算出错信号线 $\overline{\text{FERR}}$ 为低电平。

⑧ C3～C0 被称为条件码。除了条件码 C1 和堆栈故障标志 SF 一起表示堆栈状态外，这几个条件码可以用指令 SAHF 进行设置，用指令 FSTSW AX 读取，然后以此为条件实现某种

操作。

⑨ TOP 为栈顶指示器。

⑩ B 为 1，表示浮点运算器处于忙状态。

（4）控制字寄存器。控制字寄存器的低 6 位分别用来对上述 6 种故障进行屏蔽，这些屏蔽位和状态寄存器的标志位一一对应，如图 1-32 所示。另外两个 2 位的控制位 PC 和 RC 分别是精度控制和舍入控制。

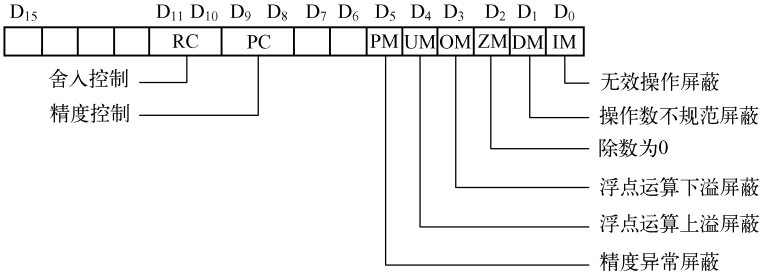

图 1-32　浮点处理单元的控制字寄存器

（5）指令指示器和数据指示器。这 2 个寄存器用来存放发生故障的指令的地址及其操作数的存储地址。

1.6　80x86 的工作方式与存储器物理地址的生成

1.6.1　80x86 的工作方式

80x86 的工作方式有实地址方式（real address mode）、虚地址保护方式 （protected virtual address mode）、虚拟 8086 方式（virtual 8086 mode）和系统管理方式（system management mode）4 种。

8086/8088 只有实地址方式 1 种工作方式。80286 有实地址方式和虚地址保护方式 2 种工作方式。80386 和 80486 有实地址方式、虚地址保护方式和虚拟 8086 方式 3 种工作方式。Pentium 除了实地址方式、虚地址保护方式和虚拟 8086 方式 3 种方式外，还增加了一种系统管理方式。

1．实地址方式

实地址方式是实在的 1MB 的物理地址空间的工作方式，实地址方式采用存储器地址分段的方法，使两个 16 位的地址实现了对 1MB 地址空间寻址的 20 位的物理地址。在实地址方式下，操作数的默认长度为 16 位，可以运行 8086 的全部指令。80x86 除了虚地址保护方式指令外，其余指令都可以在实地址方式下运行。8086 和 80286 微处理器允许 4 种存储器分段，段寄存器为 CS、DS、SS 和 ES。80386 以上微处理器允许 6 种存储器分段，段寄存器为 CS、DS、SS、ES、FS 和 GS。

2．虚地址保护方式

虚地址保护方式是支持虚拟存储器、支持多任务、支持特权级与特权保护的工作方式。在虚地址保护方式下，32 位微处理器可访问的物理空间为 4GB（2^{32} 字节），由辅存和内存提

供的虚拟空间可达 64TB（2^{46} 字节）。该方式对如此之大的虚拟存储空间采取保护措施，使系统程序和用户的任务程序之间以及各任务程序之间互不干扰地运行。最主要的保护措施是特权级和特权保护。特权级（privilege level）分为 4 级，由 2 位二进制数组成，特权级编号为 0～3，其中 0 级为最高特权级，3 级为最低特权级。每个存储段都同一个特权级相联系，只有足够级别的程序才可以对相应的段进行访问。在程序运行的过程中，通过 CPL、DPL 和 RPL 三个特权级来实施特权级保护。

CPL（current privilege level）是当前特权级，它既是代码段寄存器 CS 的最低 2 位的值，也是当前代码段的 DPL 的值，用来表示当前正在运行的程序的特权级。

DPL（descriptor privilege level）是描述符特权级，每个段的段描述符中都有 2 位 DPL 来标明此段的特权级。只有当 CPL 等于或高于 DPL 时，当前任务才能访问描述符所确定的段中的数据。

RPL（requrstor privilege level）是请求特权级，它位于数据段寄存器的最低 2 位，用来防止特权级低的程序访问特权级较高的数据段。

3．虚拟 8086 方式

虚拟 8086 方式是一种在 32 位虚地址保护方式下支持 16 位实地址方式应用程序运行的特殊工作方式。微处理器的工作过程与虚地址保护方式下的工作过程相同，但程序指定的逻辑地址又与 8086 实地址方式相同。在这种方式下操作系统可以建立多个 8086 虚拟机，每个虚拟机都认为自己是唯一运行的机器，安全地运行以实地址方式编写的 16 位应用程序。虚拟 8086 方式是具有最低特权级（特权级为 3）的保护方式。当标志寄存器的 VM 位为 1 时，微处理器进入虚拟 8086 方式。

4．系统管理方式

系统管理方式主要为系统管理而设置。该方式可使系统设计人员实现高级管理功能，例如，对电源实施管理，对操作系统和正在运行的程序实施管理，提供透明的安全性。系统管理方式由计算机内部的硬件（装有系统程序代码的 ROM）来控制。

5．4 种工作方式之间的转换

80x86 处理器在实地址方式、虚地址保护方式、虚拟 8086 方式和系统管理方式 4 种方式之间的转换关系如图 1-33 所示。在系统上电或复位之后，微处理器首先进入实地址方式。控制寄存器 CR0（80286 为机器状态字寄存器 MSW）的保护允许标志位 PE 控制微处理器是工作在实地址方式还是工作在虚地址保护方式；标志寄存器 EFLAGS 的虚拟 8086 方式标志位 VM 决定微处理器是工作在虚地址保护方式还是工作在虚拟 8086 方式。

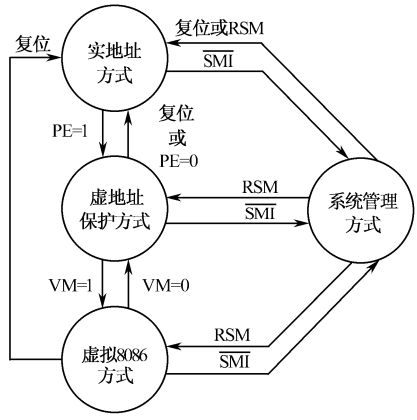

图 1-33　80x86 微处理器 4 种工作方式之间的转换关系图

一旦 Pentium 微处理器收到系统管理中断（系统管理中断引线 $\overline{\text{SMI}}$ 有效）请求，无论 Pentium 微处理器工作在实地址方式、虚地址保护方式还是虚

拟 8086 方式，便立即转换到系统管理方式。在系统管理方式中，执行从系统管理方式返回指令 RSM（Resume from System Management Mode），Pentium 微处理器便恢复保存的内容，返回到进入系统管理方式之前的工作方式。

1.6.2 80x86 存储器的分段和物理地址的生成

1．实地址方式下的存储器分段

实地址方式使用 20 条地址线寻址 1MB 存储器。8086/8088 仅有 20 条地址线 A_{19}～A_0；在实地址方式中，80286 使用 24 条地址线中的 20 条 A_{19}～A，80386、80486 和 Pentium 也仅使用 32 条地址线中的 20 条 A_{19}～A_0。

20 条地址线可以寻址多达 2^{20}（1M）字节，所以把 1M 字节的存储器分为任意数量的段，其中每一段最多可寻址 2^{16} （64K）字节，这样，每一个段就必须开始于一个能被 16 整除的地址（即该地址的最低 4 位为全 0）。段的段基址即起始地址的高 16 位地址和偏移地址一样都是 16 位无符号二进制整数，其值可为 0000H～FFFFH，故可以将存储器分为 64K 个段。存储器的分段并不是唯一的，它们可以相互重叠。对于一个具体的存储单元来说，它可以属于一个逻辑段，也可以同时属于几个逻辑段，如图 1-34 所示，地址 00000H～0FFFFH 为一个段，地址 00010H～1000FH 为一个段，……，地址 F0000H～FFFFFH 为一个段。00020H 单元既属于 00000H～0FFFFH 段，又属于 00010H～1000FH 段，同时还属于 00020H～1001FH 段。

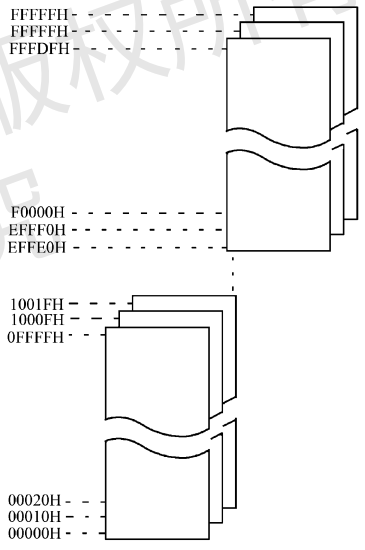

图 1-34　实地址方式下存储器段的划分

存储器中的每个存储单元都可以用两个形式的地址来表示：实际地址（或称物理地址）和逻辑地址。物理地址是用唯一的 20 位二进制数所表示的地址，微处理器与存储器交换信息时使用物理地址。程序中不能使用物理地址，而要使用逻辑地址，即段基址：偏移地址。一个物理地址可以用不同的逻辑地址表示。如图 1-34 中的物理地址 00020H，在 00000H～0FFFFH 段中的逻辑地址是 0000H：0020H，在 00010H～1000FH 段中的逻辑地址就是 0001H：0010H，而在 00020H～1001FH 段中的逻辑地址却是 0002H：0000H。

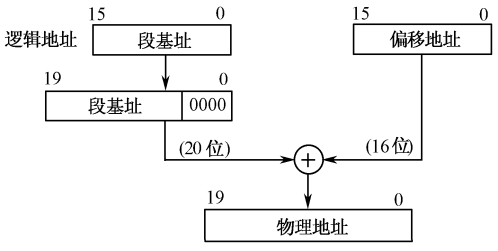

图 1-35　8086 物理地址的生成

2．实地址方式下物理地址的生成

在实地址方式下微处理器将段寄存器提供的段基址自动乘以 10H 即将 16 位二进制数逻辑左移 4 位得到 20 位的段首址，然后与 16 位的偏移地址相加得到 20 位的物理地址，并锁存在物理地址锁存器中，如图 1-35 所示。如逻辑地址 0001H：0010H 生成物理地址时，将段首址的高 16 位 0001H 左移 4 位得到段首址 00010H，再与偏移地

址 0010H 相加即可得到物理地址 00020H。

每次需要生成物理地址的时候，一个段寄存器会自动被选择，且能自动将其中的 16 位二进制数左移 4 位，再与一个 16 位的偏移地址相加，产生所需要的 20 位物理地址。

实地址方式下有 4 个段寄存器 CS、DS、SS、ES 用来存放段基址即段首址的高 16 位地址，还有 6 个 16 位的寄存器（IP、SI、DI、BX、BP、SP）用来存放偏移地址，在寻址时到底应该使用哪个段寄存器是微处理器根据执行操作的要求来确定的。若取指令，则由代码段寄存器 CS 给出段首址的高 16 位地址，指令寄存器 IP 给出要取指令的偏移地址。执行堆栈操作，被寻址的操作数的段首址的高 16 位地址和偏移地址由堆栈段寄存器和堆栈指示器给出。若是采用间址存取数据，段首址的高 16 位地址一般是由 DS 给出，偏移地址由间址寄存器 SI、DI 或 BX 给出。

在不改变段寄存器值的情况下寻址的最大范围是 64K，不可能寻址这个段以外的其他存储单元，要想超出这个段寻址就必须要改变这个段寄存器的值。若有一个任务，它的程序段，堆栈段以及数据段都不超过 64K，则在程序开始时分别给 CS、SS、DS 赋值，然后在程序中就可以不再考虑这些段寄存器，程序就可以在各自的区域中正常地工作。若某一任务所需要的存储器空间不超过 64K，则可以在程序开始时使 CS、SS、DS 相等，完全由 IP、SP 和各种存储器寻址方式确定的有效地址 EA（effective address）来确定存储器的地址。

3．虚地址保护方式下 80286 的存储器的分段和物理地址的生成

在虚地址保护方式中，80286 可产生 24 位物理地址，直接寻址能力为 16MB。和实地址方式一样，80286 将寻址空间分成若干段，一个段最大为 64KB，逻辑地址也是由两部分组成的：段寄存器的值和偏移地址。但在虚地址保护方式下的段首址是 24 位而不是实地址方式下的 20 位。而段内的偏移地址与实地址方式相同，是由各种存储器寻址方式所确定的 16 位。80286 的段寄存器是 16 位的，如何通过段寄存器中的 16 位二进制数得到 24 位的段首址呢？

在虚地址保护方式下，80286 的段寄存器不再存放段基址，而是存放一个指针，该指针称为描述符索引 DI。段首址存放在段描述符表中，通过 DI 检索该表即可得到段首址。把程序中可能用到的各种段（如代码段、数据段、附加段、堆栈段）的段首址、段长度和段属性等信息（称之为描述符）集合在一起形成一张表，称为描述符表，存放在内存的某一区域。80286 的每个描述符由 6 个字节组成，其中有 3 个字节为段首址，DI（实际上只使用了 14 位，高 2 位为 0）指向描述符的起始位置。80286 的地址转换机构根据 DI 的值找出描述符中的 24 位段首址，再与偏移地址相加，就得到 24 位物理地址，如图 1-36 所示。

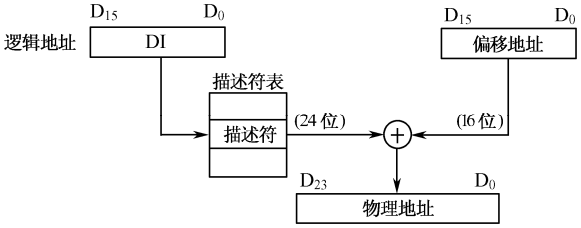

图 1-36　80286 在虚地址保护方式下物理地址的生成

由于描述符索引 DI 有 14 位，因此可以定义 2^{14} 个描述符；而对应各描述符可定义 2^{16}（64K）字节的段，所以 80286 的逻辑地址寻址能力为 $2^{14}\times2^{16}$B=1000MB（1GB）的存储空间。但 80286 的实际内存最多只有 16MB，容纳不下这么大的存储空间，所以只能将其置于辅助存储器（硬盘）上。实际工作时，将当前需要的段调入内存，用过的段返回辅助存储器，这一切都是系统自动管理的。因此，虽然系统只有 16MB 内存，但对用户来说，好像在使用 1GB 内存，于是这个 1GB 内存称之为虚拟内存。

4．虚地址保护方式下 80386、80486 和 Pentium 的存储器的分段和物理地址的生成

80386、80486 和 Pentium 工作在虚地址保护方式时，可产生 32 位物理地址，直接寻址能力为 4GB（2^{32}B）。和 80286 一样，80386、80486 和 Pentium 的逻辑地址也是由描述符索引 DI 和偏移地址两部分组成的，DI 也只用了 13 位，偏移地址不是 16 位而是 32 位。因此 80386、80486 和 Pentium 的逻辑地址可达 2^{14} 个段（13 位描述符索引 DI 和 1 位 TI），每个段的长度可达 2^{32}B＝4GB，80386、80486 和 Pentium 的虚拟内存为 $2^{14}\times2^{32}$B＝2^{46}B＝64TB，在 80286 中虚拟内存的单位是段，80286 的段最大为 64KB，在磁盘与内存之间进行调度是可行的，但当段的长度达到 4GB 就不合适了。为此在 80386、80486 和 Pentium 中将 4GB 空间以 4KB 为一页分成 1G 个等长的页面，并以页面为单位在磁盘与内存之间进行调度。

由于 80386、80486 和 Pentium 将段进行了分页处理，所以 80386、80486 和 Pentium 要经过两次转换才能得到物理地址。80386、80486 和 Pentium 的每个描述符由 8 个字节组成，其中有 4 个字节的段首址，还有 20 位的段长度和 12 位的段属性（其中有 2 位描述符特权级 DPL）。第 1 次为段转换，由段管理单元根据描述符索引 DI 的值找出描述符中的 32 位段首址，再与偏移地址相加将逻辑地址转换为线性地址；第 2 次为页转换，由页管理单元将线性地址转换为物理地址，80386、80486 和 Pentium 的物理地址生成如图 1-37 所示。从图 1-37 可以看到，如果禁止分页功能，线性地址就等于物理地址，如果进行分页处理，线性地址就不同于物理地址。

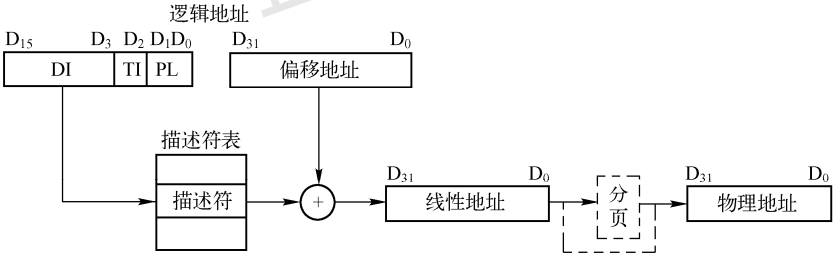

图 1-37　80386、80486 和 Pentium 在虚地址保护方式下物理地址的生成

5．虚拟 8086 方式物理地址的生成

虚拟 8086 方式是在虚地址保护方式下，能够在多任务系统中执行 8086 任务的工作方式。当 80x86 工作在虚拟 8086 方式时，所寻址的物理内存是 1MB，段寄存器的内容不再是描述符表的描述符索引而是段基址。将段寄存器的内容乘以 16（左移 4 位）就是 20 位的段首址，与偏移地址相加形成 20 位的线性地址，线性地址再经过页管理单元的分页处理，就可得到 20 位的物理地址。

习 题 1

1.1 将下列十进制数转换为十六进制数：10，64，78，80，93，100，125，255

1.2 将下列十六进制无符号数转换为十进制数：2CH，64H，D5H，100H ，378H，4FEH ，CADH

1.3 写出表 1-10 中十进制数的原码和补码，用 2 位或 4 位或 8 位十六进制数填入表 1-7 中。

表 1-10

十进制数	原码	补码	十进制数	原码	补码
18			928		
–18			–928		
30			8796		
–30			–8796		
347			65530		
–347			–65530		

1.4 用十进制数写出下列补码表示的机器数的真值：1BH，71H，80H，F8H，397DH，7AEBH，9350H，CF42H

1.5 用补码运算完成下列算式，并指出运算结果是否产生了溢出：

（1）33H+5AH （2）-29H-5DH （3）65H-3EH （4）4CH-68H

1.6 将 8 位无符号数 AAH 扩展为 16 位应为_____；将 8 位原码数 BBH 扩展为 16 位应为_____；将 8 位补码数 88H 扩展为 16 位应为_____。

1.7 将下列各组二进制数进行“与”运算。

（1）DAH∧99H （2）BAH∧56H （3）95H∧FFH

1.8 将下列各组二进制数进行“或”运算。

（1）DAH∨99H （2）F0H∨5AH （3）C6H∨45H

1.9 将下列各组二进制数进行“异或”运算。

（1）86H⊕5AH （2）BCH⊕AAH （3）DAH⊕99H

1.10 将表 1-11 中的十进制数按表中要求转换后用十六进制数填入表 1-8 中。

表 1-11

十进制数	压缩 BCD 数	非压缩 BCD 数	ASCII 码
38			
97			
105			
255			
483			
764			
1000			
1025			

1.11 下列 4 组十六进制数是十六进制数的 ASCII 码：4435H，3745H，32303030H，38413543H，请写

出各组数的十六进制数，并将其看做字节补码数或者字补码数转换为十进制数。

1.12　某 8 位移位寄存器已装入数 15H，令其补 0 左移 3 次，试用十六进制数写出移位后的结果。

1.13　8086/8088 的标志寄存器 FLAG 中包括哪几个标志位?各位的状态含义及用途如何?

1.14　8086/8088 和 80x86 各有哪些寄存器?如何分组?各有什么用途?

1.15　保护方式下的段寄存器和实地址方式下的段寄存器在组成上、内容上和使用上有何不同？在保护方式下设计了选择字这一新的数据结构，它有什么作用？

1.16　在实地址方式中，存储器的物理地址由哪两部分组成?是如何生成的?每个段与寄存器之间有何对应的要求?

1.17　在实地址方式中，设 CS=0914H，共有 243 字节长的代码段，该代码段末地址的逻辑地址和物理地址各是多少?

1.18　在实地址方式系统中，有一数据段装入内存后，若 DS=095FH 时，物理地址是 11820H。当 DS=2F5FH 时，物理地址为多少?

1.19　80486 微处理器有哪 3 种工作方式？简述各种工作方式的特点和区别。

1.20　80486 微处理器的 3 种工作方式的物理地址空间各有多大？