1.4　微型计算机运算基础

1.4.1　计算机中数的表示方法

1.4.2　计算机中的编码

1.4.3　计算机中的运算

重点解决：计算机的重要职能之一**处理数**

在计算机中如何表示一个数?

不同性质数的运算规则和算法。

1.4.1 计算机中数的表示方法

1. 几个重要概念
2. 复习不同进制数之间的互换
3. 机器数与真值
4. 带符号数的原码、反码、补码
5. 数的定点与浮点表示

1 几个重要概念

重点概念1：

计算机中的数据都是以二进制形式进行存储和运算的

重点概念2：

在计算机中存储数据时，每类数据占据固定长度的二进制数位，而不管其实际长度。一般长度为字节的整倍数

例如：在八位微机中，

整数216 存储为11011000B

整数56 存储为00111000B

重点概念3：

计算机中不仅要处理无符号数，还要处理带符号和带小数点的数。

重点概念4： 机器数与真值

2 不同进制数之间的互换

1、不同进制数转换成十进制数——按权展开法

表示不同进制数的尾部字母：

二 B ， 十六 H ，八 Q ，十 D(可略)

例：10101010B

$=1\times2^7+0\times2^6+1\times2^5+0\times2^4+1\times2^3+0\times2^2+1\times2^1+0\times2^0$

$=128+32+8+2=170$

2、十 $\longrightarrow$ 二

（1）整数部分——除以2取余法——直到商为0止

（2）小数部分——乘以2取整法——直到积为0止

或达到精度要求止

例： 100= B= H = Q

0.625= B= H= Q

（1） 十↔二

1）整数 十→二 （除2取余法）

217 → 108 → 54 → 27 → 13 → 6 → 3 → 1 → 0 ----商

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

1 0 0 1 1 0 1 1 ——余数

$(217)_{10}=(11011001)_2$

书写方向

结论：整数除2取余，直到商为0为止，

余数：按照相反的方向写下来。读数由后向前。

适用于数值比较小的情况。

2）小数十→二: 乘2取整

方法：对十进制数逐次乘2，取小数点前边系数。

【例2】将十进制小数（0.8125）$_{10}$转换为二进制小数，采用“乘2顺取整”的方法，过程如下：

$0.8125\times2=1.625$ 取整数位1

$0.625\times2=1.25$ 取整数位1

$0.25\times2=0.5$ 取整数位0

$0.5\times2=1.0$ 取整数位1

所以，$(0.8125)_{10}=(0.1101)_2$

如果出现乘积的小数部分一直不为“0”，则可以根据精度的要求截取一定的位数即可。

3)、二 $\longrightarrow$ 十　　同样可以用公式进行

——按权展开法

$(0.1001)_2 = 1\times 2^{-1}+1\times 2^{-4}=0.5+0.0625=(0.5625)_{10}$

$(0.100111)_2=1\times 2^{-1}+1\times 2^{-4}+1\times 2^{-5}+1\times 2^{-6}$

$=(0.609375)_{10}$

3、二进制数、八进制与十六进制数之间的互换

1）二 → 八　三合一

2）八 → 二　一分三

3）二 → 十六　四合一　（重点）

4）十六 → 二　一分四　（重点）

例：0111 0110 B=76H　　9BH=1001 1011B

7　6　　1001　1011

例：0.1010 110 B= 0.ACH

A　C　　不足四位补0

问：01110110B= ？Q

0.1010110B= ？Q

4、 用权表示数（2^n 2^{n-1} 2^n-1 $2^{n-1}-1$）

1） 权

n位二进制数各位的权从高位到低位依次为：

n位二进制数：$B_{n-1}B_{n-2}B_{n-3}$ …………… B_1B_0

权： $2^{n-1}2^{n-2}2^{n-3}$ ……… 2^12^0

2）用权表示数

例：11111……1111B = 2^n-1，即n个1。

01111……1111B = $2^{n-1}-1$ ，即n-1个1

最高位的权为：2^{n-1}

n=32

?

N=64

?

例：n=8,11111111B=FFH=2^8-1

01111111B=7FH=$2^{8-1}-1$

例： n位二进制数表示无符号数的范围： 0～2^n-1

n=8 0 ～2^8-1 0 ～ 255

n=16 0 ～ $2^{16}-1$ 0～65535

关于数制的小结：

(1)每一种数制都有一个固定的基数“J ”

十进制 $J = 10$,有0 ~ 9 十个不同的数值

二进制 $J = 2$, 有0 ,1两个不同的数值

十六进制$J = 16$, 有0 ~ 9 A,B,C,D,E,F十六个不同的数值

(2)各种数制都是逢“J”进位。

(3)各种数制每位的**权**：以小数点分界，

$$\ldots\ldots J^4, J^3, J^2, J^1, J^0 \;.\; J^{-1}, J^{-2}, J^{-3}, \ldots\ldots$$

$$\ldots\ldots 10^4, 10^3, 10^2, 10^1, 10^0 \;.\; 10^{-1}, 10^{-2}, 10^{-3}, \ldots\ldots$$

$$\ldots\ldots 2^4, 2^3, 2^2, 2^1, 2^0 \;.\; 2^{-1}, 2^{-2}, 2^{-3}, \ldots\ldots$$

$$\ldots\ldots 16^4, 16^3, 16^2, 16^1, 16^0 \;.\; 16^{-1}, 16^{-2}, 16^{-3}, \ldots\ldots$$

5、 数据单位

b：位（bit）是计算机的最小基本数据单位；

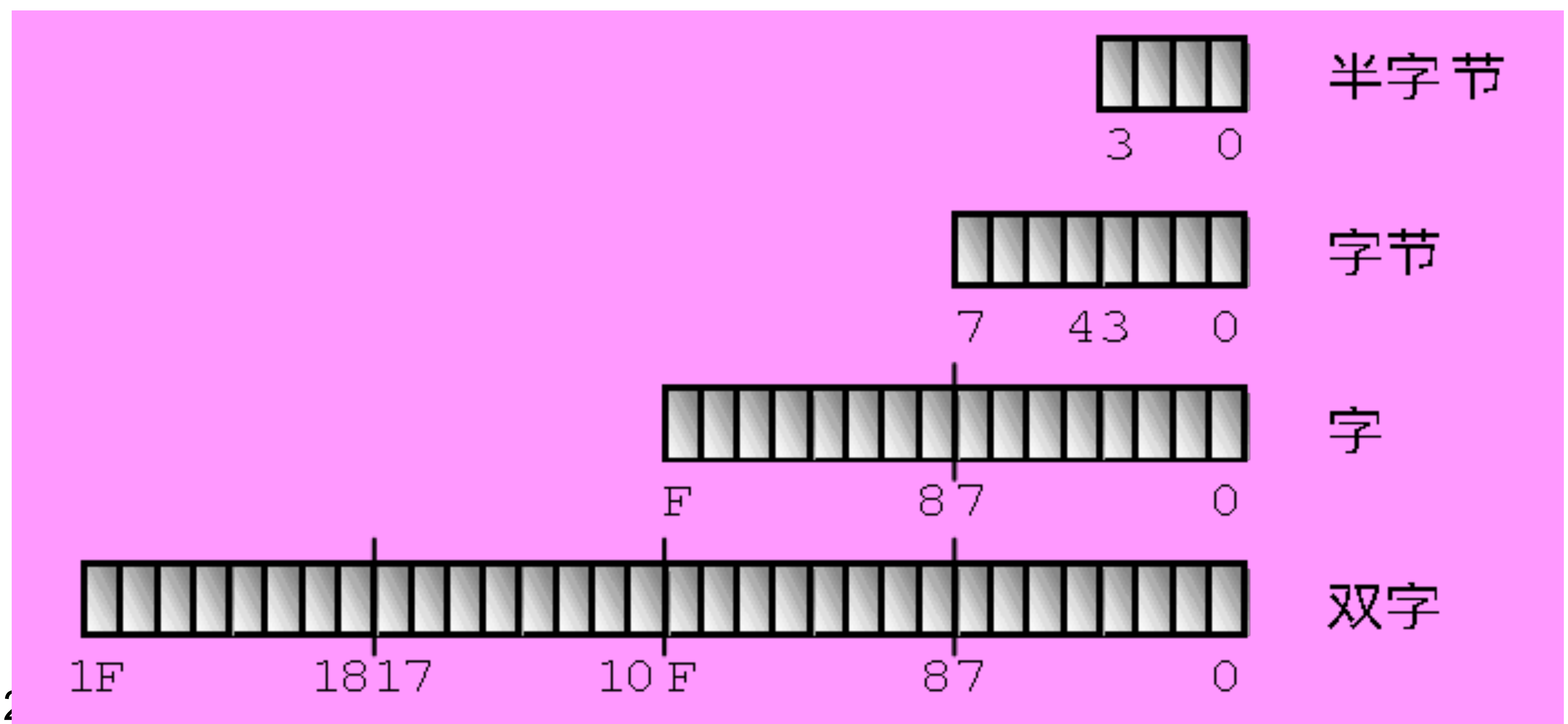
B：字节（byte）由8个二进制位组成，1B=8b;

K （Kilo） ： $1K=1024=2^{10}$；

M （Mega） ： $1M=1024K=2^{20}$；

G （Giga） ： $1G=1024M=2^{30}$；

T （ Tera ） ： $1T=1024G=2^{40}$。



3. 机器数与真值

1）机器数：能被计算机识别的数称为机器数。

2）真值：　机器数所代表的真实值称为机器数的真值。

3）对于无符号数其机器数与真值表示方法相同。

例：真值：　100=64H=01100100B

对应的机器数：64H=01100100B

n位二进制数可表示的数的范围是：$0 \sim 2^n-1$

8位二进制数可表示的数的范围是：

$0 \sim 2^8-1$，[0，FFH],[0,255]

16位二进制数可表示的数的范围是：

$0 \sim 2^{16}-1$,[0,FFFFH],[0,65535]

例：01100100B 其8位全部为数值位。

特点：无符号数的机器数与其真值为等值关系

4） 带符号数的机器数的表示方法（重点和难点）

常见的有原码、反码和补码三种表示方式。

特点：带符号数的机器数与其真值表示方法不同，两者的关系不是等值关系，仅是一一对应关系。

0	1	0	0	0	0	0	1

符号位　　　数值位

符号位：

“0” 表示正号

“1” 表示负号

例如：在八位微机中，

真值：＋65可表示成机器数（原码）为01000001B

真值：－65可表示成机器数（原码）为11000001B

[例]：

$$+52 = \overbrace{+0110100}^{\text{真值}} = \overbrace{\underbrace{0}_{\text{符号位}}\quad \underbrace{0110100}_{\text{数值位}}}^{\text{机器数}}$$

$$-52 = -0110100 = \underline{1}\quad \underline{0110100}$$

4 带符号数的原码、反码、补码 ★

（1）原码

定义： 在表示带符号数时，正数的符号位为“0”，负数的符号位为“1”，数值位表示数的绝对值，这样就得到了数的原码。

例如在八位微机中：

$$[+38]_{原}=[+100110]_{原}=00100110B$$

$$[-38]_{原}=[-100110]_{原}=10100110B$$

计算公式：对于字长为n位的机器数：

当真值$X \geq 0$时，X可表示为$+X_{n-2}X_{n-3}\ldots X_0$；

当真值$X<0$时，X可表示为$-X_{n-2}X_{n-3}\ldots X_0$，

则X的原码可定义为：

$$[X]_{原}=\begin{cases} 0X_{n-2}X_{n-3}\ldots X_0=X & 0\leq X\leq 2^{n-1}-1 \\ 1X_{n-2}X_{n-3}\ldots X_0=2^{n-1}-X=2^{n-1}+|X| & -(2^{n-1}-1)\leq X\leq 0 \end{cases}$$

可见n位原码可表示数的范围为：

$$-(2^{n-1}-1)\sim+(2^{n-1}-1)$$

则在八位微机中，原码可表示数的范围为-127至$+127$

求真值：带符号数的原码表示法简单易懂，而且与真值转换方便。

原码的缺点：

- “0”的原码有两种形式，这在运算中非常不方便。

  $[+0]_{原}$ = 00000000B

  $[-0]_{原}$ = 10000000B，即分为+0和−0

- 原码在进行两个异符号数相加或两个同符号数相减时，需做减法运算，由于微机中一般只有加法器而无减法器，所以，为了把减法运算转变为加法运算就引入了反码和补码。

原码的用途：

- 原码做乘除法运算方便，两数的符号和数值分别处理

  积的符号为两数符号位的异或运算结果

  积的数值部分为两数绝对值相乘的结果

（2）反码

定义： 正数的反码表示与原码相同；负数的反码，可将负数原码的符号位保持不变、数值位按位取反得到，或者将负数看作正数求原码，再将所有位按位取反得到。因此，在n位机器数的计算机中，数X的反码定义为：

$$[X]_{反}=\begin{cases}0X_{n-2}X_{n-3}\ldots X_0=X & 0\le X\le 2^{n-1}-1\\ 1\overline{X}_{n-2}\overline{X}_{n-3}\ldots\overline{X}_0=11\ldots1B-|X|=2^n-1-|X| & -(2^{n-1}-1)\le X\le 0\end{cases}$$

n位反码表示数的范围与原码相同，

八位二进制反码表示的范围仍是－127至＋127。

缺点： “0”的反码也有两种表示法，即＋0和－0。

$[+0]_{反}=00000000B$

$[-0]_{反}=11111111B$

例如八位微机中：

$[+11]_{原}=00001011B$	$[+11]_{反}=00001011B$
$[-11]_{原}=10001011B$	$[-11]_{反}=11110100B$
$[-38]_{原}=10100110B$	$[-38]_{反}=11011001B$
$[+127]_{原}=01111111B$	$[+127]_{反}=01111111B$
$[-127]_{原}=11111111B$	$[-127]_{反}=10000000B$
$[+0]_{原}=00000000B$	$[+0]_{反}=00000000B$
$[-0]_{原}=10000000B$	$[-0]_{反}=11111111B$

求真值： 由反码求得原码，再由原码求得真值，即可得到反码的真值。

例如：反码11011001B，符号位为1，将数值位按位取反，得到原码10100110B，其真值为－0100110B

即十进制数－38。

（3）补码（难点）

定义： 正数的补码表示与原码相同

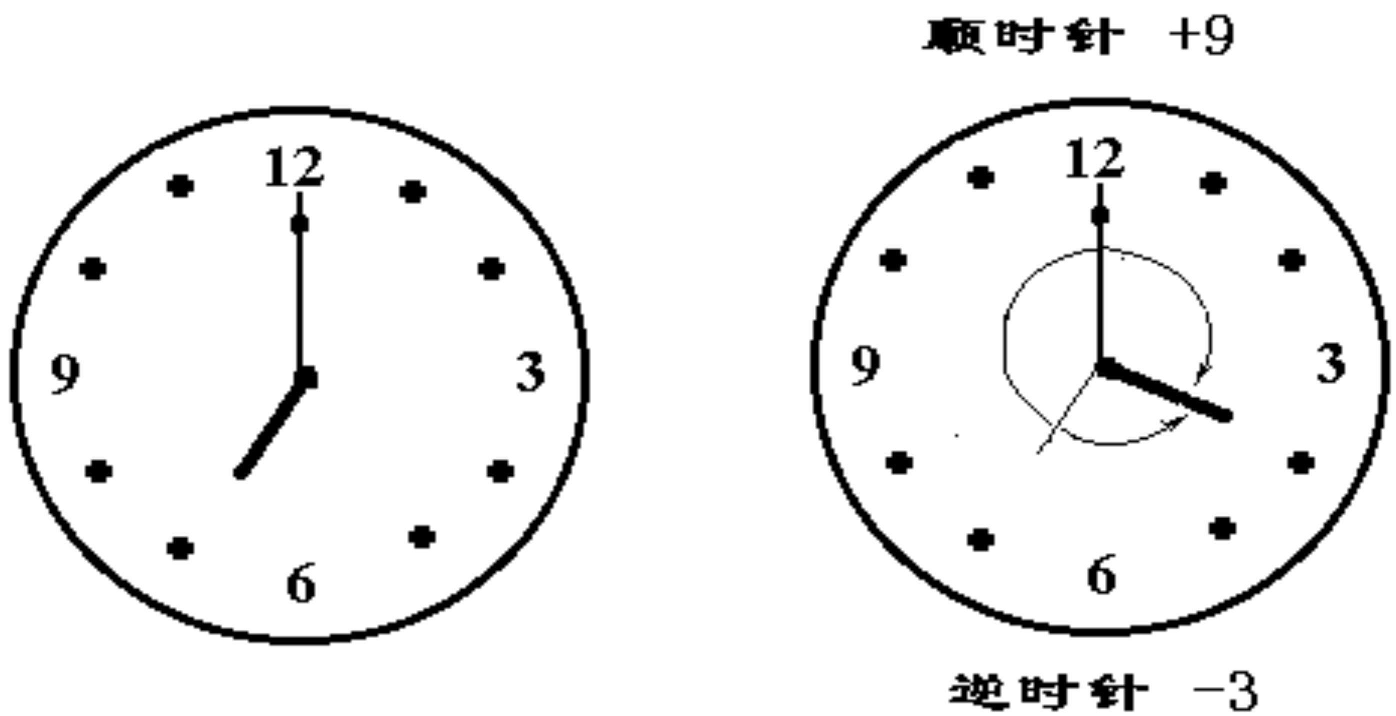
负数的补码等于它的反码末位加1

即$[X]_{补}=[X]_{反}+1$　　例如：

$[+11]_{原}=00001011B$	$[+11]_{反}=00001011B$	$[+11]_{补}=00001011B$
$[-11]_{原}=10001011B$	$[-11]_{反}=11110100B$	$[-11]_{补}=11110101B$
$[+127]_{原}=01111111B$	$[+127]_{反}=01111111B$	$[+127]_{补}=01111111B$
$[-127]_{原}=11111111B$	$[-127]_{反}=10000000B$	$[-127]_{补}=10000001B$
$[+0]_{原}=00000000B$	$[+0]_{反}=00000000B$	$[0]_{补}=00000000B$
$[-0]_{原}=10000000B$	$[-0]_{反}=11111111B$	$[-128]_{补}=10000000B$

补码的含义:

以时钟对时为例来说明，现由7点钟调到4点钟。



顺时针调： 7＋9 ＝4 （mod 12）

逆时针调： 7－3 ＝ 4 （mod 12）

9 称为 —3 对模 12 的补数。

由于时钟上超过12点时就会自动丢失一个数12，这个自动丢失的数叫做“模”（module，简写为mod）

由补码的定义得求补码公式：$(\bmod 2^n)$

$$[X]_{补}=\begin{cases}0X_{n-2}X_{n-3}\ldots X_0=X & 0\le X\le 2^{n-1}-1\\ 1\overline{X}_{n-2}\overline{X}_{n-3}\ldots\overline{X}_0+1=2^n-|X|=2^n+X & -2^{n-1}\le X<0\end{cases}$$

- 则n位补码表示数的范围为：$-2^{n-1}\sim+\ (2^{n-1}-1)$
- 八位二进制补码表示的数值范围是－128至＋127。

优点：0的补码为00000000B，只有这一种形式。

已知补码求真值：

☏ 已知正数的补码求真值

与原码相同，只要将符号位的0变为＋（正号），即得到它的真值。

☏ 已知负数的补码求真值

方法1：将负数补码的数值位按位取反再加1，将符号位的1变为－（负号），即得到它的真值。

方法2：用公式：$X=-(2^n-[X]_{补})$

已知 补码为 01111111B,其真值为+1111111B=+7FH

已知 补码为 11111111B,其真值为:

10000000B+1= 10000001B,其真值为—01H

或：X= —（2^8 — 11111111B)= —(00H-FFH)= —1

小结：已知带符号数的机器数求真值

1\. 已知正数的原码、反码、补码求真值，

只需将符号位的“0”改为正号“+”即可。

2\. 已知负数的原码，其真值只需将原码的符号位的“1”改为负号“-”即可。

3\. 已知负数的反码，先将它变为原码，再求真值。

或用公式计算： 真值 $x=-(2^n-1-[x]_{反})$

4\. 已知负数的补码，数值位取反加1，符号为改为-号，或用公式：$X=-(2^n-[X]_{补})$

例：已知带符号数的机器数为 56H，求其真值。

真值=+56H？

例：已知带符号数的机器数为 0D6H，求其真值。

- 若0D6H是原码，则真值为：-56H

  11010110B -1010110B

- 若0D6H是反码，则真值为：-29H

  -(0FFH-0D6H)

- 若0D6H是补码，则真值为：-2AH

  -(00H-0D6H)

当n=8时，几种码的表示范围		
	原码	－127至＋127
	反码	－127至＋127
	补码	－128至＋127

当n=16时，几种码的表示范围		
	原码	－32767至＋32767
	反码	－32767至＋32767
	补码	－32768至＋32767

5　数的定点与浮点表示

计算机中如何表示实数中的小数点呢?

计算机中不用专门的器件表示小数点，而是用数的两种不同的表示法来表示小数点的位置。

根据小数点的位置是否固定，数的表示方法分为定点表示和浮点表示, 相应的机器数称为定点数和浮点数。

任意一个二进制数N均可表示为：

$$N = S \cdot 2^J$$

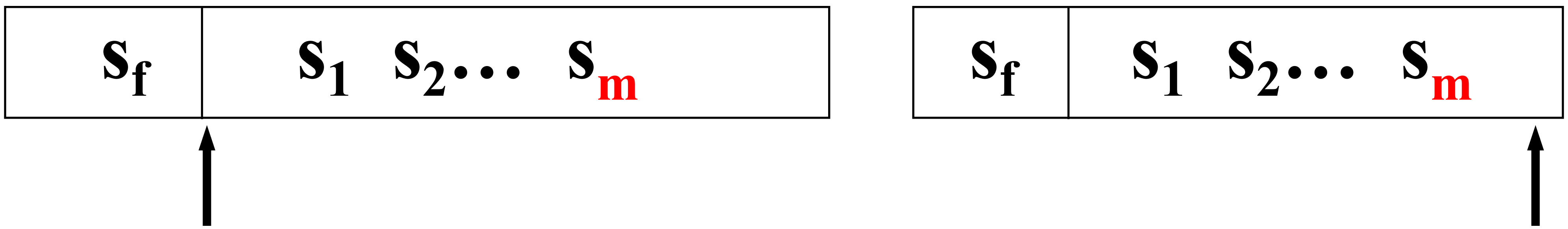
其中：

S称为数N的尾数，表示数N的全部有效数字，决定了N的精度。

J称为数N的阶码，底为2，指明了小数点的位置，决定了数N的大小范围。

（1）定点表示法

计算机在处理定点数时，常把小数点固定在**数值位**的最后面或最前面，即分为定点纯小数与定点纯整数两类，如图1-6所示。



小数点隐含位置，**定点纯小数**

小数点隐含位置，**定点纯整数**

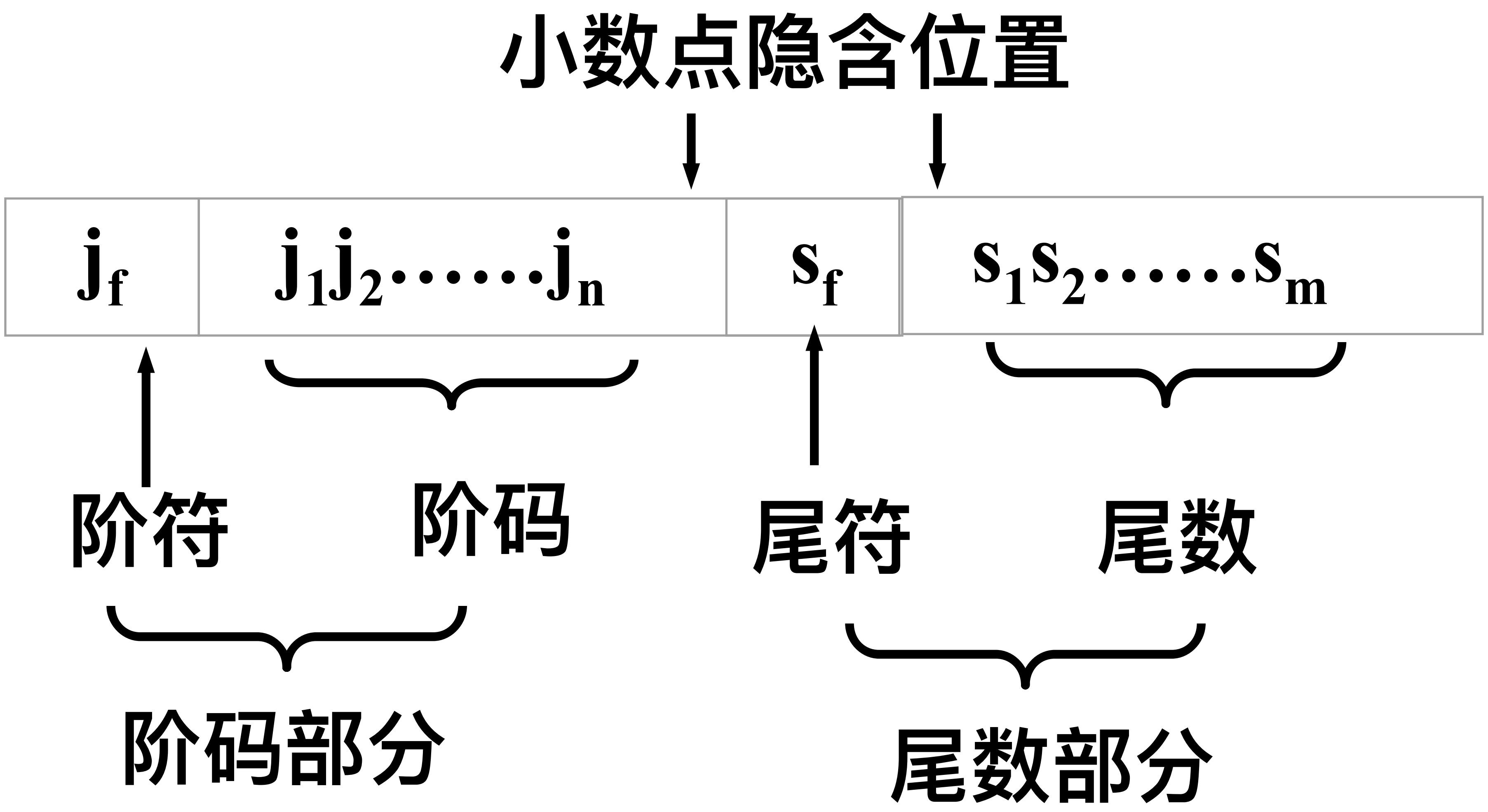
例如：

00011000B，如果看作定点纯整数，其真值为24

看作定点纯小数，其真值为0.1875

（2）浮点表示法

在浮点表示法中，小数点的位置是浮动的，阶码J可取不同的数值，则在计算机中除了要表示尾码S，还要表示阶码J。因此，一个浮点数表示为阶码和尾数两部分，**尾数一般是定点纯小数**，**阶码是定点纯整数**，其形式如下图所示。



例如，某计算机用32位表示浮点数，**尾数部分占24，为补码定点纯小数**；**阶码为8位补码定点纯整数**。用来表示一个数－469.375,先进行变换：

$$(-469.375)_{10}=(-111010101.011)_2$$
$$=(-0.111010101011)_2\times 2^{+9}$$
$$=(-0.111010101011)_2\times 2^{+1001B}$$

$$[-0.111010101011]_{补}=100010101010100000000000B$$
$$[+1001B]_{补}=00001001B$$

因此，数－469.375在该计算机中的浮点表示为：

$$\underbrace{0}_{阶符}\,\underbrace{0001001}_{阶码}\,\underbrace{1}_{尾符}\,\underbrace{00010101010100000000000B}_{尾数\,(24位)}$$

1.4.2 计算机中的编码

ASCII码： 由七位二进制编码组成，
共有128个字符编码。
包括图形字符（字母、数字、其它可见字符共96个）和控制字符（回车、空格等共32个）
其中 数字0~9的ASCII码为30H~39H，差30H
字母A~F的ASCII码为41H~46H，差37H
D7位加奇偶校验位：
无校验 D7位补0
奇校验 D7位使含1的个数为奇数个
偶校验 D7位使含1的个数为偶数个
例： 30H 00110000B D7补0为无校验和偶校验

ASCII 码是一种七位二进制编码，包括数字符号、英文字母、标点符号、控制字符等。

“ A ”	100 0001	(七位ASCII码)
	<u>0</u>100 0001	(带偶校验的8位编码)
	<u>1</u>100 0001	(带奇校验的8位编码)
“ 8 ”	011 1000	(七位ASCII码)
	<u>1</u>011 1000	(带偶校验的8位编码)
	0011 1000	(带奇校验的8位编码)

BCD编码：具有十进制位权的二进制编码。最常见的是8421码。

注意：

0000B~1001B是0~9的BCD码

1010B~1111B**是非BCD码**

例：

15 的BCD码为0001 0101B=15H

15=0FH

100=64H

100的BCD码为0001 0000 0000B=100H

存储方式

压缩的BCD码 56H 占一个存储单元

非压缩BCD码 05H 06H 占两个单元

1.4.3 计算机中的运算

计算机中的运算分为两类:

- ☏逻辑运算：逻辑"与"、"或"、"非"、"异或"等
- ☏算术运算：加、减、乘、除运算

逻辑运算

1、与　　3、非

2、或　　4、异或

算术运算

1. 带符号数补码运算及判OV
2. BCD码加/减法及十进制调整
3、算术运算小结

加/减运算电路

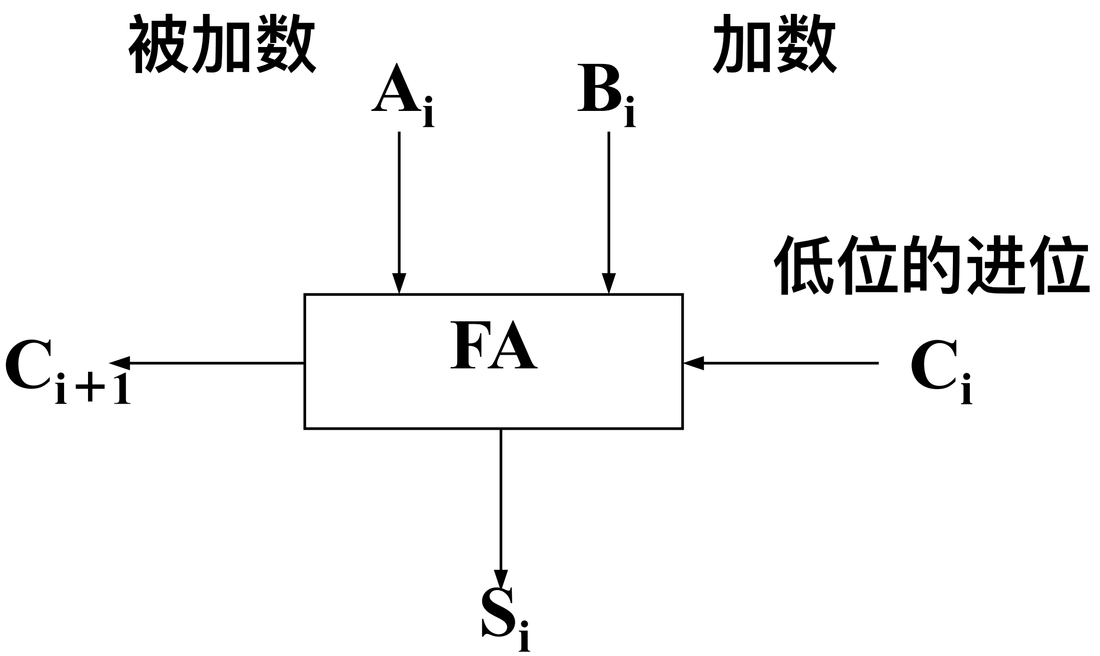


图1-8 全加器符号图

全加器真值表

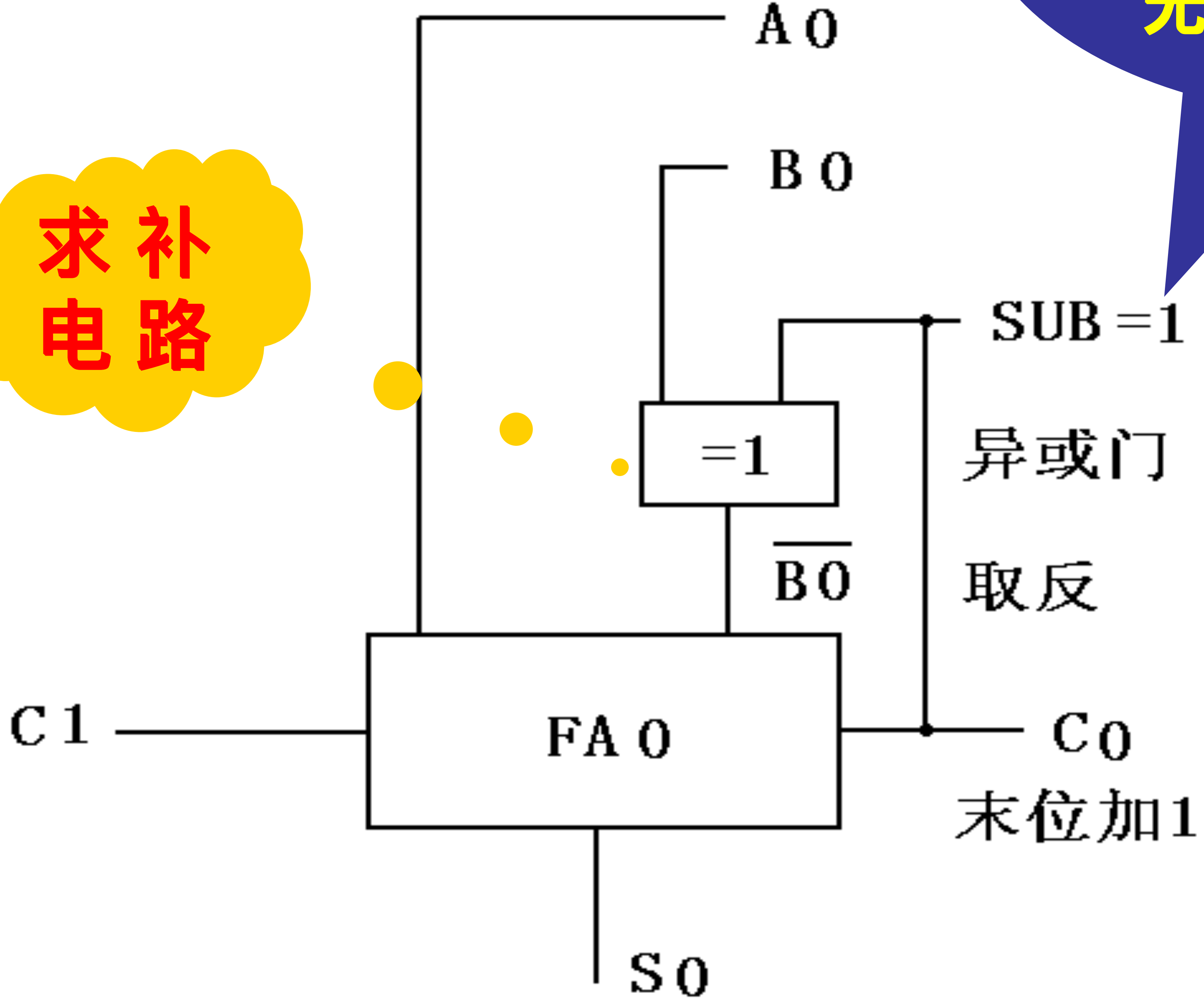
Ai	Bi	Ci	Si	Ci+1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

减法的实现

减法时SUB=1，
有取反加1功能
加法时SUB=0
无取反加1功能

加法电路+求补电路

求补
电路



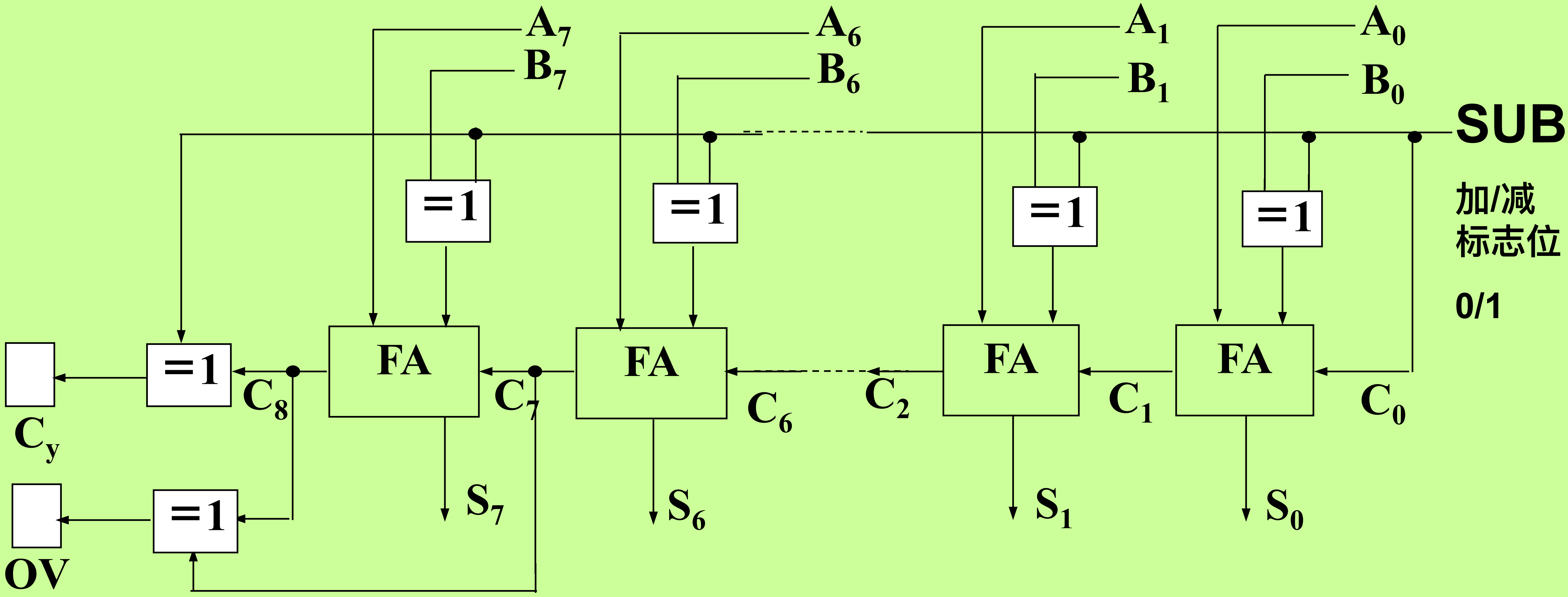


图1-9 八位微机加/减运算电路

进/借位标志CY=SUB $\oplus$C8

SUB	C8	CY
0	0	0
0	1	1
1	0	1
1	1	0

溢出标志OV=C7 $\oplus$C8

C7	C8	OV
0	0	0
0	1	1(负)
1	0	1(正)
1	1	0

二进制无符号数的四则运算

（1）加法运算

二进制加法法则为：

0＋0＝0

1＋0＝0＋1＝1

1＋1＝10

1＋1＋1＝11

例：二进制无符号数加法

1、求 187+22

	被加数	10111011B
+	加 数	00010110B
	进 位	00111110
	和	11010001B

结果：11010001B 即209

SUB=0，C8=0，CY=0

CY=SUB ⊕ C8

2、求200+200

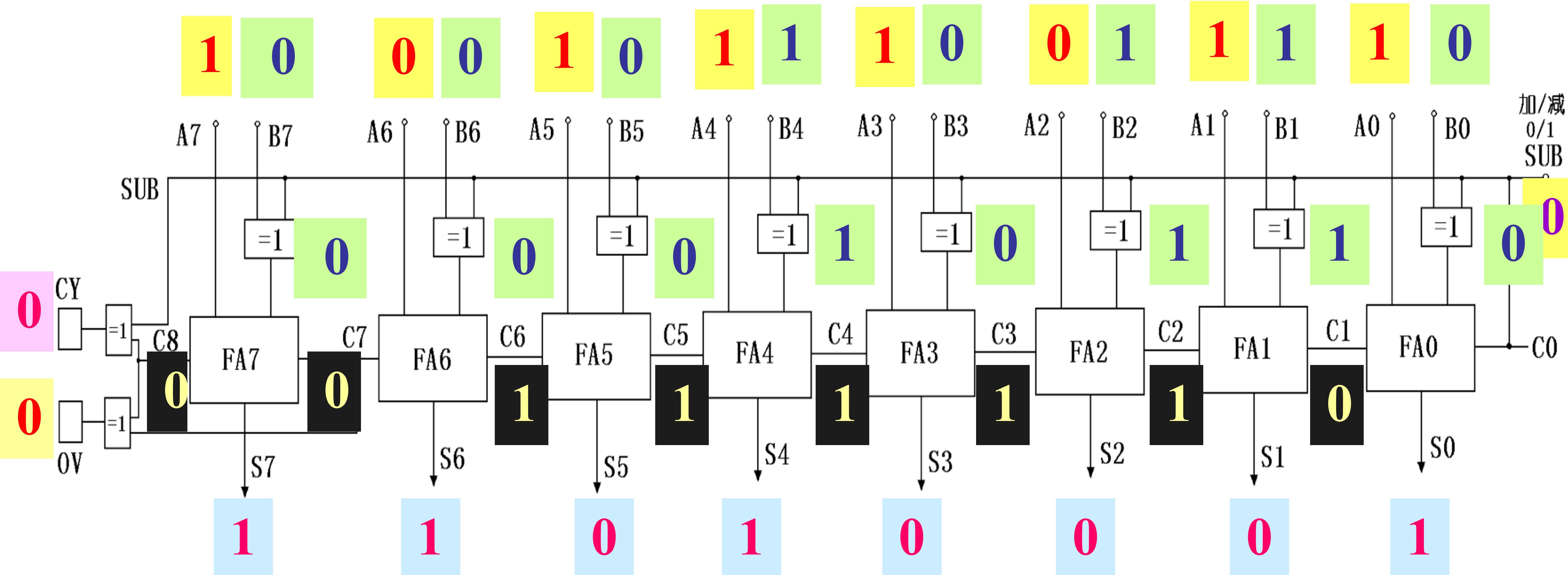
	被加数	11001000B
+	加 数	11001000B
	进 位	11001000
	和	110010000B

结果：SUB=0，C8=1，CY=1

和=进位值+8位和值

=256+10010000B

= 400



被加数　10111011B

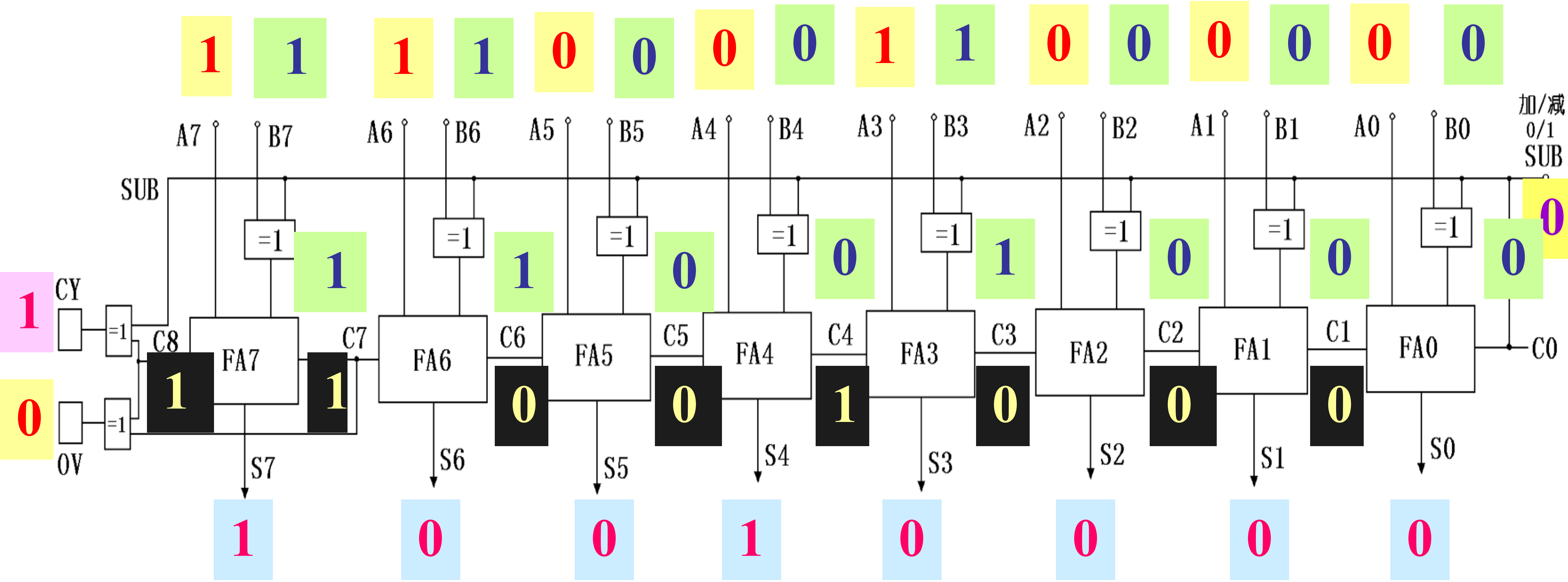
＋加　数　00010110B

进　位　00111110

和　11010001B

结果：11010001B 即209

SUB=0，C8=0，CY=0



	被加数	11001000B
+	加　数	11001000B
	进　位	11001000
	和	110010000B

结果：SUB=0，C8=1，CY=1

和=进位值+8位和值

=256+10010000B

= 400

（2） 减法运算法则：

0 − 0 = 0

1 − 0 = 1

1 − 1 = 0

0 − 1 = 1 （借1当2）

例：二进制无符号数减法

例：求 187-22

手算：

被减数	10111011B
−减 数	00010110B
借 位	00000100
差	10100101B

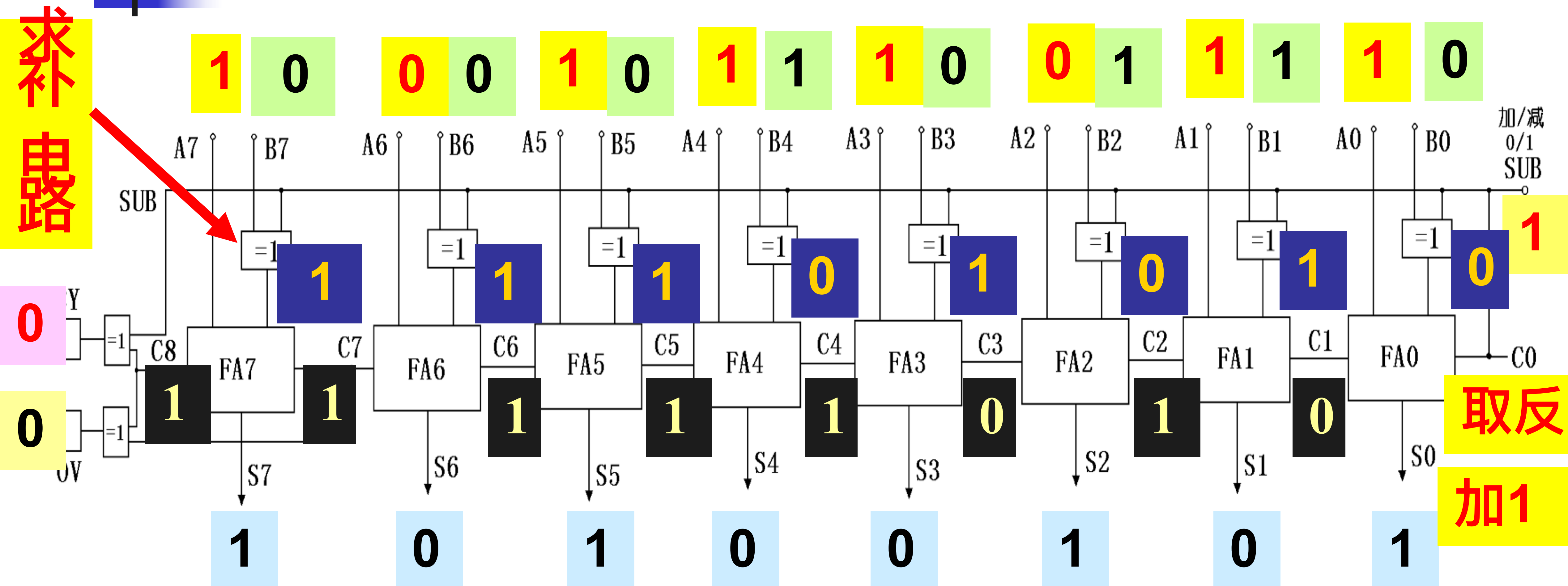
结果：无借位，差为10100101B 即165

机器算：

难点

减法 SUB=1

被减数 10111011B －减 数 00010110B



借位标志CY=SUB ⊕ C8=1 ⊕1=0

对减数求补后，加被减数 ★

例：求187-22

	被减数	10111011B=BBH
－	减数	00010110B=16H
		11101001B 取反
	＋	1 加1
＋		11101010B =EAH
	进 位	11111010
	差	10100101B =A5H

求补

结果：10100101B 即165 无借位，SUB=1， C8=1 ， CY=0

说明：直接相减无借位，求补相加有进位，反之亦然。 ★

（3） 乘法运算法则

0×0=0

0×1=1×0=0

1×1=1

```
被乘数          1001B
乘数    ×       1011B
        ---------------
                1001
               1001
              0000
        +    1001
        ---------------
乘积         1100011B
```

常用算法：

用部分积右移一位来代替被乘数左移一位，使被乘数固定在原位置，则n位全加器就可以实现乘法运算。

（4）定点整数除法运算

```
            1 1 1 0 0
      ─────────────────
101  ) 1 0 0 0 1 1 0 0
       1 0 1
       ─────
         1 1 1
         1 0 1
         ─────
           1 0 1
           1 0 1
           ─────
               0
```

常用算法：

1、移位相减法

2、连减

1. 带符号数定点补码运算及判OV ★

定点补码运算定律：

当X,Y,X+Y，X-Y均在$-2^{n-1}\sim+(2^{n-1}-1)$范围内时，则：

［X＋Y］补＝［X］补＋［Y］补

［X－Y］补＝［X］补＋［－Y］补

如果X＋Y,X-Y的值不在$-2^{n-1}\sim+(2^{n-1}-1)$范围内(n=8时 [-128，127])，则机器就产生了溢出错误,上式不成立，运算结果无意义。

溢出判别 (overflow)

若X±Y> $2^{n-1}-1$,为正溢出；

若X±Y< -2^{n-1} ,为负溢出。

判溢出的方法：

1、双进位位法 （本书主要用此法判溢出）

$OV = C8 \oplus C7$ C8、C7相同不溢出，不同溢出。

2、双符号位法——变形码

（1）定点补码加法

步骤：1、将X、Y（或－Y）转换为补码。

2、进行加法运算，符号位参与运算。

[例1-1] 在八位微机中，已知X＝＋76，Y＝＋23，求X＋Y

解： $[X]_{补}$ ＝ 01001100B

编写出程序片段：

```
MOV  A,#76;(A)=4CH=01001100B
ADD A,#23 ;(A)=4CH+17H=63H
```

OV=0

真值

补码

或：

```
MOV  A ,#4CH;(A)=4CH=01001100B
ADD A,#17H ; (A)=63H
```

$=[(+76)+(-23)]_{补}=[X+Y]_{补}$

双进位位法判溢出： OV=0 ∵ C7=1,C8=1

[例1-5] 在八位微机中，已知X＝＋76，Y＝＋69，求X＋Y

溢出后，运算结果无意义，需要将两个操作数扩大位数后，再算。

例1-5 可将76的补码写成004CH，69的补码写成0049H

计算：

0 000 0000 0100 1100B

\+ 0 000 0000 0100 1001B

0 000 0000 1001 0101B =0095H

C16 C15 OV=0，不溢出

例1-6同理，用16位二进制数表示数，-76得补码为FFB4H

-69的补码为FFBBH，再算即可。

双符号位法判断溢出——变形码

用两位来表示符号：

00表示正号，11表示负号，称为变形码。

用变形码进行加法运算时，两位符号位同数值位一起参加运算，运算后，

若运算结果的两个符号位相同，则没有溢出；

若运算结果的两个符号位不同，则发生了溢出，运算结果错误。用S_f'和S_f表示运算结果的两个符号位，则有：

$$OV = S_f' \oplus S_f$$

[例1-7] 在八位微机中，已知X＝＋76，Y＝＋69，求X＋Y

解：

$[X]_{补} = 01001100B$

$[Y]_{补} = 01000101B$

$$
\begin{array}{rl}
 & [X]_{变形码} = 001001100B \\
+ & [Y]_{变形码} = 001000101B \\
\hline
 & 010010001B
\end{array}
$$

因为$S_f'=0$，$S_f=1$，运算后，根据$S_f' \oplus S_f = 0 \oplus 1 = 1$

设置OV＝1，有溢出，结果错误。

(2) 定点补码减法运算

编写出程序片段：

```
MOV  A,#4CH  (A)=4CH=01001100B
MOV  B,#-0E9H ;(B)=0E9H
CLR  C
SUBB A,B;(A)=4CH-0E9H=63H
          OV=0
```

	00010110B	取反
+	1	加1

求补

+	00010111B
	01100011B

$01100011B = [+99]_{补} = [(+76)-(-23)]_{补} = [X-Y]_{补}$

双进位位法判溢出： OV=0 ∵ C7=0,C8=0

同补码加法一样，补码的减法运算也可能发生溢出，

因为 …… 所以其

溢出

[例1-

解：

典型算法：两个带符号数比较大小

用S表示和的符号位，OV为溢出标志位

则：$[X]_{补} - [Y]_{补}$

S	OV	比较结果
0	0	X>Y
0	1	X<Y
1	0	X<Y
1	1	X>Y

错误。

X-Y=145>127

2．BCD码加法及十进制调整

（1）BCD码的加法运算

在两个数的BCD码进行加法运算时，当低四位和高四位都无进位并且不超过9时，可得到正确的运算结果。

[例1-13] 已知X＝63，Y＝24，求X＋Y

解：

$$
\begin{array}{lrl}
 & [X]_{BCD码} & = 01100011B \\
+ & [Y]_{BCD码} & = 00100100B \\
 & 进位 & \ 01100000 \\
\hline
 & & \ 10000111B
\end{array}
$$

$10000111B = [87]_{BCD码} = [63+24]_{BCD码} = [X+Y]_{BCD码}$

[例1-14] 已知X＝68，Y＝49，求X＋Y

编写出程序片段：

```
MOV  A,#68H;(A)=68H=01101000B
ADD A,#49H ;(A)=B1H
DA  A          ;(A)=B1H+66H=17H
               CY=1 代表100
               结果：117
```

必须写BCD码
不能写真值

结果

错误

进位

时，是逢16进位为1，即按照十六进制的原则进行的运算，而BCD码是十进制数，应按照逢十进一的原则进行运算，所以应将和的低四位加6以补上多拿走的6，调整为0111B。和的高四位1011B大于9，应向高位进位，同样加上6进行调整，变为10001B。

（2）BCD码的减法运算

两个数的BCD码进行减法运算时，

当低四位或高四位都不需借位时，可得到正确的运算结果。

[例1-15] 已知X＝58，Y＝25，求X－Y

解：

$$
\begin{array}{lrl}
 & [X]_{BCD码} & = 01011000B \\
- & [Y]_{BCD码} & = 00100101B \\
\hline
 & & \ 00110011B
\end{array}
$$

$$00110011B=[33]_{BCD码}=[58-25]_{BCD码}=[X-Y]_{BCD码}$$

当低四位或高四位有借位时，按十进制运算规则，向高位借1当10，而计算机中按二进制运算规则进行，借1当作16，因此运算后必须减6进行调整。

[例1-16] 已知X＝68，Y＝49，求X－Y

解：

$$
\begin{array}{lrr}
 & [X]_{BCD码} = & 01101000B \\
- & [Y]_{BCD码} = & 01001001B \\
\hline
 & 差 & 00011111B \quad AC=1,CY=0 \\
 & - & 0110B \\
\hline
 & & 00011001B
\end{array}
$$

$$00011001B=[19]_{BCD码}=[68-49]_{BCD码}=[X-Y]_{BCD码}$$

说明:

如果指令系统中有BCD码的减法调整指令，即可直接用该指令完成上述调整。

如果指令系统中没有BCD码的减法调整指令，则不能用减法指令直接对两个BCD码进行减法运算，而需对减数求补，进行加法运算，然后用加法运算的调整指令进行调整。

对八位微机，BCD码的模为100(十进制数)，减去减数实现对减数的求补。为在八位加减运算电路中运算，将100表示成9AH，即10011010B，减去减数求补。

编写出程序片段：

CLR C

MOV A,#9AH ; (A)=9AH MOD

SUBB A,#49H; (A)=51H BCD

ADD A,#68H ; (A)=B9H 非BCD

DA A ; (A)=19H BCD

CPL C ; CY=0 无借位，差=19H$_{BCD}$

求补相加有进位，直接相减无借位，反之，有借位。

求补

法运

算

到正

确的结果。

例：求62+98=?

1、作无符号数运算，结果0A0H=160，CY=0。

一般，CY=0，结果在0~255之间，

CY=1，代表256，结果在0~256+255之间

2、做带符号补码运算，OV=1，正溢出，结果无意义。

一般，结果应在-128~127之间，超出则溢出。可扩大位数到16位再重新做。

3、做BCD码运算，必须送BCD 码，调整后，CY=1，代表100，(A)=60H，代表 60，合成后代表160。

一般，CY=0，结果在00~99之间

CY=1，代表100，结果在00~199之间

4、位数相同，性质不同的数，表示数的范围不同。

逻辑运算

计算机由专门的逻辑电路完成一些逻辑运算

逻辑运算都是位

逻辑“与”的作用：

关，

与1

屏蔽

例：将56H拆成05H和 06H

```
MOV  A,#56H    ;(A)=01010110B=56H
ANL  A,#0FH    ;(A)=00000110B=06H
MOV  30H,A     ;(30H)=06H
MOV  A,#56H    ;(A)=56H
ANL  A,#0F0H   ;(A)=01010000B=50H
SWAP   A       ;(A)=00000101B=05H
```

析取低四位
屏蔽高四位

		0
		0
		0
1	1	1

析取

清零

(2) 逻辑或运算

逻辑或的运算符为“∨ ”，其运算规

$0 \vee 0 = 0$

$0 \vee 1 = 1 \vee 0 = 1$

$1 \vee 1 = 1$

逻辑“或”的用途

1、某位置1

2、自身相或，不变

3、拼字

在八位微机中进行的逻辑或运算：

逻辑或运算的真值表

例：　求9的ASCII码

```
MOV  A,#30H ;(A)=00110000B
ORL  A,#09H; (A)=00111001B=39H
```

1	0	1
1	1	1

逻辑或运算可对字中的**某位置1**，本例中将D_7、D_4位置1。

（3）逻辑非运算

逻辑非运算的真值表

逻辑

它的运

$\overline{1}=0$

$\overline{0}=1$

	1
	0

例：求-5的补码

```
MOV A,#5 ；(A)=00000101B
CPL  A    ；(A)=11111010B
INC  A    ；(A)=11111011B=0FBH
```

-5的补码

这里“0”或“1”上面的“一横”表示“非”运算，或称“求反”，其真值表如表所示。

微型机中通常有“求反”（CPL）指令。

在机器中求一个数的补码，就是先求该数的“反”，再在末位加1得到的。

（4）异或运算

逻辑异或运算的真值表

异或运算又称为按位加

它

0

0

1

“

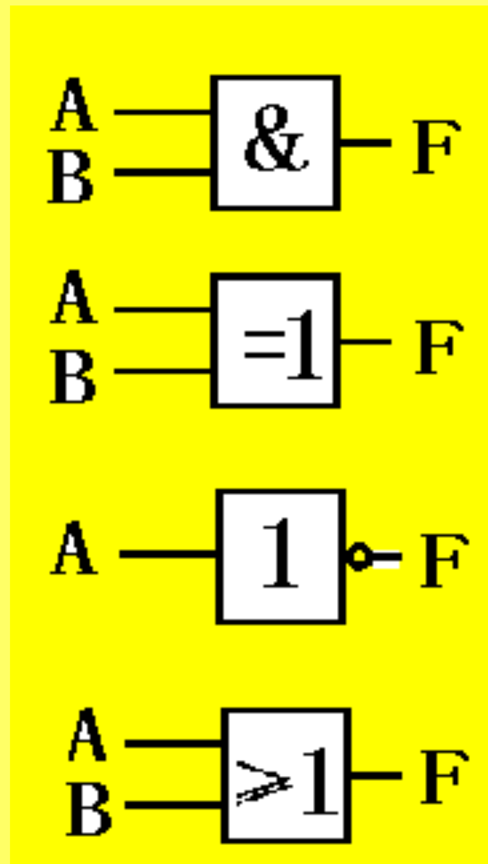
常用的逻辑门电路符号

与门符号

异或门符号

非门符号

或门符号



$F=A \cdot B$

$F=A\oplus B$

$F=\bar{A}$

$F=A+B$

反

0B

1B

10010010B

从例中不难看出，
对字中异或“1”的位变反，
异或“0”的位不变。自身异或，清零。

高四位不变
低四位变反

第一章 结束