计算机组成原理

一. 常用进位计数制

1. 二进制

计算机中信息的存储、处理和传送采用的都是二进制，不论是指令还是数据，或是多媒体信息（声音、图形、图像等），都必须采用二进制编码形式才能存入计算机中。

二进制是一种最简单的进位计数制，它只有两个不同的数码：0 和 1，即基数为 2，逢 2 进 1。任意位数的权是 2^i。

任何一个二进制数都可表示为：$(\mathrm{N})_2 = \sum_{i=n}^{-m} K_i * 2^i$

数字：0、1

后缀：B

例：1010B

2. 八进制

数字：0、1、2、3、4、5、6、7

后缀：O 或 Q

例：137.67Q

3. 十进制

在进位计数制中，每个数位所用到的数码符号的个数叫做基数。十进制是人们最熟悉的一种进位计数制，每个数位允许选用 0~9 共 10 个不同数码中的某一个，因此十进制的基数为 10.每个数位计满 10 就向高位进位，即“逢 10 进 1”。

在一个数中，数码在不同的数位上所表示的数值是不同的。每个数码所表示的数值就等于该数码本身乘以一个与它所在数位有关的常数，这个常数叫做权。

例如：十进制数 6543.21，数码 6 所在数位的权威 1000，这一位所代表的数值即为 $6 * 10^3 = 6000$，5 所在数位的权为 100，这一位所代表的数值即为 $5 * 10^2 = 500$。。。。。。

数字：0、1、2、3、4、5、6、7、8、9

后缀：D

例：1356D

4. 十六进制

十六进制数的基数为 16，逢 16 进 1。大多数计算机都是采用十六进制来描述计算机中的指令和数据的。

任何一个十六进制数可表示为： $(\mathrm{N})_{16} = \sum_{i=n}^{-m} K_i * 16^i$

数字：0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F

后缀：H

例：19EH

5. 推广

一个基数为 R 的 R 机制数可表示为

$$(\mathrm{N})_R = \sum_{i=n}^{-m} K_i * R^i$$

二. 各种进制数之间的转换

1. 二进制数转换为十六进制数

将一个二进制数转换成十六进制数的方法，是将二进制数的整数部分和小数部分分别进

行转换，即以小数点为界，整数部分从小数点开始往左数，每 4 位分成一组，当最左边的数不足 4 位时，可根据需要，在数的最左边添加若干个 0 以补足 4 位；对于小数部分，从小数点开始往右数，每 4 位分成一组，当最右边的数不足 4 位时，可根据需要，在数的最右边添加若干个 0 以补足 4 位，最终使二进制数的总的位数是 4 的倍数，然后用相应的十六进制数取而代之。

例如：111011.1010011011B=0011 1011.1010 0110 1100B=3B.A6CH。

2. 十六进制数转换为二进制数

要将十六进制数转换成二进制数，只要将 1 位十六进制数写成 4 位二进制数，然后将整数部分最左边的 0 和小数部分最右边的 0 去掉即可。

例如：3B.328H = 0011 1011.0011 0010 1000B = 3B.A6CH。

3. 二进制数转换为十进制数

要将一个二进制数转换成十进制数，只要把二进制数的各位数码与它们的权相乘，再把乘积相加，就得到对应的十进制数，这种方法称为按权展开相加法。

例如：100011.1011B=$1*2^5+1*2^1+1*2^0+1*2^{-1}+1*2^{-3}+1*2^{-4}$=35.6875D。

4. 十进制数转换为二进制数

要将一个十进制数转换成二进制数，通常采用的方法是基数乘除法。这种转换方法是对十进制数的整数部分和小数部分分别进行处理，整数部分用除基取余法，小数部分用乘基取整法，最后将它们拼接起来即可。

1) 整数——除基取余法

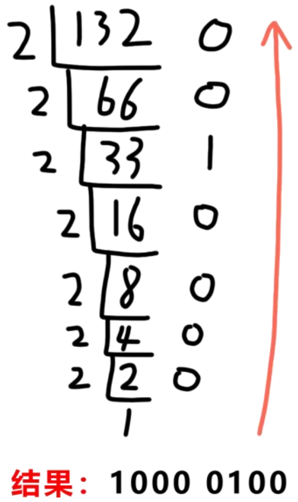

图 1. 十进制转二进制（整数）

2) 小数——乘基取整法

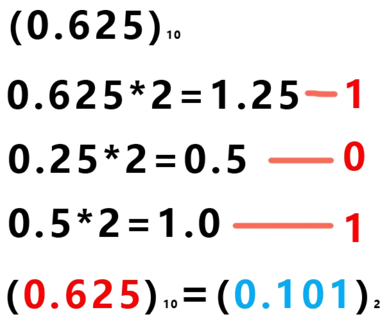

图 2. 十进制转二进制（小数）

三. 数的机器码表示

1. 机器数与真值

二进制数有正负之分，如 $N_1 = +0.101101$，$N_2 = -0.101101$，则 N_1 是个正数，N_2 是个负数。机器不能直接把符号+、-表示出来，为了能在计算机中表示正负数，必须引入符号位，即把正负符号也用 1 位二进制数码来表示。把符号位和数值位一起编码来表示相应的数的表示方法包括原码、补码、反码、移码等。

1) 机器数

用二进制数 0 或 1 来表示数的符号，0 表示正号，1 表示负号。小数点隐含在某一固定位置上，不占存储空间。机器数的位数受机器字长的限制，超过机器字长的数值位要舍去。

机器数可分为，无符号数和带符号数。无符号数，其机器字长的所有二进制位均表示数值。带符号数，其第一个二进制位为符号位，其余为数值部分。

2) 真值

真值，即用$\pm|A|$表示的实际数值。

例：8 位机器数为 11011011

若为无符号整数，其真值为 219。

若为带符号整数，且采用源码表示，则最高位为符号位，11011011 表示二进制数，-1011011，其真值为-91。

3) 原码

符号位为 0 表示正数，为 1 表示负数，数值部分用二进制数的绝对值表示的方法称为原码表示法，通常用$[X]_{原}$表示 X 的原码。

例如，要表示+59 和-59 的原码。假设机器数的位数 8 位（机器的字长为 8 位），最高位是符号位，其余 7 位是数值位，那么，+59 和-59 的原码分别表示为

$[+59]_{原}$=0011 1011　　　　$[-59]_{原}$=1011 1011

a. 注意

0 的原码有两个值，有“正零”和“负零”之分，机器遇到这两种情况都当做 0 处理。

$[+0]_{原}$=0000 0000　　　　$[-0]_{原}$=1000 0000

b. 定义式

$$[X]_{原}=\begin{cases} X,\ 0 \le x < 2^{n-1} \\ 2^{n-1} - X,\ 2^{n-1} < x \le 0 \end{cases}$$

c. 真值、机器数、原码互转

例：当 X=+0.1101 时，$[X]_{原}$=0.1101，在机器中表示为0↓1101

当 X=-0.1101 时，$[X]_{原}$=1.1101

当 X=+1110 时，$[X]_{原}$=01110，在机器中表示为 01110↓

当 X=-1110 时，$[X]_{原}$=11110

① 当 X 为“+”时，X_0=0；当 X 为“-”时，X_0=1。

$$[X]_{原} = \underset{\text{数值位不变}}{\overset{\text{0 变为+，1 变为−}}{\longleftrightarrow}} \text{真值 X。}$$

$$[X]_{原} = \underset{\text{数值位不变}}{\overset{\text{符号取反位}}{\longleftrightarrow}} [-x]_{原}\text{。}$$

4) 补码

补码的定义：把某数 X 加上模数 k，称为以 K 为模的 X 的补码。

$[X]_{补}$=K＋X

因此，正数的补码的最高位为符号 0，数值部分为该数本身；负数的补码的最高位为符号 1，数值部分为用模减去该数的绝对值。

求一个二进制数的补码的简便方法是：正数的补码与其原码相同；负数的补码是符号位不变，数值位逐位取反（求其反码），然后在最低位加 1。

注意：0 的补码只有一种形式，就是 n 位 0。

a. 真值、补码、原码互转

当 X≥0，则$[X]_{补}$=$[X]_{原}$=X，且置符号位为 0。

当 X<0，则 X 与$[X]_{补}$互转，置符号位为 1，将 X 的数值位按位取反，末位加 1，即得$[X]_{补}$。若$[X]_{补}$与$[X]_{原}$互转，符号位不变，数值位按位取反，末尾加 1。

b. $[X]_{补}$与$-[X]_{补}$的关系

计$[X]_{补}$称为机器正数，$-[X]_{补}$称为机器负数。求$-[X]_{补}$，也称为对$[X]_{补}$的求补或变补。

$[X]_{补}$↔$-[X]_{补}$：找低位第一个 1，该 1 和其后的 0 不变，其他位取反。

补码的移位关系：

右移：符号位不变，数值位右移，空位补与符号位相同的代码。$[X]_{补}$右移一位得到$[\frac{X}{2}]_{补}$。

左移：符号位不变，数值位左移，空位补 0，$[X]_{补}$左移一位得到$[2X]_{补}$。

5) 反码

引入反码的目的是便于求负数的补码。正数的反码与原码相同，负数的反码是符号位不变，数值位逐位取反。

反码实质上是补码的一个特例，区别在于反码的模比补码的模小一个最低位上的 1。

例如：$[+59]_{反}$=$[+59]_{原}$=0011 1011，而$[-59]_{原}$=1011 1011，因此，[-59]反=1100 0100。

注意：0 的反码也有两个，$[+0]_{反}$=0000 0000，$[-0]_{反}$=1111 1111。

当 X≥0，则$[X]_{反}$=X，且置符号位为 0。

当 X<0，置符号位为 1，将 X 的数值位按位取反，即得$[X]_{反}$。

a. 反码与原码、补码及真值的关系

当 X≥0，则$[X]_{补}$=$[X]_{原}$=$[X]_{反}$=X，且置符号位为 0。

当$[X]_{原} \underset{\textit{数值位按位取反}}{\overset{\textit{符号位不变}}{\longleftrightarrow}} [x]_{反} \underset{\textit{符号位不变，末位减}1}{\overset{\textit{符号位不变，末位加}1}{\rightleftharpoons}} [X]_{补}$

$[X]_{反} \underset{\textit{数值位按位取反}}{\overset{\textit{符号位的}1\textit{变为“}-\textit{”}}{\longleftrightarrow}} X$

四. 位运算符

1. 与 – &

参加运算的两个数，按二进制位进行“与”运算。

运算规则：只有两个数的二进制同时为 1，结果才为 1，否则为 0。（负数按补码形式参加按位与运算）

即 0 & 0= 0 ，0 & 1= 0，1 & 0= 0， 1 & 1= 1。

例：3 &5 即 00000011 & 00000101 = 00000001 ，所以 3 & 5 的值为 1。

1) “与运算”的特殊用途

a. 清零

如果想将一个单元清零，即使其全部二进制位为 0，只要与一个各位都为零的数值相与，结果为零。

b. 取一个数中指定位

比如取数 X=1010 1110 的低 4 位，只需要另找一个数 Y，令 Y 的低 4 位为 1，其余位为 0，即 Y=0000 1111，然后将 X 与 Y 进行按位与运算（X&Y=0000 1110）即可得到 X 的指定位。

例：设 X=10101110，

取 X 的低 4 位，用 X & 0000 1111 = 0000 1110 即可得到；

还可用来取 X 的 2、4、6 位。

c. 判断奇偶

只要根据最未位是 0 还是 1 来决定，为 0 就是偶数，为 1 就是奇数。因此可以用 if ((a & 1) == 0)代替 if (a % 2 == 0)来判断 a 是不是偶数。

2. 或 – |

参加运算的两个数，按二进制位进行“或”运算。

运算规则：参加运算的两个数只要两个数中的一个为 1，结果就为 1。

即 0 | 0= 0 ,1 | 0= 1，0 | 1= 1, 1 | 1= 1。

例：2 | 4 即 00000010 | 00000100 = 00000110，所以 2 | 4 的值为 6。

负数按补码形式参加按位或运算。

1) “或运算”的特殊用途

常用来将一个数据的某些位设置为 1。

比如将数 X=1010 1110 的低 4 位设置为 1，只需要另找一个数 Y，令 Y 的低 4 位为 1，

其余位为 0，即 Y=0000 1111，然后将 X 与 Y 进行按位或运算（X|Y=1010 1111）即可得到。

3. 异或 – ^

参加运算的两个数，按二进制位进行“异或”运算。

运算规则：参加运算的两个数，如果两个相应位为“异”（值不同），则该位结果为 1，否则为 0。

即 0 ^ 0=0 ， 0 ^ 1= 1 ， 1 ^ 0= 1 ， 1 ^ 1= 0 。

例： 2 ^ 4 即 00000010 ^ 00000100 =00000110 ，所以 2 ^ 4 的值为 6 。

1) “异或运算”的特殊作用

a. 翻转指定位

比如将数 X=1010 1110 的低 4 位进行翻转，只需要另找一个数 Y，令 Y 的低 4 位为 1，其余位为 0，即 Y=0000 1111，然后将 X 与 Y 进行异或运算（X^Y=1010 0001）即可得到。

例：X=10101110，使 X 低 4 位翻转，用 X ^ 0000 1111 = 1010 0001 即可得到。

b. 与 0 相异或，保留原值

X ^ 0000 0000 = 1010 1110。

c. 交换两个数

```
void Swap(int &a, int &b)
{
    if (a != b)
    {
        a ^= b;
        b ^= a;
        a ^= b;
    }
}
```

4. 取反 – ~

参加运算的一个数，按二进制位进行“取反”运算。如果二进制位是 0，则取反后为 1；如果二进制位是 1，则取反后为 0。要注意的是，符号位也是同样的操作。

运算规则：~1=0； ~0=1；

1) 取反运算符的特殊作用

a. 使一个数的最低位为零

使 a 的最低位为 0，可以表示为：a & ~1。~1 的值为 1111 1111 1111 1110，再按"与"运算，最低位一定为 0。因为" ~"运算符的优先级比算术运算符、关系运算符、逻辑运算符和其他运算符都高。

5. 左移 – <<

左移，相当于乘法。将一个数的各二进制位全部左移若干位（最左边的二进制位丢弃，右边补 0）。

例：a = a << 2 ， a=5。则是将 a 的二进制位左移 2 位，右补 0。

a = 0000 0000 0000 0000 0000 0000 0000 0101

则 a << 2 = 0000 0000 0000 0000 0000 0000 0001 0100

1 << 5，相当于 1 × (2^5) = 32

1 << 0，相当于 1 × (2^0) = 1

因此，对于正数，若左移时舍弃的高位不包含 1，则每左移一位，相当于该数乘以 2。

6. 右移 – >>

右移，相当于除法。将一个数的各二进制位全部右移若干位，右移过程中符号位不变，低位溢出则舍弃，并用符号位补足溢出的高位，即正数左补 0，负数左补 1，右边丢弃。

16 >> 3，相当于 16 / (2^3) = 2

7. 无符号右移 – >>>

无符号右移运算是将操作数所有二进制值逐位右移若干位，包括最高位符号位，也跟着右移，低位溢出并舍弃，高位补 0。要注意的是，无符号右移（>>>）中的符号位（最高位）也会跟着变。

8. 方法示例

注意，位运算只支持整数，不支持 float 和 double，逻辑右移的符号各种语言不太相同，<u>位运算的优先级比较低</u>，建议使用括号来确保运算顺序。

1) 获得 int 型最大值

(1 << 31) – 1; 其结果为 2147483647，也可以写成~(1 << 31)。

2) 交换两数

交换两个数可以使用位异或，a ^= b b ^= a a^=b。

3) 绝对值

(n ^ (n >> 31)) – (n >> 31)，n >> 31 取得 n 的符号位，若 n 为正数，n >> 31 等于 0，若 n 为负数，n >> 31 等于-1。

若 n 为正数 n ^ 0 = 0,数保持不变，若 n 为负数则有 n ^ -1 ，这需要计算 n 和 -1 的补码，然后进行异或运算，结果 n 变号并且为 n 的绝对值减 1，再减去-1 就是绝对值。

4) 取两个数的最大值

b & ((a-b) >> 31) | a & (~(a-b) >> 31)，如果 a >= b，则(a-b) >> 31 为 0，否则为-1。

在 C 语言中可以写成，x ^ ((x ^ y) & -(x < y))，如果 x < y 则 x < y 返回 1，否则返回 0，与 0 做与运算结果为 0，与-1 做与运算结果不变。

5) 取两个数的最小值

a & ((a-b) >> 31) | b & (~(a-b) >> 31)，如果 a >= b，(a-b) >> 31 为 0，否则为-1。y ^ ((x ^ y) & -(x < y))，如果 x<y x<y 返回 1，否则返回 0，与 0 做与运算结果为 0，与-1 做与运算结果不变。

6) 判断符号是否相同

(x ^ y) >= 0，true 表示 x 和 y 有相同的符号， false 表示 x，y 有相反的符号。

7) 判断两数是否相等

判断两数是否相等可以使用异或(^)运算符，如果两个十进制数是相同的，那么它们的二进制数也是相同的，也就是说对这两个二进制数进行按位异或，那么其结果就一定为 0。当这两个数不相同时，那么所得的结果就一定不为 0。

x ^ y == 0

8) 判断一个数的奇偶性

(n & 1) == 1，当数 n 与 1 进行按位与操作时，结果为 1 就为奇数，结果为 0 就为偶数。

9) 求两个整数的平均值

(x + y) >> 1，也可以写成((x ^ y) >> 1) + (x & y)，(x^y) >> 1 可以得到 x，y 之中一个为 1 的位并除以 2，x&y 得到 x，y 都为 1 的部分，加一起就是平均数了。

10) 计算 n+1

-~n

11) 计算 n-1

~-n

12) 取相反数

~n + 1，也可以写为(n ^ -1) + 1。

13) 去掉当前最低位

X >> 1，如果 x 为 101101，那么右移一位以后就变为了 10110。

14) 在最低位后加一个 0

X << 1，如果 x 为 101101，那么右移一位以后就变为了 1011010。

15) 在最低位后加一个 1

X << 1 | 1，如果 x 为 101101，那么左移一位再按位或就变为了 1011011。

16) 把最低位变为 1

X | 1，如果 x 为 101101，那么其与 1 进行按位或就变为了 101101。

17) 把最低位变为 0

X & -2，如果 x 为 101101，那么其与-2(1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1110)进行按位与就变为 101100。

18) 从低位到高位，把第 k 位变成 1

x | (1 << (k-1))，如果 x 为 101101，K 为 3，则结果为 101101。

19) 从低位到高位，把第 k 位变成 0

x & ~ (1 << (k-1))，如果 x 为 101101，K 为 3，则结果为 101001。

20) 右数第 k 位取反

x ^(1 <<(k-1)) ，如果 x 为 101001，K 为 3，则结果为 101101。

21) 取末 k 位

x & (1 << k-1) ，如果 x 为 1101101，K 为 5，则结果为 1101。

22) 低位到高位，取第 k 位

x >> (k-1) & 1，如果 x 为 1101101，K 为 4，则结果为 1。

23) 从低位到高位，把低 k 位变成 1

x | ((1 << k)-1) ，如果 x 为 101001，K 为 4，则结果为 10111。

24) **从低位到高位，把低 k 位取反**

x ^ (1 << k-1) ，如果 x 为 101001，K 为 4，则结果为 100110。

25) **把右边连续的 1 变成 0**

x & (x+1) ，如果 x 为 100101111 则结果为 100100000。

26) **从低位到高位，把第一个 0 变成 1**

x | (x+1) ，如果 x 为 100101111 则结果为 100111111。

27) **从低位到高位，把右边连续的 0 变成 1**

x | (x-1) ，如果 x 为 11011000 则结果为 11011111。

28) **从低位到高位，取右边连续的 1**

(x ^ (x+1)) >> 1，如果 x 为 100101111 则结果为 1111。

29) **从低位到高位，去掉第一个 1 的左边**

x & -x，如果 x 为 100101000 则结果为 1000。