



计算机组成原理

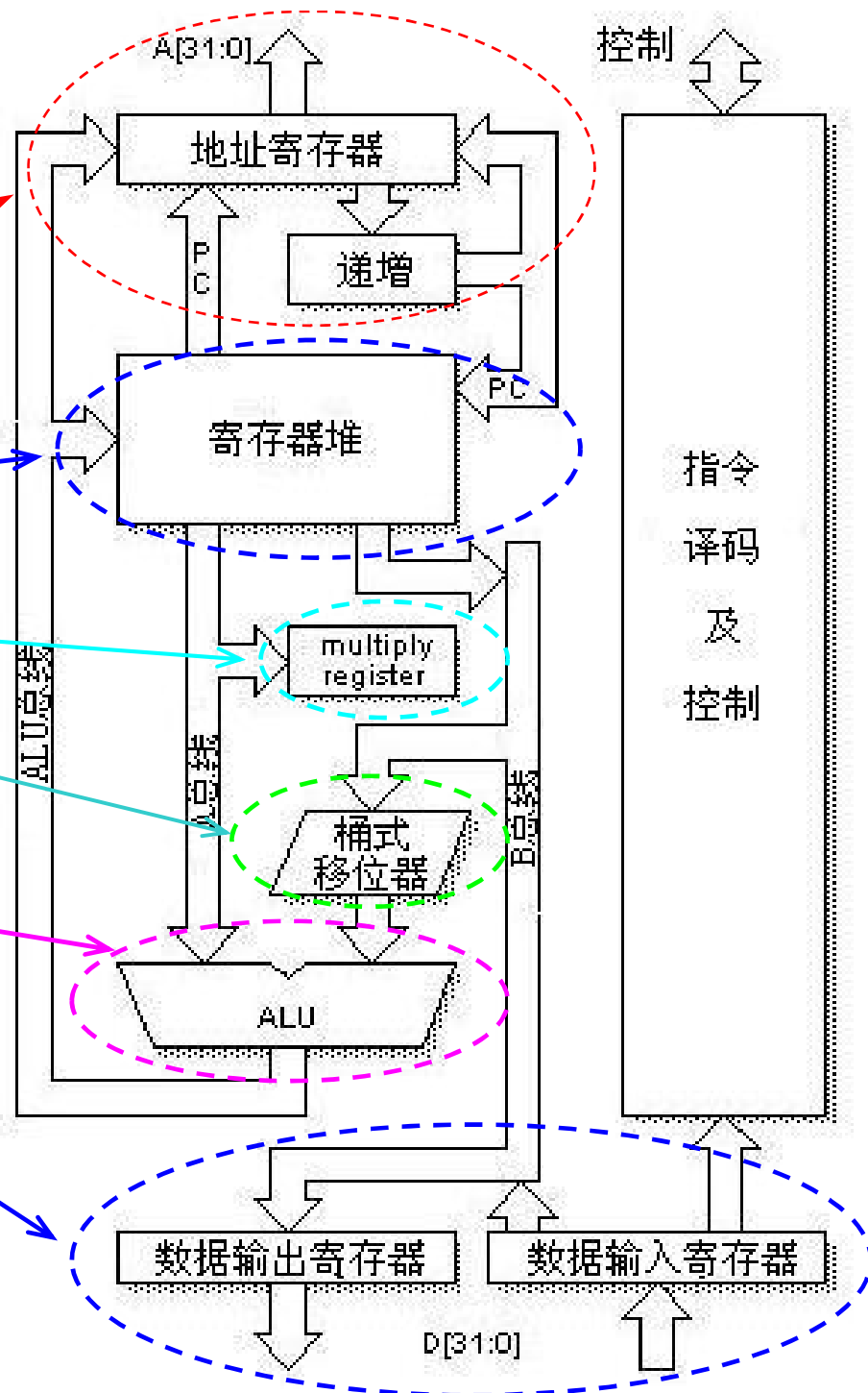
第6章 计算机的运算方法

llxx@ustc.edu.cn

wjluo@ustc.edu.cn

ARM7数据通路

- 地址端口
- 寄存器堆
- 乘法器
- 桶式移位器
- ALU
- 数据端口



内容



- 0. 数据的表示方法和转换
进位计数制，进位计数制之间的转换
- 1. 无符号数和有符号数
原码、补码、反码和移码
- 2. 数的定点表示和浮点表示
- 3. 定点运算
- 4. 浮点四则运算
- 5. 算术逻辑单元ALU

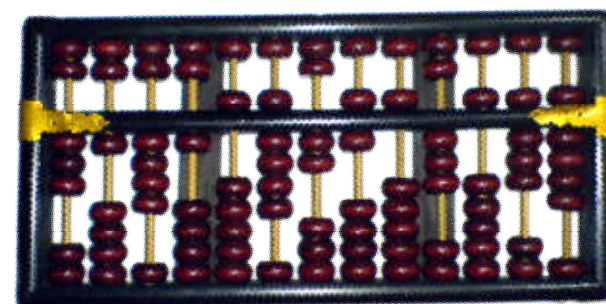
算筹，运筹

- 大约270根一束，随身携带
 - 春秋（前550年）～宋元时期
- 采用十进位位置值制
 - “按位计数法”：个位纵，十位横，空位表示0
 - 比古巴比伦（前729年）的60进位位置值制方便
 - 比古罗马（前8世纪/前6世纪）的十进位非位置值制先进
 - $DCCXII = 500 + 100 + 100 + 12 = (712)_{10}$



■ 陕西千阳出土西汉骨筹

| | | | | | | | | | |
|----|---|---|---|---|---|---|----|-----|------|
| 纵式 | | | | | | ⊥ | ⊥⊥ | ⊥⊥⊥ | ⊥⊥⊥⊥ |
| 横式 | — | = | ≡ | ≡ | ≡ | ⊥ | ⊥ | ⊥ | ⊥ |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |



进位计数制



- **进位计数制**：用少量的数字符号（也称数码），按先后次序把它们排成数位，由低到高进行计数，计满进位，这样的方法称为进位计数制。
- **基数**：进位制的基本特征数，即所用到的**数字符号个数**。
 - 例如10进制：0-9 十个数码表示，基数为10
- **权**：进位制中各位“1”所表示的值为该位的权。
 - $123.45 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2}$
 - 等式左边为**并列**表示法，等式右边为**多项式**表示法
- **常见的进位制**：2，8，10，16进制。



进位计数制之间的转换

- **R进制转换成十进制的方法**
- **十进制转换成二进制方法**
- **二进制和八进制之间的转换**
- **二进制和十六进制之间的转换**



R进制转换成十进制的方法

- 按权展开法: 先写成多项式, 然后计算十进制结果.

- $$\begin{aligned} N &= d_{n-1}d_{n-2}\cdots d_1d_0d_{-1}d_{-2}\cdots d_{-m} \\ &= d_{n-1} \times R^{n-1} + d_{n-2} \times R^{n-2} + \cdots + d_1 \times R^1 + \\ &\quad d_0 \times R^0 + d_{-1} \times R^{-1} + d_{-2} \times R^{-2} + \cdots + d_{-m} \\ &\quad \times R^{-m} \end{aligned}$$

写出 $(1101.01)_2$, $(10D)_{16}$ 的十进制数



十进制转换成二进制方法

- 一般分为两个步骤：
 - 整数部分的转换
 - 基数除法：除**2**取余法
 - 减权定位法
 - 小数部分的转换
 - 基数乘法：乘**2**取整法



除基取余法

- 除基取余法：把给定的数除以基数,取余数作为最低位的系数,然后继续将商部分除以基数,余数作为次低位系数,重复操作，直至商为0。

例如：用基数除法将 $(327)_{10}$ 转换成二进制数

| | | | | |
|---|--|-----|--|----|
| 2 | | 327 | | 余数 |
| 2 | | 163 | | 1 |
| 2 | | 81 | | 1 |
| 2 | | 40 | | 1 |
| 2 | | 20 | | 0 |
| 2 | | 10 | | 0 |
| 2 | | 5 | | 0 |
| 2 | | 2 | | 1 |
| 2 | | 1 | | 0 |
| 2 | | 0 | | 1 |

$$(327)_{10} = (101000111)_2$$

减权定位法



将十进制数依次从二进制的最高位权值进行比较，
若够减则对应位置1，减去该权值后再往下比较，若
不够减则对应位为0，重复操作直至差数为0。

例如：将 $(327)_{10}$ 转换成二进制数

$$256 < 327 < 512$$

| | |
|------------------|---|
| $327 - 256 = 71$ | 1 |
| $71 < 128$ | 0 |
| $71 - 64 = 7$ | 1 |
| $7 < 32$ | 0 |
| $7 < 16$ | 0 |
| $7 < 8$ | 0 |
| $7 - 4 = 3$ | 1 |
| $3 - 2 = 1$ | 1 |
| $1 - 1 = 0$ | 1 |





乘基取整法

• **乘基取整法(小数部分的转换)**：把给定的十进制小数乘以2, 取其整数作为二进制小数的第一位, 然后取小数部分继续乘以2, 将所的整数部分作为第二位小数, 重复操作, 直至得到**所需要的**二进制小数。

例如: 将 $(0.8125)_{10}$ 转换成二进制小数.

整数部分

$$2 \times 0.8125 = 1.625$$

1

$$2 \times 0.625 = 1.25$$

1

$$2 \times 0.25 = 0.5$$

0

$$2 \times 0.5 = 1$$

1



$$(0.8125)_{10} = (0.1101)_2$$



二进制与八进制、十六进制之间的转换

二 <----> 八

| | |
|-----|---|
| 000 | 0 |
| 001 | 1 |
| 010 | 2 |
| 011 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | 7 |



二 <----> 十六

| | |
|------|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |

| | |
|------|---|
| 1000 | 8 |
| 1001 | 9 |
| 1010 | A |
| 1011 | B |
| 1100 | C |
| 1101 | D |
| 1110 | E |
| 1111 | F |

二进制转换成十六进制



- 例：(110110111 .01101)₂转换成16进制。

高位补零，凑足四位 分组起点 低位补零，凑足四位

二进制: 1,1011,0111.0110,1

二进制: 0001,1011,0111.0110,1000

十六进制: 1 B 7 . 6 8

$$(10110111.01101)_2 = (1B7.68)_{16}$$



十六进制转换成二进制

方法：每位十六进制数用四位二进制数表示。

例如： $(7AC.DE)_{16}$

$$=(0111, 1010, 1100.1101, 1110)_2$$

$$=(11110101100.1101111)_2$$



6.1 无符号数和有符号数

原码、补码、反码和移码

无符号数vs.有符号数

| b3b2b1b0 \ b6b5b4 | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|-------------------|------|------|------|------|------|------|------|------|
| | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 |
| 0000 | NUL | DLE | SP | 0 | @ | P | | p |
| 0001 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 0010 | STX | DC2 | " | 2 | B | R | b | r |
| 0011 | ETX | DC3 | # | 3 | C | S | c | s |
| 0100 | EOT | DC4 | \$ | 4 | D | T | d | t |
| 0101 | ENQ | NAK | % | 5 | E | U | e | u |
| 0110 | ACK | SYN | & | 6 | F | V | f | v |
| 0111 | BEL | ETB | ' | 7 | G | W | g | w |
| 1000 | BS | CAN | (| 8 | H | X | h | x |
| 1001 | HT | EM |) | 9 | I | Y | i | y |
| 1010 | LF | SUB | * | : | J | Z | j | z |
| 1011 | VT | ESC | + | : | K | [| k | { |
| 1100 | FF | FS | , | < | L | \ | l | |
| 1101 | CR | GS | - | = | M |] | m | } |
| 1110 | SO | RS | . | > | N | ^ | n | ~ |
| 1111 | SI | US | / | ? | O | _ | o | DEL |

- 无符号数：不带符号位的数
 - 数值：可用于计数、地址指针等
 - 可用全部字长来表示数值大小。
 - 如8位无符号数的取值范围是 $0 \sim 255$ ($2^8 - 1$)。
 - 字符：ASCII码
- 有符号数：带符号位的数
- 机器字长：计算机进行一次整数运算所能处理的二进制数据的位数（即定点运算）
 - CPU内部数据通路的宽度(通用寄存器、ALU)
 - 16位寄存器
 - 无符号数： $0 \sim 65535$
 - 一位符号位的有符号数： $-32768 \sim +32767$



有符号数的编码

- 真值vs.机器数

- 真值：正、负号加某进制数绝对值的形式称为真值。

- 如二进制真值： $X=+1011$ $y=-1011$

- 机器数：符号数码化的数称为机器数，
如： $X=01011$ $Y=11011$

- 在本章中， n 表示字长的有效位（数值位）， X 表示真值。



机器数的表示方法

- 一旦符号数字化以后，符号和数值就形成了一种新的编码。
- 在运算过程中，符号位能否和数值部分一起参加运算？如果参与运算，符号位又需要做哪些处理？
 - 这些问题都与符号位和数值位所构成的编码有关。
- 机器数有四种**编码表示方法**：原码、补码、反码和移码。

1. 原码表示法(Original code?)



- 用 “0”表示正号，用 “1”表示负号，数值位用真值的绝对值表示。
 - 整数的符号位与数值位之间用逗号 “,” 隔开；
 - 小数的符号位与数值位之间用小数点 “.” 隔开。
- 约定：
 - 在本章中， n 表示字长的有效位（数值位，不含符号位）， X 表示真值（含符号）。
 - 如： $y = -1011011$ ， $|y|_{\text{原}} = 1,1011011$
 - 则： $X = -1011011$ ， $n = 7$ ，字长 = 8



整数原码的定义

$$[X]_{\text{原}} = \begin{cases} 0, X & 2^n > X \geq 0 \\ 2^n - X & 0 \geq X > -2^n \end{cases}$$

0的原码有两种表示方式:

$[+0]_{\text{原}} = 0,0000000$; $[-0]_{\text{原}} = 1,0000000$

例：完成下列数的真值到原码的转换

$X_1 = + 1011011$ $[X_1]_{\text{原}} = 0,1011011$

$X_2 = - 1011011$ $[X_2]_{\text{原}} = 1,1011011$



小数原码的定义

$$[X]_{\text{原}} = \begin{cases} X & 1 > X \geq 0 \\ 1-X & 0 \geq X > -1 \end{cases}$$

例：完成下列数的真值到原码的转换

$X1 = + 0.1011011$

$[X1]_{\text{原}} = 0.1011011$

$X2 = - 0.1011011$

$[X2]_{\text{原}} = 1.1011011$



原码特点

- **直观**
 - 表示简单，易于同真值之间进行转换；
 - 实现乘除运算规则简单。
- **进行加减运算十分麻烦**
 - 本来是加法运算却可能要用减法器实现。
 - 当两个操作数符号不同且做加法运算时，先要判断两个数绝对值的大小，然后将绝对值大的数减去绝对值小的数，结果的符号以绝对值大的数为准。
- **0的表示不惟一**

2. 反码表示法 (1's complement coding)



- **反码的概念：**

- 正数的表示与原、补码相同；
- 负数的反码符号位为1，数值位是将原码的数值按位取反，就得到该数的反码表示。

$$1\overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4$$

- 反码通常用来作为原码求补码，或者由补码求原码的中间表示。

整数反码



$$[X]_{\text{反}} = \begin{cases} 0, X & 2^n > X \geq 0 \\ (2^{n+1} - 1) + X & 0 \geq X > -2^n \quad (\text{mod } (2^{n+1} - 1)) \end{cases}$$

例:

$X_1 = +1011011$, $[X_1]_{\text{反}} = 0, 1011011$

$X_2 = -1011011$, $[X_2]_{\text{反}} = 1, 0100100$

$$\begin{array}{r} 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ -\ 1\ 0\ 1\ 1\ 0\ 1\ 1 \\ \hline 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0 \end{array}$$

$[+0]_{\text{反}} = 00000000$; $[-0]_{\text{反}} = 11111111$



小数反码

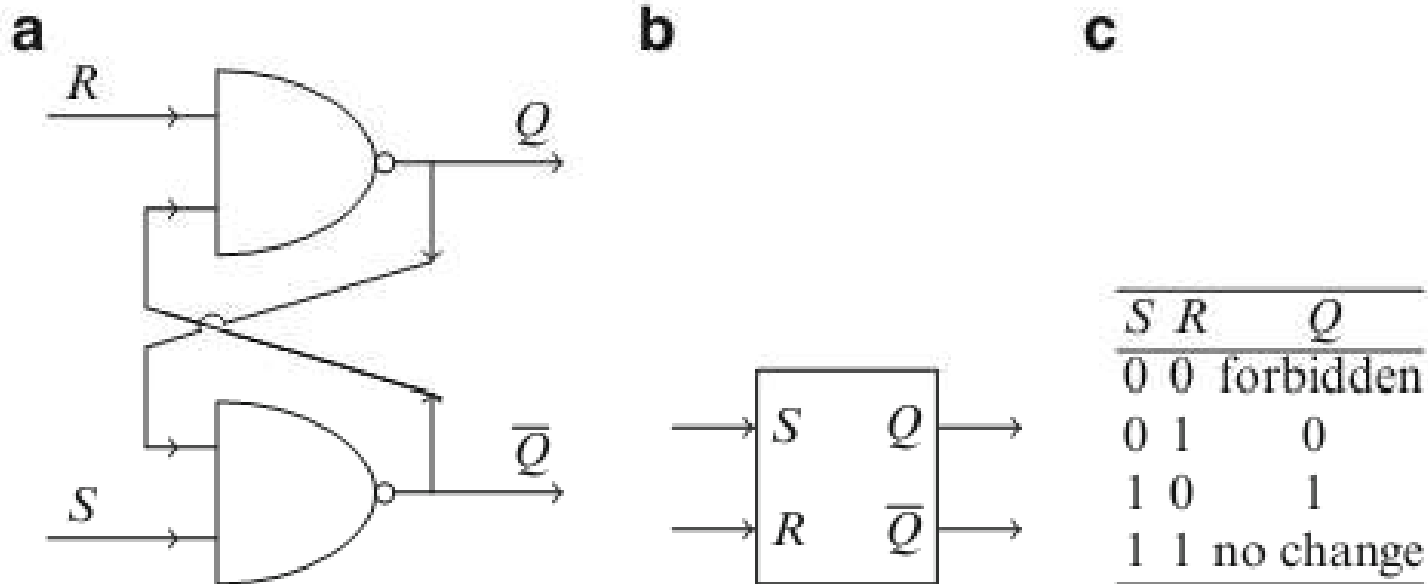
$$[X]_{\text{反}} = \begin{cases} X & 1 > X \geq 0 \\ (2 - 2^{-n}) + X & 0 \geq X > -1 \pmod{(2 - 2^{-n})} \end{cases}$$

$$X_1 = +0.1011011, [X_1]_{\text{反}} = 0.1011011$$

$$X_2 = -0.1011011, [X_2]_{\text{反}} = 1.0100100$$

$$\begin{array}{r} 1.1111111 \\ - 0.1011011 \\ \hline 1.0100100 \end{array}$$

complement: 反, 补



A SR NAND latch: a) implementation, b) icon, c) functional behavior

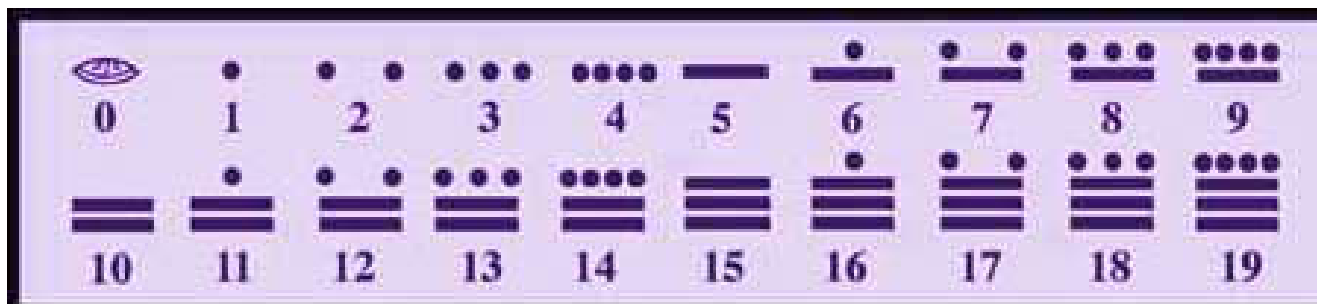
The input signals S and R (for “set” and “reset”) are used to change the value of the output.

"the output signals Q and \bar{Q} are always opposite and represent the value stored by the latch (Q) and its **complement** (\bar{Q})"

补码 (2's complement coding)



- **模**：计量器具的容量，或称为模数。
 - M位字长**整数的模**值为 2^M
 - 4位字长的机器表示的**二进制整数**为：
0000 ~ 1111 共16种状态，模为 $16=2^4$ 。
 - 一位符号位的**纯小数的模**值为2
- **补码定义**：正数的补码就是正数的本身，负数的补码是原负数加上模。

A table showing the 16-point binary code (BCD) for digits 0 through 9. Each digit is represented by a 4-bit binary code, shown as a horizontal bar with dots above it. The digits are arranged in two rows of five.

| | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |



整数补码的定义

$$[x]_{\text{补}} = \begin{cases} 0, X & 2^n > X \geq 0 \\ 2^{n+1} + X & 0 > X \geq -2^n \end{cases} \pmod{2^{n+1}}$$

例：完成下列数的真值到补码的转换

$$X1 = +1011011$$

$$[X1]_{\text{补}} = 0, 1011011$$

$$X2 = -1011011$$

$$[X2]_{\text{补}} = 2^{7+1} + x = 1, 0100101$$



小数补码的定义

$$[x]_{\text{补}} = \begin{cases} X & 1 > X \geq 0 \\ 2+X & 0 > X \geq -1 \end{cases} \pmod{2}$$

例：完成下列数的真值到补码的转换

$$X1 = + 0.1011011$$

$$[X1]_{\text{补}} = 0.1011011$$

$$X2 = - 0.1011011$$

$$[X2]_{\text{补}} = 1.0100101$$

例：0的补码（补码中“零”只有一种表示形式）

$$[+ 0.00000000] = 0.00000000$$

$$[-0.00000000] = 2 + (-0.00000000) \pmod{2} = 0.00000000$$



原码与补码之间的转换

- 原码求补码
- 先看**整数**原码和补码之间的转换

$$[X]_{\text{原}} = \begin{cases} 0, X & 2^n > X \geq 0 \\ 2^n - X & 0 \geq X > -2^n \end{cases} \quad [x]_{\text{补}} = \begin{cases} 0, X & 2^n > X \geq 0 \\ 2^{n+1} + X & 0 > X \geq -2^n \end{cases}$$

正数的原码和补码显然一致。

对于负数：设 $n=4$, $x = -x_1x_2x_3x_4$

$$\begin{aligned} [x]_{\text{补}} &= 2^{n+1} + x = 10,0000 - x_1x_2x_3x_4 = 11111 + 00001 - x_1x_2x_3x_4 \\ &= 1\overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + 00001 \end{aligned}$$

符号位除外，对原码每位取反，末位加1。

对小数原码也同样成立。反过来，由补码求原码也同样成立。



原码与补码之间的转换

- 原码 \leftrightarrow 补码

正数 $[X]_{\text{补}} = [X]_{\text{原}}$

负数 符号位除外，每位取反，末位加1

例： $X = -1001001$

$[X]_{\text{原}} = 1, 1001001$, $[X]_{\text{补}} = 1, 0110110 + 1 = 1, 0110111$

$[X]_{\text{补}} = 2^{7+1} + X = 1\ 0000\ 0000 - 1001001 = 1, 0110111$

$$\begin{array}{r} 10\ 000000 \\ -\quad 1001001 \\ \hline 1,0110111 \end{array}$$



由 $[X]_{\text{补}}$ 求 $[-X]_{\text{补}}$ (求机器负数)

解：以小数补码为例。设 $[y]_{\text{补}} = y_0 y_1 y_2 \dots y_n$

第一种情况，正数， $[y]_{\text{补}} = 0.y_1 y_2 \dots y_n$

所以 $y = 0.y_1 y_2 \dots y_n$ ，故 $-y = -0.y_1 y_2 \dots y_n$

则 $[-y]_{\text{补}} = 1.\bar{y}_1 \bar{y}_2 \dots \bar{y}_n + 2^{-n}$

第二种情况，负数， $[y]_{\text{补}} = 1.y_1 y_2 \dots y_n$

所以 $[y]_{\text{原}} = 1.\bar{y}_1 \bar{y}_2 \dots \bar{y}_n + 2^{-n}$

$$y = -(0.\bar{y}_1 \bar{y}_2 \dots \bar{y}_n + 2^{-n})$$

$$-y = 0.\bar{y}_1 \bar{y}_2 \dots \bar{y}_n + 2^{-n}$$

$$\text{则} [-y]_{\text{补}} = 0.\bar{y}_1 \bar{y}_2 \dots \bar{y}_n + 2^{-n}$$

- **运算过程：连同符号一起将补码各位取反，末位再加1。**

补码最大的优点就是将减法运算转换成加法运算



- $[X]_{\text{补}} - [Y]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}}$

例如 $X=(11)_{10}=(1011)_2$ $Y=(5)_{10}=(0101)_2$

已知字长 $n=5$ 位

$$[X]_{\text{补}} - [Y]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}}$$

$$= 0,1011 + 1,1011 = 10,0110 = 0,0110 = (6)_{10}$$

注：最高1位已经超过字长故应丢掉

变形补码



- 为了便于判断运算结构是否溢出，某些计算机还采用了一种**双符号位的补码**表示方法，称为变形补码。
- 假定变形补码的有效数值部分位数为 n ，则负数变形补码的表示定义为：
 - ① 负整数： $[X]_{\text{补}} = 2^{n+2} - X$
 - ② 负小数： $[X]_{\text{补}} = 4 + X$
- 因为这种补码小数的模数为4，因此也称模4补码。
- 在双符号位中，左符是真正的符号位，右符用来判别“溢出”。

双符号位溢出判断法



双符号含义：

- 00**表示运算结果为正数；
- 01**表示运算结果正溢出；
- 10**表示运算结果负溢出；
- 11**表示运算结果为负数。

第一位符号位为运算结果的真正符号位。



4. 移码（增码）表示法

- 引入移码的原因：当真值用补码表示时，由于符号位和数值部分一起编码，难于从补码形式上**直接判断**其真值的大小。
 - 例如：x=21，y=-21， $[X]_{\text{补}}=0,0010101$ ， $[y]_{\text{补}}=1,1101011$ 。从二进制码看，会得出 $11101011 > 00010101$ 的结论。

移码定义： $[X]_{\text{移}} = 2^n + X$ $2^n > X \geq -2^n$

移码就是真值加上一个常数。

例：

$X_1 = 101\ 0101$

$[X_1]_{\text{补}} = 0, 101\ 0101$

$[X_1]_{\text{移}} = 1, 101\ 0101$

$X_2 = -101\ 0101$

$[X_2]_{\text{补}} = 1, 010\ 1011$

$[X_2]_{\text{移}} = 0, 010\ 1011$

X=0时，

$[+0]_{\text{移}} = 2^7 + 0 = 1, 000\ 0000$

$[-0]_{\text{移}} = 2^7 - 0 = 1, 000\ 0000$

移码表示中0的也是唯一的。

移码与补码符号位相反。



码制表示法小结

- 原码：真值的直观表示；
 - 补码：真值的加减运算；
 - 原码符号位不变，数值位“取反加一”。
 - 移码：补码的大小比较；
 - 移码与补码的数值位相同，只是符号位相反。
 - 变形补码：补码运算溢出判断；
 - 双符号位补码。
-
- ① $[X]_{\text{原}}$ 、 $[X]_{\text{反}}$ 、 $[X]_{\text{补}}$ 用 “0”表示正号，用 “1”表示负号；
 $[X]_{\text{移}}$ 用 “1”表示正号，用 “0”表示负号。
 - ② 如果X为正数，则 $[X]_{\text{原}} = [X]_{\text{反}} = [X]_{\text{补}}$ 。
 - ③ 如果X为0，则 $[X]_{\text{补}}$ 、 $[X]_{\text{移}}$ 有唯一编码， $[X]_{\text{原}}$ 、 $[X]_{\text{反}}$ 有两种编码。
 - ④ 求 $[-X]_{\text{补}}$ ，将 $[X]_{\text{补}}$ 连同符号一起各位取反，末位加1。



6.2 数的定点表示和浮点表示

定点表示

浮点表示

IEEE 754标准

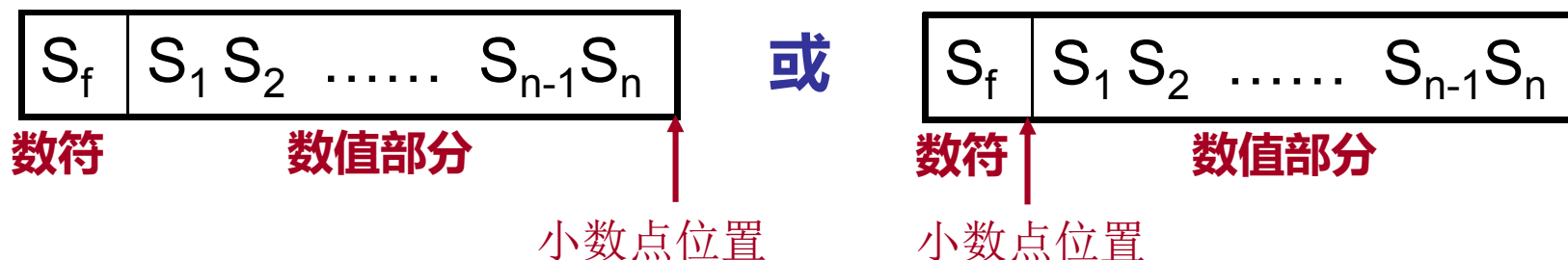
相关概念



- **数值范围**：一种数据类型所能表示的最大值和最小值。
- **数据精度**：实数所能表示的有效数字位数。
- 数值范围和数据精度均与使用多少位二进制位数以及编码方式有关。
- 计算机用数字表示正负，隐含规定小数点。采用“定点”、“浮点”两种表示形式。



1. 数的定点表示方法



- (1) 定点整数——小数点位置固定在数的**最低位**之后
若采用原码，则范围为： $-(2^n-1) \sim 2^n-1$
其中 n 表示数值位的位数。
- (2) 定点小数——小数点位置固定在数的**符号位**之后、数值最高位之前。
若采用原码，则范围为： $-(1-2^{-n}) \sim 1-2^{-n}$



2. 数的浮点表示方法

(1) 浮点数的表示

- 把字长分成阶码j和尾数S两部分。其根据就是：

$$N = S \times r^j$$

S为尾数，j为阶码，r为基值。在计算机中，基可取2、4、6、8或16等。

- 以基数r=2为例，数N可写成下列不同形式：

$$N = 11.0101$$

$$= 0.110101 \times 2^{10} \text{ (尾数为纯小数且规格化)}$$

$$= 1.10101 \times 2^1$$

$$= 1101.01 \times 2^{-10}$$

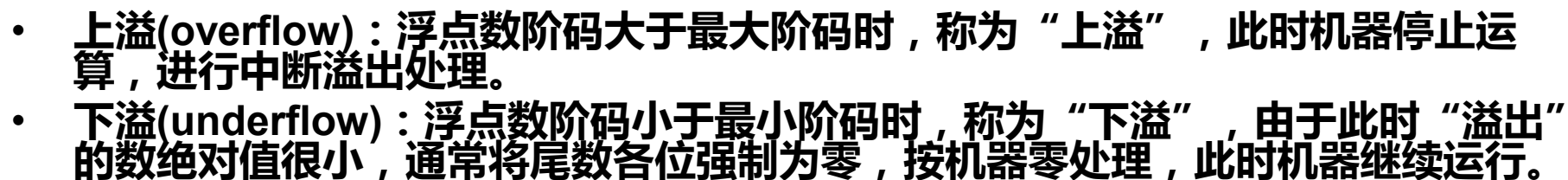
$$= 0.00110101 \times 2^{100} \text{ (尾数为纯小数)}$$



Diagram illustrating the structure of a floating-point number:

- 阶符** (Sign of Exponent): j_f
- 阶码的数值部分** (Numerical part of Exponent): $j_1 j_2 \dots j_{n-1} j_n$
- 数符** (Sign of Mantissa): S_f
- 数值部分** (Numerical part of Mantissa): $S_1 S_2 \dots S_{n-1} S_n$

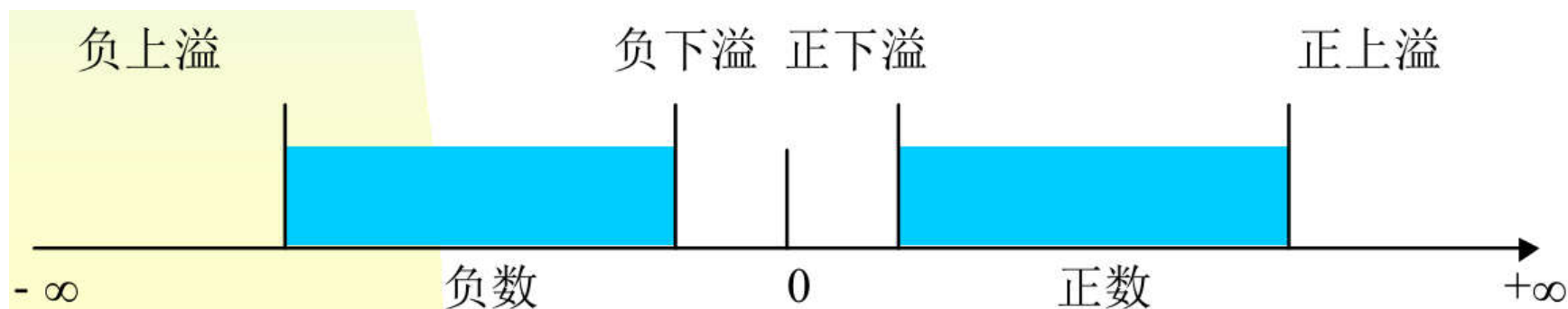
The vertical line between the exponent and mantissa sections indicates the **小数点位置** (Decimal point position).



机器零



- **case1** : 当一个浮点数尾数为0时, 不论其阶码为何值;
- **case2** : 阶码等于或小于它所能表示的最小值时, 不管其尾数为何值。





浮点数的规格化

- 为了提高数据精度以及便于浮点数的比较，规定浮点数的尾数用纯小数的形式。
 - 字长固定的情况下提高表示精度的措施：
 - ① 增加尾数位数（但减小了数值表示范围）。
 - ② 采用浮点规格化形式。
- 规格化数
 - 二进制原码表示时，尾数最高位为1的浮点数
 - 规格化：出现 $0.0xx\dots x$ 就不是规格化数
 - 左规：尾数左移1位，阶码减1。
 - 右规：尾数右移1位，阶码加1。

采用双符号位的补码表示的规格化数



- 尾数S的规格化是指尾数满足条件：

$$\frac{1}{2} \leq |S| < 1$$

- 如果采用双符号位补码，则
当 $S > 0$ 时，其补码规格化形式为

$$[S]_{\text{补}} = 00.1 \times \times \dots \times$$

当 $S < 0$ 时，其补码规格化形式为

$$[S]_{\text{补}} = 11.0 \times \times \dots \times$$

(为啥最高位为0?)

- 左规：00.0xx...x或11.1xx...x
- 右规：01.xx...x或10.1xx...x



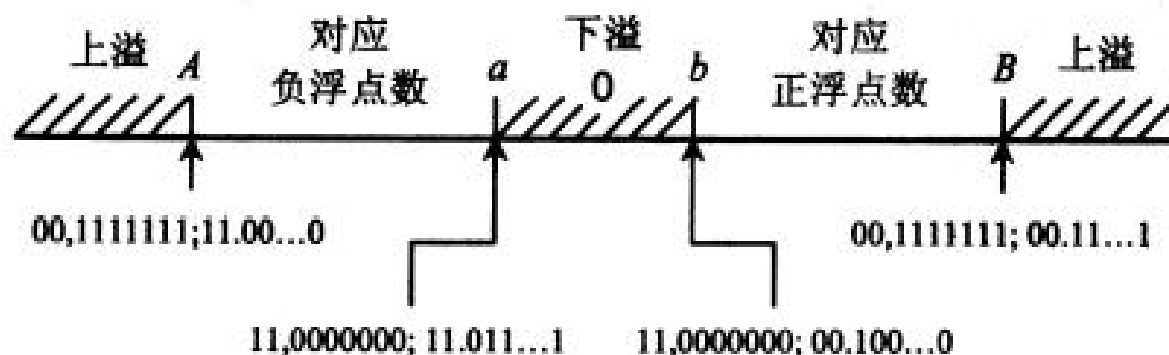
浮点数的表示范围和精度

- ① 一旦浮点数的位数确定以后，合理分配阶码和尾数的位数，直接影响浮点数的表示范围和精度。
 - ① 阶码越长，表示范围越大；
 - ② 尾数越长，表示精度越高。
- ② 基数对数的表示范围也有影响。
 - 一般来说，基数 r 越大，可表示的浮点数范围越宽，而且所表示的数的个数也越多。但 r 越大，浮点数的精度反而下降。
 - 例如： $r=16$ 的浮点数，引起规格化的尾数最高三位可能出现零，故与其尾数位数相同的 $r=2$ 的浮点数相比，后者可能比前者多三位精度。
 - 基数是隐含的，浮点机中一旦基数确定后就不再变了。

溢出判断



- 变形补码表示：若机器数为补码，尾数为规格化形式，并假设阶符取2位，阶码取7位，数符取2位，尾数取n位，则它们能表示的补码在数轴上的表示范围：



- 在浮点规格化中已指出，当尾数出现 $01.\times\times\dots\times$ 或 $10.\times\times\dots\times$ 时，并不表示溢出，只有将此数右规后，再根据阶码的符号来判断浮点运算结果是否溢出。即
阶码 $[j]_{\text{补}}=01$ ， $\times\times\dots\times$ 为上溢。
阶码 $[j]_{\text{补}}=10$ ， $\times\times\dots\times$ 为下溢，按机器零处理



3. 定点数与浮点数的比较

- ① 数的表示范围：当浮点机和定点机中的数的位数相同时，浮点数的表示范围比定点数大得多。
 - ② 数的精度：当浮点数为规格化数时，其精度比定点数高。
 - ③ 浮点运算步骤比定点运算多，运算速度比定点低，运算线路比定点复杂。
 - ④ 溢出判断
 - 定点数的溢出——根据数值本身判断
 - 浮点数的溢出——根据规格化后的阶码判断
-
- 通用的大型机采用浮点数，或同时采用定、浮点数；
 - 小型、微型及某些专用机、控制机采用定点数。
 - 当需作浮点运算时，可通过软件实现，也可通过外加的浮点扩展硬件（如协处理器）来实现。



例题1

设某机器用32位表示一个实数，阶码部分8位（含1位阶符），用定点整数补码表示；尾数部分24位（含数符1位），用规格化定点小数补码表示，基数为2。

1. 求 $X=256.5$ 的浮点表示格式

$$X=(256.5)_{10}=+(100000000.1)_2=+(0.1000000001 \times 2^{+9})_2$$

8位阶码为： $(+9)_{\text{补}}=0,000\ 1001$

24位尾数为： $(+0.10\ 0000\ 0001)_{\text{补}}$

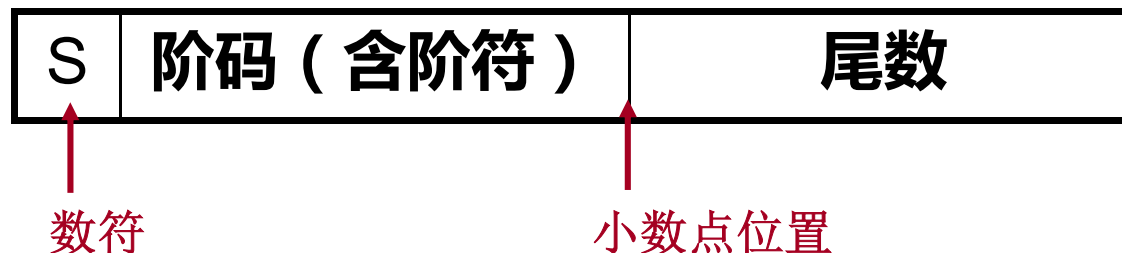
$$=0.100\ 0000\ 0010\ 0000\ 0000\ 0000$$

所求256.5的浮点表示格式为：

0,000 1001; 0.100 0000 0010 0000 0000 0000



IEEE754标准格式



- 现代计算机中浮点数采用的标准。
- 按IEEE标准，常用的浮点数有三种

| | 符号位S | 阶码 | 尾数 | 总位数 |
|-----------|------|----|----|-----|
| 短实数/单精度 | 1 | 8 | 23 | 32 |
| 长实数/双精度 | 1 | 11 | 52 | 64 |
| 临时实数/扩展精度 | 1 | 15 | 64 | 80 |



IEEE754标准格式

- 阶码用移码，阶码的真值都被加上一个常数（偏移量），如短实数、长实数和临时实数的偏移量分别用16进制表示为7FH、3FFH、3FFFH。
- 尾数部分通常用规格化表示，即非“0”的有效位最高位总是“1”。
- 但在IEEE标准中，尾数有效位呈如下形式：

1▲ffff...fff

其中▲表示假想的二进制小数点。在实际表示中，对于短实数和长实数，这个整数位的“1”被省略，称为“隐藏位”；对临时实数不采用隐藏位方案。



小结

- **数据的表示方法和转换**
 - 二进制、八进制、十进制、十六进制
- **无符号数和有符号数**
 - 原码、补码、反码、移码
- **数的定点表示和浮点表示**
- **作业：6.1 , 6.5 , 6.6 , 6.12**



休息是为了走更远的路！