

第六章 数据的机器级表示

数据的机器级表示

- 使用高级程序设计语言——C语言求解问题
 - 最基本的介绍（二-五章）
- 当一个C程序执行时，在计算机内部究竟发生了什么？
 - 需要从最底层——计算机内表示数值的方式开始

位和数据类型

数字

- 在现代计算机里，所有信息都是采用**数字化**的形式表示的
- 整数、小数、文字、图像、声音等等

信息的最小单位——位/比特

- 在计算机内部，数以亿计、微小、快速的电子元件控制电子的流动
 - 这些元件对电路中**电压的有无**做出反应
- 如果存在电压用“1”表示，不存在电压用“0”表示
 - “0”和“1”被称为**比特**（bit），或**位**
 - “**二进制位**”（binary digit）的缩写

两种稳定状态

- 自然界的事物通常具有两种稳定的状态
 - 存储信息的**磁盘**
 - 使用1和0表示磁化和未磁化
 - **光盘**
 - 使用1和0表示凹（聚光）和凸（散光）

接近0和远离0

- 计算机并不是区分电压的绝对不存在（即0）和绝对存在（即1）
- 计算机电路区分的是接近0的电压和远离0的电压
 - 如果计算机把3.3伏的电压表示为1，把0伏的电压表示为0，那么2.9伏的电压也会被视作1，而0.2伏的电压会被当作0

二进制 对 十进制

- 如果采用十进制，就需要测量电压的具体值
 - 不仅比测量有和无要复杂
 - 而且要求电路的电压值必须稳定，不允许从3.3伏波动到2.9伏

识别多个数值

- 计算机要解决一个真正的问题，必须能唯一的识别出许多不同的数值，而不仅仅是0和1
- 为了唯一的识别出多个数值，必须对**多个位进行组合**

位组合

- 如果用8位（对应8根线路上的电压）
 - 使用01001110表示某一个特定值，用11100111表示另一个值
 - 最多能区分出256（即 2^8 ）个不同的值
- 如果有 k 位，最多能区分出 2^k 个不同的值
 - 这些 k 位的每一种组合都是一个编码，对应着某个特定的值

位运算

- 除了需要表示出不同的数值之外，还需要对这种表示出来的信息进行运算
 - 1679年，德国数学家莱布尼茨发表了一篇关于二进制表示及**算术运算**的论文
 - 英国数学家乔治·布尔在1854年给出了二进制的**逻辑运算**，布尔代数即由此得名
 - 这些工作奠定了现代计算机工作的**数学基础**

数据类型

- 同一个数值，多种表示方法
 - 整数6
 - 十进制计数法，“6”
 - 一元计数法，“正一”
 - 罗马字符，vi
 - 二进制计数法，“0110”
- 某种表示法，编码，运算
 - 数据类型

计算机中的数据类型

- 如果在计算机上能对以某种表示法编码的信息进行运算，这种特殊的表示法就可以被称为**数据类型**
- 在大多数计算机上存在着多种数值表示方法
 - 用来表示进行算术运算的正负整数的**二进制补码整数**
 - 用来表示从键盘输入计算机或显示在计算机显示器上的字符的**ASCII码**
 - 类似于十进制“科学计数法”的**浮点数数据类型**

整数数据类型

无符号整数

- 任务执行的次数
- 课程的选课人数
-

位置计数法——十进制

- 十进制系统
 - 十进制数286
 - 2在286中的位置决定了它表示200 (2×10^2)，而8则表示 8×10^1
- 位置计数法，或**定位数制**
- 在十进制系统里，10被称为数制中的**基数**或基

位置计数法——二进制无符号整数

- 采用位置计数法，二进制数表示无符号整数，基数为2，二进制数为0和1
 - 如果使用8位有效数字来表示数值，则数字30可以表示为00011110
 - $0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

二进制无符号整数

- 使用2位数
00, 01, 10, 11
- 使用3位数
000, 001, 010, 011, 100, 101, 110, 111
- 使用 k 位数, 就可以表示从0到 2^k-1 共 2^k 个整数

有符号整数

- 将 k 位的 2^k 个不同的数字分为两半，一半表示正数，另一半表示负数
- $k=4$, $2^k=16$
 - 表示从+1到+7的正数，以及从-1到-7的负数——14个
 - 剩下2个
 - 0和?
 - 首先考虑：从+1到+7，从-1到-7，到底是哪些码字与其一一匹配？

正数

- 正数按照位置计数法直接表示
- k 位，用 2^k 个码字的一半来表示从0到 $2^{k-1}-1$ 的正数，最高位有一个0

$k=4$ 的正数

- 最大的正数：7
- 以0开头

二进制表示	十进制
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

负数——原码

- $k=4$, $-1 \sim -7$?
- 最高位0: 正数 (+)
- 最高位1: 对应的负数 (-)

二进制表示	十进制
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

负数——反码

- $k=4$, $-1 \sim -7$?
- 对正数“按位取反”

二进制表示	十进制
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-7
1001	-6
1010	-5
1011	-4
1100	-3
1101	-2
1110	-1
1111	-0

4+(-3)

- 采用与十进制加法相同的规则
 - 自右向左，一列一列的运算，如果某列的加法有进位，则立即加至它的左列

- 原码表示法

$$\begin{array}{r} 0100 \ (4) \\ + \ 1011 \ (-3) \\ \hline = \ 1111 \ (-7) \end{array}$$

- 反码表示法

$$\begin{array}{r} 0100 \ (4) \\ + \ 1100 \ (-3) \\ \hline = \ 0000 \ (0) \end{array}$$

负数——补码

- $k=4$
- $0 \sim +7$
 - 位置记数法
- $-8 \sim -1$
 - 尽可能使逻辑电路最简单

二进制表示	十进制
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

算术逻辑单元

- 对负数表示法的选择是基于尽可能使逻辑电路最简单的想法
- 几乎所有的计算机都使用相同的基本结构——算术逻辑单元 (Arithmetic and Logic Unit, **ALU**) 来进行加法运算
- ALU并不知道 (也不关心) 所加的两个位组合表示什么, 它只是简单的将二进制数相加

$$\begin{array}{r} 0100 \quad (4) \\ + \quad \underline{1101} \quad (-3) \\ \hline = \quad 0001 \quad (1) \end{array}$$

$$A + (-A) = 0$$

$$\begin{array}{r}
 0101 \ (+5) \\
 + \textcolor{red}{?} \ (-5) \\
 \hline
 = 0000 \ (0)
 \end{array}$$

$$\begin{array}{r}
 0101 \ (+5) \\
 + \textcolor{red}{1011} \ (-5) \\
 \hline
 = 0000 \ (0)
 \end{array}$$

二进制表示	十进制
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

1000 ?

- 在对每个数值加0001后，应得到正确的结果
- 1000: -8

$$\begin{array}{r} 1000 (?) \\ + \quad 0001 (1) \\ \hline = \quad 1001 (-7) \end{array}$$

二进制表示	十进制
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

1111、0000、0111、1000

- 1111和0000：-1和0

$$\begin{array}{r} 1111 \text{ (-1)} \\ + \quad 0001 \text{ (1)} \\ \hline = \text{(1)0000 (0)} \end{array}$$

- 在做补码算术运算时
 - 这个进位总是被忽略
- 0111：+7
- 1000：-8

二进制表示	十进制
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

K位组合

- $-2^{k-1} \sim 2^{k-1}-1$
 - 0后面跟k-1个1: $2^{k-1}-1$
 - 1后面跟k-1个0: -2^{k-1}
 - k个1: -1
- 32位ALU
 - 二进制补码类型整数从-2147483648到2147483647

-A的表示

$$\begin{array}{r}
 A \\
 + \quad \underline{A \text{ 的反码}} \\
 \hline
 = \quad 11 \cdots 11 \quad (-1) \\
 + \quad \underline{00 \cdots 01 \quad (1)} \\
 \hline
 = \quad 00 \cdots 00 \quad (0)
 \end{array}$$

●-A的表示

- 把A的反码加1
- “取反加1”

示例

- -6的二进制补码表示是什么（采用4位表示）？

1. A: +6, 0110

2. A的反码 1001

3. 1001

+ 0001

= 1010 (-6)

- 验证

0110 (6)

+ 1010 (-6)

= (1)0000

二进制-十进制

- 一个8位的二进制补码数采取如下格式：

- $a_7 \ a_6 \ a_5 \ a_4 \ a_3 \ a_2 \ a_1 \ a_0$

- 1. 检查最前面的 a_7 。如果是0，该整数是正数，就可以直接计算其数值。如果是1，该整数是负数，“取反加1”。
 - 2. 计算：
$$a_6 \times 2^6 + a_5 \times 2^5 + a_4 \times 2^4 + a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$$
 - 3. 如果原数值是负数，在前面加一个负号前缀即可

示例

$$\begin{aligned} X &= 01110110_{\text{two}} \\ &= 2^6 + 2^5 + 2^4 + 2^2 + 2^1 = 64 + 32 + 16 + 4 + 2 \\ &= 118_{\text{ten}} \end{aligned}$$

$$\begin{aligned} X &= 11110010_{\text{two}} \\ -X &= 00001110 \\ &= 2^3 + 2^2 + 2^1 = 8 + 4 + 2 \\ &= 14_{\text{ten}} \\ X &= -14_{\text{ten}} \end{aligned}$$

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

十进制-二进制

- 如果一个**正**的二进制数的最右端的数字为1，这个数为**奇数**；否则为**偶数**
- 8位二进制数（正数）
$$a_6 \times 2^6 + a_5 \times 2^5 + a_4 \times 2^4 + a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$$
- 如何找到 a_i （ $i = 0, 1, \dots, 6$ ）的值？

示例: +123

- 正数, 最高位 a_7 : 0

$$123 = a_6 \times 2^6 + a_5 \times 2^5 + a_4 \times 2^4 + a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$$

- 奇数, a_0 : 1
- 在等式两端同时减去1

$$122 = a_6 \times 2^6 + a_5 \times 2^5 + a_4 \times 2^4 + a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2^1$$

- 在等式两端同时除以2

$$61 = a_6 \times 2^5 + a_5 \times 2^4 + a_4 \times 2^3 + a_3 \times 2^2 + a_2 \times 2^1 + a_1 \times 2^0$$

- 奇数, a_1 : 1
- 在等式两端同时减去1

$$60 = a_6 \times 2^5 + a_5 \times 2^4 + a_4 \times 2^3 + a_3 \times 2^2 + a_2 \times 2^1$$

- 在等式两端同时除以2

$$30 = a_6 \times 2^4 + a_5 \times 2^3 + a_4 \times 2^2 + a_3 \times 2^1 + a_2 \times 2^0$$

- $a_2=0$

$$15 = a_6 \times 2^3 + a_5 \times 2^2 + a_4 \times 2^1 + a_3 \times 2^0$$

- $a_3=1$

$$7 = a_6 \times 2^2 + a_5 \times 2^1 + a_4 \times 2^0$$

- $a_4=1$

$$3 = a_6 \times 2^1 + a_5 \times 2^0$$

- $a_5=1$

$$1 = a_6 \times 2^0$$

- $a_6=1$

- 二进制表示为 0111 1011

除2取余

2	123	余数	
2	61	1	低位
2	30	1	
2	15	0	
2	7	1	
2	3	1	
2	1	1	
	0	1	高位

- 将余数从高位向低位依次排列
- 1111011
- 正数, 最高位为0
- 二进制表示为 0111 1011

总结

- 已知十进制数 N ，通过以下步骤可得其 k 位补码表示
 - 1. 首先将 N 的绝对值通过“除2取余”的方法，得到其绝对值的二进制表示；
 - 2. 如果原来的十进制数为正，则在二进制数前加0，得到 k 位结果；
 - 3. 如果原来的十进制数为负，在二进制数前加0，得到 k 位，然后再计算出这个补码的负数（即“取反加1”），得到结果。

示例

2		215		余数
2		107		1
2		53		1
2		26		1
2		13		0
2		6		1
2		3		0
2		1		1
		0		1

结果 $(215)_{10} = (0000000011010111)_2$
或写为: $215\text{D} = 0000000011010111\text{B}$

注: 使用16位编码

算术运算

- 与十进制运算十分相似
- 加法运算
 - 从右到左进行，每次一位。在每次运算后，产生一个“和”与一个“进位”，进位在1的后面产生

$$\begin{array}{r} 0011 \quad (3) \\ + \quad 0100 \quad (4) \\ \hline 0111 \quad (7) \end{array}$$

$$\begin{array}{r} 0111 \quad (7) \\ + \quad \quad \quad (-4) \\ \hline \quad \quad \quad (3) \end{array}$$

减法运算

$$\begin{array}{r} 0111 \text{ (7)} \\ - 0100 \text{ (4)} \\ \hline \end{array}$$

$$\begin{array}{r} 0111 \text{ (7)} \\ + \underline{1100 \text{ (-4)}} \\ (1) \text{ } \mathbf{0011} \text{ (3)} \end{array}$$

$$x + x$$

- 把一个数 x 加上它自身
- 每位上的数字都向**左移**了一位

61+61

- 61可以表示为

$$0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

- $61 + 61 = 2 \times 61$, 可以表示为

$$2 \times (0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0)$$

- 也就是

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

- 每位上的数字都向左移了一位

恰当的位数

- 为减少占用空间，会采用恰当的位数来表示数值
 - 6
 - 用4位 (0110)
 - 用16位 (0000 0000 0000 0110)
 - -6
 - 用4位 (1010)
 - 用16位 (1111 1111 1111 1010)

不同长度数值做加法

- 为了对两个具有不同长度的数值做加法，首先必须将它们表示为相同的长度

$$\begin{array}{r} 00000000000001110 \\ + \quad \quad \quad 1100 \\ \hline \end{array}$$

?

14+ (-4)

$$\begin{array}{r} (14) \\ + (12) \\ \hline 0000 0000 0001 (26) \end{array}$$

$$\begin{array}{r} (14) \\ + (-4) \\ \hline 0000 0000 0000 (10) \end{array}$$

符号扩展

- 如果用0来扩展一个正数的左端，它的值不会改变
- 如果用1来扩展一个负数的左端，其值亦不会改变
- 在这两种情况中扩展的都是符号位，这种运算被称为符号扩展（Sign-EXTension, **SEXT**）
- 用于对不同长度的数值之间的运算

溢出

- 使用4位补码数据类型，计算2+6:

$$\begin{array}{r} 0010 \\ + 0110 \\ \hline 1000 \end{array}$$

- 计算结果为-8，**为什么**会出现错误？
 - 2加6等于8，大于+7，即大于0111，而0111是使用4位的补码数据类型能够表示的最大正数，因此8不能用4位的补码表示出来

溢出示例

$$\begin{array}{rcl} & 0011 & (3) \\ + & \underline{0110} & (6) \\ & 1001 & (-7) \end{array}$$

$$\begin{array}{rcl} & 1101 & (-3) \\ + & \underline{1010} & (-6) \\ & 0111 & (7) \end{array}$$

检测溢出

- 如果
 - 两个数符号相同
 - 和的符号不同
- 一个负数和一个正数的和永远不会出现溢出

习题 (1)

- 书面作业

- 6.3

- 6.4

- 6.5

- 6.7

- 6.8

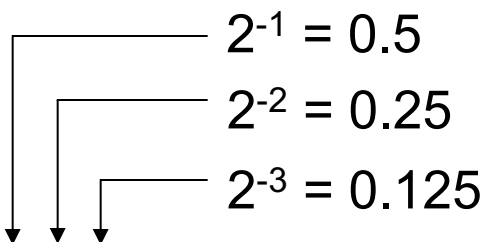
浮点数数据类型

小数——定点小数

- 十进制，小数点，定点表示法（*fixed point*）
 - $3456.78 = 3 \cdot 10^3 + 4 \cdot 10^2 + 5 \cdot 10^1 + 6 \cdot 10^0 + 7 \cdot 10^{-1} + 8 \cdot 10^{-2}$
- 二进制
 - $00011001.110 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} \Rightarrow 25.75$
($2^{-1} = 0.5$ and $2^{-2} = 0.25$, etc.)

定点小数算术运算

- 没有新的运算——补码整数算术运算
- 小数点对齐


$$\begin{array}{r} 00101000.101 \quad (40.625) \\ + \quad 11111110.110 \quad (-1.25) \\ \hline 00100111.011 \quad (39.375) \end{array}$$

十进制数转换成二进制数

- (1)整数部分，采用除2取余法（倒除法）
- (2)小数部分，采用乘2取整法

示例

例如：将 $(0.6875)_{10}$ 转换成二进制数

	取整数部分
$\begin{array}{r} 0.6875 \\ \times 2 \\ \hline 1.3750 \\ 0.3750 \end{array}$	1
$\begin{array}{r} 0.3750 \\ \times 2 \\ \hline 0.7500 \end{array}$	0
$\begin{array}{r} 0.7500 \\ \times 2 \\ \hline 1.5000 \\ 0.5000 \end{array}$	1
$\begin{array}{r} 0.5000 \\ \times 2 \\ \hline 1.0000 \\ 0.0000 \end{array}$	1

结果 $(0.6875)_{10} = (0.1011)_2$

精度

- 如果十进制小数不能用有限位的二进制数表示，则
- 根据精度取几位
 - 例如： $(0.414)_{10} \approx (0.01101)_2$ （取5位）
 - 或写为： $0.414D \approx 0.01101B$ （取5位）

数值范围

- 阿佛加德罗常数 6.023×10^{23} ?
- 32位补码整数
 - $-2147483648 \sim 2147483647$
 - $-2^{31} \sim +2^{31}-1$
 - 精度31位，数值范围 2^{31}

精度 对 数值范围

- 6.023×10^{23}
 - 数值范围——需要79位（二进制补码整数）
 - 精度——只需要4个十进制数（6023）
- 6.626×10^{-34}
 - 需要>110位
- 问题：表示精度的位数过多，而表示范围的位数却不足

浮点数类型

- 类似于科学记数法
- 6.023×10^{23}
 - 符号： +
 - 有效数字： 6.023
 - 分数
 - 指数： 23

浮点数类型

- 十进制：科学计数法
 - $1.602\ 176\ 462 \times 10^{-19}$
 - $1 \times 10^{85} \text{ cm}^3$
- 二进制：浮点数
 - 定点数
 - $00011001.110 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2}$
 $\Rightarrow 25.75$
 - 浮动小数点
 - $00011001.110 \Rightarrow 1.1001110 \times 2^4$
 - 需要表示出：符号、指数（范围）、分数（精度）
 - IEEE（国际电气和电子工程师协会）浮点数算术运算标准

IEEE-754 单精度 (*float*) 浮点数



- 32 位
- 符号 (S)
 - 0代表正数, 1代表负数
- 指数: 8位
 - 无符号整数, 0 ~ 255
- 分数: 23 位

IEEE 浮点数算术运算标准

$$N = \begin{cases} (-1)^s \times 1.\textit{fraction} \times 2^{\textit{exponent}-127}, & 1 \leq \textit{exponent} \leq 254 \\ (-1)^s \times 0.\textit{fraction} \times 2^{-126}, & \textit{exponent} = 0 \\ \textit{Infinity}(+and-), & \textit{exponent} = 255, \textit{fraction} = 0 \\ \textit{NaN}(\textit{not a number}), & \textit{exponent} = 255, \textit{fraction} \neq 0 \end{cases}$$

- 符号位: $(-1)^s$
- 指数和分数
 - $1 \leq \textit{exponent} \leq 254$
 - $\textit{exponent} = 0$
 - $\textit{exponent} = 255$

$$1 \leq \text{exponent} \leq 254$$

$$N = (-1)^S \times 1.\text{fraction} \times 2^{\text{exponent} - 127}$$

- 指数：8 位
 - 无符号整数，表示从0到255
 - 为了表示出负数，实际上被表示的指数是该无符号整数减去127后得到的数值：
 - $2^8 \Rightarrow 10000111$ (= 135)
 - $2^{-125} \Rightarrow 00000010$ (= 2)
- 分数：23 位
 - 规格化，实际表示的是24位的精度
 - 包括23位分数和二进制小数点左端的没有明确表示出来的一位1

exponent=00000000

$$N = (-1)^S \times 0.\textit{fraction} \times 2^{-126}$$

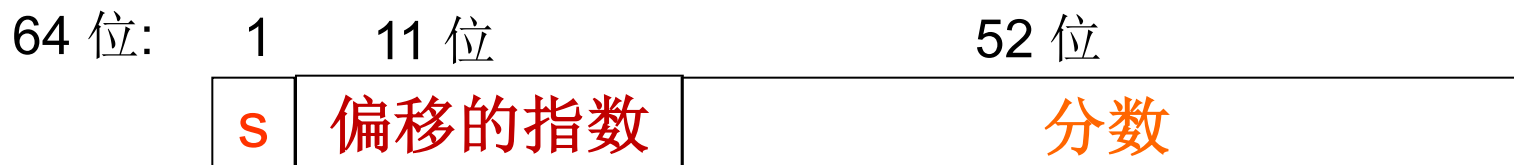
- 指数是-126，得到的有效数字以0开头，后面是23位二进制分数部分
 - 0 00000000 000010000000000000000000
 - 开头的0表示它是正数
 - 接下来的8位，是一个零指数，意味着它的指数是-126
 - 后面的23位形成0. 000010000000000000000000，即 2^{-5}
 - $2^{-5} \times 2^{-126}$ ，得到 2^{-131}
- 这样就能表示很小的数
 - 当分数部分也全为0时，此时将表示数值0

exponent=11111111

$$N = \begin{cases} \text{Infinity}(+and-), & \text{fraction} = 0 \\ \text{NaN(not a number)}, & \text{fraction} \neq 0 \end{cases}$$

- 如果符号位为0，分数域全为0，那么该数表示**正无穷**
 - C语言中，计算“10.0/0”时出现
- 如果符号位为1，分数域全为0，那么该数表示**负无穷**
 - -10.0/0
- 如果分数域不全为0，那么该数表示“**非数值**”
 - 0.0/0.0
 - 0.0*inf
 - inf/inf
 - inf-inf

IEEE 双精度浮点数



$$N = (-1)^s \times 1.\text{fraction} \times 2^{(\text{biased exp.} - 1023)}$$

- **double precision (64 bit)**

- **范围 和 精度:**

- ◆ **32 位:**

- 精度=> 大约 7 位
- 范围=> 大约 $10^{+/-38}$

- ◆ **64 位:**

- 精度=> 大约 15 位
- 范围=> 大约 $10^{+/-306}$

-45.8125

- 45.8125的二进制数表示为：0101101.1101
- 规格化为 1.011011101×2^5 ，即
 $1.011011101 \times 2^{132-127}$
- 符号位应为1，负数
- 指数部分为10000100，即无符号整数132
- 分数部分是小数点后的23位，
01101110100000000000000

1 10000100 0110 1110 1000 0000 0000 000

示例

- 0 01111010 000000000000000000000000
- 最高位为0，表示该数是个正数
- 跟着的8位代表无符号整数122，减去127，得到实际指数为-5
- 最后23位都是0
- 因此这个数表示为
 $+1.000000000000000000000000 \times 2^{-5}$ ，也就是
 $1/32$

示例

- 0 10000101 111000011110000000000000
- 指数域
 - 无符号整数133，由于 $133-127=6$ ，所以指数为+6
- 将分数域的小数点左边加一个1形成
1. 11100001111
- 将小数点向右移动6位，得到1111000. 01111，
即120. 46875

示例

- 1 10000010 001010000000000000000000
- 符号位为1，表示负数
- 指数是130，表示130-127，即指数为3
- 将分数域的小数点左边加一个1形成1.00101
- 将小数点向右移动3位，得到1001.01，即-9.25

示例

- 0 11111110 111111111111111111111111
- 符号为正
- 指数是 $254-127$ ，即 $+127$
- 将分数域的小数点左边加一个1形成
1.111111111111111111111111111111，约等于2
- 因此，结果约为 2^{128}

示例

- 1 00000000 000000000000000000000001
- 符号为负
- 指数域全为0，表示指数是-126
- 将分数域的小数点左边加一个0形成 2^{-23}
- 因此，结果为 $2^{-23} \times 2^{-126}$ ，等于 -2^{-149}

加法运算

- 与科学计数法类似
- 加法运算，需经过五步完成
 - 1、对指数操作：首先使二数的指数相等
 - 2、分数运算：经指数相等操作后，即可直接对分数做加法运算
 - 3、结果规格化：对运算结果进行规格化处理
 - 4、舍入操作：对丢失的位进行舍入处理
 - 5、判断溢出：判断指数是否溢出

十六进制表示法

十六进制表示法

- 由二进制的位置计数法发展而来
- 方便人们手工处理二进制数位

二进制-十六进制

- 16位二进制位组合（无符号整数）

$a_{15} \ a_{14} \ a_{13} \ a_{12} \ a_{11} \ a_{10} \ a_9 \ a_8 \ a_7 \ a_6 \ a_5 \ a_4 \ a_3 \ a_2 \ a_1 \ a_0$

- 计算如下：

$$\begin{aligned} &2^{15} \times a_{15} + 2^{14} \times a_{14} + 2^{13} \times a_{13} + 2^{12} \times a_{12} + 2^{11} \times a_{11} + 2^{10} \times a_{10} + 2^9 \times a_9 \\ &+ 2^8 \times a_8 + 2^7 \times a_7 + 2^6 \times a_6 + 2^5 \times a_5 + 2^4 \times a_4 + 2^3 \times a_3 \\ &+ 2^2 \times a_2 + 2^1 \times a_1 + 2^0 \times a_0 \end{aligned}$$

$$\begin{aligned} = &2^{12} \times [2^3 \times a_{15} + 2^2 \times a_{14} + 2^1 \times a_{13} + 2^0 \times a_{12}] + 2^8 \times [2^3 \times a_{11} \\ &+ 2^2 \times a_{10} + 2^1 \times a_9 + 2^0 \times a_8] + 2^4 \times [2^3 \times a_7 + 2^2 \times a_6 + 2^1 \times a_5 \\ &+ 2^0 \times a_4] + 2^0 \times [2^3 \times a_3 + 2^2 \times a_2 + 2^1 \times a_1 + 2^0 \times a_0] \end{aligned}$$

$$= 16^3 \times h_3 + 16^2 \times h_2 + 16^1 \times h_1 + 16^0 \times h_0$$

十六进制

$$h_3 = 2^3 \times a_{15} + 2^2 \times a_{14} + 2^1 \times a_{13} + 2^0 \times a_{12}$$

$$h_2 = 2^3 \times a_{11} + 2^2 \times a_{10} + 2^1 \times a_9 + 2^0 \times a_8$$

$$h_1 = 2^3 \times a_7 + 2^2 \times a_6 + 2^1 \times a_5 + 2^0 \times a_4$$

$$h_0 = 2^3 \times a_3 + 2^2 \times a_2 + 2^1 \times a_1 + 2^0 \times a_0$$

• h_3 、 h_2 、 h_1 、 h_0 的值

- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

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

二进制-十六进制

- 01101100010111110011110101101110
- 首先，将这个位组合按照每四位进行分割
- 0110 1100 0101 1111 0011 1101 0110 1110
- 然后，将每四位转换为相等的十六进制数
- 6 C 5 F 3 D 6 E
- 其十六进制表示为 “**x**6C5F3D6E”

十六进制-二进制

- 将十六进制转换为二进制，则只需将每一位十六进制转换为对应的4位二进制组合即可
 - 十六进制 “x41”
 - 二进制表示 “0100 0001”

十六进制-十进制

- 十六进制数xE20A?
- 如果十六进制数E20A表示一个16位的**二进制补码整数**
- 首先将E20A的每个16进制符号转换为二进制，即1110 0010 0000 1010
- 通过这个二进制表示，就可以知道该整数是负数，因为最高位为1
- 再转换为十进制

注意

- 十六进制计数法的采用方便了人们对数据的表达
- 它能表达：整数、浮点数……
 - IEEE-754 fp numbers
 - $25.75 \Rightarrow$ 0 1000 0011 100 1110 0000 0000
0000 0000 \Rightarrow x41CE0000
 - $-0.75 \Rightarrow$ 1 01111110 100000000000000000000000
 \Rightarrow xBF400000

八进制数 (octal)

- 八进制数与二进制数之间的转换
 - 一位八进制数相当于3位二进制数，所以八进制数转换成二进制数，或二进制数转换成八进制数很方便
- 例如： $(563)_8 = (101, 110, 011)_2$
 $(0.764)_8 = (0.111, 110, 100)_2$

习题(2)

- 书面作业
 - 6. 10
 - 6. 11
 - 6. 12
 - 6. 13

ASCII码

ASCII

- 如何表示从键盘输入计算机或显示在显示器上的字符？
- American Standard Code for Information Interchange
- [æski], 美国信息交换标准码, 美国国家标准局制定

ASCII

- 键盘上的每个键被一个唯一的ASCII码所识别
- 8个二进制位表示
- 在键盘上敲击某个键时，相应的8位码被存储，并提供给计算机

标准ASCII码

D: 十进制表示
H: 十六进制表示

字符	ASCII		字符	ASCII		字符	ASCII		字符	ASCII	
	D	H		D	H		D	H		D	H
NUL	0	00	SP	32	20	@	64	40	`	96	60
SOH	1	01	!	33	21	A	65	41	a	97	61
STX	2	02	"	34	22	B	66	42	b	98	62
ETX	3	03	#	35	23	C	67	43	c	99	63
EOT	4	04	\$	36	24	D	68	44	d	100	64
ENQ	5	05	%	37	25	E	69	45	e	101	65
ACK	6	06	&	38	26	F	70	46	f	102	66
BEL	7	07	'	39	27	G	71	47	g	103	67
BS	8	08	(40	28	H	72	48	h	104	68
HT	9	09)	41	29	I	73	49	i	105	69
LF	10	0A	*	42	2A	J	74	4A	j	106	6A
VT	11	0B	+	43	2B	K	75	4B	k	107	6B
FF	12	0C	,	44	2C	L	76	4C	l	108	6C
CR	13	0D	-	45	2D	M	77	4D	m	109	6D
SO	14	0E	.	46	2E	N	78	4E	n	110	6E
SI	15	0F	/	47	2F	O	79	4F	o	111	6F
DLE	16	10	0	48	30	P	80	50	p	112	70
DC1	17	11	1	49	31	Q	81	51	q	113	71
DC2	18	12	2	50	32	R	82	52	r	114	72
DC3	19	13	3	51	33	S	83	53	s	115	73
DC4	20	14	4	52	34	T	84	54	t	116	74
NAK	21	15	5	53	35	U	85	55	u	117	75
SYN	22	16	6	54	36	V	86	56	v	118	76
ETB	23	17	7	55	37	W	87	57	w	119	77
CAN	24	18	8	56	38	X	88	58	x	120	78
EM	25	19	9	57	39	Y	89	59	y	121	79
SUB	26	1A	:	58	3A	Z	90	5A	z	122	7A
ESC	27	1B	;	59	3B	[91	5B	{	123	7B
FS	28	1C	<	60	3C	\	92	5C		124	7C
GS	29	1D	=	61	3D]	93	5D	}	125	7D
RS	30	1E	>	62	3E	^	94	5E	~	126	7E
US	31	1F	?	63	3F	_	95	5F	DEL	127	7F

标准ASCII码

- 第0~32 (20_H) 号及第127 (7F_H) 号 (共34个) 是**控制字符或通讯专用字符**，其中第32号 (20_H) 是空格 (Space)
- 第33~126号 (共94个) 是**可见字符**，包括了阿拉伯数字，英文字母，英文标点等，可以通过标准键盘直接输入

特点

- 阿拉伯数字与其ASCII码之间的关系？
- 大写字母与相应的小写字母ASCII码之间的关系？
- 字母表顺序与ASCII码之间的关系？
- 其他字符？
 - <http://www.unicode.org/>

没有新的运算- 整数算术和逻辑运算

其他数据类型

- 字符串

- 字符序列，以 **NUL (0) 结束**

- 图像

- 像素矩阵

- 黑白: 1位 (1/0 = black/white)
 - 彩色: red, green, blue (RGB) (每个8位)
 - (0, 0, 0) black, (255, 0, 0) red, (255, 255, 255) white
 - 其他属性: 透明度

- 硬件支持

- MMX

- 声音

- 定点数序列

C语言中的数据类型

C语言中的数据类型——深入理解

- `int`，二进制补码整数类型
- `char`，ASCII码
- `double`，双精度浮点数
- 每种类型采用多少位二进制来表示，与具体计算机的指令集结构和编译器有关

char

- char 类型采用ASCII码表示
- 如果key是char类型的变量：
 - `(('a' <= key) && (key <= 'z')) || (('A' <= key) && (key <= 'Z'))`
- 表达式 “`'a' <= key`” 中的 “`<=`” 运算符比较的就是变量key和字符a的**ASCII码的大小**
 - **整数运算**

混合类型表达式

- 算术运算表达式
 - “ $i + 3.1$ ”，其中 i 被声明为`int`类型， 3.1 是浮点型字面常量
 - 将整数转换为浮点数，然后进行计算
- 整数与字符型运算
 - 字符型被转化为整数类型后再进行计算
 - “ $x + 'a'$ ”，如果`int`类型变量 x 取值为 1 ，字符`'a'`的ASCII码为 97 ，则计算“ $1 + 97$ ”，表达式的值为 98

输入输出的格式说明

- 格式说明 “%d”
- 输出
 - 它使得列在格式用字符串后面的数值被显示为十进制数输出
 - 事实上，是将一个存储的二进制补码整数**转化**为ASCII码字符输出

```
printf ("25 plus 76 in decimal is %d. \n", 25 + 76);  
25 plus 76 in decimal is 101.
```

- 输入
 - 它把从键盘输入的数字解释为十进制数值
 - 事实上，是将输入字符的ASCII码**转化**为一个二进制补码整数存储

%X

- 格式说明 “%x”
- 输出
 - 将数值以十六进制数的形式被显示出来
 - `printf ("25 plus 76 in hexadecimal is %x. \n", 25 + 76);`
 - 25 plus 76 in hexadecimal is 65.
- 输入
 - 它把从键盘输入的数字解释为十六进制数值
 - 事实上，将输入字符的ASCII码转化为一个二进制补码整数存储
 - `scanf ("%x", &valueX);`
 - 输入: A, A0, a, ae, 10, 0XA, 0x10
 - 存储: 10, 160, 10, 174, 16, 10, 16

%c

- 格式说明 “%c”
- 输出
 - 将该数值直接解释为ASCII字符显示
 - `printf ("25 plus 76 as a character is %c.\n", 25 + 76);`
 - 25 plus 76 as a character is **e**.
- 输入
 - `scanf ("%c", &grade);`
 - 事实上，这个过程是将输入的字符的ASCII码进行存储

输出示例： %d, %x, %o, %c

```
printf ("25 plus 76 in decimal is %d. \n", 25 + 76);
```

```
printf ("25 plus 76 in hexadecimal is %x. \n", 25 + 76);
```

```
printf ("25 plus 76 in octal is %o. \n", 25 + 76);
```

```
printf ("25 plus 76 as a character is %c. \n", 25 + 76);
```

- %d: 将“25 + 76”的结果以十进制数的形式显示出来
 - 将二进制“0110 0101”转换为“101”三个字符显示
- %x: 将“25 + 76”的结果以十六进制数的形式显示出来
 - 将二进制“0110 0101”转换为“65”两个字符显示
- %o: 将“25 + 76”的结果以八进制数的形式显示出来
 - 将二进制“01 100 101”转换为“145”两个字符显示
- %c: 将“25 + 76”的结果直接解释为ASCII字符显示
 - 二进制“0110 0101”解释为字符“e”

25 plus 76 in decimal is 101.

25 plus 76 in hexadecimal is 65.

25 plus 76 in octal is 145.

25 plus 76 as a character is e.

%f

- 如果使用printf输出浮点数，则应该使用相应的格式说明“%f”
 - 将二进制浮点数**转化**为如“3.140000”形式的字符序列
- 如果变量radius被声明为float类型，需要使用格式说明“%f” **输入**
 - scanf ("%f", &radius);
 - 这个过程将键盘输入的如“15.0”类型的字符**转**化为**float类型**浮点数，并进行存储

%lf

- 如果变量radius被声明为double类型，则需要使用格式说明“%lf”：
 - scanf ("%lf", &radius);
 - 将键盘输入的如“15.0”类型的字符**转化为double类型**浮点数，并进行存储

十六进制字面常量

- 十六进制字面常量使用前缀0x表示，零扩展

```
int memoryAddress = 0x30000000;
```

```
int valueE = 0xE;
```

```
printf("%d\n", valueE);
```

按位运算符

- 移至第七章

- **62-63=1是个错误的等式，能不能移动一个数字使得等式成立**

习题(3)

- 书面作业

- 6. 14
- 6. 15
- 6. 17

- 上机作业

- 6. 18
- 6. 19