Learn Computer System

Chirality

目录

1 信息的表示

1.1 整数的表示

1.1.1 整型的数据类型

下面两个表格给出了 C 语言的各种整型在 32 位和 64 位两种程序下的大小以及范围。

类型名	占用字节数 (Byte)	最小值	最大值
char	1	-128	127
unsigned char	1	0	255
short	2	-32768	32767
unsigned short	2	0	65535
int	4	-2147483648	2147483647
unsigned int	4	0	4294967295
long	4	-2147483648	2147483647
unsigned long	4	0	4294967295
int32_t	4	-2147483648	2147483647
uint32_t	4	0	4294967295
int64_t	8	-9223372036854775808	9223372036854775807
uint64_t	8	0	18446744073709551615

Table 1: 32 位程序的 C 语言数据类型大小和范围

类型名	占用字节数 (Byte)	最小值	最大值
char	1	-128	127
unsigned char	1	0	255
short	2	-32768	32767
unsigned short	2	0	65535
int	4	-2147483648	2147483647
unsigned int	4	0	4294967295
long	8	-9223372036854775808	9223372036854775807
unsigned long	8	0	18446744073709551615
int32_t	4	-2147483648	2147483647
uint32_t	4	0	4294967295
int64_t	8	-9223372036854775808	9223372036854775807
uint64_t	8	0	18446744073709551615

Table 2: 64 位程序的 C 语言数据类型大小和范围

可以看出每种整型都分为有符号和无符号两种, 了解它们的编码方式就可以快速得到它们的范围.

1.1.2 无符号数的编码

假设一个 ω 位的 01 向量是一个无符号整数的编码, 记

$$\boldsymbol{x} = [x_{\omega-1}, x_{\omega-2}, \ldots, x_0]$$

其中 $x_i = 0$ or $1, \quad i = 0, 1, \ldots, \omega - 1$. 我们用记号 $\mathrm{B2U}_\omega(\boldsymbol{x})$ 表示 $\boldsymbol{x}$ 所代表的无符号整数的值, 那么就有

$$\mathrm{B2U}_\omega(\boldsymbol{x}) = \sum_{i=0}^{\omega-1} x_i 2^i$$

在数学上很容易证明无符号整数与编码按这种规则的对应是双射 (单射且满射).

下面给出一些例子:

$$\begin{aligned}
&\mathrm{B2U}_4([0101]) = 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \\
&\mathrm{B2U}_4([1111]) = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 15 \\
&13 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = \mathrm{B2U}_4([1101])
\end{aligned}$$

不难得出结论, ω 位编码表示的无符号整型的最小值是 0, 对应编码为 ω 个 0, 最大值是 $2^\omega - 1$, 对应编码为 ω 个 1.

1.1.3 有符号数的补码 (two's-complement) 编码

假设一个 ω 位的 01 向量是一个有符号整数的补码编码, 记

$$\boldsymbol{x} = [x_{\omega-1}, x_{\omega-2}, \ldots, x_0]$$

其中 $x_i = 0$ or $1, \quad i = 0, 1, \ldots, \omega - 1$. 我们用记号 $\mathrm{B2T}_\omega(\boldsymbol{x})$ 表示 $\boldsymbol{x}$ 在补码编码规则下所代表的有符号整数值, 那么就有

$$\mathrm{B2T}_\omega(\boldsymbol{x}) = -x_{\omega-1} 2^{\omega-1} + \sum_{i=0}^{\omega-2} x_i 2^i$$

最高位 $x_{\omega-1}$ 称为符号位, 权重为 $-2^{\omega-1}$, 容易发现符号位为 1 时值为负数, 反之为正数, 并且补码表示有符号整数也是双射.

下面给出一些例子:

$$\begin{aligned}
&\mathrm{B2T}_4([0101]) = -0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \\
&\mathrm{B2T}_4([1111]) = -1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = -1 \\
&-3 = -1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = \mathrm{B2T}_4([1101])
\end{aligned}$$

补码编码的 ω 位有符号数最大值为 $2^{\omega-1} - 1$, 对应的编码为 $[0, 1, \ldots, 1]$, 最小值为 $-2^{\omega-1}$, 对应的编码为 $[1, 0, \ldots, 0]$. 我们也容易得到如下结论: 如果两个 ω 位补码编码除了符号位都相同, 那么它们的大小相差 $2^{\omega-1}$.

下表给出一些同样的编码表示为无符号数和用补码规则表示有符号数的结果

hex	binary	unsigned int	signed int
0xE	1110	14	-2
0x0	0000	0	0
0x5	0101	5	5
0x8	1000	8	-8
0xD	1101	13	-3
0xF	1111	15	-1

从表中可以注意到如下规律: 如果编码最高位是 0, 那么无符号和有符号表示的整数值相同; 如果编码最高位是 1, 那么它表示的无符号数与其补码规则表示的有符号数的和是 2^{ω}, 这个规律在数学上的证明也是显然的.

最后, 我们可以证明, 如果两个使用补码表示符号数的等长编码的每一位都不同, 那么它们表示的符号数的和为-1, 这也就是说对于一个符号数 x, 将它的二进制补码表示按位取反, 即可得到-x-1 的补码表示.

1.1.4 反码与原码

反码 (Ones' Complement) 的规则为

$$\mathrm{B2O}_{\omega}(\boldsymbol{x}) = -x_{\omega-1}(2^{\omega-1}-1) + \sum_{i=0}^{\omega-2} x_i 2^i$$

原码 (Sign-Magnitude) 的规则为

$$\mathrm{B2S}_{\omega}(\boldsymbol{x}) = (-1)^{x_{\omega-1}} \cdot (\sum_{i=0}^{\omega-2} x_i 2^i)$$

这两种有符号整数的编码方式已经很少采用了, 其中一个原因是非双射性, 对于数字 0, 除了使用 $[0,\ldots,0]$ 表示外, 反码还可以使用 $[1,\ldots,1]$ 表示, 原码还可以使用 $[1,0,\ldots,0]$ 表示. 现代机器几乎全部使用补码, 所以之后的符号数表示我们都默认是使用补码的表示.

1.1.5 无符号和有符号的类型转换

C 语言中的强制类型转换本质上就是对于同一个编码的不同解读, 比如将 32 位有符号整数-1 强制转换为无符号整数

```
1 int a = -1;
2 unsigned int b = a;
```

输出变量 b, 结果为 4294967295. 因为-1 的 32 位补码表示为 0xFFFFFFFF, 将其解析为无符号数便得到了 32 位的最大无符号数.

反过来, 定义一个无符号数, 转换为有符号数

```
unsigned int a = 4294967294u;
int b = a;
```

a 的编码为 0xFFFFFFFE, 解析为有符号数变为-2.

我们可以得到下列结论:

- ω 位的有符号数 x, 强制转换为 ω 位的无符号数 y, 则有

$$y = \begin{cases} x, & x \geq 0, \\ x + 2^{\omega}, & x < 0 \end{cases}$$

- ω 位的无符号数 x, 强制转换为 ω 位的有符号数 y, 则有

$$y = \begin{cases} x, & x \leq 2^{\omega-1} - 1, \\ x - 2^{\omega}, & x \geq 2^{\omega-1} \end{cases}$$

C 语言中对含有无符号的表达式运算有一些奇特的处理, 例如对于有符号和无符号的数字比较大小, C 语言会默认将有符号数强制类型转换为无符号数, 考虑以下代码:

```
int max_int = 0x7FFFFFFF; // 2147483647
int min_int = -max_int - 1;
cout << "-1 < 0u ?" << ((-1 < 0u) ? "True" : "False") << endl;
cout << max_int << "u > " << min_int << " ? "
<< ((2147483647u > min_int) ? "True" : "False") << endl;
cout << max_int << " > int(2147483648u) ? "
<< (max_int > int(2147483648u) ? "True" : "False") << endl;
```

其输出为

```
-1 < 0u ? False
2147483647u > -2147483648 ? False
2147483647 > int(2147483648u) ? True
```

出现这样的结果就是 C 语言的强制类型转换导致的, 在表达式 $-1 < 0u$ 中, 编译器会将-1 强行转换为无符号数再比较, -1 的补码为 0xFFFFFFFF, 解析为无符号数会变为最大的无符号数. 表达式 $2147483647u > -2147483648$ 中, 右侧数字是最小的 32 位符号数, 补码为 0x80000000, 解析为无符号数变为 2^{31}, 大于左侧的 $2^{31} - 1$. 最后一个表达式 $2147483647 > int(2147483648u)$, 右侧数字由无符号转为有符号, 编码是 0x80000000, 转为符号数变为最小 32 位符号数.

1.1.6 整型编码的扩展

扩展指将低位数补全位数变为高位数, 这样的操作显然是安全可控的. 对于无符号数, 扩展只需要高位补 0 即可, 例如四位无符号数 0x8, 扩展到八位, 十六位, 三十二位分别为 0x08, 0x0008,0x00000008.

1.1.7 整型编码的截断

1.2 整数运算

1.2.1 无符号加法