

# 1 数据表示与存储

## 1.1 数据的类型及大小

表 1: 数据的类型及大小

类型	字节数	最小值	最大值 (signed)	最大值 (unsigned)
char	1	-128	127	255
short	2	-32768	32767	65535
int	4	-2147483648	2147483647	4294967295
long	8	-9223372036854775808	9223372036854775807	18446744073709551615
void*				
float	4	$1.17549 \times 10^{-38}$	$3.40282 \times 10^{38}$	
double	8	$2.22507 \times 10^{-308}$	$1.79769 \times 10^{308}$	

## 1.2 计算值域

T: 用  $n$  位表示数字

(signed) T 可表示  $-2^{n-1} \sim 2^{n-1} - 1$

(unsigned) T 可表示  $0 \sim 2^n - 1$

## 1.3 补码

对应正数补码的“各位取反、末位加 1”

$$\begin{aligned}
 &+23 = 00010111 \\
 &\text{按位取反} = 11101000 \\
 &\quad + \quad 1 \\
 &-23_{\text{补码}} = 11101001
 \end{aligned}$$

模 ( $2^n$ ) 减去该负数的绝对值

$$\begin{array}{r}
 10000000 \\
 - 00010111 \\
 \hline
 11101001
 \end{array}$$

## 1.4 GDB 查看数据

>(gdb) x/4xb

b - byte (8-bit value)

h - halfword (16-bit value)

w - word (32-bit value)

g - giant word (64-bit value)

o - octal

x - hexadecimal

d - decimal

u - unsigned decimal

t - binary

f - floating point

a - address

c - char

s - string

i - instruction

### 1.5 浮点数

浮点数表示为  $(-1)^s \cdot M \cdot 2^E$

二进制位数	s 符号位	exp 指数	frac 尾数	总计
float	1	8	23	32
double	1	11	52	64

#### 1.5.1 规格化数 $exp \neq 0$ 且 $exp \neq 11\dots 1$

(unsigned)  $exp = E + Bias$

Bias (偏置值) =  $2^{k-1} - 1$ , k 为  $exp$  的二进制位数

偏置值 Bias	
float	127
double	1023

例 1: 十进制整数  $\rightarrow$  二进制浮点数

$$\text{float } F = 15213.0$$

化为二进制数:

$$15213_{10} = 11101101101101_2$$

$$= 1.1101101101101_2 \times 2^{13}$$

计算 frac:

$$M = 1.\underline{1101101101101}_2$$

$$\text{frac} = \underline{11011011011010000000000}_2$$

计算 exp:

$$E = 13 \quad \text{来自化为二进制时的指数}$$

$$Bias = 127$$

$$\text{exp} = 140 = 10001100_2$$

结果:

$$0 \quad 10001100 \quad 11011011011010000000000$$

s          exp                          frac

例 2: 二进制浮点数  $\rightarrow$  十进制数

无符号数, 4 位阶码 (Bias=7), 3 个小数位

$$1001 \quad 111$$

exp    frac

计算  $E = \text{exp} - Bias$

$$= 1001_2 - 7_{10}$$

$$= 2_{10}$$

计算  $M = 1.\text{frac} = 1.111$

化为十进制:

$$1.111 \times 2^2 = 111.1_2 \quad \text{小数点右移 2 位}$$

$$= \frac{15}{2} = 7.5$$

#### 1.5.2 非规格化数 $exp = 0$

frac = 00...0 表示 0

frac  $\neq$  00...0 表示接近 0 的小数  $(-1)^s \cdot M \cdot 2^E$

非规格化数 $E = 1 - Bias$	
float	-126
double	-1022

#### 1.5.3 特殊值 $exp = 11\dots 1$

frac = 00...0 表示  $\infty$

frac  $\neq$  00...0 表示 NaN

$$1.0/0.0 = -1.0/-0.0 = +\infty$$

$$1.0/-0.0 = -\infty$$

$$\sqrt{-1.0} = \infty - \infty = \infty \times 0 = \text{NaN}$$

#### 1.5.4 舍入 (到偶数)

末两位	动作	例子(保留一位小数)
01	舍	$11.00\underline{1}_2 \rightarrow 11.0_2$
11	入	$10.0\underline{11}_2 \rightarrow 10.1_2$
10	强迫结果为偶数 (末尾为 0) 010 舍, 110 入	$10.0\underline{10}_2 \rightarrow 10.0_2$ $10.1\underline{10}_2 \rightarrow 11.0_2$

## 2 程序的机器级表示

### 2.1 计算数组元素的地址

计算  $T * D[R][C]$  元素  $D[i][j]$  的地址:

$\&D[i][j] = \&D[0][0] + \text{sizeof}(T) \times (C \cdot i + j)$

假设  $\text{sizeof}(T) = k$ , 将  $D[i][j]$  复制到  $\%eax$  中

asm:  $D$  in  $\%rdi$ ,  $i$  in  $\%rsi$ ,  $j$  in  $\%rdx$

```
1 leaq (%rsi,%rsi,$C-1), %rax
```

```
2 leaq (%rdi,%rax,$k), %rax
```

```
3 movl (%rax,%rdx,$k), %rax
```

结果为  $D + k \cdot C \cdot i + k \cdot j$

即  $D + \text{sizeof}(T) \times (C \cdot i + j)$

### 2.2 其他内容

内容	操作数计算方式	栈	gdb 常用操作
页码	P121	P164	P194

## 3 链接

### 3.1 符号表 (.symtab)

C 语言表示	类型	符号强度	节	说明
<code>void swap();</code>	全局	强	<code>.text</code>	函数在 <code>.text</code>
<code>extern int buf[];</code>	外部	—	实际定义所在位置	默认 UND(未解析的引用符号)
<code>int *bufp0 = &amp;buf[0]</code>	全局	强	<code>.data</code>	初始化的全局变量
<code>int *bufp1;</code>	全局	弱	COMMON	未初始化的全局变量
<code>void p() {   static int i = 1; }</code>	局部	强,不同 函数可重	<code>.data</code> 或 <code>.bss</code>	未初始化或初始化为 0 在 <code>.bss</code> 初始化为其他在 <code>.data</code>
<code>void q() {   int j = 2; }</code>	都不是	—	节里没有,在栈里	链接器不看局部变量

### 3.2 链接顺序

```
$ gcc -static -o prog2c main2.o ./libvector.a
```

E 将被合并以组成可执行文件的所有目标文件集合

U 当前所有未解析的引用符号的集合

D 当前所有定义符号的集合

开始 E、U、D 为空,首先扫描 `main2.o`,将其加入 E,将未找到的符号加入 U,定义的符号加入 D。

再扫描 `./libvector.a`,将匹配到的 U 中的符号转移到 D 并加入到 E,同时将未找到的符号加入 U。

最后搜索标准库 `libc.a`,处理完 `libc.a` 时,U 一定是空的,D 中符号唯一,否则错误。

### 3.3 重定位

PC 相对地址下重定位值计算公式:

$$\text{ADDR}(\text{r.symble}) - ((\text{ADDR}(\text{.text}) + \text{r.offset}) - \text{r.addend})$$

在 asm 中表示为 `4004de: e8 05 00 00 00 callq 4004e8 <sum>`