

2.58 编写过程 `is_little_endian`，当在小端格式机器上编译和运行时返回 1，在大端格式机器上编译运行时则返回 0。这个程序应该可以运行在任何机器上，无论机器的字长是多少。

设计思路

小端法是最低有效字节在前面，而大端法是最高有效字节在前面，区分两种格式只需比较低地址存放的数据即可。这里采用联合体 `union` 来进行设计，`union` 中的数据都是共享内存地址的，并且从低地址开始存放。`Union duan` 定义中有 4 个字节的 `int` 型变量 `a`，1 个字节的 `char` 型变量 `c`，这里比较 `c` 和 `a` 的低 8 位（16 进制下），如果一致就说明是最低有效字节在前面，为小端格式，反之为大端格式。

实现测试代码如下：

```
#include <stdio.h>
#include <stdlib.h>
union duan//定义联合体 duan
{
    int a;
    char c;
} d;
int is_little_endian()//判断大小端的函数
{
    d.a=0x11134567;
    /*令 d.a=0x11134567 如果是小端模式，int 下 d.a 中的 67 会存放在低地址处，而调用 d.c 时经过隐形类型转换，也在低地址处存放最后只需比较是否一致就好*/
    if(d.c==0x67)//c 和 a 的低 8 位（16 进制下）进行比较
    {
        printf("小端模式\n");return 1;
    }
    else
    {
        printf("大端模式\n");return 0;
    }
}
int main()//主函数测试部分
{
    printf("%d\n",is_little_endian());
    return 0;
}
```

2.71 你刚刚开始在一家公司工作，他们要实现一组过程来操作一个数据结构，要将 4 个有符号字节封装成一个 32 位 `unsigned`。一个字中的字节从 0（最低有效字节）编号到 3（最高有效字节）。分配给你的任务是：为一个使用补码运算和算术右移的机器编写一个具有如下原型的函数：

```
/* Declaration of data type where 4 bytes are packed into an unsigned
*/
```

```
typedef unsigned packed_t;
```

```
/* Extract byte from word. Return as signed integer */
```

```
int xbyte(packed_t word, int bytenum)
```

也就是说，函数会抽取出指定的字节，再把它符号扩展为一个 32 位 int。你的前任（因为水平不够高而被解雇了）编写了下面的代码：

```
/* Failed attempt at xbyte */
```

```
int xbyte(packed_t word, int bytenum)
{
    return (word >> (bytenum << 3)) & 0xFF;
}
```

A. 这段代码错在哪里？

B. 给出函数的正确实现，只能使用左右移位和一个减法。

解答

A 这段代码得到的结果是 unsigned，而并非扩展为 signed 的结果。

B 使用 int 型变量，将待抽取字节左移到最高字节，再右移到最低字节即可修改后的代码如下：

```
int xbyte(packed_t word, int bytenum) //bytenum 指第几个字节
{
    int ret = word << ((3 - bytenum)<<3); //将抽取字节左移到最高位，并用 int 型表示
    return ret >> 24; //再右移到最低位
}
```

2.84 给定一个浮点格式，有 k 位指数和 n 位小数，对于下列数，写出阶码 E 、尾数 M 、小数 f 和值 V 的公式。另外，请描述其位表示。

A. 数 5.0。

B. 能够被准确描述的最大奇整数。

C. 最小的正规格化数的倒数。

解答

公式： $E = \text{exp} - \text{Bias} = \text{exp} - (2^{k-1} - 1)$; $M = 1 + f$; f 根据具体浮点数而定; $V = (-1)^s * M * 2^E$ 。

A 数 5 用二进制表示为 $101.0 = 1.01 * 2^2$, $E = 2$, $M = 1.01$, $f = 0.01$, $V = 1.01 * 2^2$, $\text{exp} = E + \text{Bias} = 2 + (2^{k-1} - 1) = 2^{k-1} + 1 = (2^{k-1} + 1)_2$

其位表示为：

s	exp	frac
0	$(2^{k-1} + 1)_2$	01000...00 (01 后面位用 0 补全)

B $(\text{exp})_{\max} = (11 \cdots 10)_2 = 2^k - 2$ (这里的 $11 \cdots 10$ 有 k 位), $(E)_{\max} = \text{exp} - \text{Bias} = 2^k - 2 - (2^{k-1} - 1) = 2^{k-1} - 1$, $M = (1.11 \cdots 1)_2 = (2^{n+1} - 1) * 2^{-n}$, $f = (0.11 \cdots 1)_2$, $V = M * 2^E = (2^{n+1} - 1) * 2^{E-n}$, 要使其为最大的奇整数，则需 $E = n$, 此时 $\text{exp} = E + \text{Bias} = n + (2^{k-1} - 1)$, $M = (2^{n+1} - 1) * 2^{-n}$, $V = 2^{n+1} - 1$ 。

其位表示为：

s	exp	frac
0	$(n + 2^{k-1} - 1)_2$	111...1111

C 最小的正规格化数 x , exp 是 $00\cdots 1$, frac 为全 0, $E=\text{exp}-\text{Bias}=1-(2^{k-1}-1)=2-2^{k-1}$, 所以 $x=1*2^{(2-2^{k-1})}$, 设其倒数是 y , 满足 $x*y=1$, 所以最小正规格化数的倒数是 $2^{(2^{k-1}-2)}$ 。其位表示为:

s	exp	frac
0	$(2^k-3)_2$	00...0000 (全为 0)

2.88 我们在一个 `int` 类型为 32 位补码表示的机器上运行程序。`float` 类型的值使用 32 位 IEEE 格式, 而 `double` 类型的值使用 64 位 IEEE 格式。我们产生随机整数 x 、 y 和 z , 并且把它们转换成 `double` 类型的值:

```
/* Create some arbitrary values */
int x = random();
int y = random();
int z = random();
/* Convert to double */
double dx = (double) x;
double dy = (double) y;
double dz = (double) z;
```

对于下列的每个 C 表达式, 你要指出表达式是否总是为 1。如果它总是为 1, 描述其中的数学原理。否则, 列举出使它为 0 的参数的例子。请注意, 不能使用 IA32 机器运行 GCC 来测试你的答案, 因为对于 `float` 和 `double`, 它使用的都是 80 位的扩展精度表示。

- A. `(double) (float) x == dx`
- B. `dx + dy == (double) (x+y)`
- C. `dx + dy +dz == dz + dy +dx`
- D. `dx * dy * dz == dz * dy *dx`
- E. `dx / dx == dy /dy`

解答

A 值为 1。`int` 转 `float`, 再由 `float` 转 `double`, 由于 `double` 有更大的范围, 所以精度不会损失, 两者仍然相等。

B 值可能为 0。`dx+dy` 不会发生溢出, 而 `x+y` 发生溢出, 如 `x=INT_MAX, y=1`, 这时 `x+y` 溢出进行转化后, 两者不相等。

C 值为 1。`double` 类型进行加法计算不会发生溢出

D 值为 1。进行乘法运算, 中间结果可能会发生溢出, 但是最终结果是相等的 (经过调试验证)

E 值可能为 0。当 `dx=0, dy=1` 时, `dx/dx=nan, dy/dy=1`, 两者不相等。

2.92. 遵循位级浮点编码规则, 实现具有如下原型的函数:

```
/* Compute -f. If f is NaN, then return f. */
float_bits float_negate(float_bits f);
```

对于浮点数 f , 这个函数计算 $-f$ 。如果 f 是 NaN, 你的函数应该简单的返回 f 。测试你的函数, 对参数 f 可以取的所有 2^{32} 个值求值, 将结果与你使用机器的浮点运算得到的结果相比较。

解答

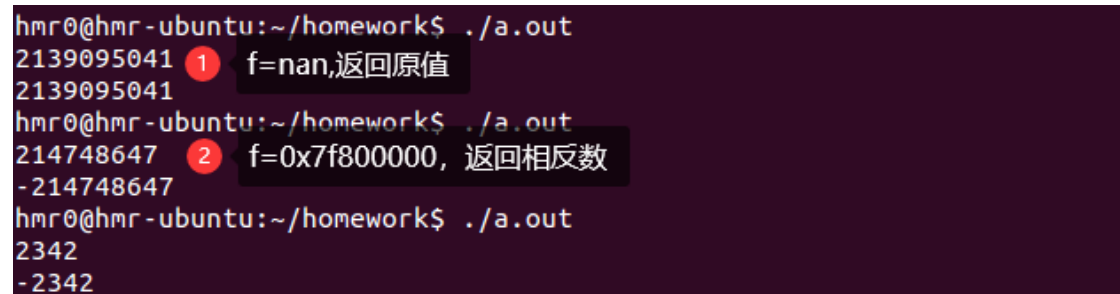
分析

nan 用 IEEE 标准表示时,exp 为全 1,frac 不为 0,这里可以用 f 与 0x7f800000 进行相与 (0x7f800000 就是 exp 为全 1, frac 全为 0 的情况), 如果结果仍然为 0x7f800000, 但是 f 不等于 0x7f800000, 则说明 f 为 nan, 返回原值。

编写的函数如下

```
#include <stdio.h>
typedef unsigned float_bits;//取别名
float_bits float_negate(float_bits f)
{
    if((f & 0x7f800000)==0x7f800000 &&(f !=0x7f800000))
    {
        return f;
    }
    else return -f;
}
int main()
{
    float_bits f;
    scanf("%d",&f);
    printf("%d\n",float_negate(f));
    return 0;
}
```

测试结果:



```
hmr0@hmr-ubuntu:~/homework$ ./a.out
2139095041 ① f=nan,返回原值
2139095041
hmr0@hmr-ubuntu:~/homework$ ./a.out
214748647 ② f=0x7f800000, 返回相反数
-214748647
hmr0@hmr-ubuntu:~/homework$ ./a.out
2342
-2342
```