《计算机系统》

实验二

班级：信安 2101

学号：202108060121

姓名：杨华

目录

1 实验项目一

1.1 项目名称

1.2 实验目的

1) 替换 bits.c 中各个函数中的 return，格式要求如下所示：

int Funct(arg1, arg2, …) {

/* brief description of how your implementation works */

int var1 = Expr1;

…

int varM = ExprM;

varJ = ExprJ;

varN = ExprN;

eturn ExprR;

}

其中，每一个“Expr”只能使用如下规则：

① 数字只能使用 0 到 255（0xff），不能使用像 0xffffffff 这样大的数字

② 函数参数和局部变量(没有全局变量)

③ 一元运算目：! ~

④ 二元运算目：& ^ | + << >>

2.bits.c 中所给的 15 个函数都是缺失的，需要用上每个函数被允许的操作去实现所要求的功能。

3.下面的操作不被允许：

① 使用任何控制结构，如 if, do, while, for, switch 等。

② 定义或使用任何宏。

③ 在此文件中定义任何其他函数。

④ 调用任何库函数。

⑤ 使用任何其他的操作，如&&, ||, -, or ?:

⑥ 使用任何形式的 casting

⑦ 使用除 int 以外的任何数据类型。这意味着你不能使用数组、结构等。

对于需要你执行浮点运算的问题，编码规则较不严格。允许使用循环和条件控制也可以同时使用 int 和 unsigned。可以使用任意整数和无符号常量。

1.3 实验资源

利用学习通上下载的 datalab-handout 包

2 实验任务

2.1 实验任务 A

任务名称：**实验说明**

实验的目标是修改bits.c的副本，以便它通过所有在btest中进行测试而不违反任何编码准则。

1、使用 dlc 编译器（./dlc）自动检查代码是否符合标准。

命令：unix> ./dlc bits.c

说明：如果代码没有问题，dlc 会直接返回，否则，它会打印标记问题的消息。

命令：unix> ./dlc -e bits.c

说明：dlc 打印每个功能使用的操作员数量。

2、使用 btest 进行测试

命令：unix> make btest

unix> ./btest [可选命令行参数]

说明：编译和运行 btest 程序（每次更改 bits.c 都要执行 make btest 重新编译）。

命令：unix> ./btest

说明：测试所有功能的正确性并打印出错误信息。

命令：unix> ./btest -g

说明：以紧凑的形式测试所有功能、错误消息。

命令：unix> ./btest -f foo

说明：测试函数 foo 的正确性。

命令：unix> ./btest -f foo -1 27 -2 0xf

说明：用特定参数测试函数 foo 是否正确。

3.助手程序

ishow 和 fshow 程序可以查看整数和浮点表示，都需要单个十进制或十六进制数作为参数。

要构建它们，执行命令：unix> make

示例用法：

unix> ./ishow 0x27

十六进制= 0x00000027，有符号= 39，无符号= 39

unix> ./ishow 27

十六进制= 0x0000001b，有符号= 27，无符号= 27

unix> ./fshow 0x15213243

浮点值 3.255334057e-26

位表示形式 0x15213243，符号= 0，指数= 0x2a，分数= 0x213243

标准化：+1.2593463659 X 2 ^（ - 85）

linux> ./fshow 15213243

浮点值 2.131829405e-38

位表示形式 0x00e822bb，符号= 0，指数= 0x01，分数= 0x6822bb

标准化： +1.8135598898 X 2 ^（ - 126）

2.2 实验任务 B

1)BitAnd

要求：只用~ |实现 x&y

操作符限制：~ |

操作符使用数量限制：8

思路：运用德摩根律

```
int bitAnd(int x, int y) {
  return ~(~(x)|~(y));
}
```

2）getByte

要求：取出 x 中的 n 号字节

编号从低位到高位从 0 开始

操作符使用数量限制：6

思路：将 x 右移 n*8 位之后取出低 8 位的值

```
int getByte(int x, int n) {
return (x >> (n << 3)) &255;
}
```

3）logicalShift

要求：将 x 逻辑右移 n 位（0<=n<=31)

操作符使用数量限制：20

思路：将 x 的最高位除去后右移 n 位（保证高位补 0），然后使用|操作符手动将最高位移动到的位置置上 x 的最高位。

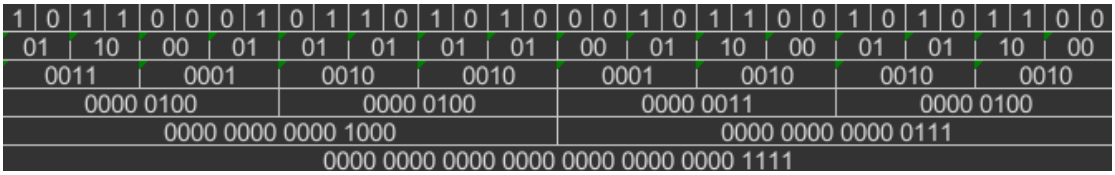
```
int logicalShift(int x, int n) {
    int a = 1 << 31;
    return ((x & ~a) >> n) | ((!!(x & a)) << (32 + ~n));
}
```

4）bitCount

要求：统计 x 的二进制表示中 1 的数量

操作符使用数量限制：40

思路：利用二分法和掩码实现，下面是参考连接



https://stackoverflow.com/questions/3815165/how-to-implement-bitcount-using-only-bitwise-operators

```
int bitCount(int x) {
      int mask1 = 0x55;
      int mask2 = 0x33;
      int mask3 = 0x0F;
      int result = 0;
      mask1 = mask1 | (mask1 << 8);
      mask1 = mask1 | (mask1 << 16);
      mask2 = mask2 | (mask2 << 8);
      mask2 = mask2 | (mask2 << 16);
      mask3 = mask3 | (mask3 << 8);
      mask3 = mask3 | (mask3 << 16);

      result = (x & mask1) + ((x >> 1) & mask1);
      result = (result & mask2) + ((result >> 2) & mask2);
      result = (result & mask3) + ((result >> 4) & mask3);
      return (result + (result >> 8) + (result >> 16) + (result >> 24)) & 0xff;
}
```

5）bang

要求：不使用!实现!操作符

操作符限制：~ & ^ | + << >>

操作符使用数量限制：12

思路：

!操作符的含义是 0 变为 1，非 0 变为 0，我们自然可以想到要做的是区分非零和零，零相对的非零数有一个非常明显的特征是-0=0，而对于非零数，取负后必定是一正一负而不可能相等，利用这一点，可以得出非零数与它的相反数进行|运算后符号位一定为 1，我们将符号位取出并取反就可以返回正确的值。

```
int bang(int x) {
  int op=~x+1;//the opposite of the two's complement
  return 1&(1^((x|op)>>31));
}
```

6）tmin

要求：返回补码表示的整型的最小值

操作符使用数量限制：4

思路：按照补码的权值理解，只要将权为-32 的位置为 1 即可

```
int tmin(void) {
  return 1<<31;
}
```

7)fitBitS

要求：如果 x 可以 l 只用 n 位补码表示则返回 1，否则返回 0

1 <= n <= 32

操作符使用数量限制：15

思路：

如果 x 可以用 n 位补码表示，那么左移多余的位的个数后再右移回来的数值一定与原值相等，这个方法利用了左移后溢出的位会被丢弃，而右移回来时的是补符号位，如果丢弃了 1 或者右移时补的是 1 都会导致值的改变，而这两种情况也正说明了 x 不可以只用 n 位补码表示。

```
int fitsBits(int x, int n) {
    int temp = n+(~0);          //n+0xffffffff指n-1，全1在机器中表示-1。
    int tempx = x >> temp;          //x右移n-1位
    return   (!tempx|!(tempx+1));//判断是否全为0或全为1

}
```

8) divpwr2

要求：计算 x/(2^n) 0 <= n <= 30 结果向零取整

操作符使用数量限制：15

思路：对于正数，直接将 x 右移 n 位，如果是负数，如果能整除则不用，如果不能整除则要加一。

```
int divpwr2(int x, int n) {
    int a=1<<31;
    int isnegative=!!(a&x);
    int divisible=(!!((~(a >> (32 + ~n))) & x));
    return (x>>n)+(isnegative&divisible);

}
```

9）negate

要求：计算-x

操作符使用数量限制：5

```
int negate(int x) {
  return ~x+1;
}
```

10）isPositive

要求：如果 x 大于 0 返回 1，否则返回 0

操作符使用数量限制：8

思路：检测符号位与 x 是否为 0 即可。

```
int isPositive(int x) {
    int temp = (x>>31) & 0x1;          //取符号位
    int ans = !(  temp |  !x );        //判断是否为负数或0
    return ans;
}
```

11)isLessOrEqual

要求：如果 x 小于等于 y 则返回 1，否则返回 0

操作符使用数量限制：24

思路：首先取 x，y 的符号位判断是否同符号，再计算 y-x，如果不同符号直接和 signx 相与

如果相同符号则看 y-x 的结果

```
int isLessOrEqual(int x, int y) {
   int signx = (x >> 31) &0x1;          //取x的符号位
    int signy = (y >> 31) &0x1;          //取y的符号位
    int isSameSign=! (signx ^ signy);// 对符号位异或取反，判断是否相同
    int p=! ( ( (~x)+1+y ) >> 31);      //计算y-x，并取结果的符号位p为1表示x<=y(x，y同号)
    return ((isSameSign & p) | ((~isSameSign) & signx)) ;
}
```

12）ilog2

要求：返回 x 求以 2 为底的对数的结果 向下取整

操作符使用数量限制：90

思路：因为 x 一定为正数，则先找到最高位的一将其后面全部填充为一，再用 bitcount 的二分法计算出全部的一的数量，log2 的结果即为数量减一

```
int ilog2(int x) {
   int result=0;
   int mask1=0x55;
   int mask2=0x33;
   int mask3=0x0F;
   x=x|(x>>16);
   x=x|(x>>8);
   x=x|(x>>4);
   x=x|(x>>2);
   x=x|(x>>1);
   mask1 = mask1 | (mask1 << 8);
   mask1 = mask1 | (mask1 << 16);
   mask2 = mask2 | (mask2 << 8);
   mask2 = mask2 | (mask2 << 16);
   mask3 = mask3 | (mask3 << 8);
   mask3 = mask3 | (mask3 << 16);
   result = (x & mask1) + ((x >> 1) & mask1);
      result = (result & mask2) + ((result >> 2) & mask2);
      result = (result & mask3) + ((result >> 4) & mask3);
      result = (result + (result >> 8) + (result >> 16) + (result >> 24)) & 0xff;
    return result+~1+1;

}
```

13)float_neg

要求：返回-f 的位级表示

本题及以下所有的题目都采用 unsigned int 来存放位级表示

所有的浮点类型都为 float

如果输入为 NaN，返回 NaN

操作符使用数量限制：10

思路：对于一般的浮点数，我们只需要对它的符号位取反就可以了。需要特殊处理的只是无穷与 NaN 这两种非规格化的情况

```
unsigned float_neg(unsigned uf) {
 unsigned result=uf^(1<<31);//change the sign bit
 if ((uf & 0x7F800000) == 0x7F800000 && (uf & 0x007FFFFF)) {
            result = uf;
     }
      return result;
}
```

14) i2f

要求：实现由 int 到 float 的类型转换

操作符使用数量限制：30

```
unsigned float_i2f(int x) {
   unsigned ux=x,sign=ux>>31,carry=0;
   int d=0;
   if(!x) return 0;
   if(sign) //negative
{
    x=~x+1;
    ux=x;
}
    while(!(x&0x80000000))
{
    x<<=1;
    d++;//the closest 1
}
    ux<<=(d+1); //frag

    carry=((ux&0x1FF)>0x100)|((ux&0x300)==0x300);
    return (sign<<31)+((31-d+127)<<23)+(ux>>9)+carry;


}
```

15)float_twice

要求：返回 2*f 的位级表示

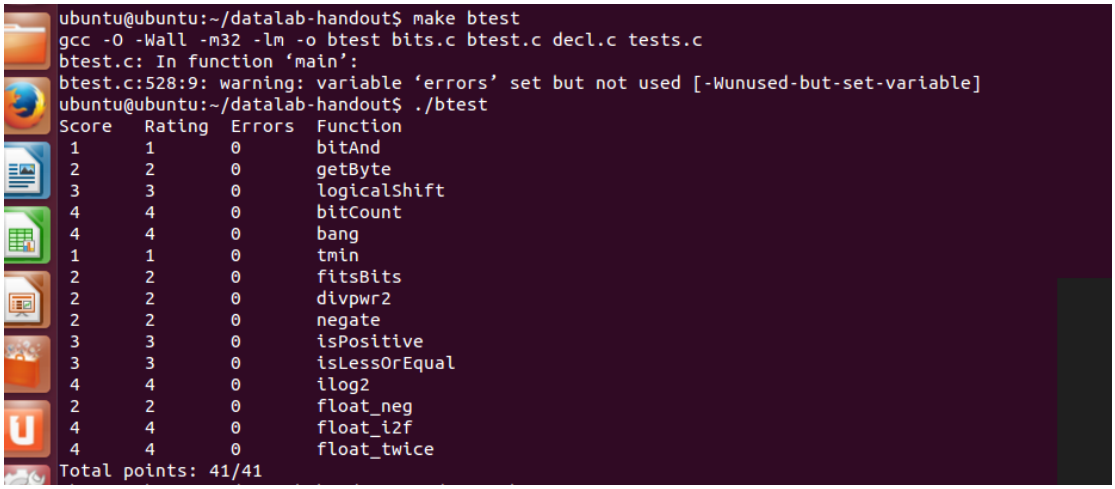
操作符使用数量限制：30

思路：先考虑特殊值，如果是特殊值直接返回即可，再考虑非规格数，左移动一位然后保留符号位，如果是规格化数，直接阶码加一

```
unsigned float_twice(unsigned uf) {
  if((uf&0x7f800000)==0x7f800000)
        return uf;//special number
  else if(!(uf&0x7f800000))//subnormal number
        return (uf<<1)|(uf&0x80000000);//directlt mul 2 and careof the sign bit
        else //normal number
                return uf+0x00800000;// E directly plus 1
}
```

运行成功界面

```
ubuntu@ubuntu:~/datalab-handout$ ./dlc -e bits.c
dlc:bits.c:143:bitAnd: 4 operators
dlc:bits.c:160:getByte: 3 operators
dlc:bits.c:172:logicalShift: 11 operators
dlc:bits.c:196:bitCount: 31 operators
dlc:bits.c:207:bang: 6 operators
dlc:bits.c:216:tmin: 1 operators
dlc:bits.c:231:fitsBits: 7 operators
dlc:bits.c:246:divpwr2: 14 operators
dlc:bits.c:256:negate: 2 operators
dlc:bits.c:268:isPositive: 5 operators
dlc:bits.c:282:isLessOrEqual: 15 operators
dlc:bits.c:312:ilog2: 44 operators
dlc:bits.c:330:float_neg: 6 operators
dlc:bits.c:360:float_i2f: 23 operators
dlc:bits.c:379:float_twice: 8 operators
```


ubuntu@ubuntu:~/datalab-handout$ make btest
gcc -O -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c
btest.c: In function ‘main’:
btest.c:528:9: warning: variable ‘errors’ set but not used [-Wunused-but-set-variable]
ubuntu@ubuntu:~/datalab-handout$ ./btest
Score Rating Errors Function
1 1 0 bitAnd
2 2 0 getByte
3 3 0 logicalShift
4 4 0 bitCount
4 4 0 bang
1 1 0 tmin
2 2 0 fitsBits
2 2 0 divpwr2
2 2 0 negate
3 3 0 isPositive
3 3 0 isLessOrEqual
4 4 0 ilog2
2 2 0 float_neg
4 4 0 float_i2f
4 4 0 float_twice
Total points: 41/41

3 总结

3.1 实验中出现的问题

1. 一开始对 int 的存储和-x=~x+1 的性质没有深入的了解，做题很困难，上网查阅才逐渐掌握
2. 在做 bitCount 时，不理解题目的做法，查阅网上解法才理解这个精妙的算法

3.2 心得体会

这个实验综合性很高，通过这个实验锻炼了我的思维能力和编程能力，同时通过这个实验，我更加深刻的理解了在底层数据是如何被存储和操作的，收获很大。