Data Lab

1900012901 王泽州

November 5, 2020

Section 1

Data Lab 基本要求

整数部分规则

- 常数在 $[0, 255]$ 之间
- 不使用全局变量
- 不能使用一些运算符，例如 $*$ $-$
- 不能使用控制结构
- 数据类型只有 int

浮点数部分规则

- 可以使用大常数，例如 0x80000000u
- 数据类型只有 int unsigned，并且运算符都可以使用
- 可以使用控制结构，例如 if – else

测试代码基本流程

注意虚拟机内存分配小于 4GB 第 16 个函数会产生奇怪错误

1. 代码通过 ./btest
2. ./bddcheck/check.pl −f function_name 所有数据测试
3. ./driver.pl 查看最终得分

Section 2

我实现的 Data Lab

前三个 lab

return a value of -1

```
int minusOne(void) {
    return ~0;
}
```

return x -> y in propositional logic - 0 for false, 1

```
int implication(int x, int y) {
    return !x | y;
}
```

return low_bit

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

Rotate x to the left by n

符号位不能直接右移，单独拿出来之后右移即可

```
int rotateLeft(int x, int n) {
    int _1=(~0);
    int n_1=n+_1; //n-1
    int all=(1<<n_1) + _1; //2^(n-1)-1
    int c=32+(~n_1); //32-n
    int t=all<<c;
    int asd=((x>>31)&1);

    t= (t&x)>>c;

    return (x<<n)|t|(asd<<n_1);
}
```

same as x ? y : z

直接左移 31 右移 31 得到 0 或-1 即可

```
int conditional(int x, int y, int z) {
    int t1=!x; // for z
    int t2=!t1;// for y

    t1=(t1<<31)>>31;
    t2=(t2<<31)>>31;
    return (t1&z)|(t2&y);
}
```

Compute !x without using !

就是用小于和大于把 0 的情况去掉

```
int bang(int x) {
    int t0=(x>>31)&1; //x<0
    int t1=(((~x)+1)>>31)&1;//x>0
    return 2+(~(t0 | t1));
}
```

return 1 if y is one more than x, and 0 otherwise

判断 sgn(x)=sgn(x+1) 且 x!=-1

```
int oneMoreThan(int x, int y) {
    int t=(~x)+y; //y-x-1
    int t_3=((x>>31)^((x+1)>>31))&(~x);
    //check the signed bit
    return (!t)&(!t_3);
}
```

return 1 if x can be represented as an n-bit, two's complement integer

若 $x > 0$，则范围一定在 $[0, 2^{n-1} - 1]$ 内
若 $x < 0$，则判断去掉低 $n - 1$ 位，高位是否都是 1

```
int fitsBits(int x, int n) {
    int _1=(~0); //-1
    int n_1=n+_1;  //n-1
    int all=(1<<n_1)+_1; //2^(n-1)-1
    int t=x+(~(x&all))+1;

    return (!t) | (!((t>>n_1)+1));
}
```

multiplies by 5/8 rounding toward 0

小班课上讲过的怎样处理负数向上取整

```
int multFiveEighths(int x) {
    int x_5 = (x+(x<<2));
    int t = x_5 >> 31; //check x*5 < 0?
    return (x_5+(7&t))>>3;
}
```

multiplies by 2, saturating to Tmin or Tmax if overflow

把最高位的两位拿出来
若 $x > 0$，则判断其第 30 位是否为 1
若 $x < 0$，同理判断其第 30 位是否为 1，即和 2^{30} 比较

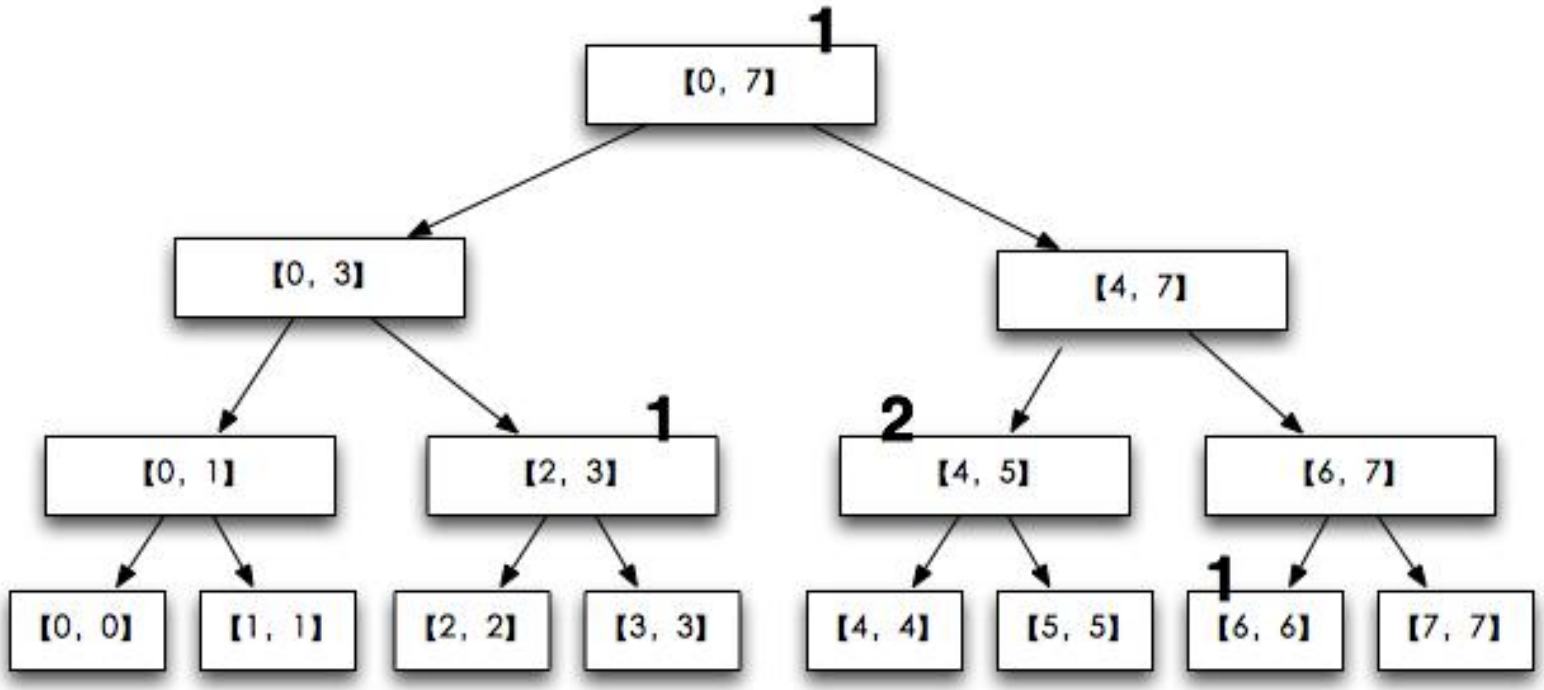
```
int satMul2(int x) {
    int sgn = (x>>30)&3;
    int _1 = ~0;

    int sgn1 = sgn + _1; // sgn -1
    int sgn2 = sgn1 + _1; //sgn -2
    int tc1= !(sgn1);
    int tc2= !(sgn2);
    int tc3= !(tc1+tc2);

    return ((tc1<<31)+(~tc1)+1)+(tc2<<31)+(((tc3<<31)>>31)&(x<<1));
}
```

modThree && ilog2

我这两个写的是一种思路，所以合在一起讲
首先我们肯定能想到 $4^k \equiv 1$，则肯定要枚举位做
因为没有循环和条件语句，因此每一位枚举肯定不行
因此自然能想到类似二分的方法，只需要 $\log 32$ 步即可



modThree && ilog2 代码实现

~~不知道这么丑的代码我怎么写出来的~~

```c
int modThree(int x) {
    int cmp0=(x>>31)&1;
    int _1=(~0);
    int sgn=0;

    int all=(1<<16)+_1;
    x=(x&all)+(x>>16); //2^16 bit
    x=(x&255)+(x>>8);//2^8 - 1
    x=(x&15)+(x>>4);//2^4 - 1
    x=(x&3)+(x>>2);//2^2-1
    x=(x&3)+(x>>2);//twice

    //for 1xx
    sgn=(x>>2)&1;
    x=x+sgn+(~(sgn<<2))+1;

    //x < 0 && result !=0
    cmp0=cmp0 & (!!x);
    x=x+1+(~(cmp0|(cmp0<<1)));

    sgn=!(x+(~2));
    sgn=sgn|(sgn<<1);
    x=(x+(~sgn)+1);

    return x;
}
```

```c
int ilog2(int x) {
    int ans=0;
    //middle

    int _1=(~0);
    int all=(1<<16)+_1;//2^16-1
    int fc1=(x>>16),fc2=x&all,fc3=!fc1,fc4=!fc3;

    ans+=(fc4<<4)&16;
    x=((fc3+_1)&fc1)+((fc4+_1)&fc2);

    all=255; //2^8-1
    fc1=(x>>8),fc2=x&all,fc3=!fc1,fc4=!fc3;
    ans+=(fc4<<3)&8;
    x=((fc3+_1)&fc1)+((fc4+_1)&fc2);

    all=15; //2^4-1
    fc1=(x>>4),fc2=x&all,fc3=!fc1,fc4=!fc3;
    ans+=(fc4<<2)&4;
    x=((fc3+_1)&fc1)+((fc4+_1)&fc2);


    all=3; //2^2-1
    fc1=(x>>2),fc2=x&all,fc3=!fc1,fc4=!fc3;
    ans+=(fc4<<1)&2;
    x=((fc3+_1)&fc1)+((fc4+_1)&fc2);


    fc3=!(((x+(~1))>>31)&1);

    return ans+fc3;
}
```

Return bit-level equivalent of absolute value of f for floating point argument f.

先把高位设为 0，然后判断是否大于 inf 即可

```
unsigned float_abs(unsigned uf) {
    int t=uf&0x7fffffff;
    if(t>0x7f800000) return uf;
    return t;
}
```

Return bit-level equivalent of expression (float) x

唯一的问题在于舍掉的低位进位的问题

当 $\text{res} \neq \frac{1}{2}$ 时很简单，直接四舍五入即可

当 $\text{res} = \frac{1}{2}$ 时，需要用到向偶数进位的原则

注意直接做就好，如果之前的位都是 1 则会直接进位到阶码

```
unsigned float_i2f(int y) {
    int sgn=0,i=22,len=0;
    unsigned ans=0,x=y;

    if(!y) return 0;
    if(y<0) sgn=1,x=(~y)+1;
    while ((x>>len)/2) ++len;
    ans=(len+127)<<23;
    len=len-1;

    while ((~len) && (~i)) {
        ans=ans|(((x>>len)&1)<<i);
        i=i-1;
        len=len-1;
    }

    if(~len) {
        int val=1<<len;
        int t=x&(val*2-1);
        if(t==val) ans=ans+(ans&1);
        else ans=ans+(t>val);
    }

    if(sgn) return ans+(0x80000000u);
    return ans;
}
```

Return bit-level equivalent of expression (int) f for floating point argument f.

比上一题要简单很多

强制转换我们都知道是直接扔掉最后的位

所以这个直接做就行

```
int float_f2i(unsigned uf) {
    int sgn=uf&(1<<31);
    int all=(uf>>23)&255;
    int ans=1,len=all-127,i=22;

    if(all<127) return 0;
    if(all>=159) return 0x80000000u;
    if(len==31) return (1<<31);

    while (len && (~i)) {
        ans=ans<<1|(uf>>i&1);
        --i;
        --len;
    }
    ans=ans<<len;

    if(sgn) return -ans;
    return ans;
}
```

Return bit-level equivalent of the expression 2.0^-x

只有 1000... 的情况，直接判断写在哪一位即可
注意非规格化形式

```
unsigned float_negpwr2(int x) {
    if(x<-128) return 0x7f800000;
    if(x>149)  return 0;
    if(x>=127) return 1<<(149-x);
    x=127-x;
    return x<<23;
}
```

总结

~~操作字符用的有点多~~

```
Correctness Results     Perf Results
Points  Rating  Errors  Points  Ops     Puzzle
1       1       0       2       1       minusOne
2       2       0       2       2       implication
2       2       0       2       3       leastBitPos
3       3       0       2       15      rotateLeft
3       3       0       2       9       conditional
4       4       0       2       9       bang
2       2       0       2       11      oneMoreThan
2       2       0       2       13      fitsBits
3       3       0       2       6       multFiveEighths
3       3       0       2       20      satMul2
4       4       0       2       43      modThree
4       4       0       2       57      ilog2
2       2       0       2       2       float_abs
4       4       0       2       30      float_i2f
4       4       0       2       19      float_f2i
4       4       0       2       8       float_negpwr2

Score = 79/79 [47/47 Corr + 32/32 Perf] (248 total operators)
```

Section 3

他人的实现思路

如何用一个操作字符做多次同样的操作

使用 switch 即可

加上 if(x) 不算操作符，便可以实现

```
int i=1;
int ans=read();//rand();
while (i) {
    ans=ans<<1;
    switch (i) {
        case 1:
            i=2;
            break;
        case 2:
            i=3;
            break;
        default:
            i=0;
    }
}
cout<<ans<<endl;
```

Return bit-level equivalent of absolute value of f for floating point argument f.

```
unsigned float_abs(unsigned uf) {
    int t=0,i=1;
    int val1=0,val2=0,val3=0;
    while (i) {
        switch(i) {
            case 1:
                t=0x7fffffff;break;
            case 2:
                t=0x7f800000;break;
            default:
                t=0x7fffff;
        }
        t=t&uf;
        switch(i) {
            case 1:
                i=2,val1=t;break;
            case 2:
                i=3,val2=t;break;
            default:
                i=0,val3=t;
        }
    }
    switch(val2) {
        case 0x7f800000:
            if(val3) return uf;
            break;
    }
    return val1;
}
```

总结

这就是 data lab 所有内容了