COMPUTER STRUCTURE

PA2: Numbers and Bit Manipulations

Computer science & engineering

2017-18538 황선영

1. bitOr

```
int bitOr(int x, int y) {
        return ~((~x)&(~y));
  //return 2;
}
```

Or을 not과 and만으로 표현하려면 nand에 not을 붙이면 된다.

2. isAsciiDigit

```
int isAsciiDigit(int x) {
        int a = 0x30;
        int b = 0x3a;
        return !((x + (~a+1)) >> 31) & (x + (~b+1)) >> 31;
  //return 2;
}
```

a보다 크거나 같으면서 b보다 작아야 하므로 -a와 x의 합의 sign bit이 0이어야하고 -b와 x의 합의 sign bit이 1이어야한다.

3. getByte

```
int getByte(int x, int n) {
        int byte = x >> (n << 3);
        return byte & 0xFF;
  //return 2;
}
```

x로부터 byte를 추출하기 전에 n*8을 해준 후 그만큼 x를 right shift 한다. 그 후 맨앞의 bit를 sign bit으로 인식하지 않도록 0xFF와 &한 값을 return한다.

4.byteSwap

```
int byteSwap(int x, int n, int m) {
        int a = n << 3;
        int b = m << 3;
        int swap = 0xFF & ((x>>a) ^ (x>>b));
        x = x ^ (swap << a);
        x = x ^ (swap << b);
        return x;
```

n과 m을 각각 8배 해준 후 x를 각각 a와 b만큼 right shift해준다. 그 후 x와 xor하여 달라진 부분만 바꾼다.

5. bang

```
int bang(int x) {
        int sign = x >> 31;
        int negSign = (~x+1) >> 31;
        return (sign|negSign) + 1;
//return 2;
}
```

0인 경우 sign bit과 0을 negate한 후의 sign bit이 같다. 나머지 숫자는 그 둘이 다르다. 그 점을 이용하여 짠 코드다.

6. fitsShort

```
int fitsShort(int x) {
        int a = x >> 15;
        return !(a ^ 0) + !(a ^ (~0));
```

x를 short길이만큼 shift한 후 그 값을 0과 xor했을 때의 negate, 그 값을 0의 negate와 xor했을 때의 negate를 더한 값을 return 한다.

7. isTmax

```
int isTmax(int x) {
        return !((~x^(x+1)) | !(~x));
//return 2;
}
```

x가 Tmax일 때만 1을 return하도록 코드를 짰다.

8. negate

```
int negate(int x) {
        return ~x+1;
 // return 2;
}
```

X를 negate하는 방법은 x의 bits를 반전시킨 후 1을 더하는 것이다.

9. sign

```
int sign(int x) {
        return (x>>31) | (!!x);
 //   return 2;
}
```

X의 sign bit과 x를 두번 ! 한 값(그 결과는 0 또는 1이다.)을 or하면 x의 sign bit을 return 한다.

10. addOK

```
int addOK(int x, int y) {
        int add = x + y;
        return !(((add ^ x) & (add ^ y)) >> 31);
  return 2;
}
```

X+y를 했을 때 sign bit이 반전되지 않았는지 확인하는 코드이다.

11. isPositive

```
int isPositive(int x) {
        int a = !(x >> 31);
        return !x^a;
  //return 2;
}
```

0도 포함이 된다면 그냥 sign bit을 return 하면 되는데 양수만 판별해야 하므로 다음과 같이 코드를 짰다.

12. staMul2

```
int satMul2(int x) {
        int Tmin = 1 << 31;
        int sign = x >> 31;
        int double_x = 0;
        int overflow = 0;
        int sat2 = 0;

        double_x = x+x;
        overflow = (x ^ double_x) >> 31;
        sat2 = overflow & (sign ^ (~Tmin));
        return (sat2) | ((~overflow) & double_x);
  //return 2;
}
```

Sign은 x의 sign bit을 나타낸다. double_x는 x+x로 나타낼 수 있다. Overflow는 x와 double_x의 sign bit이 다른 지 확인하는 변수이다. Sat2는 만약 overflow일 경우 그 값을 saturate해주는 변수이다. return시에 sat2와 overflow, double_x를 모두 고려하여 return 한다.

13. absVal

```
int absVal(int x) {
        int a = x >> 31;
        return (a + x) ^ a;
//  return 2;
}
```

a는 x의 sign bit이다. 만약 x가 양수라면 a가 0이므로 return 값도 x일 것이고, x가 음수라면 a가 1이어서 x의 절댓값을 return할 것이다.

14. float_neg

```
unsigned float_neg(unsigned uf) {
        int neg = 0x80000000 ^ uf;
        int exp = 0x7F800000 & uf;
        int frac = 0x007FFFFF & uf;
        if((exp == 0x7F800000) && frac)
                return uf;
        return neg;
 //return 2;
}
```

uf를 FP 방식으로 다루고 sign bit을 바꾸어주는 방식으로 구현하였다. NaN인 경우는 if문을 통해 예외처리 하였다. exp의 bit이 모두 1이고 frac이 0이 아닌 경우 uf를 return한다.

15. float_half

```
unsigned float_half(unsigned uf) {
        int exp = uf & 0x7F800000;
        int sign = uf & 0x80000000;
        int frac = uf & 0x007FFFFF;
        if(exp == 0x7F800000) return uf;
        if((!exp) || (exp == 0x00800000))
        {
                frac = frac | exp;
                frac = (uf & 0x00FFFFFF) >> 1;
                frac += ((uf & 3) >> 1) & (uf & 1);
                return (sign | frac);
        }
        return (sign | ((exp - 1) & 0x7F800000) | frac);

        //return 2;
```

Exp, sign, frac은 각각 uf의 exp, sign, frac을 받는다. 만약 exp가 그대로라면 uf는 NaN이나 무한대이므로 uf를 리턴한다. 변수와 결과 모

두 single-precision형태로 해석되도록 조작을 거쳤다.

16. float_f2i

```
int float_f2i(unsigned uf) { //error
        unsigned sign = uf >> 31;
        unsigned exp  = (uf >> 23) & 0xFF;
        unsigned frac = (uf & 0x7FFFFF);
        unsigned bias = 0x7F;
        unsigned result = frac;

        if(exp == 0xFF)
                return 0x80000000u;

        if(exp < bias)
                return 0x0;

        exp -= bias;

        if(exp >= 31)
                return 0x80000000u;

        if(exp > 22)
                result = frac << (exp - 23);
        else
                result = frac >> (23 - exp);

        result += 1 << exp;

        if(sign)
                result = -result;

        return result;
}
```

첫번째 if문은 NaN과 무한대를 판별한다. 두번째 if문은 denomalized number과 노멀하지만 bias case보다 작은 exp를 판별한다. 세번째는 overflow를 판별하고 네번째 if문은 shift한 후 result를 얻는다. 다섯번째 if문은 sign bit가 1일 경우 부호를 바꾼다.