Intro. to Computer SW Systems Lab Report

[Data Lab: Floating Point]

20220100 박기현

kihyun@postech.ac.kr

명예 서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

Problem 1. negate(x): return -x

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

Problem 1은 -x를 반환하는 함수를 구현하는 문제이다.

2의 보수의 성질에 따라 x의 bit를 반전한 후 1을 더해주면 -x 값을 구할 수 있으므로 ~x + 1을 통해 구현할 수 있다.

Problem 2. isLess(x, y): if x < y then return 1, else return 0

```
int isLess(int x, int y) {
    int x_sign = x >> 31;
    int y_sign = y >> 31;
  return ((x_sign ^ y_sign) & (!y_sign)) | ((!(x_sign ^ y_sign)) & ((x + (~y + 1)) >> 31));
}
```

Problem 2는 x가 y보다 작으면 1을, 그렇지 않으면 0을 반환하는 함수를 구현하는 문제이다.

우선, x와 y의 부호가 같은 경우와 다른 경우로 나누어 생각해 볼 수 있다.

따라서 x와 y의 부호를 알기 위해서는 MSB(Most Significant Bit)를 확인하면 되므로 >>(Right Shift)를 이용하여 MSB의 값을 각각 x_sign 변수와 y_sign 변수에 저장한다.

그 후 ^(XOR)을 통해 x와 y의 부호가 같은 경우와 다른 경우에 따른 연산이 진행되도록 한다.

먼저, x와 y의 부호가 다른 경우는 y가 음수이면 x가 양수, y가 양수이면 x가 음수이므로 y_sign의 !(Logical Negation)을 통해 구현할 수 있다. (y가 음수인 경우 y_sign이 1이지만, x는 양수로써 y보다 크므로 0이 반환되어야 한다. 따라서 !1 = 0으로 옳은 값을 반환할 수 있다. 반대로 y가 양수인 경우 y_sign이 0이지만, x는 음수로써 y보다 작으므로 1이 반환되어야 한다. 따라서 !0 = 1로 옳은 값을 반환할 수 있다.)

다음으로, x와 y의 부호가 같은 경우는 y의 부호를 바꾼 후 더한 결과의 부호 값을 반환하면 성립하므로 ((x + (~y + 1)) >> 31)로 구현할 수 있다. (x와 y의 부호가 양수인 경우에서 y의 부호를 바꿔 더했을 때 양수, 즉 결과 값의 MSB가 0이면 x가 더 크다는 뜻이므로 그대로 0을 반환하면 되고, 음수, 즉 결과 값의 MSB가 1이면 x가 더 작다는 뜻이므로 그대로 1을 반환하면 된다. 반대로 x와 y의 부호가 음수인 경우에서 y의 부호를 바꿔 더했을 때, 양수, 즉 결과 값의 MSB가 0이면 x가 더 크다는 의미이므로 그대로 0을 반환하면 되고, 음수, 즉 결과 값의 MSB가 1이면 x가 더 작다는 의미이므로 그대로 1을 반환하면 된다.)

Problem 3. float_abs(uf): return absolute value of f

```
unsigned float_abs(unsigned uf) {
    unsigned int e = (uf >> 23) & 0x000000FF;
    unsigned int f = uf & 0x007FFFFF;

    if(!(e ^ 0x000000FF) && f)
        return uf;
    else
        return (0x7FFFFFFF & uf);
}
```

Problem 3은 f의 절댓값을 반환하는 함수를 구현하는 문제이다.

먼저, 지수부와 가수부를 구한다.

이때, 지수부의 bit가 모두 1이고 가수부가 0이 아니면, NaN(Not a Number)이므로 argument를 그대로 반환한다.

이외의 경우에는 부호부를 0으로 만들면 되므로 0x7FFFFFFF 값과 &(And) 연산을 이용하여 구현할 수 있다.

Problem 4. float_twice(uf): return 2 * f

```
unsigned float_twice(unsigned uf) {
```

```
    unsigned int s = (uf >> 31) & 0x00000001;
    unsigned int e = (uf >> 23) & 0x000000FF;
    unsigned int f = uf & 0x007FFFFF;

    if(!(e ^ 0x000000FF))
    {
        return uf;
    }
    else
    {
        if(!e)
        {
            f <<= 1;
            if(f >> 23)
            {
                f &= 0x007FFFFF;
                e++;
            }
        }
        else
        {
            e++;
        }
        return (s << 31) | (e << 23) | f;
    }
}
```

Problem 4는 f의 2배 값을 반환하는 함수를 구현하는 문제이다.

먼저, 부호부와 지수부, 가수부를 구한다.

이때, 지수부의 bit가 모두 1이면, NaN(Not a Number) 또는 infinity이므로 argument를 그대로 반환한다.

이외의 경우 중 지수부의 bit가 모두 0이면(denormalized case이면), 가수부를 1만큼 Left Shift하고, 그 결과가 overflow인 경우에는 지수부에 1을 더해준다. 만약 그렇지 않으면(Normalized case이면), 지수부에 1을 더해준다.

모든 연산을 끝낸 후, 다시 부호부와 지수부, 가수부를 합쳐 f의 2배 값을 반환할 수 있다.

Problem 5. float_i2f(x): return (float)x

```
unsigned float_i2f(int x) {
    unsigned int s = x & 0x80000000;
    unsigned int E = 31;
    unsigned int Bias = 127;
    unsigned int e = E + Bias;
    unsigned int f;
    unsigned int rounding;

    if (!(x ^ 0x00000000))
    {
        return 0;
    }
    else
    {
        if(s)
        {
            x = ~x + 1;
        }

        while (!(x & 0x80000000))
        {
            x <<= 1;
            E--;
        }

        e = E + Bias;
        f = (x & 0x7FFFFFFF) >> 8;

        x = x & 0x000000FF;
        rounding = (x > 0x00000080) || ((x == 0x00000080) && (f & 0x00000001));
```

```
        if (!(e ^ 0x000000FF) && x)
        {
            return 0x7FC00000;
        }
        else
        {
            if(rounding)
            {
                f++;
                if(f >> 23)
                {
                    f = 0;
                    e++;
                }
            }

            return s | (e << 23) | f;
        }
    }
}
```

Problem 5는 int x 값을 float x로 변환한 값을 반환하는 함수를 구현하는 문제이다.

먼저, 부호부를 구하고, 지수부와 가수부를 구하기 위한 변수를 선언한다.

예외로 x가 0인 경우에는 0을 반환한다.

부호부가 1이면 음수이므로, 2의 보수 성질을 이용하여 x의 양수 값을 구해준다.

그후, 지수부를 구하기 위해 MSB가 1이 될 때까지 Left Shift하여 E(exponent) 값을 구하고 Bias(=127) 값을 더해 지수부를 구한다.

MSB를 제외한 나머지 bit를 취해 가수부에 해당하는 위치로 Right Shift하여 가수부를 구한다.

이때, M(Significand)이 23 bit 이상인 경우 근사가 필요하므로, 근사의 경우를 살펴본다.

LSB(Least Significant Bit)로부터 8 bit까지의 값이 중간값(=0x80)보다 크거나(R(Round Bit)와 S(Sticky Bit)가 모두 1인 경우), 중간값과 같을 때 가수부의 LSB(Least Significant Bit)가 1(R(Round Bit)와 G(Guard Bit)가 모두 1인 경우)이면 반올림한다.

반올림은 가수부에 1을 더하고, 만약 overflow된 경우 지수부에도 1을 더해주는 것으로 수행할 수 있다.

최종적으로 부호부와 지수부, 가수부를 합쳐 float x 값을 반환할 수 있다.

Problem 6. float_f2i(uf): return (int)f

```
int float_f2i(unsigned uf) {
    unsigned int s = (uf >> 31) & 0x00000001;
    unsigned int e = (uf >> 23) & 0x000000FF;
    unsigned int f = uf & 0x007FFFFF;
    unsigned int Bias = 127;
    int E = e - Bias;

    if(!(e ^ 0x000000FF) || E >= 31)
    {
        return 0x80000000u;
    }
    else if (E < 0)
    {
        return 0;
    }
    else
    {
        if(E < 23)
        {
            f >>= (23 - E);
        }
        else if(E > 23)
        {
            f <<= (E - 23);
        }

        f += (1 << E);
```

```
        if(s)
        {
            f = ~f + 1;
        }

        return f;
    }
}
```

Problem 6은 float x 값을 int x로 변환한 값을 반환하는 함수를 구현하는 문제이다.

먼저, 부호부와 지수부, 가수부를 구한다.

E(exponent) 값은 지수부 값에 Bias(=127)을 빼줌으로써 구할 수 있다.

우선, 지수부의 bit가 모두 1인 경우(NaN 또는 infinity인 경우) 또는 E(exponent) 값이 31보다 크거나 같은 경우, 0x80000000u를 반환한다. 또한, E(exponent) 값이 음수인 경우, 0을 반환한다.

이외의 경우에는 E(exponent)가 23보다 작으면 (23 - E)만큼 Right Shift를 진행하고, 23보다 크면 (E - 23)만큼 Right Shift를 진행하여 M(Significand)의 소수 부분을 구하고, 1을 E(exponent)만큼 Left Shift한 값을 더해주어 M(Significand)의 정수 부분을 구한다.

이때, 부호부를 통해 음수이면 2의 보수 성질을 이용하여 음수 값을 구해준다.

이 과정을 모두 마친 값이 int x 값이므로 원하는 값을 반환할 수 있다.