컴퓨터 SW시스템 개론 Lab 2 Report

20220923 이윤혁

1. 개요

Lab 2에서는 bitwise operation과 float의 구조를 이용하여 주어진 함수를 구현한다.

2. 구현

2.1. negate

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

2의 보수를 이용해 주어진 수의 부호를 바꾸는 함수를 작성했다.

2.2. isLess

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

x와 y의 부호가 같고 x – y의 부호가 음수이거나, x가 양수이고 y가 음수인 경우를 검사하여 x < y의 참 거짓을 판별하는 함수를 작성했다.

2.3. float_abs

```
unsigned float_abs(unsigned uf) {
  if ((!(((uf << 1) >> 24) ^ 255) && (uf << 9))) return uf;
  return (~(1 << 31)) & uf;
}
```

if문을 이용해 uf가 Nan인 경우(exp가 0b11111111, fraction이 0)를 판별 uf를 그대로 반환하도록 함수를 작성하였다. uf가 Nan이 아닌 경우 sign비트를 0으로 바꿔주고 반환하도록 함수를 작성했다.

2.4. float_twice

```
unsigned float_twice(unsigned uf) {
    unsigned s = uf >> 31;
    unsigned e = uf >> 23;
    if (!((e & 255) ^ 255)) return uf;
    if (!((e & 255) ^ 254)) return (e + 1) << 23;
```

```
    if (!(e & 255)) return (e << 23) | (uf << 9) >> 8;
    return (s << 31) | (((e & 255) + 1) << 23) | ((uf << 9) >> 9);
}
```

x >> 31은 x가 양수일 때 0x0000000, x가 음수일 때 0xFFFFFFFF이다. 이를 이용해 x에 xor 연산을 취해주면, x가 양수일 때 x값 그대로, x가 음수일 때 x의 2의 보수를 얻을 수 있다. !!(x >> 31)는 양수일 때 0, 음수일때 1이므로 이를 더해주면 x의 절댓값을 구할 수 있다.

2.5. float_i2f

```
unsigned float_i2f(int x) {
  int s, e, f;
  int len = 1;
  int guard;
  int round, sticky;
  // Nomalize case
  if (!x)
    return x;
  if (!(x ^ 0x80000000))
    return 0xCF000000;

   // Sign
  s = x & 0x80000000;
  if (s)
    x = ~x + 1;

  // count length of MSB ~ LSB
  while ((x >> len) ^ 0)
    len++;

  // Get float attr
  e = len + 126;
  x = x << (32 - len);
  f = (x >> 8) & 0x7FFFFF;

  // Get round attr
  guard = f & 1;
  round = x & 0x00000080;
  sticky = x & 0x0000007F;
```

```
  // Round
  if ((len > 24) && round && (sticky || guard)) {
    f++;
    if (f >> 23) {
      e++;
      f = 0;
    }
  }
   e = e << 23;
   return s | e | f;
}
```

int가 float으로 바뀔 때 nomalized case를 고려하여 특이값들을 예외처리 해주었다.

sign비트를 추출해 x를 절댓값을 취해주고, x의 MSB부터 LSB까지 길이를 측정해 round 처리 전 fraction의 길이를 계산했다. 길이를 이용해 exp를 계산하고, fraction으로부터 guard bit, round bit, sticky bit를 추출해 round 여부를 판별한 후 exp와 fraction을 round 처리했다. 그 후 sign, exp, fraction을 bitwise or 처리해 위치에 맞게 이어붙여 반환하도록 함수를 작성하였다.

2.6. float_f2i

```
int float_f2i(unsigned uf) {

    unsigned int s, e, f;
    int x;
    s = uf >> 31;
    e = (uf << 1) >> 24;
    f = (uf << 9) >> 9;

    if (!(e ^ 0xFFu)) return 0x80000000;
    if (e < 127u) return 0;
    if (e >= 158u) return 0x80000000;
    e = e - 127;
    x = 1 << e; // restore front 1

    // fraction pos < int pos
    if (e >= 22u) {
        x = x + (f << (e - 23));
        if (s) x = ~x + 1; // sign
        return x;
```

```
    }

    // fraction pos > int pos
    x = x + (f >> (23 - e));
    if (s) x = ~x + 1; // sign
    return x;
}
```

uf가 float에서 int로 바뀔 때 문제가 되는 특이값들 (exp가 0b11111111, bias 처리 후 exp가 음수 혹은 31이상(int overflow))를 조기에 처리해주었다.

exp에서 bias를 빼주어 실제 int의 2의 지숫값을 계산하고, int -> float으로 캐스팅하는 과정에서 날아간 int의 첫 1을 복구해주었다. 그 후 uf에서 fraction의 MSB 위치와 exp를 비교해 fraction이 int로 캐스팅 했을 때 올바른 위치에 있도록 shift 시킨다. 그 후 sign을 체크해 원래 float의 값이 음수인 경우 int에 2의 보수를 취하여 올바른 음의 int가 반환되도록 함수를 작성하였다.

3. 결과

Score	Rating	Errors	Function
2	2	0	negate
3	3	0	isLess
2	2	0	float_abs
4	4	0	float_twice
4	4	0	float_i2f
4	4	0	float_f2i

btest 실행결과 다음과 같은 결과를 얻었다. 이 밖에도 다른 값들을 넣었을 때 예상 값과 결과가 일치하여, 구현이 잘 된 것을 확인할 수 있었다.