CSED211: Data Lab2 Report

20230784 / 유재원

1. Overview

본 과제는 C 언어에서 정수 및 부동 소수점과 관련된 여러 상황을 제한된 문법 기능만을 이용해 해결하는 과정에서 비트 단위의 연산 및 정수, 부동 소수점 자료형의 처리를 연습하고 그 원리를 이해하는데 의의가 있다.

2.1. negate

본 문제는 int x에 대하여 절대값이 같고 부호가 반대인 x를 구하는 문제로, Two's complement를 구하는 문제이다. 정의에 따라 모든 부호를 반전한 뒤 1을 더해 구할 수 있다.

단, 예외적으로 Tmin에 대하여는 절대값이 같고 부호가 반대인 x가 없으므로 Tmin을 그대로 출력한다. 이는 Two's complement 의 정의에 부합한다. 또한 모든 부호를 반전한 뒤 1을 더하면 잘 구해진다.

실제 구현은 다음과 같다. ~ 연산자를 이용하여 모든 부호를 반전한 후 1을 더한다.

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

2.2. isLess

본 문제는 두 수 x, y를 준 후에 y가 더 크다면 1을, 그렇지 않다면 0을 반환하는 문제이다. x와 y의 차이를 이용하여 이를 구할 수 있다.

목적 달성을 위해서는 x, y, x-y 의 부호만 고려하면 된다. 먼저 x-y를 구해야 한다. - 연산자는 이용할 수 없으므로, x-y 와 동치인 x+(-y)를 이용할 것이다. -y의 경우 2.1 문제에서 구한 방법을 이용해 구하였다.

Case 1) x < 0 y >= 0 인 경우 x <y 이다. 0 이상인 수는 음수보다 항상 크기 때문이다.
x 가 음수일 때 MSB 는 1, Y가 0 이상일 때 MSB 는 0 이므로 x와 ~y를 & 연산할 경우 x < y 일 때에만 MSB가 1로 설정되도록 할 수 있다.

Case 2) x 와 y의 부호가 같고, x – y 의 부호가 음수인 경우 x < y 이다. x – y < 0 이라면 부등호를 기준으로 양변에 y를 더해 x < y를 만들 수 있기 때문이다.
x 와 y의 부호가 같음을 판정하기 위해서 x ^ y 의 결과에 ~을 해 준다. 이 경우 x와 y 의 결과가 같을 때에만 1, 그 외에는 0이 ~ 연산의 결과로 MSB가 됨을 확인할 수 있다.
또한 x-y 의 부호가 음수여야 하기에 위 결과와 x-y를 & 연산할 경우 x < y 인 상황에만 MSB 가 1, 아닐 경우 MSB가 0으로 정해짐을 알 수 있다.

위 두 경우를 통하여 x < y 인 상황에 MSB를 1로 설정했다면 그것을 >> 연산을 통해 LSB 로 내리고 나머지 비트에 대한 정보를 지우면 된다.

실제 구현은 다음과 같다. diff 변수를 선언하여 x – y 의 결과를 담고, Case 1과 Case 2의 결과를 Or 연산한 후 MSB를 LSB로 >> 연산을 이용해 옮긴 후 1과 & 연산을 통해 LSB 외에 나머지 비트에 대한 정보를 지웠다.

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

```
}
```

2.3 float_abs

본 문제는 절대값이 같고 부호가 양수인 float 자료형의 비트를 구하는 문제이다. 대부분의 상황에서 MSB를 0으로만 설정해주면 되지만, NaN일 경우에 대한 예외 처리가 필요하다.

NaN 일 경우를 판정해보자. NaN 은 e 가 11111111, frac 이 0 이 아닌 경우이다.

먼저 e가 11111111인 경우를 찾아보자. float에서 inf를 나타내는(exp 가 11111111 이며 frac 이 0 인) inf 변수를 선언해 준 뒤 uf를 inf 와 & 연산한다. 그러면 exp 부분의 정보만 남는데, 그것이 만약 inf 와 같다면 exp가 11111111인 경우이다.

frac 이 0이 아닌 경우를 판정해보자. uf에 << 9 연산을 취해주면 frac 부분과 뒤에 0이 9개 들어있다. 만약 이 값이 0이라면 frac 이 0인 경우고, 그 값이 0이 아니여서 true 로 판정되는 경우는 frac 이 0이 아닌 경우이다.

위에서 NaN 인 경우를 판정하기 위한 모든 조건을 찾았으므로 두 경우를 && 연산해 NaN을 최종적으로 판단하고, 그 때에는 uf를 반환하도록 예외처리를 한다. NaN 이 아닌 경우 0x7fffffff 과 & 연산을 하면 MSB를 반드시 0으로 만들고 나머지 비트에 대한 정보는 유지할 수 있다.

실제 코드 구현은 다음과 같다.

```
unsigned float_abs(unsigned uf)
{
    unsigned int inf = 0x7f800000;
    if ((uf & inf) == inf && (uf << 9))
    {
        return uf;
    }
    return uf & 0x7fffffff;
}
```

2.4 float_twice

본 문제는 특정 float 값의 2배 값을 구하여 반환하는 문제이다. 크게 세 가지 경우로 나누어 확인할 수 있다.

Case 1) 이미 inf 또는 NaN 일 경우

문제의 조건에 따라 이 경우에는 uf를 그대로 반환한다. 이 경우를 판정하기 위해 exp 부분만 1인 inf를 0x7f800000 으로 선언하고, 그것과 uf를 & 연산하여 그 결과가 inf 와 같다면 exp 부분이 11111111 인 것이므로 inf 또는 NaN 임을 판정할 수 있다.

Case 2) 만약 uf 와 inf 의 & 연산 결과가 0 이라면 이는 exp 가 00000000인 것으로 Denormalized 인 경우이다. 이 때에는 uf 의 비트를 저장한 후 uf를 << 1 연산 해 준 결과와 or 처리를 하면 2배 값을 얻을 수 있다. 왜냐하면 Denormalized 의 경우 0.f1f2f3... * E^-126 인데, 이것을 2배 해 준 결과는 frac 부분을 그저 2배 해 주면 된다. 만약 frac 부분을 두배 해 주었더니 1.f1'f2'... 이 되었다면 그 결과는 exp 에 1을 더해 Normalized가 된다.

Case 3) 이외에는 Normalized 이다. 이 때에는 그저 exp 의 값을 1 만큼 더해주면 된다. (exp 는 frac 의 23 비트 후에 있으므로 1 <<23의 값, 즉 0x800000을 더해줘야 한다. 이미 exp 가 11111111인 경우에 대한 처리를 했으므로 1을 더할 때 따로 처리해야 할 예외는 없다.

코드 구현은 다음과 같다.

```
unsigned float_twice(unsigned uf)
{
    unsigned int inf = 0x7f800000;
    if ((uf & inf) == inf)
    {
```

```
        // 만약 이미 inf 또는 NaN 이라면 바로 반환.
        return uf;
    }
    if (!(uf & inf))
    {
        // Denormalized 일 경우
        return (uf & 0x80000000) | (uf << 1);
    }
    // Normalized 일 경우
    return uf + 0x800000;
}
```

2.5 float_i2f

본 문제는 int를 float로 변환하는 문제이다.

먼저 x 의 부호를 잘 가져와두자. x & 0x80000000 로 MSB를 가져올 수 있다. 이후 x 의 절대값을 가져오자. sign 이 1이라면 음수인 것이기에 –x를, sign 이 0이라면 양수인 것이기에 x를 택한다.

절대값을 abs_val 이라 하자. abs_val 에 대하여 E 값을 구해야 한다. (원본이 int 이기에 denormalized 일 수 없다.) abs_val의 값을 변수에 담고, 오른쪽으로 쉬프트해가며 0인지 확인한다. 0이 되었을 때 쉬프트한 횟수가 abs_val 의 유효한 비트 수이다. 가장 앞의 1 부분은 1.xxx 에서의 1 부분이므로 이를 고려하면 쉬프트한 횟수 –1 이 E 값이 된다.

구한 E 값을 이용해 frac 와 round 부분을 나누어 구해보자. frac 는 abs_val을 왼쪽으로 32 – E 번 만큼 쉬프트해주어 MSB부터 frac 이 채워지도록 한 후 오른쪽으로 9번 쉬프트해 구할 수 있다. frac이 상위 9비트를 제외하고 채워져 실제 frac 의 비트와 같아지도록 구할 수 있다.

round 값은 abs_var을 왼쪽으로 32-E 만큼 쉬프트해 MSB부터 frac 이 채워지도록 한 후 0x1FF 와 & 연산을 하여 하위 9비트를 round 로 가져올 수 있다. 이후 만약 round 의 값이 256 보다 크거나(나머지가 0.5 보다 크거나), 256 이면서 frac 의 LSB 가 1이라면 (나머지가 0.5 이고 반올림하면 짝수가 되는 상황이기에) 반올림을 수행한다. 이는 frac 의 값을 1만큼 더해주면 된다. 만약 frac 의 값이 0이 되며 하위 23비트 외에 24번째 비트로 오버플로 되어 exp 의 값을 1 더해주게 되더라도 마지막에 부호, exp, frac 부분을 모두 더해줄 때 반영된다.

```
unsigned float_i2f(int x)
{
    unsigned int sign = x & 0x80000000;
    unsigned int abs_val = (sign) ? -x : x;
    unsigned int shifted_val = abs_val;
    unsigned int len_bit = 0;
    unsigned int exp, E, frac, round;
    if (x == 0)
    {
        return 0;
    }
    while (shifted_val)
    {
        len_bit++;
        shifted_val >>= 1;
    }
    E = len_bit - 1;
    exp = E + 127;
    frac = abs_val << 32 - E >> 9;
```

```
    round = (abs_val << 32 - E) & 0x1FF;
    if (round > 256 || round == 256 && frac & 1)
    {
        frac++;
    }
    return sign + (exp << 23) + frac;
}
```

2.6 float_f2i

본 문제는 float 변수를 int로 바꾸는 문제이다.
uf를 오른쪽으로 23번 쉬프트한 후 0xff 와 & 연산을 취해 exp 부분을 추출한다. 또한 bias 는 자명히 0x7f 이다.
E 값은 아까 구한 exp에서 bias를 빼서 구할 수 있다.
만약 E 값이 0보다 작다면 (이 경우에 Denormalized 도 포함된다.) 1보다 작은 값이 되므로 결과는 자명히 0이다. (float를 int로 바꿀 때에는 소수점을 버린다.)
만약 exp 가 11111111 이거나(float 가 inf 또는 NaN) E 값이 31보다 크다면(int 로 바꿀 때 오버플로우 발생) 문제의 조건에 따라 0x80000000u를 반환한다.
위 두 경우가 아니라면 1.frac~을 만들어 주기 위해 frac 에다가 1 << 23을 더해준다. 지금 변수에 들어있는 값은 1.frac * 2^23이다. 이를 다시 실제 2^E에 맞게 수정해야 하는데, 편의를 위해 7 만큼 더 왼쪽으로 쉬프팅해 1.frac * 2^30으로, 최상위 부호 비트에 0, 그다음 비트는 1, 그다음 frac이 오도록 한 후 30-E 만큼 다시 오른쪽으로 쉬프팅하면 원래 구하려는 1.frac * 2^E를 얻을 수 있다.
반약 float 의 부호가 음수였다면 res 의 two's complement 로 음수를, 아닐 경우 양수를 그대로 반환한다.

```
int float_f2i(unsigned uf)
{
    unsigned int exp = uf >> 23 & 0xff;
    unsigned int bias = 0x7f;
    int E = exp - bias;
    unsigned int frac = uf & 0x7fffff;
    unsigned int sign_bit = uf & 0x80000000;
    unsigned int res;
    if (E < 0)
    {
        // E 가 0보다 작다면(Denormalized인 경우도 여기 포함) 0을 반환
        return 0;
    }
    if (exp == 0xff || E > 31)
    {
        // 만약 초기 인풋이 inf 또는 NaN 이거나 결과가 INT 기준 오버플로라면 바로 반환.
        return 0x80000000u;
    }
    res = frac + (1 << 23) << 7 >> 30 - E;
    return sign_bit ? ~res + 1 : res;
}
```

3. Results

본 과제를 통하여 정수, 부동 소수점 자료형의 활용 및 평소 인지하지 못한 형 변환 과정에서의 규칙 등을 이해할 수 있었다.

4. References

None.