Lab Homework 2 Report

이번 랩 시간에 2's Complement 연산과 Floating point의 비트 표현에 대해 배웠다. 2's Complement에서 x와 부호가 반대인 –x를 연산하기 위해서는 ~x+1로 계산해야 한다. IEEE의 표준 Floating point 표현법을 배우고, floating point number를 비트로 표현하는 법을 배웠다. Exp가 00…0일 경우, 11…1인 경우, 그 외의 경우는 각각 Denormalized Value, Special Value, Normalized Value이다. 이와 관련해 6개의 문제가 이번 과제로 나왔고, 각 문제의 해결 과정은 다음과 같다.

1. Problem1 (negate)

- 본 과제의 Problem 1은 정수 x를 입력 받아 –x를 반환하는 함수를 구현하는 문제이다.
- '~' 연산자를 사용 후 1을 더해 해당 기능을 구현하였다. 함수에서 return하는 값은 다음과 같다.

  ~x + 1

2. Problem2 (isLess)

- 본 과제의 Problem2는 두 정수 x, y를 입력 받아 x가 y보다 작으면 1, 그렇지 않으면 0을 반환하는 함수를 구현하는 문제이다.
- 먼저 y가 가장 작은 정수이면 함수의 return값은 항상 0이므로 변수 s를 설정하여 이를 확인하였다. 이때 mMax는 signed int에서 표현할 수 있는 가장 작은 정수이다.
- 앞 문제에서 구현한 –y 값을 변수 my로 선언하여 값을 넣고, x-y의 값, 즉 x+my가 0보다 큰지 작은지 판단하여 두 수의 크기를 비교하였다.
- 이때 덧셈 과정에서 overflow가 발생하는 경우를 처리하기 위해 MSB와 그 아래 bit에서 발생하는 carry를 각각 c1, c2로 변수를 선언하여 처리하였다.
- c1과 c2가 다른 경우에 overflow가 발생하고, overflow가 발생하면 결과 값이 반대가 되므로 XOR 연산자를 사용하여 처리하였다.
- 각 변수의 선언은 다음과 같다.

  int my = ~y + 1;

  int mMax = 1 << 31;

int c2 = ((x | mMax) + (my | mMax)) >> 31;

int c1 = (((x >> 31) & (my >> 31))) | ((x >> 31 | my >> 31) & c2);

int s = !!(y ^ mMax);

- 함수에서 return하는 값은 다음과 같다.

  s & ((((x + my) >> 31) & 1) ^ (c1 ^ c2))

3. Problem3 (float_abs)

- 본 과제의 Problem3는 unsigned로 표현된 floating point number를 입력 받아 절댓값을 반환하는 함수를 구현하는 문제이다.
- floating point의 표현 중 exp 부분이 11...1이고, fractional part가 0이 아닌 NaN의 경우를 예외처리하기 위해 if문과 변수 e를 선언하여 사용하였다.
- 그 외의 경우 양수는 입력 값을 그대로, 음수는 sign 값을 바꿔 return하였다.

4. Problem4 (float_twice)

- 본 과제의 Problem4는 unsigned로 표현된 floating point number를 입력 받아 2를 곱한 값을 반환하는 함수를 구현하는 문제이다.
- floating point의 표현 중 exp 부분이 11...1인지 확인하기 위한 변수 e1, 00...0인지 확인하기 위한 변수 e2를 선언하였다.
- exp 부분이 11...1인 경우 입력 값을 그대로 return하였고, 00...0인 경우 다시 음수인 경우와 양수인 경우로 나누어 양수일 때는 입력 값을 두 번 더한 값을 반환하였고, 음수일 때는 입력 값을 두 번 더한 값에 부호를 붙여 반환하였다.
- 그 외의 경우 exp 값에 1을 더해 2를 곱하는 연산을 처리하였다.

5. Problem5 (float_i2f)

- 본 과제의 Problem5는 signed integer를 입력 받아 floating point로 표현한 수를 반환하는 함수를 구현하는 문제이다.
- sign bit을 제외한 모든 bit이 0인 경우 예외처리 해주었다.
- sign bit, exp, f를 각각 구하여 OR 연산을 통해 floating point number를 만들었다. 각각의

연산 과정은 다음과 같다.

- x의 right shift, left shift를 차례로 연산하여 sign bit을 구한 후 음수인 경우 x에 ~x+1을 넣어 –x를 통해 연산하도록 하였다.
- x의 sign bit을 제외한 부분을 i에 넣고 while 문을 통해 i에서 최초로 1이 나오는 bit가 몇 번째인지 계산 후에 E를 계산하였고, E에 bias를 더해 exp 값을 구했다. 이때 bias는 2^(8-1)-1로 계산하였다.
- 다음으로 f를 구하기 위해 i에서 지워야 할 첫 번째 bit가 0인 경우 나머지를 버리고, 1인 경우 그 뒤에 bit들 중 하나라도 1이 있는 경우 result의 LSB를 1만큼 증가시켰다. 이때 overflow가 발생할 경우 f 부분을 0으로 만든 후 exp에 1을 더해주었다. 지워야 할 첫 번째 bit가 0이고, 그 뒤에 bit들이 모두 0인 경우 Round to even을 통해 Rounding 해주었다. 이때에도 마찬가지로 overflow가 발생할 경우 f 부분을 0으로 만든 후 exp에 1을 더해주었다.
- f 부분(코드에서 i에 해당)을 연산할 때 편의를 위해 다음과 같은 변수를 선언하였다.

  unsigned Ri = i >> 9; //i에 right shift 9만큼 해준 값

  unsigned c_Ri = Ri + 1; //Ri에 carry가 발생한 경우

  unsigned c_exp = exp + 1; //overflow가 발생하여 exp에 1을 더해주는 경우

  int isOverflow = !(c_Ri ^ 0x00800000); //overflow가 발생하는지 여부를 판단하기 위해 선언한 변수
- sign bit은 변수 sign, exp는 변수 exp에 left shift를 23만큼 해준 값, f는 변수 i이다. 함수의 return값은 다음과 같다.

  return sign | exp << 23 | i;

6. Problem6 (float_f2i)

- 본 과제의 Problem6은 unsigned로 표현된 floating point number를 입력 받아 int형 정수를 반환하는 함수를 구현하는 문제이다.
- unsigned로 표현된 floating point number의 e=11...1인 경우, 즉 infinite number이거나 NaN인 경우 예외처리 해주었다.
- 32-bit signed 정수로 나타낼 수 있는 수의 범위는 -2^31부터 2^31-1이다. E=exp-bias로 계산할 수 있고, int형으로 표현할 수 없는 범위 밖의 수도 예외처리 해주었다.

- 먼저 int형으로 나타낼 수 있는 범위 아래의 수들은 0을 return하도록 하였다.

  다음으로 int형으로 나타낼 수 있는 범위보다 큰 수들은 0x80000000u를 return하도록 하였다.

- int형으로 나타낼 수 있는 범위의 수들에 대해 연산하였다. exp를 구하고, E=exp-bias를 활용하여 E 값을 계산하였다. 그리고 fractional part를 포함하여 1.XXX 형식으로 표현될 수를 변수 M에 저장하였다.

- 변수 M에서 실제 소수점은 존재하지 않지만 소수점이 존재한다고 가정한 뒤 적절한 shift 연산을 통해 int형 변수 x에 저장했다. 먼저 E가 23보다 작은 경우 M에 23 + (~E + 1)만큼, 즉 23-E만큼 right shift한 값을 x에 저장하고, E가 23보다 크거나 같은 경우 M에 E-23만큼 left shift한 값을 x에 저장하였다.

- 그런 다음 입력 값의 sign bit가 1인 경우 ~x+1, 즉 –x를 return하고, sign bit이 0인 경우 x를 그대로 return하였다.