CSED211: Data Lab2 Report

20230024 / 문요준

Overview

이 Lab 은 실수에 대한 bit-level 연산을 위한 C 프로그래밍 퀴즈이고, 인자 x 에 대해 -x 를 반환하는 negate(x), x<y 인지 확인하는 isLess(x, n), 인자 f 에 대해 절댓값을 계산하는 float_abs(uf), 2*f 를 연산하는 float_twice(uf), 인자 x 에 대해 (float) x 를 연산하는 float_i2f(x), (int)ff 를 연산하는 float_f2i(uf) 함수들을 완성하는 문제로 구성되어 있다.

1. Question #1. negate(x)

2-1. Explanation.

이 함수는 입력된 정수 x 의 부호를 반전시켜 음수를 반환하는 함수이다. 2 의 보수 연산을 사용하여 구현되며, 정수 x 는 -x 로 변환된다.

2-2. Solution.

2 의 보수는 다음의 과정으로 구할 수 있다. x 의 모든 비트를 반전(1 -> 0, 0 -> 1)시키면 ~x 가 된다. 반전된 값에 1 을 더하면 2 의 보수를 구할 수 있다. 즉, ~x + 1 이 -x 와 동일하다.

2-3. Implementation.

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

2. Question #2. isLess(x, n)

2-1. Explanation.

이 함수는 두 정수 x 와 y 를 비교하여 x 가 y 보다 작으면 1 을 반환하고, 그렇지 않으면 0 을 반환하는 함수이다. 정수의 부호와 값의 차이를 고려하여 x < y 인지 판단해야 하므로, 부호가 다를 때와 같을 때 각각 다른 방식으로 처리했다. 음수와 양수를 비교하거나 두 값이 양수 혹은 음수일 때의 처리를 비트 연산으로 구현했다.

2-2. Solution.

이 함수는 두 가지 경우를 처리해야 한다. x 와 y 의 부호가 다르면, x 가 음수일 때 무조건 y 보다 작다. 이를 부호 비트를 이용하여 처리한다. x 의 부호 비트가 1 이고, y 의 부호 비트가 0 일 경우 x < y 임이 확실하므로 이 경우 바로 1 을 반환한다. 부호가 같을 때는 x - y 의 결과를 기반으로 비교한다. x - y 의 부호가 음수이면 x 가 y 보다 작다는 의미이므로, 이를 통해 비교를 수행한다.

방법은 다음과 같다. x - y 를 계산하는 대신 2 의 보수를 사용하여 x + (~y + 1)으로 계산한다. 그리고 x 와 y 의 부호를 각각 추출하여 부호가 다른지 여부를 XOR 연산으로 확인한다. 최종적으로, 부호가 다르면 x 의 부호가 음수일 때 1 을 반환하고, 부호가 같으면 x - y 의 부호를 통해 비교한다.

2-3. Implementation.

```
int isLess(int x, int y) {
  int diff = x + (~y + 1);
  int x_sign = (x >> 31) & 1;
  int y_sign = (y >> 31) & 1;
  int diff_sign = (diff >> 31) & 1;

  int signDifferent = x_sign ^ y_sign;
  int xLessAtDifferentSign = signDifferent & x_sign;
  int xLessAtSameSign = ~signDifferent & diff_sign;

  return xLessAtDifferentSign | xLessAtSameSign;
}
```

1. x - y 연산:

- x - y 를 계산하기 위해 diff = x + (~y + 1) 연산으로, y 의 2 의 보수를 더해 x - y 를 계산한다.

2. 부호 추출:

- x_sign = (x >> 31) & 1: x 의 부호 비트를 추출한다.
- y_sign = (y >> 31) & 1: y 의 부호 비트를 추출한다.
- diff_sign = (diff >> 31) & 1: x - y 의 부호 비트를 추출한다.

3 . 부호가 다른 경우 처리:

- signDifferent = x_sign ^ y_sign: x 와 y 의 부호가 다를 때 1 이 되고, 같으면 0 이 된다.
- xLessAtDifferentSign = signDifferent & x_sign: 부호가 다를 때 x 가 음수이면 1 을 반환하여 x < y 임을 나타낸다.

4. 부호가 같은 경우 처리:

- xLessAtSameSign = ~signDifferent & diff_sign: 부호가 같을 때, x - y 의 결과에 따라 x 가 작으면 1 을 반환한다.

최종적으로, xLessAtDifferentSign | xLessAtSameSign 은 둘 중 하나가 1 이면 1 을 반환하고, 그렇지 않으면 0 을 반환하여 x < y 인지 여부를 결정한다.

3. Question #3. float_abs(uf)

3-1. Explanation.

이 함수는 입력된 부동소수점 값 uf 의 절대값을 반환하는 함수이다. 이 함수는 비트 수준에서 부동소수점 값의 부호를 제거하고, 절대값을 반환한다. 부동소수점은 부호 비트 (1 비트, 최상위 비트로 0 이면 양수, 1 이면 음수), 지수 비트 (8 비트, 수의 크기를 나타냄), 가수 비트 (23 비트, 소수점 이하의 값을 나타냄) 세 부분으로 이루어져 있다. 이 함수는 부호 비트만 제거하고, 지수와 가수 비트를 그대로 유지하는 방식으로 절대값을 계산한다. 단, 주어진 값이 NaN 일 경우에는 절대값을 계산하지 않고 원래 값을 반환한다.

3-2. Solution.

부동소수점 값의 절대값을 구하는 과정에서, 부호 비트만 0 으로 설정하고 나머지 비트를 그대로 유지한다. 부호 비트를 제거하기 위해, 부호 비트를 0 으로 만들고 나머지 비트를 그대로 유지한다. 이를 위해 0x7FFFFFFF 라는 마스크(최상위 비트를 제외한 나머지 비트가 모두 1)를 사용하여 부호 비트(최상위 비트)를 0 으로 만든다. 그리고 부동소수점 값이 NaN 인지 확인하기 위해, 지수 비트가 모두 1 이면서 가수 비트가 0 이 아닌 경우를 검사한다. 이를 위해 지수 비트만 추출하는 마스크 0x7F800000 를 사용하여, 지수 비트가 모두 1 인지를 확인하고, 가수 비트가 0 이 아닌지 검사한다. NaN 일 경우, 절대값을 계산하지 않고 원래 값을 반환한다.

3-3. Implementation.

```
unsigned float_abs(unsigned uf) {
  unsigned mask_sign = 0x7FFFFFFF;
  unsigned abs_val = uf & mask_sign;

  unsigned mask_exp = 0x7F800000;
  unsigned mask_frac = 0x007FFFFF;

  if ((uf & mask_exp) == mask_exp && (uf & mask_frac) != 0) {
    return uf;
  }
  return abs_val;
}
```

1. 부호 비트 제거:

- uf & mask_sign 는 부호 비트를 제거하고, 나머지 비트는 그대로 유지한다. mask_sign = 0x7FFFFFFF 는 최상위 비트를 제외한 나머지 비트가 모두 1 인 마스크이다. 이 연산을 통해 uf 의 부호 비트를 0 으로 만들어 절대값을 계산한다.

2. NaN 확인:

- mask_exp = 0x7F800000 는 지수 비트를 추출하기 위한 마스크이고, mask_frac = 0x007FFFFF 는 가수 비트를 추출하기 위한 마스크이다. 만약 uf 의 지수 비트가 모두 1 이면서 가수 비트가 0 이 아닌 경우는 NaN 이다. 이 경우 절대값을 계산하지 않고 원래 값을 반환한다.

4. Question #4. float_twice(uf)

4-1. Explanation.

이 함수는 부동소수점 값 f 를 2 배 한 값을 반환한다. 지수 비트를 증가시키거나, 지수 비트가 모두 0 인 경우에는 가수 비트를 왼쪽으로 시프트하여 처리한다. 입력값이 NaN 인 경우 원래 값을 그대로 반환한다.

4-2. Solution.

이 문제를 해결하기 위한 과정은 다음과 같다. 입력 값 uf 에서 부호 비트, 지수 비트, 가수 비트를 분리한다. 그리고 입력 값이 NaN 인지 확인하기 위해 지수 비트가 모두 1(즉, exp == 0xFF)이고, 가수 비트가 0 이 아닌 경우를 검사한다. 이 경우, 입력 값을 그대로 반환한다. 이때 지수 비트가 0 일 경우, 가수 비트를 왼쪽으로 1 비트 시프트하여 2 배를 만든다. 이때 가수 비트의 최상위 비트가 1 로 변하면 지수를 1 로 증가시키고, 가수 비트의 최상위 비트를 제거해준다. 지수 비트가 0 이 아닌 경우, 지수 비트를 1 증가시켜 부동소수점 값을 2 배로 만든다. 만약 지수 비트가 0xFF 로 증가하면 가수 비트는 0 으로 설정되고, 이는 무한대를 의미한다. 최종적으로 부호 비트, 지수 비트, 가수 비트를 결합하여 2 배가 된 부동소수점 값을 반환한다.

4-3. Implementation.

```
unsigned float_twice(unsigned uf) {
  unsigned sign = uf & 0x80000000;
  unsigned exp = (uf >> 23) & 0xFF;
  unsigned frac = uf & 0x007FFFFF;

  if (exp == 0xFF) {
    return uf;
  }

  if (exp == 0) {
    frac = frac << 1;
    if (frac & 0x00800000) {
      exp = 1;
      frac &= 0x007FFFFF;
    }
  } else {
    exp += 1;
    if (exp == 0xFF) {
      frac = 0;
    }
  }

  return sign | (exp << 23) | frac;
}
```

1. 부호, 지수, 가수 분리:

- sign = uf & 0x80000000 는 부호 비트를 추출하고, exp = (uf >> 23) & 0xFF 는 지수 비트를 추출하며, frac = uf & 0x007FFFFF 는 가수 비트를 추출한다.

2. NaN 처리:

- exp == 0xFF 일 때, 입력 값이 NaN 이거나 무한대이므로 원래 값을 반환한다.

3 . 지수비트가 0 일 때:

- 지수 비트가 0 이면 가수 비트를 왼쪽으로 1 비트 시프트하여 2 배로 만든다. 만약 시프트 후 가수의 최상위 비트가 1 로 바뀌면 지수를 1 로 증가시키고, 가수의 최상위 비트를 제거한다.

4. 지수비트가 0 이 아닐 때:

- 지수 비트가 0 이 아니면 지수를 1 증가시킨다. 만약 지수가 0xFF 가 되면, 가수를 0 으로 설정하여 무한대를 나타낸다.

최종적으로 sign | (exp << 23) | frac 을 통해 부호, 지수, 가수를 결합하여 최종적으로 2 배가 된 부동소수점 값을 반환한다.

5. Question #5. float_i2f(x)

5-1. Explanation.

이 함수는 입력된 정수 x 를 지수 가수 연산, 반올림 처리를 통해 부동소수점 값으로 변환하여 반환하는 함수이다.

5-2. Solution.

정수를 부동소수점 값으로 변환하는 과정은 다음과 같다. 입력 값이 0 일 경우, 부동소수점 표현도 0 이므로 바로 0 을 반환한다. 정수 x 가 음수인 경우, 부호 비트를 1 로 설정하고 2 의 보수를 사용하여 절대값을 구한다. 정수의 이진 표현에서 가장 왼쪽에 있는 1 의 위치를 찾는다. 이는 부동소수점의 지수 값을 계산하는 데 필요하다. 이 위치를 기준으로 지수는 shift + 127 이 된다. x 의 이진 표현을 왼쪽으로 정렬하여 소수점 이하 부분인 가수 비트을 계산한다. 가수는 23 비트로 표현되며, 상위 비트를 가수 비트로 추출한다. 가수 비트를 23 비트로 표현할 때, 하위 비트를 잘라내야 한다. 이때 반올림을 고려하여, 잘린 부분이 0x80 이상이거나 반올림 규칙에 맞으면 가수에 1 을 더해 반올림을 수행한다. 만약 반올림 후 가수 비트가 23 비트를 넘어가면, 지수를 1 증가시키고 가수 비트를 다시 정렬한다. 부호 비트, 지수 비트, 가수 비트를 합쳐 최종적으로 부동소수점의 비트 표현을 반환한다.

5-3. Implementation.

```
unsigned float_i2f(int x) {
  unsigned sign, exp, frac, round_part, round_mask;
  unsigned abs_x;
  int shift;

  if (x == 0) {
    return 0;
  }

  sign = x & 0x80000000;

  if (x < 0) {
    abs_x = ~x + 1;
  } else {
    abs_x= x;
  }

  shift = 31;
  while ((abs_x & (1 << shift)) == 0) {
    shift--;
  }

  exp = shift + 127;

  abs_x = abs_x << (31 - shift);
  frac = (abs_x >> 8) & 0x007FFFFF;

  round_mask = 0x000000FF;
  round_part = abs_x & round_mask;

  if (round_part > 0x80 || (round_part == 0x80 && (frac & 1))) {
    frac += 1;
    if (frac & 0x00800000) {
      frac = 0;
      exp += 1;
    }
  }
  return sign | (exp << 23) | frac;
}
```

1. 부호 비트 설정:

- sign = x & 0x80000000 는 x 의 부호 비트를 추출하여 양수인 경우 0, 음수인 경우 1 을 설정한다.

2. 절대값 계산:

- x 가 음수일 경우 2 의 보수를 사용하여 절대값을 계산한다. x < 0 일 때 ~x + 1 로 음수를 양수로 변환한다.

3 . 최상위 1 비트 위치 찾기:

- 정수의 최상위 1 비트 위치를 찾기 위해 반복문을 사용하여 shift 값을 감소시키면서 가장 왼쪽의 1 을 찾는다.

4. 지수 계산:

- 지수는 최상위 1 비트의 위치(shift)에 127 을 더한 값이다. 이는 부동소수점 표현에서 지수를 나타내는 방식이다.

5. 가수 계산:

- abs_x 를 왼쪽으로 정렬한 후, 상위 23 비트를 가수로 추출한다. frac = (abs_x >> 8) & 0x007FFFFF 는 가수 비트만 추출하는 과정이다.

6. 반올림 처리:

- 잘려나간 하위 비트를 기준으로 반올림을 수행한다. round_part > 0x80 이거나, 반올림 규칙에 맞을 경우 가수에 1 을 더한다. 반올림 후 가수가 23 비트를 넘으면 지수를 1 증가시키고 가수를 정리한다.

6. Question #5. float_f2i(uf)

6-1. Explanation.

이 함수는 부동소수점 값을 정수로 변환하는 함수이다. 범위를 벗어나거나 NaN 또는 무한대일 경우 0x80000000u 를 반환한다. 정수 범위에 미치지 못하는 너무 작은 값은 0 으로 반환한다.

6-2. Solution.

먼저 NaN 또는 무한대 값이 입력될 경우, 지수 비트가 255(0xFF)가 된다. 이때는 변환이 불가능하므로 0x80000000u 를 반환한다. 지수는 E = exp - 127 로 계산되며, E 는 실제로 수를 2 의 몇 제곱으로 곱해야 하는지 나타낸다. 만약 E < 0 이라면 소수 부분이므로, 정수로 변환하면 0 이 된다. 만약 E > 30 이면 32 비트 정수의 범위를 초과하므로 0x80000000u 를 반환한다. 가수는 frac | 0x00800000 로 상위 비트를 1 로 설정하여 정규화된 가수를 만든다. E 에 따라 가수를 왼쪽 또는 오른쪽으로 시프트하여 2 의 제곱 승에 해당하는 값을 곱해준다. E 가 23 보다 크면 왼쪽 시프트, 23 보다 작으면 오른쪽 시프트를 한다. 마지막으로, 부호 비트가 1 이면 음수로 변환하고, 그렇지 않으면 양수로 반환한다.

6-3. Implementation.

```
int float_f2i(unsigned uf) {
  unsigned mantissa;
  unsigned sign = uf >> 31;
  unsigned exp = (uf >> 23) & 0xFF;
  unsigned frac = uf & 0x007FFFFF;

  int E = exp - 127;

  if (exp == 255) {
    return 0x80000000u;
  }

  if (E < 0) {
    return 0;
  }

  if (E > 30) {
    return 0x80000000u;
  }

  mantissa = (frac | 0x00800000);

  if (E > 23) {
    mantissa = mantissa << (E - 23);
  } else {
    mantissa = mantissa >> (23 - E);
  }

  if (sign) {
    return -mantissa;
  } else {
    return mantissa;
  }
}
```

1. 부호, 지수, 가수 추출:

- sign = uf >> 31: 부호 비트를 추출하여 양수면 0, 음수면 1 로 설정한다.
- exp = (uf >> 23) & 0xFF: 지수 비트를 추출하여 부동소수점의 지수 부분을 구한다.
- frac = uf & 0x007FFFFF: 가수 비트를 추출하여 소수 부분을 구한다.

2. NaN 또는 무한대 처리:

- 지수 비트가 255 일 경우 이는 NaN 또는 무한대를 의미하므로, 변환할 수 없다. 이때는 0x80000000u 를 반환한다.

3 . 지수 값 처리:

- 지수 값 E = exp - 127 을 계산하여 실제 지수 값을 구한다.
- 만약 E < 0 이라면, 이는 소수점 이하의 값이므로 0 을 반환한다.
- E > 30 이면 정수로 변환할 수 있는 범위를 초과하므로 0x80000000u 를 반환한다.

4. 가수 값 처리:

- mantissa = (frac | 0x00800000)로 가수의 최상위 비트를 설정하여 정규화된 가수를 만든다.
- E 에 따라 가수를 왼쪽 또는 오른쪽으로 시프트하여 실제 정수 값을 계산한다.

5. 부호 처리:

- sign 비트에 따라 양수 또는 음수로 변환하여 최종적으로 정수 값을 반환한다.

Results

```
[yojun313@programming2 datalab-floating-point]$ ./btest
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
Total points: 19/19
```