CSED211 LAB1 REPORT

20220778 표승현

#negate

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

2의 보수 형식을 취해 음수를 반환한다.

isLess

```
int isLess(int x, int y) {
    int signX = (x >> 31) & 1;
    int signY = (y >> 31) & 1;

    int signDiff = signX ^ signY;
    return (signDiff & signX) | (!signDiff & ((x + (~y + 1)) >> 31 & 1));
}
```

Sign 부호가 같을 때와 다를 때를 각각 고려한다. 부호가 다르다면 x가 음수일 때 1을 출력하게 한다. 부호가 같을 때에는 x에서 y를 뺀 값이 양수인지 음수인지를 판단하여 대소를 비교한다.

#float_abs

```
unsigned float_abs(unsigned uf) {
    unsigned mask = 0x7FFFFFFF;
    unsigned exp = uf & 0x7F800000;
    unsigned frac = uf & 0x007FFFFF;
```

```
    if (exp == 0x7F800000 && frac != 0) {
        return uf;
    }
    else {
        return uf & mask;
    }
}
```

Exp가 모두 1이고, frac이 1이 아니면 Nan값을 그대로 출력하게 한다. 이외에는 sign비트가 항상 0이 나오도록 mask를 &연산해준다.

```
#float_twice
unsigned float_twice(unsigned uf){
  unsigned sign = uf & 0x80000000;
  unsigned exponent = uf & 0x7F800000;
  unsigned fraction = uf & 0x007FFFFF;

  if (exponent == 0x7F800000) {
    return uf;
  } else if (exponent == 0) {
    fraction <<= 1;
    if (fraction & 0x00800000) {
      exponent = 0x00800000;
    }
  } else {
    exponent += 0x00800000;
    if (exponent == 0x7F800000) {
```

```
        return sign | 0x7F800000;
    }
  }

  return sign | exponent | fraction;
}
```

Sign, exp와 frac부분을 각각 추출한다. 먼저 exp의 비트가 모두 1인 경우 special value를 그대로 출력한다. 이외의 경우에는 2배 연산을 실시한다. Exp가 모두 0인 비정규화 값인 경우에는 frac에 right shift 연산을 취해 2배를 해준다. 이때 frac이 오버플로우되면 exponent의 lsb를 1로 설정하여 정규화한다. 정규화 값인 경우에는 exp에 1을 더해 2배 연산을 수행한다. Exp가 모두 1이 된다면 special value를 출력한다. 이후 sign, exp, frac을 모두 합쳐 2배 연산값을 출력한다.

#float_i2f

```
unsigned float_i2f(int x) {
    unsigned sign = 0;
    unsigned ux, exp, frac, round;

    if (x == 0) return 0;

    if (x < 0) {
        sign = 1;
        ux = -x;
    } else {
        ux = x;
    }

    exp = 31;
```

```
    while (!(ux & 0x80000000)) {
        ux <<= 1;
        exp--;
    }
    frac = (ux >> 8) & 0x7FFFFF;

    round = (ux & 0xFF) > 0x80 || ((ux & 0xFF) == 0x80 && (frac & 1));

    if (round) {
        frac++;
        if (frac > 0x7FFFFF) {
            frac = 0;
            exp++;
        }
    }

    exp += 127;

    return (sign << 31) | (exp << 23) | frac;
}
```

X가 0인 경우 그대로 0을 출력한다. 0을 표현하는 float도 모든 비트가 0이기 때문에 같은 결과를 보인다.

연산을 용이하게 하기 위해 sign 비트를 0으로 통일한다.

절댓값을 취한 정수의 유효숫자의 시작이 msb가 되도록 while문을 이용해 left shift 해준다. 이때 exp를 31로 설정한 후 loop가 돌 때마다 감소시켜 자릿수를 계산한다. 얻은 유효숫자를 8비트만큼 right shift하여 frac의 비트 자리와 맞춰준다. 정수의 비트가 23비트가 넘어가는 경우 부동소수점 유효숫자로 다 표현할 수 없는데, 이때 rounding을 수행한다.

(ux & 0xFF) > 0x80: 최상위 8비트가 0xFF인 경우 0.5에 가장 가까운데 이보다 크면 반올림한다.

((ux & 0xFF) == 0x80 && (frac & 1)): 23비트 이후가 0.5이고 frac의 최하위 비트가 1이면 홀수이므로 짝수로 반올림한다.

이러한 조건에 부합하면 frac을 1 증가시켜 반올림한다. Frac이 오버플로우되면 frac을 0으로 설정하고 exp를 1 증가시킨다. exp에는 2^7-1=127을 더해준다.

이후 부동소수점 각 요소에 맞게 비트 자리를 조정한 후 합친 값을 출력한다.

#float_f2i

```
int float_f2i(unsigned uf) {
    int sign = uf >> 31;
    int exp = ((uf >> 23) & 0xFF) - 127;
    int frac = (uf & 0x7FFFFF) | 0x800000;

    if (exp < 0) return 0;
    if (exp > 30) return 0x80000000u; // 오버플로우

    if (exp > 23) frac <<= (exp - 23);
    else frac >>= (23 - exp);

    if (sign) frac = -frac;

    return frac;
}
```

Float형 입력으로부터 sign, exp, frac을 추출한다. Sign의 경우 msb이다. Exp는 e-(2^k-1)+1을 통해 구할 수 있다. Frac은 맨 뒷자리 23비트를 추출한 후 맨 앞자리에 1비트를 돌려주면 된다.

Exp가 0보다 작으면 정수로 표현할 때 0이 된다.

Exp가 30을 초과하면 32비트 정수로 표현할 수 없으므로 오버플로우이다. 따라서 0x80000000u를 반환한다.

Exp가 23 초과면 frac에 2의 exp-23 제곱을 수행한다. Frac이 1.xxx … 의 23개의 유효숫자 비트

로 표현되기 때문이다.

반대로 exp가 23이하면 2의 23-exp 제곱만큼 frac에서 나눠준다.

이후 sign 값을 설정해준 후 반환한다.