CSED211 LAB1 REPORT

20220778 표승현

#Homework1-1

```
int bitNor(int x, int y) {
  //to be implemented
  return ~x & ~y;
}
```

~(x|y)는 드모르간 법칙에 의해 (~x)&(~y)로 표현될 수 있다. ~과 &만으로 구현하기 위해 변형된 이 식을 이용한다.

#Homework1-2

```
int isZero(int x) {
  //to be implemented
  return !x;
}
```

!는 논리연산자로 bitwise 연산자와 달리 전체적 값에 대한 연산을 수행한다. X가 0이면 1을, 이외의 값이면 1을 반환한다.

#Homework1-3

```
int addOK(int x, int y) {
  //to be implemented
        int signX = x >> 31;
        int signY = y >> 31;

        int sum = x + y;
        int signS = sum >> 31;

        int output = ((signX ^ signS) & (signY ^ signS)) >> 31;
        return !output;
}
```

오버플로우 발생 여부를 기준으로 덧셈의 유형을 구분하자면

- 양수+양수

- 양수+음수 (또는 음수+양수)

- 음수+음수

과 같이 나눌 수 있다.

양수+음수 (또는 음수+양수)처럼 서로 다른 부호의 수를 더할 때에는 오버플로우가 발생하지 않는다. 서로 절댓값을 작게 만들어주는 역할을 하여 비트 제한으로 인한 표현의 한계를 넘어설 수 없다.

오버플로우가 발생할 수 있는 위의 경우를 제외한 같은 부호끼리의 덧셈이다. 양수+양수의 경우 합의 결과가 표현의 범주를 넘어서게 되면 음수의 값을 내놓게 된다. 반대로 음수+음수의 경우에는 오버플로우가 발생하는 경우 양수의 값을 내놓게 된다. 오버플로우가 발생할 때 부호가 역전되는 현상에 주목하여 구현한다.

Right shift 연산을 통해 x, y의 부호를 추출한다. X+y의 값을 구한 후 right shift 연산을 통해 sum의 부호도 추출한다. X의 부호와 sum의 부호, y의 부호와 sum의 부호가 동시에 달라야 한다. 이를 논리식으로 작성하면 !((signX ^ signS) & (signY ^ signS))과 같이 표현할 수 있다.

#Homework1-4

```
int absVal(int x) {
  //to be implemented
        int signBit = x >> 31;
        int output = (signBit&((~x)+1))|((~signBit)&x)
  return output;
}
```

절댓값을 구하는 함수를 구현하기 위해 x가 양수인지 음수인지를 먼저 판단한다. 양수라면 그대로 값을 반환한다. 음수일 경우 2's complement 변환을 취해 같은 절댓값의 양수로 바꿔준 후 반환한다.

#Homework1-5

```
int logicalShift(int x, int n) {
  //to be implemented
        int mask = (~0 << (31 + (~n + 1))) << 1;
        int shiftedX = x >> n;
        int output = shiftedX & mask;
  return output;
}
```

n번째 비트 이후의 비트를 필터링하기 위해, n번째 비트까지는 1, 그 이후는 0의 값을 가지는 32비트 mask를 구한다. ~0을 통해 모든 비트가 1인 값을 가진 후 31-n의 수만큼 left shift를 취해준다. 그러면 n+1번째 비트까지만 1의 값을 가지는 값을 가질 수 있고, 1만큼 추가적인 left shift를 취해주면 n번째 비트까지만 1을 가지는 mask를 얻을 수 있다. X를 n만큼 right shift 취해준 값에 mask를 &연산해주면 bitwise 연산을 통해 n+1번째 비트부터 모두 0의 값을 가지는 logical shift된 값을 얻을 수 있다.