Lab 1

20200437 김채현

Problem 1

```
int bitAnd(int x, int y) {
    return ~(~x | ~y);
}
```

본 문제는 ~과 |을 이용하여 bitwise And를 구현하는 문제이다. or과 not은 functionally complete 하므로 and를 구현할 수 있다. 드모르간 법칙을 쓰면 x & y = ~(~x | ~y)이다.

Problem 2

```
int addOK(int x, int y) {
    int xMSB = (x >> 31) & 1; //negative 1, positive 0
    int yMSB = (y >> 31) & 1;
    int difMSB = xMSB ^ yMSB; //if different sign 1, if same 0
    int sumMSB = ((x + y) >> 31) & 1; //negative 1, positive 0
    int sameSumMSB = !(sumMSB ^ xMSB); //if sum and x are same sign 1, otherwise 0
    return difMSB | sameSumMSB;
}
```

본 문제는 x+y를 수행했을 때 overflow가 발생하는 지를 확인하는 함수 addOK를 구현하는 문제이다. x의 MSB를 확인하기 위해 (x>>31)&1 연산을 수행해주었고 만약 x가 negative라면 결과값이 1이, positive라면 0이 나올 것이다. y의 MSB를 확인하기 위해서도 같은 과정을 수행해주었다.

만약 x, y의 부호가 다르다면, 즉 x의 MSB와 y의 MSB가 다르다면 overflow가 발생하지 않을 것이므로 difMSB 값이 1이라면 overflow가 발생하지 않는다고 말할 수 있다.

x, y의 부호가 같을 때, x+y를 더한 값의 부호와 x, y의 부호가 같다면 overflow가 발생하지 않은 것이다. 따라서 x+y의 MSB인 sumMSB와 x의 MSB가 같다면 sameSumMSB 값이 1이 될 것이고 overflow가 발생할 것이다.

difMSB 값과 sameSumMSB 값 중 하나만 1이라면 overflow가 발생하지 않은 것이므로 difMSB | sameSumMSB 연산값을 return 하면 된다.

Problem 3

```
int isNegative(int x) {
    return (x >> 31) & 1; //if negative 1, positive 0
}
```

본 문제는 x가 음수인지 양수인지 판별하는 함수 isNegative를 구현하는 문제이다. x의 MSB가 1인지 0인지 알기 위해 x>>31을 해준다. MSB가 0이라면 ....0으로 끝날 것이고, 1이라면 ....1로 끝날 것이다. 1과 bitwise And 연산을 실행해주었을 때 0으로 끝난다면 0이, 1로 끝난다면 1이 나올 것이다. x<0인 경우 1을 return, 반대 경우에 0을 return하라고 하였으므로 연산의 결과값을 return하면 된다.

Problem 4

```
int logicalShift(int x, int y)
{
    int leftShift = 32 + (~y + 1); //32-y
    int bitMask = ~((~0) << leftShift); //000...111 that has y 0's
    return (x >> y) & bitMask;
}
```

본 문제는 logical right shift를 구현하는 문제이다. C에서는 right shift로는 arithmetic shift를 수행하기 때문에 logical right shift를 위해서 따로 구현이 필요하다. 따라서 shift의 결과로 왼쪽에 0만큼의 y가 있어야 하는데, 이를 위해서는 먼저 x>>y로 right arithmetic shift를 수행해준 후 000....111 (y개의 0을 가지고, 32-y개의 1을 가진)과의 bitwise And연산을 수행해주면 된다. 000....111 (y개의 0을 가지고, 32-y개의 1을 가진)를 만들기 위해서는 ~0에 32-y만큼 left shift를 수행해준 후 -그러면 111....000 (y개의 1을 가지고, 32-y개의 0을 가진) 이 만들어진다. – bitwise Not 연산을 수행해주면 된다.

Problem 5

```
int bitCount(int x) {
    int bitMask = 1 | 1 << 8 | 1 << 16 | 1 << 24; //00..1 00..1 00..1 00..1
    int byteMask = 0xFF; //00..0 00..0 00..0 11..1
    int bitSum = (x & bitMask) + ((x >> 1) & bitMask) + ((x >> 2) & bitMask) + ((x >> 3) & bitMask)
        + ((x >> 4) & bitMask) + ((x >> 5) & bitMask) + ((x >> 6) & bitMask) + ((x >> 7) & bitMask);
    //store sum of 1's in each byte at bytes
    int byteSum = (bitSum & byteMask) + ((bitSum >> 8) & byteMask) + ((bitSum >> 16) & byteMask)
        + ((bitSum >> 24) & byteMask);
    // sum of 1's in bytes
    return byteSum;
}
```

본 문제는 x에 있는 1의 개수를 반환하는 함수 bitCount를 구현하는 문제이다. 각 byte에 있는 bit 1의 개수를 센 다음에 이를 합하는 방법으로 문제를 구현하였다. bitMask는 각 byte에 있는 bit 1의 개수를 세기 위한 mask이다. 00...1 00...1 00...1 00...1로 각 byte에서 마지막 bit만 1로 이루어진 4 byte integer이다. bitSum은 각 byte에 있는 1의 개수를 각 byte에 저장하도록 한다. 이를 위해서는 bitSum을 4 byte integer가 아닌 각 byte를 앞에 24개의 0을 가진 하나의 integer 느낌

으로 보아야한다.

다음으로 각 byte에 저장된 1의 개수를 합해야 한다. 계산을 위해 mask로 00..0 00..0 00..0 11..1 (24개의 0을 가지고, 8개의 1을 가진) 숫자를 만들었다. 물론 byte mask란 말은 쓰이지 않지만 편의를 위해 변수 이름을 byteMask로 정하였다. bitSum와 byteMask의 bitwise And 계산으로 끝에서 마지막 byte의 1의 개수를, bitSum>>8과 byteMask의 bitwise And 계산으로 끝에서 두 번째 byte의 1의 개수를, ... 이런 식으로 4개의 byte들에 저장된 1의 개수를 각각 더하여 byteSum에 저장해준 후 return한다.