2019 시스템 프로그래밍
- Lab 03. DataLab-

제출일자	2019.10.06
분 반	01
이 름	임준규
학 번	201602057

실습 1

1	bitOr(int x, int y)

&와 ~을 이용해 |를 만드는 함수.
&구문에 ~이 적용되면 &가 |이 된다는 점을 집중적으로 생각하여 설계하였다.
만들어 낸 & 구문 전체에 ~을 씌워 x|y를 만들기 위하여 거꾸로 생각해,
x|y에 ~두개를 붙여보았다.

~(~(x|y))=~(~x&~y)

2	tmax(void)

2의 보수 중 가장 큰 값을 반환하는 것이 목표이다.
2의 보수는 모든 정수의 범위가 2의 보수의 범위이므로
가장 큰 값을 반환하는 것과 같다 생각했다.
int형 정수의 최대값은 0b01111...과 같으므로 0b1000...에 ~연산을 한 것과 동일하다.

~(1<<31)

3	negate(int x)

signed int에서의 음수는 2의 보수라 볼 수 있다.
이 문제는 x의 2의 보수를 반환하라는 것으로 볼 수 있다.
2의 보수는 ~연산 이후에 1만 더하면 된다.

~x+1

4	getByte(int x, int n)

n자리의 byte의 값를 반환하는 문제이다.

이 문제를 해결하며 비트마스크라는 것을 배웠는데,
이는 추출이 필요한 비트/바이트에만 값을 줘서 &연산으로 그 값을 추출하게 해준다.

우선 1byte는 8bit로 구성이 된다. n의 자리에 있는 byte를 0자리의 byte로 데려와

추출하기 위해, 8n만큼의 shift가 필요하다.
이때 곱셈 연산자를 사용하지 못하므로 shift연산자로 곱셈을 표현해주어야 한다.
8n = n>>3과 같이 표현 할 수 있으므로, 0번째 byte로 추출해주기 위하여
x>>(n<<3)로 표현 할 수 있다. 하지만 이 경우는 n이 최상위 byte를 지정하지 않으면
남은 byte도 함께 추출되므로 마스크를 통해 필요한 값만 추출해준다.
우리는 첫 8bit만 필요하므로 첫 8bit가 모두 1인 255=0xFF를 사용하여 추출해주자.

(x>>(n<<3))&0xFF

5	float_abs(unsigned uf)

이 문제에서 생각해야 할 것을 두가지로 정의하고 시작했다.
1. 절댓값을 만든다.
2. NaN값을 처리한다.
1을 먼저 해보면 수의 절대값을 확인하려면, 부호만 맞추면 된다 생각하여
최상위 비트만 0이고 나머지가 다 1인 수와 &연산하면 되리라 생각했다.
01111... = 0x7FFFFFFF로 표현할 수 있고, 연산해주면
-> 0x7FFFFFFF&uf 이다.
2에서 아직 NaN에 대한 개념이 많이 부족해 많이 헤맸다.
우선 NaN은 exp가 다 1인 수인데, 이걸 수로 표현하면 0x7F800001보다 크거나 같은 값이 해당된다.
이 조건을 만족할 시 uf값을 반환해주고 아니면 0x7FFFFFFF&uf연산한 절댓값을 반환해주면 된다.

6	addOK(int x, int y)

접근은 이제 부호가 같을 때, x+y와 x의 최상위 비트가 다르면, 불가능 하다고 생각했다.
그래서 1. 다를 땐 무조건 1 반환, 2. 부호 동일, 최상위 비트가 같으면 1반환으로 설계했다.
이 1,2를 |연산하면 정답일 거라 생각했다.
1. x, y의 최상위 비트를 추출하여, ^연산했다. diff라고 변수선언을 하고 다를 때 1이 반환.
2. ~diff이고, x의 최상위 비트와 합의 최상의 비트가 다를 때를 확인.
->(((x+y)>>31)^(x>>31))&1 과 같이 하면 다를 때 1 같을 때 0을 추출할 수 있다.
같을 때 1이 필요하므로 2번 식을 설계하면
~diff&~((((x+y)>>31)^(x>>31))&1) = !(diff|(((x+y)>>31)^(x>>31))&1)라고 할 수 있다.
1과 2를 |연산하면 답이다.

diff|(!(diff|(((x+y)>>31)^(x>>31))&1))

7	replaceByte(int x, int n, int c)

4번 문제와 유사하다. 4번에서 배웠던 마스크의 원리를 잘 이용해 보았다.
이 문제는 nbyte자리만 0인 x와 nbyte자리만 c값이고 나머진 0 인 수를 더하면 된다.
위의 x는 n자리만 0이고 나머진 1인 마스크와 연산을 해주면 된다.
0xFF<<(n<<3)을 해주면 그 byte만 1인 값을 만들 수 있는데, 이 값을 ~연산 해주면 된다.
->~(0xFF<<(n<<3))&x ... 1
다음 c만 그 자리로 만드는 것은 문제 4를 반대로 하는 것과 같다.
->c<<(n<<3) ... 2
위에서 도출된 1과 2를 더하면 답이다.

(~(0xFF<<(n<<3))&x) + (c<<(n<<3))

8	isGreater(int x, int y)

x>y로 갈 경우 조건을 확인해야하는 것이 많다고 느껴, !(x<=y)로 진행하였다.
y-x>=0을 만들 것이다. 부호가 다를 때 산술적으로는 가능하지만,
비트가 삭제되어 값이 이상해질 때를 대비해 부호가 다를 때는 무조건 참이 되도록
설정하고, 부호가 같을 때는 연산 시 최상위 비트가 0일 경우 참이 되도록 설정했다.

6번문제에서 사용한 diff함수를 그대로 가져와 사용하였다. ※6번문제 참고
첫 번째 조건은 diff함수와 부호가 다를 경우 x가 음수여야 하므로 위 조건도 &연산
해주었다.
diff&(x>>31) ... 1
두 번째 조건은 부호가 같을 때 y-x의 최상위 비트가 0 일 때 참(1)이 되도록 설계하였다.
~diff&~((y+(~x+1))>>31)=!(diff|((y+(~x+1))>>31)) ... 2
위에서 계산한 1과 2의 식이 하나라도 만족하면 참이므로 |연산 해준다.
(diff&(x>>31))|(!(diff|((y+(~x+1))>>31)))
여기서 끝내면 안된다. 처음에 !연산을 해서 만들었으므로 다시 !을 붙여줘야 우리가 필요한
식이 될 것이다.

!((diff&(x>>31))|(!(diff|((y+(~x+1))>>31))))