2021 시스템 프로그래밍

- DataLab -

제출일자	2021.10.4
분 반	01
이 름	조해창
학 번	201702897

실습 1

1	[isTmax]

핵심 : !0 은 1 !(0이 아닌 숫자) 은 0이다. x 가 -1일 때 예외처리

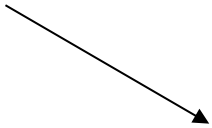
!연산은 0을 1로 만들고 나머지 수를 0으로 만들기 때문에 maximum 값인 0x7FFFFFFF만 0으로 만들어서 !연산을 사용하기로한다.
x 가 maximum일 때 x + (x + 1) 은 모든 비트가 1 인 -1이다. 하지만 x 가 -1 일 때도 결과값은 -1을 가지기 때문에 예외처리를 해야한다.
x 가 -1 일 때만 !(~x)이 1이고 나머지 숫자들은 0이기 때문에 x + (x + 1) 값에 !(~x)을 더해준다. 따라서 x 가 maximum 일 때만 x + (x + 1) + !(~x) 은 -1 값을 가진다.
결국 x 가 maximum 일 때만 ~(x + (x + 1) + !(~x))은 0을 가지기 때문에 이 수식에 !연산을 붙인 !(~(x + (x + 1) + !(~x))) 이 답이 된다.

2	[bitXor]

핵심 : 드모르간의 법칙

X ^ Y 은 X' & Y | X & Y' 로 표현할 수 있다. 하지만 | 연산은 사용할 수 없으므로 드모르간의 법칙을 이용하여 ((X' & Y)' & (X & Y')')'로 나타낸다.

X' & Y		
	X'	X
Y'	0	0
Y	1	0



X' & Y | X & Y' == ((X' & Y)' & (X & Y')')'		
	X'	X
Y'	0	1
Y	0	0

X & Y'		
	X'	X
Y'	0	1
Y	0	0

3	[copyLSB]

핵심 : arithmetic shift은 msb에 부호 비트를 유지한다.

Arithmetic shift 는 msb에 부호 비트를 유지한다는 것을 이용하기 위해서 shift left 를 31 bits 하여 lsb의 값을 부호 비트에 위치 시킨다. 부호 비트로 이동된 lsb의 비트를 모든 비트에 복사하기 위해서 right shift 31 bits를 한다.

4	[float_abs]

핵심 : single precision에서는 msb는 부호비트 exp 는 8비트 frac은 23비트이다. NaN 는 exp의 모든 비트가 1이고 frac 이 0이 아닐 때이다.

입력값 uf 에서 exp을 얻기 위해 0x7F800000 와 & 연산을 하고, frac을 얻기 위해 0x007FFFFF과 & 연산을 한다.
NaN 일 때는 exp의 모든 비트가 1이고 frac 이 0 이 아닐 때 이므로 위에서 얻은 uf 의 exp값과 frac값을 비교연산하여 분기 시킨다. NaN 일 때는 입력값을 그대로 반환하고 아니면 부호 비트를 0으로 만들어 주기 위해 0x7FFFFFFF과 & 연산을 한다.

5	[bitMask]

핵심 : ~0를 left shift연산하여 특정 범위는 1 bit로 만든다.

~0를 highbit+1 만큼 right shift 하여 msb부터 highbit바로 전 까지 비트를 1로 만들고 이것을 leftSide 라고 한다.
~0을 lowbit 만큼 right shift 하고 ~를 사용하여 lsb부터 lowbit바로 전 까지 비트를 1로 만들고 이것을 rightSide 라고 한다.
leftSide | rightSide 는 highbit 부터 lowbit 까지 비트가 0 이고 나머지 비트는 1이다.
따라서 !(leftSide | rightSide)가 답이 된다. 만약 highbit 가 lowbit보다 작으면 leftSide | rightSide는 모든 비트가 1이 되므로 !(leftSide | rightSide) 는 0이 된다.

6	[rotateRight]

핵심 : left shift 연산은 lsb에 0을 채우고 arithmetic right shift 는 msb에 부호비트를 채운다.

오른쪽으로 n bits 만큼 회전 하여 왼쪽부터 들어가는 비트를 구하기 위해 x 를 32 + (~n + 1)) 만큼 왼쪽으로 shift 연산하여 addLeftSide 라는 변수에 저장한다.
x를 오른쪽으로 n bits 만큼 shift 값을 shiftedRight라는 변수에 저장한다. 이때 shiftedRight에는 arithmetic shift로 인해 msb부터 n 개의 부호 비트가 추가되어있기 때문에 이를 없애기 위해서 추가된 n개의 부호비트를 제외한 모든 수가 1 bit인 signMask생성하여 shiftedRight 와 & 연산을 한다. signMask 는 ~(~0 << 32 + (~n + 1)) 이다.
마지막으로 addLeftSide 와 부호비트가 제거된 shiftedRight 를 | 연산한다.

7	[isAsciiDigit]

핵심 : arithmetic shift right를 (자료형의 크기 – 1) 만큼 해서 결과값이 0 이면 양수 -1 이면 음수이다.

0x30 <= x <= 0x39는 0 <= x – 0x30 이고 0 <= 0x39 – x 인 것과 동일하다. 각 조건을 a, b 라고 한다.
arithmetic shift right를 lsb 까지 하여 결과값이 0이면 양수 -1 이면 음수인 것을 이용하여 a 와 b의 결과값의 부호를 판단한다. 따라서 !(a >> 31) & !(b >> 31) 은 x 가 위의 조건에 만족할 때만 1을 반환한다.

8	[bang]

핵심 : 0은 자기자신과 2의보수의 부호비트 둘다 0인 유일한 숫자 이다.

0은 자기자신과 2의보수의 부호비트 둘다 0이기 때문에 x | (~x + 1) 의 부호 비트는 x가 0일 때를 제외하고 모두 1이다. 이 결과값을 (자료형의 크기 - 1)만큼 arimetic shift right 하면 x가 0일 때는 0 이기 때문에 결과값에 1을 더하면 1이 되고, 그 외의 숫자는 -1을 이기때문에 1을 더하면 0이다.