# HW 1 REPORT


| | | |
|---|---|---|
| 수강 과목 | : | 시스템소프트웨어(059) |
| 학 과 | : | 정보컴퓨터공학부 |
| 이 름 | : | 김태익 |
| 학 번 | : | 202155544 |
| 제출 일자 | : | 2025. 04. 04. |

# Integer Problems

**absVal**
x의 절댓값을 출력한다.

x가 양수라면 그대로, x가 음수라면 2의 보수를 취하면 구할 수 있다.

**addOK**
overflow 없이 x+y 연산이 가능한지 판단한다.

overflow는 x 와 y의 부호가 같은데, x+y 결과의 부호가 x 혹은 y와 다르면 나타난 것으로 확인할 수 있다.
따라서 x, y, x+y의 sign bit를 구하고, xor 연산을 통해 부호를 비교하면 된다.

**allEvenBits**
짝수 번째 bit들이 모두 1인지 확인한다.

0b101010.....1010형태의 mask를 만든뒤 x & mask == x 인지 확인하면 된다.

**bang**
!x, 즉 x == 0이면 1을 출력하고 그 외에는 0을 출력한다.

x | -x 를 하면 x가 0일때는 0, 그 외에는 MSB가 1이 나온다.
따라서 오른쪽 shift를 통해 0x00000000 혹은 0xFFFFFFFF로 만든 후 1을 더하면 원하는 동작을 수행할 수 있다.

**bitAnd**
& 연산을 ~와 |으로 연산한다.

드모르간의 법칙을 적용해서 xy를 !(!x + !y)로 구하면 된다.

**divpwr2**
x를 2^n으로 나누는 값을 구하는데, Round to Zero 효과를 적용해야한다.

단순히 2^n으로 나누면 x가 양수일때는 문제가 없지만, 음수일때는 소수점 버림이 아님 내려짐이 발생한다.
따라서 sign bit를 판별해 x가 음수일때만 2^k - 1의 bias를 더한 뒤 나누면 된다.

**getByte**
x에서 n번째 바이트를 추출한다. n은 0(LSB)에서 3(MSB)이다.

n 만큼 오른쪽 shift를 수행하고, 0xFF와 AND 연산을 통해 쉽게 구할 수 있다.

**howManyBits**
2의 보수 시스템에서 x를 표현하기 위해 최소 몇 비트가 필요한지 구하는 문제다.

양수의 경우 가장 왼쪽 1의 위치, 음수의 경우 가장 왼쪽 0의 위치로 계산할 수 있는데,
두 가지 경우를 고려하는 것도 번거롭고 보수를 취해도 필요한 비트 수는 같기 때문에
x가 음수일 경우 보수를 취해 양수의 경우만 취급한다.
그 다음 가장 가장 왼쪽 1의 위치를 구하기 위해 이진 탐색을 한다.
현재 남아있는 비트를 절반씩 쪼개면서 상위 n비트에 1이 존재하면 n을 저장 후 ,
다시 하위 n비트를 탐색하기 위해 n 만큼 오른쪽 shift를 한다.
이 과정을 한 비트만 남을때 까지 반복하고,
저장한 n의 총합에 sign bit를 위한 1비트를 더하면 몇 비트가 필요한지 알 수 있다.

**isLessorEqual**
x <= y일때 1을 반환, 그렇지 않으면 0을 반환하는 문제다.

처음엔 x - y <= 0을 확인해 sign bit를 출력하려 했으나,
x = y, 즉 x - y = 0이면 sign bit가 0이 되기 때문에 1이 아닌 0을 출력하는 문제가 있었다.
y - x >= 0 을 확인해 sign bit의 NOT만 구하면 x == y인 경우를 따로 처리하지 않아도 된다.

또한, overflow의 경우도 처리해야한다.
overflow가 발생하는 경우는 x와 y의 부호가 다를때이므로, 그럴땐 x의 부호를 출력하도록 처리했다.
x < 0, y > 0 -> 무조건 x <= y가 참이므로 x의 sign bit인 1 출력
x > 0, y < 0 -> 무조건 x <= y가 거짓이므로 x의 sign bit인 0 출력

**leastBitPos**
Least Significant 1의 위치. 즉 가장 오른쪽에 있는 1의 위치를 구하는 문제다.

단순히 x & -x 를 구하면 된다. 2의 보수에서 -x를 연산하는 과정을 생각하면
보수를 취한 뒤 1을 더하므로 자연스레 가장 오른쪽에 있는 1만 남게 된다.

**logicalShift**
오른쪽 shift를 했을때 sign bit가 채워지는 Arithmetic Shift가 아닌 0이 채워지는 Logical Shift를 구현하는 문제다.

이 머신에선 자동으로 Arithmetic Shift를 하므로,
먼저 shift를 한 뒤에 shift를 한 만큼의 상위 n비트를 0으로 바꾸면 된다.
그러기 위해 상위 n 비트가 0이고 나머지가 1인 mask를 구한 후 AND 연산을 했다.

**replaceByte**
x의 n번째 바이트를 c로 대체한다.

x의 n번째 바이트를 0으로 만든 후, c를 더하는 방식으로 구현했다.
그러기 위해 n번째 바이트만 0이고 나머지는 1인 mask를 구하고,
그 위치에 더하기 위해 c 를 n * 8만큼 왼쪽 shift 한 후 더했다.

**upperBits**
상위 n비트가 1인 결과를 출력하는 문제이다.
0b10000...을 n-1만큼 right shift하는 것으로 구할 수 있으나, n이 0일때를 따로 처리해야한다.
!!n은 n이0일때만 0, 그 외에는 1이므로 !!n << 31을 n-1만큼 오른쪽 shift하면 된다.

# Floating Point Problems

**float_f2i**
floating point 형식의 입력을 int형으로 변환해서 출력하는 문제이다.

floating point의 형식 그대로 Sign비트, Exponent, Fraction으로 나눠서 저장한 후 정수로 변환하면 된다.
다만 Denormalized 값이랑 Special 값에 대한 처리, int의 범위를 벗어난 수에 대한 처리가 필요하다.
Exponent가 0xFF이면 무한대 혹은 NaN이므로 문제에서 요구한대로 0x80000000u를 반환,
Exponent가 0이면 0이므로 0을 반환한다.

또한 Exponent에서 bias인 127을 뺀 값인 E의 값으로 int의 범위 내에 있는지 확인한다.
32비트 int는 -2^31 ~ 2^31 - 1까지 표현 가능하므로 E가 31보다 크면 범위를 벗어나 0x80000000u를 반환한다.
또한 E가 0보다 작으면 결과가 0보다 작은 소수가 되기 때문에 0을 반환한다.

예외값들에 대한 확인을 완료하고나서 fraction의 leading 1을 붙이고
E의 값에 따라 E - 23만큼 왼쪽 shift 혹은 23 - E만큼 오른쪽 shift 하고,
마지막으로 sign bit가 1이면 -를 붙여서 음수로 만들면 정수로 변환이 완료된다.

**float_twice**
floating point 수에 2를 곱한 값을 floating point 형식으로 출력하는 문제이다.

2를 곱하는 방법은 두 가지가 있는데,
1) Exponent 부분에 1을 더한다
2) Fraction 부분을 1만큼 왼쪽 shift 한다

Normalized Number일 경우에 1번으로 해결되지만,
Denormalized Number일 경우 문제가 생기므로 2번으로 처리한다.

Exponent의 값이 0이면 Denormalized Number일 경우 Fraction << 1한 값을 반환하고,
Exponent가 11111111일 경우 Nan 혹은 무한이므로 인자 그대로 반환한다.

그 외에는 Normalized Number이므로 Exponent 부분에 1을 더한 값을 반환한다.