컴퓨터 SW 시스템 개론
LAB 1

20190650 김관호

1. 배운 것

이번 Data Lab에서는 integer를 어떻게 bit로 표현하는지를 배웠다. 또한, &, ^, |, ~, >>, <<와 같은 비트 연산자를 배웠다. 비트 연산자의 경우 연산자 우선순위에서 산순 연산자보다 앞서거나 뒤처지는 것들이 섞여 있어 웬만하면 괄호로 계산하는 것이 안전하다고 생각했다.

2. 구현 과정

bitNor(int x, int y)

~, &를 이용해 NOR 연산자를 구현하면 된다. ~는 모든 비트를 반전시키고, NOR 연산자는 두 bit가 0일 때만 1이 된다. 이 점을 생각하면 ~를 &로 묶을 수 있다. **~x & ~y**.

isZero(int x)

x가 0이면 1을, 그 외라면 0을 반환한다. Logical Not 연산자 !를 이용하면 된다.

addOK(int x, int y)

x, y를 더했을 때 overflow가 난다면 0을, 나지 않는다면 1을 반환한다. x, y는 int로 들어오므로 두 수의 부호가 다르면 overflow가 발생하지 않는다. overflow는 두 수의 부호가 같으면서, 더한 결과값과 부호가 다를 때 발생한다. x, y, x+y 세 수의 부호를 알면 overflow가 일어났는지 확인할 수 있게 된다.
x+y가 x와 부호가 다르고, y와도 부호가 다르면 overflow가 일어난 것이다. 이 때 0을 반환시키면 된다. 이를 연산자로 표현하면 !(((sign_x ^ sign_sum) & (sign_y ^ sign_sum)) & 1) 이 된다.

absVal(int x)

x의 절댓값을 반환한다. x가 음이 아닌 수라면 x를 x가 음수라면 ~x+1을 반환하면 된다. 조건문을 대체하기 위해 x의 sign bit을 이용한다. x의 sign bit에 따라 x >> 31의 결과는 0xFF..FF, 0x00..00이 된다. 이 결과와 x를 XOR 연산자로 묶으면 x가 음수일 때는 반전된 비트로, 음이 아닐 때는 그대로 표현된다. 음수라면 반전시킨 비트에 1을 더해줘야 하므로 sign bit & 1을 결과 값에 더해준다.

logicalShift(int x, int n)

int는 right shift가 arithmetic인데 이를 logical로 바꿔주면 된다. n만큼 shift했을 때 부호에 따라 most significant bit쪽에서 1이 n개 생길 수 있는데, 이 비트들을 0으로 만들어주면 logical shift 효과를 낼 수 있다.
이를 위해 처음 n개 비트는 0으로, 나머지 비트는 1로 된 logical_shift_helper를 만들었다. arithmetic shift한 값과 이 값을 & 연산하면 logical shift효과를 낼 수 있다.
logical_shift_helper는 shift 연산자를 이용했는데, 1 << 31 >> n << 1이다. 먼저, 1을 가장 끝인 sign bit 위치로 보낸다. 그 다음 right shift는 arithmetic shift임을 이용해 n만큼 right shift하면서 1을 갖고온다. 이 때 1의 개수는 n+1이므로 왼쪽으로 한 칸 밀어 1의 개수를 맞춰준다.