CSED211: Data Lab1 Report

20230784 / 유재원

1. Overview

본 과제는 비트 연산자 등 몇 가지 제한된 C언어의 기능을 이용해 정수형 데이터의 저장 및 처리를 진행하는 과정에서 기본적인 데이터 처리에 대한 원리를 이해하는 데에 그 의의가 있다.

2.1. bitNor

~과 &를 이용해 ~(x|y)를 계산하는 문제이다. Bool Algebra에서의 드모르간 법칙을 이용하면 쉽게 계산할 수 있다.

2.2. isZero

input이 0인지 결정하는 함수이다. !(Logical not) 연산자를 사용할 수 있기에 쉽게 구현 가능하다.

2.3 addOK

int 덧셈 시에 오버플로가 일어나는지 결정하는 함수이다. 최하위 비트의 연산 결과를 저장 후에 x와 y를 오른쪽으로 shift 후 저장해둔 연산 결과를 더한 뒤에 부호를 판단하는 것을 처음에 고려했으나 별로 아름다운 해결책은 아닌 것으로 보여 int 덧셈 시에 오버플로가 일어나는 상황은 다음 두 상황만이 존재한다는 사실을 상기하여 그것을 응용하여 작성하였다.

1. 양수 + 양수 = 음수
2. 음수 + 음수 = 양수

x + y의 결과를 미리 계산한 결과를 x, y에 각각 xor 한다. 그 결과를 and 할 경우 x+y 의 결과가 x와도 다르고 y와도 다른 상황만을 골라낼 수 있다. 최상위 비트만을 확인할 것이기에 오른쪽으로 31비트만큼 shift 한다. 결과가 1일 경우 x+y 의 결과가 x와도 다르고 y와도 다른 것이므로 오버플로가 발생했으니 0을 반환하도록, Logical not을 취하여 반환한다.

단 위와 같은 구현은 일단 계산을 한 후 실제로 오버플로가 일어났는지 확인하는데, 계산 전에 오버플로가 일어날 것인지를 판단하는 방법은 없을지 궁금하다. (위에서 처음 생각한 방법 또한 결국 계산을 통해 확인하는 것이라 볼 수 있다.) 실제로 계산해 보기 전까지는 오버플로가 일어날지 판단할 수 없을 것 같기도 하다.

2.4 absVal

주어진 input의 절대값을 반환하면 된다. 처음 코드 작성은 그저 비트를 반전(~)한 후 1을 더해주는 방식으로 구현하였으나 0x80000000 (Tmin)의 경우 그것의 절대값은 똑같이 Tmin 이지만 위와 같이 구현하면 0이 나오기에 다른 방법이 필요했다. (또한 이미 양수일 경우 음수가 나오는 문제도 있었다.)

이를 해결하기 위해 현재 부호에 대한 정보를 잘 저장해 보자. mask에 x >> 31의 결과를 담는다. 이때 x가 양수라면 mask=0, x가 음수라면 mask=-1(0xFFFFFFFF)일 것이다.

x가 양수일 경우 x+mask ^ mask의 결과는 자명이 x이다.
x가 음수일 경우 x-1 ^ -1의 결과는, x에다가 1을 뺀 후(-1을 더한 후) 모든 비트를 반전한다. 이 결과가 절댓값으로 잘 mapping 됨은 직접 3bit에 대하여 table을 작성해 확인할 수 있다.

bit	int	abs	abs bit
0b000	0	0	0b000
0b001	1	1	0b001
0b010	2	2	0b010
0b011	3	3	0b011
0b100	-4	-4	0b100
0b101	-3	3	0b011
0b110	-2	2	0b010
0b111	-1	1	0b001

2.5 logicalShift

int 자료형에서 logical shift를 수행하는 문제이다. 기본적으로 C언어에서 int 자료형에 대해 bit shift 연산은 산술 shift를 수행하므로 고민을 조금 하였다. 오른쪽으로 3칸 shift 한다고 치면 지금 상태가 무엇이었든 최상위 3비트를 0으로 만들면 된다. 따라서 이 경우 최상위 3비트가 0, 나머지 비트가 1인 mask를 만들어 원본을 산술 shift 한 결과와 and 연산을 취하면 될 것 같다.
먼저 0x80000000을 만들기 위해 1 << 31을 이용했다. 이후 그것을 오른쪽으로 n만큼 shift 하면 최상위 n+1비트가 1, 나머지 비트가 0이다. 마지막으로 왼쪽으로 한 번 shift 후 ~을 하면 원하는 mask를 얻을 수 있다. 이때 처음부터 오른쪽으로 n-1 번 shift 하지 않는 이유는 n=0일 경우 unintended behavior가 발생하기 때문에 이를 방지하기 위함이다. (처음 코드 작성 시에는 n-1 번 shift 하는 방향으로 구현하였으나 n=0인 case에서 wrong answer임을 확인할 수 있어 수정하였다.)

3. Results

이 실습으로 비트 연산자의 이용 및 컴퓨터상에서의 정수 표현과 그 응용을 연습할 수 있었다.

4. References

https://dojang.io/mod/page/view.php?id=188 - 코드 작성 시 불필요한 괄호 사용을 줄이기를 원해 연산자 우선순위를 참고해 가능한 괄호가 필요한 부분에만 괄호를 이용하였다. 단 특정 상황에서 의도한 대로 동작은 되지만 컴파일러가 괄호로 explicit 하게 표현하라 경고하는 부분에는 괄호를 넣었다.