[Lab1] DataLab report

Homework: Problem 1

풀이: 드모르간의 법칙을 이용해주었습니다.

a & b는 ~~(a & b)와 같습니다. 여기서, ~(~(a&b))로 생각해볼 수 있고 ~(a&b)는 드모르간의 법칙에 의해 ~a | ~b가 됩니다.

따라서 a & b는 ~(~a | ~b)와 같으므로 이 값을 리턴해주면 &를 사용하지 않고 bitwise and연산을 구현할 수 있습니다.

Homework: Problem 2

풀이: Q&A 게시판에서 x, y를 unsigned로 가정하면 된다고 하셨습니다. 그래서 부호를 고려해줄 필요가 없어졌기 때문에 간단하게 처리할 수 있었습니다. max ops의 제한이 없어 간단하게 LSB부터 MSB까지 캐리를 구하는 것으로 문제를 해결했습니다.

먼저 LSB의 대응되는 비트끼리 더했을때 캐리를 계산합니다. 모두 1인 경우만 캐리가 1이되고 나머지는 캐리가 0이 됨을 직관적으로 쉽게 파악할 수 있습니다.

그리고 이 캐리와 x의 1번 비트, y의 1번 비트를 더했을때 캐리를 계산합니다. 이때는 1이 2개 이상 있으면 캐리가 1이되고 나머지 경우는 캐리가 0이 됨을 쉽게 알 수 있습니다.

따라서 다음의 경우를 or로 묶어서 답을 얻을 수 있습니다.

1. 모두 1인 경우

2. 캐리는 0 나머지는 1

3. x의 1번 비트는 0, 나머지는 1

4. y의 1번 비트는 0, 나머지는 1

그리고 위 과정을 MSB인 31번 비트까지 반복해줍니다.

이 결과 캐리의 최종 값이 1이면 값이 넘어가 오버플로우가 발생한 것이므로 0을, 캐리의 최종 값이 0이면 오버플로우가 발생하지 않은 것이므로 1을 리턴하는 것으로 문제를 해결할 수 있습니다.

Homework: Problem 3

풀이: signed int형에서 MSB가 1이면 음수, MSB가 0이면 양수 또는 0입니다. 그리고 이 함수는 음수면 1을, 음수가 아니면 0을 리턴해야하기 때문에 MSB를 리턴해주면 문제가 해결됩니다.

Homework: Problem 4

풀이: 먼저 x의 MSB가 0인 경우는 arithmetic shift가 수행되어도 logical shift와 결과가 같기 때문에 문제가 발생하지 않습니다. 즉, 해결해야할 것은 MSB가 1일때 추가되는 1들을 어떻게 0으로 바꾸느냐하는 문제입니다.

생각해보면 결국 arithmetic shift의 결과에서 MSB를 포함하여 y개의 비트를 0으로 바꾸어주면 문제가 해결됩니다. 그렇다면 MSB를 포함하여 y개의 비트가 1이고 나머지가 0인 어떤 수 temp를 생각해봅시다. 이 경우 ~temp와 arithmetic shift의 결과인 x>>y를 & 연산으로 묶어주면 MSB를 포함한 y개의 비트는 0이되고, 나머지 부분은 x>>y와 같습니다.

따라서 이런 수 temp를 만들면 문제가 해결됩니다. 그래서 먼저 MSB가 1이고 나머지는 0인 수 1<<31을 정의합니다. 그리고 이 수에 >>y를 해주면 arithmetic shift이므로 MSB를 포함하여 y+1개의 비트가 1이되고, 나머지 비트는 0이 됩니다. 그런데 필요한것은 MSB를 포함하여 y개의 비트가 1이 되는 것이므로 <<1을 해주면 temp가 완성되게 됩니다.

그래서 위에서 언급했듯이 ~temp와 x>>y를 & 연산으로 묶어주면 답을 구할 수 있습니다.

Homework: Problem 5

풀이: int는 총 32개의 비트를 가집니다. 그런데 max ops 제한이 40이므로 단순히 모든 비트를 살펴보기에는 제한을 초과해버립니다. 4개의 비트를 동시에 체크하는 것으로 이 문제를 해결했습니다.

먼저 32개의 비트를 4개씩 묶어 줍니다. 편의상 이 묶음을 '칸'이라고 하겠습니다. 그리고 칸의 4개의 비트를 LSB에 가까운 비트부터 0번비트, 1번비트, 2번비트, 3번비트라고 하겠습니다.

그리고 0x11111111이라는 checker를 만들어줍니다. 그러면 checker에서 1인 비트는 각 칸의 0번 비트에만 존재하게 됩니다. 따라서, 이 checker와 x를 & 연산으로 묶어주면 칸의 0번 비트가 1이면 1이, 0이면 0이 칸의 0번 비트에 저장되게 됩니다.

그리고 x>>1과 checker를 & 연산으로 묶어주면 칸의 1번 비트가 1인지 아닌지를 구할 수 있습니다. 같은 방법으로 x>>2, x>>3과 checker를 & 연산으로 묶어줄 수 있습니다.

이렇게 구한 4가지 값을 모두 더한 것을 sum이라고 하면 sum의 8개의 칸에 담겨 있는 값은 대응되는 x의 칸에 포함된 1의 개수가 됩니다.

구해야하는 것은 x 전체에 포함된 1의 개수이기 때문에, 결국 앞서 구한 sum의 8개의 칸에 담긴 값을 다 더해주면 답을 구할 수 있습니다.

sum>>0, sum>>4, sum>>8, … , sum>>28과 0x0000000F를 & 연산으로 묶는 것으로 각 칸의 값을 얻어낼 수 있습니다. 따라서 얻어낸 값을 다 더해주면 max ops제한을 만족하면서 문제를 해결할 수 있습니다.