2020년 2학기

Assignment #1

과목 : 컴퓨터SW시스템개론

담당 교수 : 김종

학과 : 컴퓨터공학과

학번 : 20190439

이름 : 오승훈

povis id : sho0927

|  |
| --- |
| < 명예서약(Honor code) >  “나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.” |

**Problem : Bitwise Manipulations and Floating-Point Operations**

1. 문제의 개요

본 프로그램은 다음의 내용을 포함한다.

- operation 수가 한정된 상태에서 bitwise operator를 이용하여 bitwise function을 만드는 것을 기본적으로 다룬다. Bitwise operation에 대한 내용을 다룰 때는 제어문의 사용을 금지하고 있다.

- 우리가 공부를 한 float형에서 자료형이 저장되어 있는 방법에 대해서 이해를 하고, floating point에 대한 이해를 돕는 것을 기본으로 다루고 있다. 또한 여기서 구현은 제어문 사용이 가능하며 single precision으로 float형이 저장되어 있다고 가정을 한다. 32개의 bit 중 첫 bit는 sign bit로 사용이 되고, 2 ~ 9 번째 bit는 exponent를 표현하기 위해서 10 ~ 32번째 bit의 경우 fraction part를 표현하기 위해서 사용되는 것을 이용하게 된다.

- 32-bit representations of integer를 2`s complement의 형태로 활용하며, right shift를 수행할 경우 arithmetical shift가 발생한다는 것을 전제로 하고 있다.

a. Bit Manipulation

* Problem 1: x | y 라는 bitor을 ~ 과 & bitwise operation을 이용하여 구현을 한다.
* Problem 2: x + y 를 했을 때 overflow가 발생한 case에 대해서는 1을 출력을 하고 overflow가 발생하지 않은 경우 0을 반환하는 함수 구현을 한다.
* Problem 3: x가 주어졌을 때 오른쪽으로 n번 logical Shift가 일어나는 함수를 구현한다.
* Problem 4: 현재 주어진 x에서 32 bit 중 1인 bit의 수는 몇 개인지 counting 하는 것을 목표로 한다.
* Problem 5: 2’s complement에서 –x 구현을 하는 것을 목표로 한다.

b. Floating Point Operation

* Problem 1: float에서 –f 를 출력하는 것을 목표로 한다.
* Problem 2: int 로 주어진 값을 float 형으로 반환하는 것을 목표로 한다.
* Problem 3: 주어진 float형을 2배로 하는 것을 목표로 한다.

2. 알고리즘

본 프로그램 작성을 위한 알고리즘을 Pseudo 코드 형태로 나타내면 다음과 같다.

|  |
| --- |
| Pseudo-algorithm for E-card game |
| *// 프로그램에 필요한 변수들은 미리 선언해놓은 것으로 가정한다*  1 bitOr  1 ~(~x & ~y);  2 addOK  Check = x + y 의 sign bit  sign\_x = x의 sign bit  sign\_y = y의 sign bit  !(!((~sign\_y & ~check) | (sign\_x & check) | (~sign\_x & sign\_y)))  3 negate  1 ~x+1  4 logicalShift  right\_x = x를 n 번 오른쪽으로 arithmetic shift한 값  memory = ((0x01 << 31) >> n ) << 1 arithmetic shift가 되어 sign bit가 복사된 횟수  ~memory & right\_x  5 bitCount  factor = 0x11 | ( 0x11 << 8) 0x1111을 만드는 과정  factor = factor | (factor << 16) 0x11111111 을 만듦  div\_4 = (x & factor) + ((x>>1) & factor) + ((x>>2) & factor) + ((x>>3) & factor) 4개의 합을 한 곳으로 합침 총 32 -> 8개로 줄어듦 (bit가 있는 곳이)  div\_4 = div\_4 + (div\_4 >> 16) 8 -> 4개로 줄어듦  (div\_4 & 0xF) + ((div\_4 >> 4) & 0xF) + ((div\_4 >> 8) & 0xF) + ((div\_4 >> 12) & 0xF)  Bit의 합들을 저장해둔 4개 값을 다 더해주기 위한 과정 |
|  |

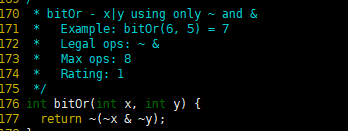
3. 프로그램 구조 및 설명

a. bitwise operation

1) bitOr

&, ~을 이용하여 |을 나타내기 위해서는 드모르간 법칙을 사용하면 된다.

A | B = ((A | B)’)’ = ~(~ A & ~ B) 이 됨을 알 수 있다.



따라서 코드는 위와 같다.

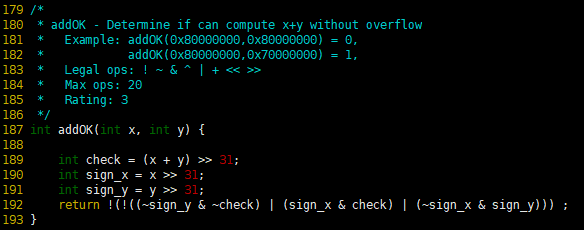
2) addOK

overflow가 일어나는 조건을 생각을 해보면 sign\_x 와 sign\_y의 bit가 다른 경우에는 overflow가 일어나지 않음을 알 수 있다. 그러므로 sign\_x와 sign\_y 값이 같은 경우에는 sign\_x / sign\_y 와 x와 y가 더한 값의 sign bit가 다른 경우 overflow가 일어난 경우라고 생각하면 된다. 그러므로

sign\_x , sign\_y 그리고 x+y의 sign bit를 모아둔 후 sign\_x sign\_y check\_bit를 XYZ라고 한다면

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X|YZ | 00 | 01 | 11 | 10 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

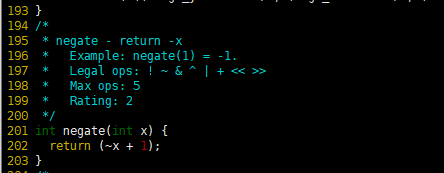
임을 볼 수 있으므로 표에 1이 적혀있는 상황일 때 1을 return 하면 되므로



따라서 코드는 위와 같다.

3) negate

2`s complement에서, -x = ~x + 1이다.

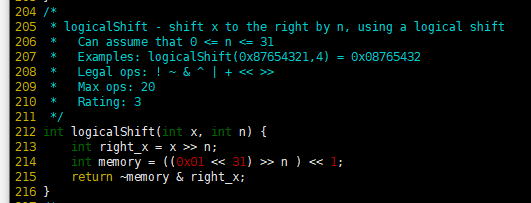


따라서 코드는 위와 같다.

4) logicalShift

Right\_Shift에서 arithmetic shift와 logical shift의 차이점은 arithmetic shift는 sign bit가 복사가 되어 왼쪽 bit에 나타난다는 것이고 logical shift에서는 right shift를 하면 shift를 하여 비어진 bit가 0으로 채워진다는 것이다.

그렇다면 right shift를 한 후 shift한 만큼 0으로 채워주면 된다는 것을 알 수 있다.

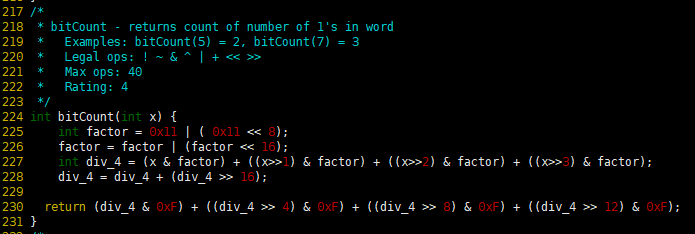


따라서 코드는 위와 같다.

5) bitCount

Bitcount를 할 때에는 32bit에 대해서 한 번씩 오른쪽으로 shift해가면서 1이면 값에 1을 더하게 진행하면 가장 쉬운 코딩 방법이 된다. 하지만 이렇게 되면 operation 수를 넘기게 되므로 이와 operation을 줄이기 위한 과정이 필요하다.

그렇기 때문에 32 bit에 존재하는 값들을 정해진 8bit 안에 들어가도록 줄이고 그 후 정해진 4bit에 값이 더해지게 한 후 첫 번째 bit에 모든 값들의 합이 존재하도록 해주면 된다.

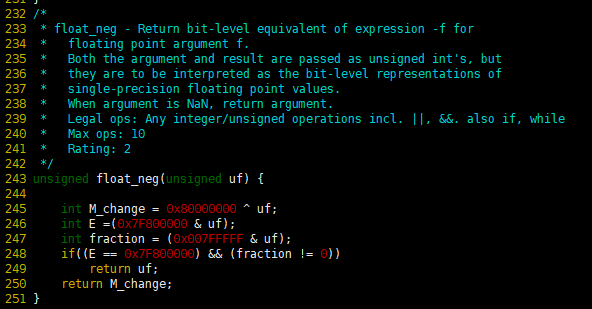


따라서 코드는 위와 같다.

b. Floating Point Operation

1) float\_neg

Float neg의 경우 sign bit 값만 바꿔주면 되기 때문에 sign\_bit와 0x80000000 의 XOR operation을 진행해주면 bit 값이 바뀌게 된다. 그리고 NaN 값인 경우 uf 를 그대로 반환하라고 했기 때문에 EXP 값과 fraction part를 구해서 NaN 이면 uf 를 반환하고 아닌 경우 sign\_bit를 바꾼 값을 반환해주게 되면 된다.

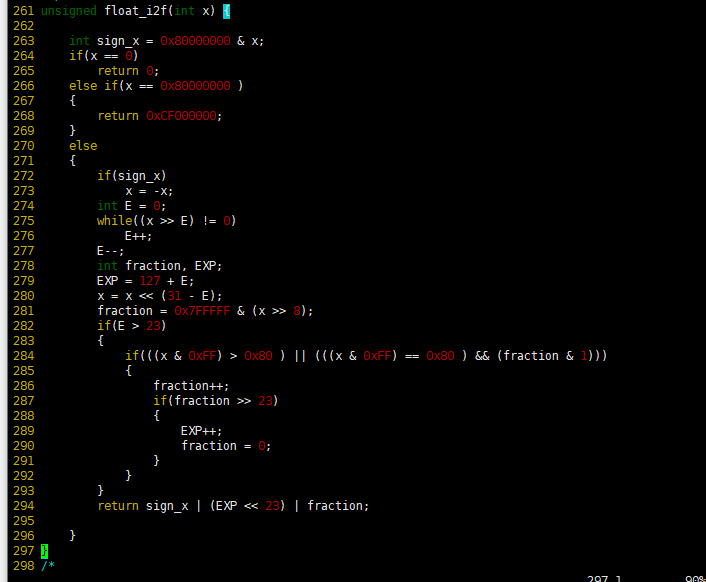


따라서 코드는 위와 같다.

2) float\_i2f

Float\_i2f의 경우 int를 float형으로 바꾸는 과정이기 때문에 몇 가지 경우를 미리 반환을 해줘야하는데 만약 x 가 0과 0x80000000인 경우 반환 값이 정해져있으므로 그 값을 반환 시켜준다.

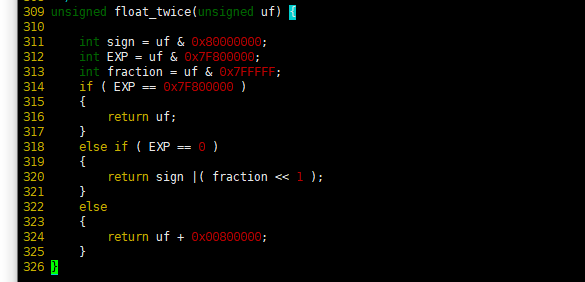
그 값들이 아닌경우 int 값이 1.xxxxx 값이 되도록 하는 EXP 값을 계산을 해주고, 그 후 fraction part를 넣어주는 과정을 거치게 되는데 이 때 rounding을 고려해줘야한다. 만약 EXP 값이 23이 넘어가 fraction part에 값이 존재하게 된다면 GRS 를 고려해줘야하는데 G는 fraction part 23bit 안에 들어갈 수 있는 맨 끝 값이고 그 후 남는 부분의 첫 부분은 R 그 뒷부분들은 S로 존재한다. 만약 RS == 00 01 이면 버림을 진행하고, RS == 11 이면 fraction part를 올림을 진행해주고, RS == 10 이 된다면 G 값이 1이라면 fraction을 올림을 0이라면 버림을 진행해주면 된다. 또한 올림을 진행해서 만약 fraction part가 23bit을 넘어가는 case가 발생할 수 있는데 그렇게 되면 EXP 값을 하나 늘리고 그 후 fraction part는 0이 되면 된다.



따라서 코드는 위와 같다.

3) float\_twice

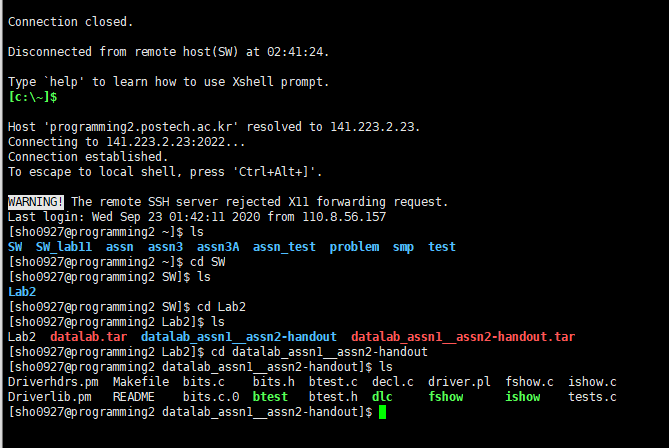
2\*f를 계산하기 위해서 EXP 이 0x7F800000 인 경우 EXP 가 0인 경우 그리고 그 외의 경우들을 나눠서 생각해야하는데 EXP가 0x7F800000 인 경우에는 그 값 그대로 반환을 하면 되고, EXP 이 0인 경우 fraction part를 left shift를 진행하면 2배가 된다. EXP가 그 외의 경우에는 EXP 값을 1 늘려주면 된다.



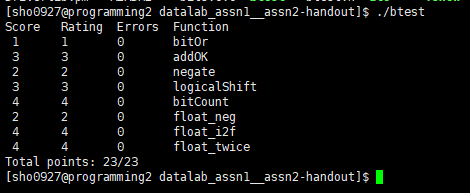
따라서 코드는 위와 같다.

4. 프로그램 실행방법 및 예제

Xshell을 이용하여 리눅스 환경에 접속한다. tar 파일을 드래그앤 드롭을 이용하여 업로드하고 tar 명령어를 이용하여 압축을 풀어주고, datalab\_assn1\_\_assn2-handout에 들어간다.



vi 편집기를 이용하여 bits.c 프로그램을 작성한다. 그 후 make 명령어를 통해 btest 파일을 만들고 ./btest 명령어를 통해 테스트를 진행한다.



5. 토론

처음 rounding을 배우지 않은 상태에서 float\_i2f 값에 문제가 생기는 상황이 발생하는 경우가 처음에는 이해를 하기 어려워서 힘들었던 것 같다. 하지만 rounding을 배운 후에는 어떻게 구현을 해야할지에 대한 감을 잡게 되어서 코딩을 진행해 나갈 수 있었던 것 같다. 그리고 또한 원래는 operation의 수를 고려하지 않고 코딩을 했었기 때문에 내가 원하는 방식대로 코딩을 진행하였지만 이번에는 operation의 수를 고려해야했기 때문에 코딩을 하고도 operation의 수가 Max Ops를 넘긴 case들이 있어 코드를 변경을 해야하는 상황이 나오기도 했다. 또한 bitwise OP에 대해서 익숙하지 않았는데 간단한 code들로도 진행을 할 수 있어서 신기함을 경험한 것 같다.

6. 결론

본 과제를 통해 bitwise operator를 이용하여 기능을 구현하고, bit 개념이 익숙하지 않았는데 그 부분에 대해서 다가갈 수 있는 LAB이 되었던 것 같다. 또한 실제 컴퓨터에서 2’s complement로 저장이 되어 있다는 내용을 이해를 할 수 있게 되었고, floating point lab에서는 실제 float 형에서 값이 sign bit 1 EXP 8 fraction 23 의 bit 수로 구성이 되어있고, 실제 값을 우리가 bit에 맞게 넣어주면 원하는 값을 구해낼 수 있다는 것을 더 자세히 알 수 있는 LAB이 되었던 것 같다.