어셈블리 프로그래밍 설계 및 실습

보고서

실험제목: Floating-Point

실험일자: 2018년 10월 09일 (화)

제출일자: 2018년 10월 25일 (화)

학 과: 컴퓨터공학과

담당교수: 이형근 교수님

실습 분반: 화 6,7 , 수 5

학 번: 2017202010

성 명: 박유림

1. 제목 및 목적

A. 제목

Floating-Point

B. 목적

ARM에는 floating point를 해주는 instruction이 존재하지 않는다. 때문에 floating point가 입력되었을 때 해당 연산을 수행시켜주는 assembly code가 필요하다.

Floating point의 adder를 구현하여 floating point의 덧셈과 뺄셈에 대하여 알아본다.

Normalize 과정에서 어떤 경우에는 right로, 어떤 경우에는 left로 shift 되는지를 고려하며 코드를 작성하여 본다.

2. 설계 (Design)

A. Pseudo code

1. Problem1

   Cmem << r1

   Cmem << r2

   -Add할 floating point를 memory로부터 읽어온다.

   r10 = 0Xffffff;

   r11 = 1

   -Mantissa값의 Normalize를 위한 변수를 선언

   r3 = r1 << 31; r4 = r2 << 31

   -r1, r2의 sign big을 가져온 후 각각 r3, r4에 저장

   If((r1<<1) == (r2<<1)) { r8=0; return; } else continue

   -만약 sign bit를 제외한 나머지 bit들이 모두 같은 경우, 0을반환하고 그렇지 않다면 아래의 코드를 더 실행한다.

   r5 = r1 << 1; r5 = r5 >> 24; r6 = r6 <<1; r6 = r6 >> 24;

   -r1, r2의 exponent값을 가져온 후 각각 r5, r6에 저장

```
r7 = r1 << 9; r7 = r7>> 9 ;          r8 = r2 << 9; r8 = r8 >>9;
```

-r1, r2의 mantissa값을 가져온 후 각각 r7, r8에 저장

```
If( r5 < r6 ) { r9 = r6 – r5; r7 = r7 >> r9 }
else { r9 = r5 – r6; r8 = r8 >> r9 }
```

-만약 r1의 exponent값이 r2의 exponent값보다 작은 경우 r9에 r6-r5를 저장, 반대의 경우에는 r5-r6를 저장한다. 이 값이 곧 shift num이다. Exponent 값이 작은 값의 mantissa 값을 shift num만큼 오른쪽으로 shift한다.

```
If( r3 != r5 ){ r7 = r7 + r8 }
else {
        if( r7 > r8 ) r7 = r7 + r8
        else r7 = r8 – r7
 }
```

-부호가 같으면 mantissa값을 더해주고 다르면 두 mantissa중 큰 값에서 작은 값을 빼준다.

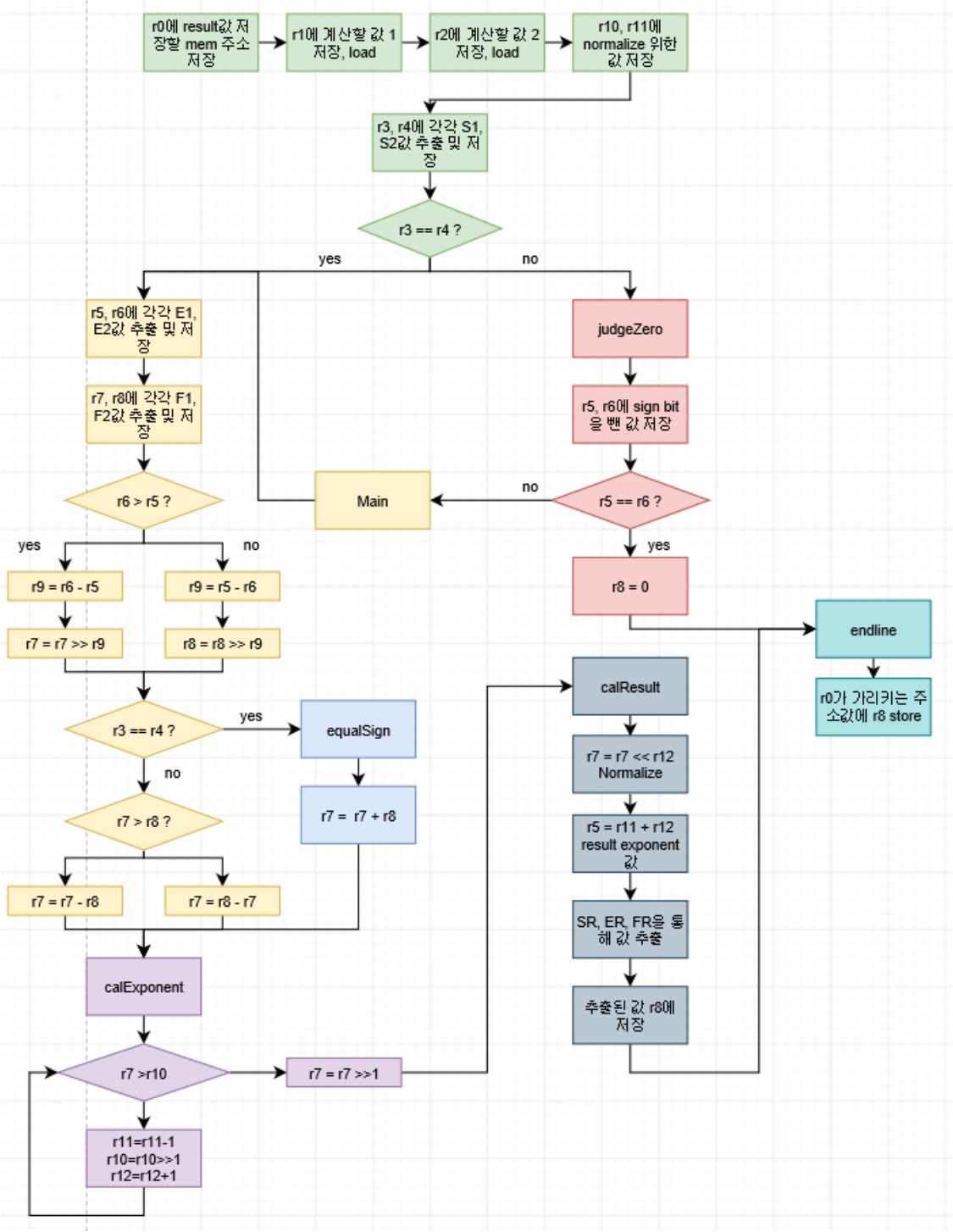
```
While(true){
        If( r7 > r10 ) { r7 = r7 >> 1; break; }
        else { r11 = r11 – 1; r10 = r10 >> 1; r12 = r12 +1; }
}
```

-Normalize 과정을 거친다.

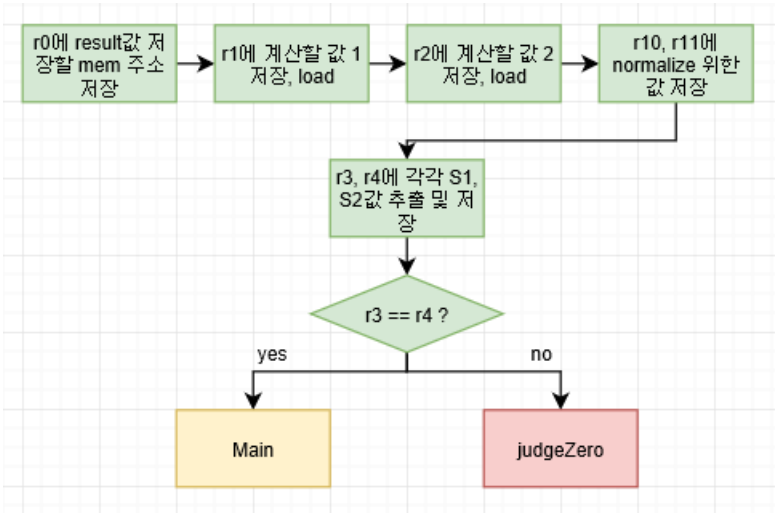
-수정이 완료된 mantissa값과 exponent값을 사용하여 결과를 도출한다.

B. Flow chart 작성

1. Problem1



1) Start



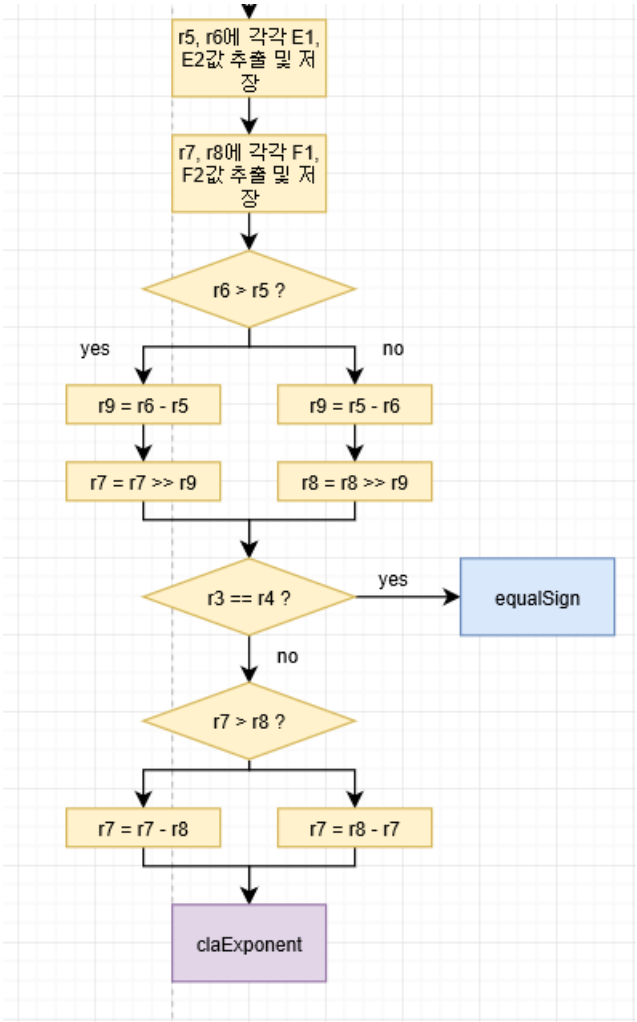
r0에 result값을 저장할 memory주소를 저장시킨다. (0x40000000)

r1, r2에 계산할 floating point값을 저장과 load를 시켜준다.

r10, r11에 normalize를 위한 값을 초기화해준다.

r3, r4에 각각 r1, r2를 오른쪽으로 shift 31번 해주어 S1, S2값을 저장시킨다. (이때, S = sign bit)

2) Main



r5, r6에 r1, r2의 왼쪽으로 1번 shift, 오른쪽으로 24번 shift한 값 즉 E1, E2값 저장. (이때, E = exponent)

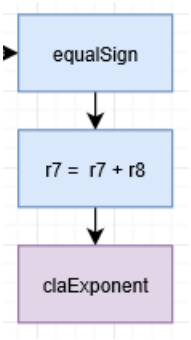
r7, r8에 r1, r2의 왼쪽으로 9번 shift, 오른쪽으로 9번 shift한 값 즉 F1, F2값 저장. (이때, F = fraction (Mantissa))

만약 r1의 exponent가 r2의 exponent보다 작다면 shift num을 저장하는 r9에 r6-r5를 저장하고 r7을 r9만큼 오른쪽으로 shift한다. 반대의 경우에는 r9에 r5~r6를 저장하고 r8을 r9만큼 오른쪽으로 shift한다.

만약 r3와 r4의 값이 같다면 equalSign으로 넘어가고 같지 않다면 아래의 과정을 행한다.

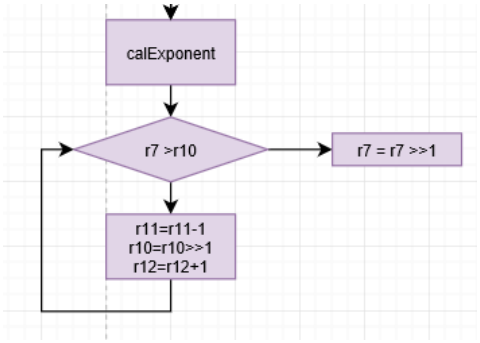
만약 r7이 r8보다 크다면 r7 = r7 – r8을, 반대의 경우에는 r7 = r8 – r7을 진행한다. 후 calExponent로 넘어간다.

3) equalSign



r7 = r7 + r8 연산을 진행 한 후 calExponent로 넘어간다.

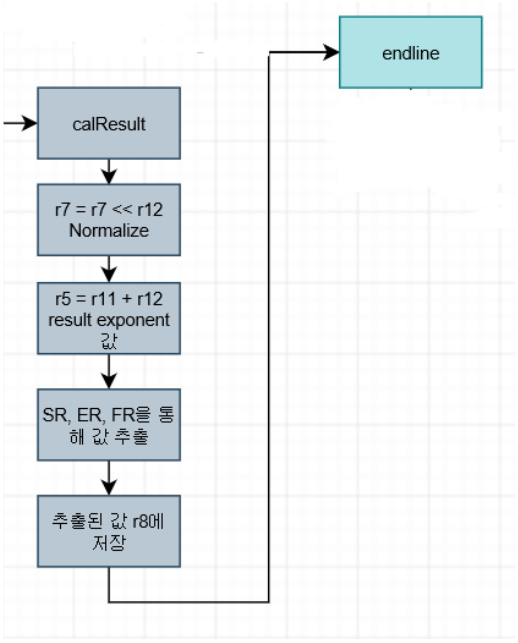
4) calExponent



r7이 r10보다 커질 때까지 r11 = r11 – 1, r10 = r10 >> 1, r12 = r12 +1 연산을 진행한다.

만약 r7이 r10보다 크다면 r7 = r7 >> 1 연산을 진행한다.

5) calResult



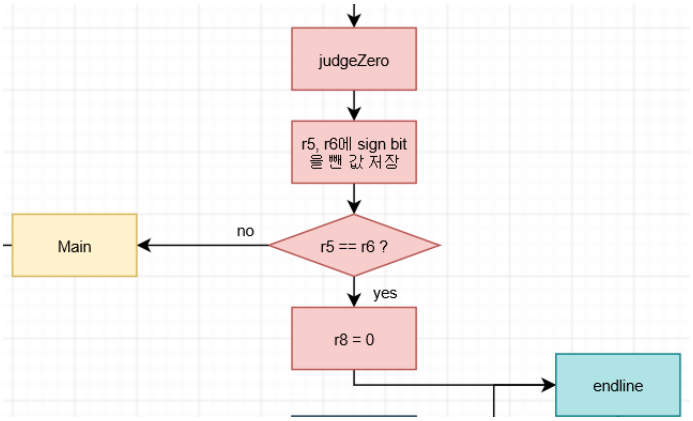
r7 = r7 << r12를 통해 normalize해준다.

r5 = r11 + r12를 통해 result exponent 값을 추출해준다.

위의 과정을 통해 얻어낸 S_R, E_R, F_R을 통해 Floating point addition의 result값을 추출한다.

추출된 result값을 r8에 저장한다.

6) judgeZero

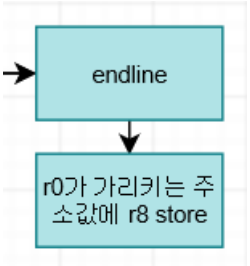


r5에 r1을 오른쪽으로 1번 shift한 값을 저장한다. 마찬가지로 r6에 r2를 오른쪽으로 1번 shift한 값을 저장한다.

r5와 r6의 값이 같다면 r8에 0을 저장한 후 endline으로 이동한다.

반대로 값이 다르다면 Main으로 이동한다.

7) Endline

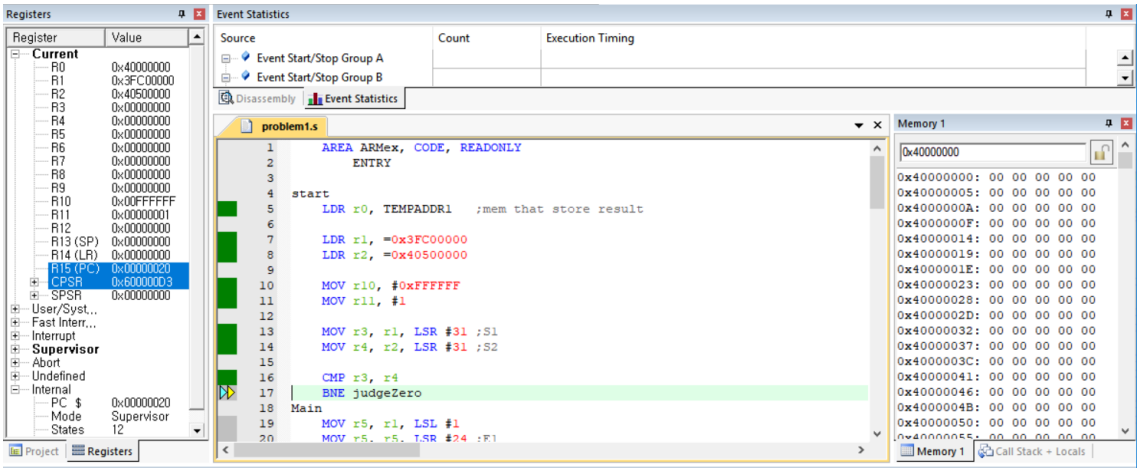


r8에 result값이 저장되어 있다. 따라서 r0가 가리키는 주소값에 8번 register에 있는 값을 store해준다.

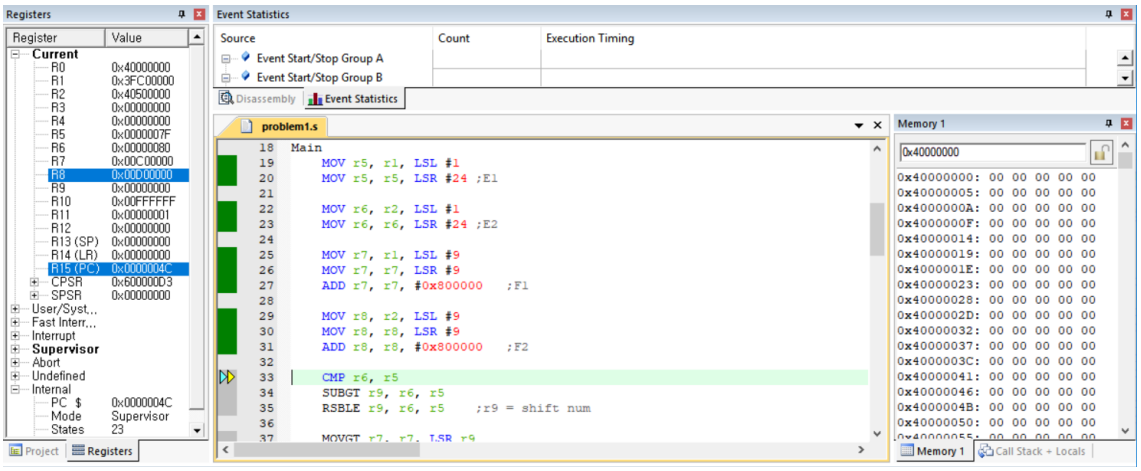
C. Result

1. Problem1

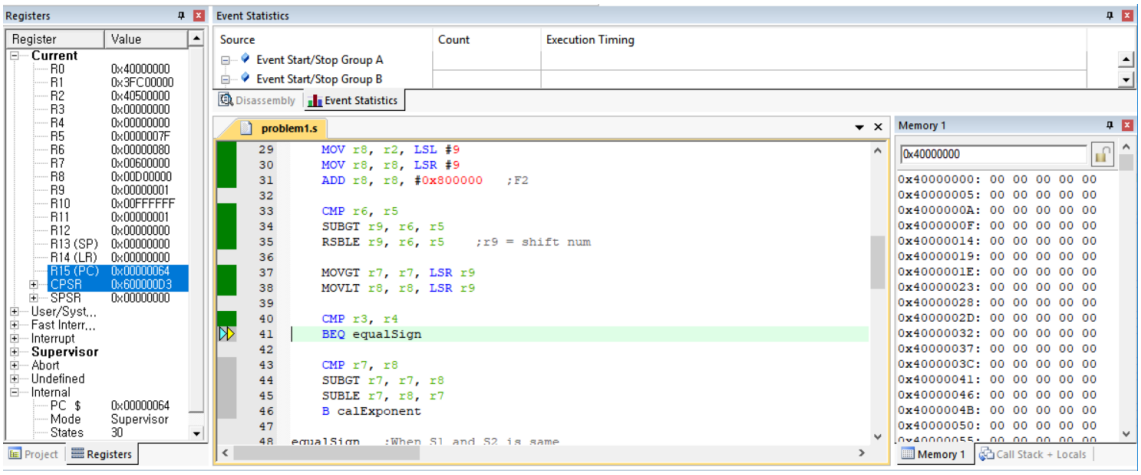
1) 0x3FC00000 + 0x40500000 = 0x40980000



start구문을 통해 필요한 값들을 register에 초기화해준다.

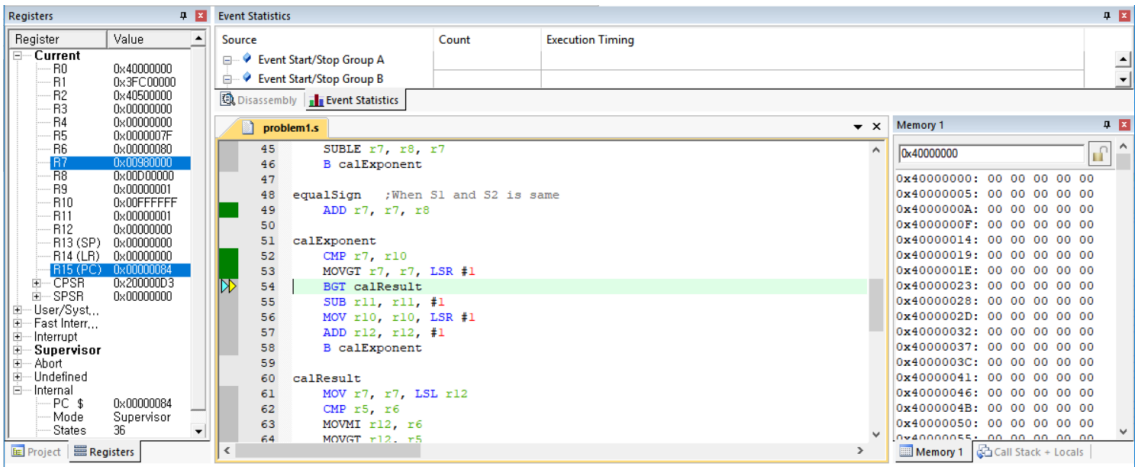


Main구문을 통해 E_1, E_2, F_1, F_2의 값을 구한다.

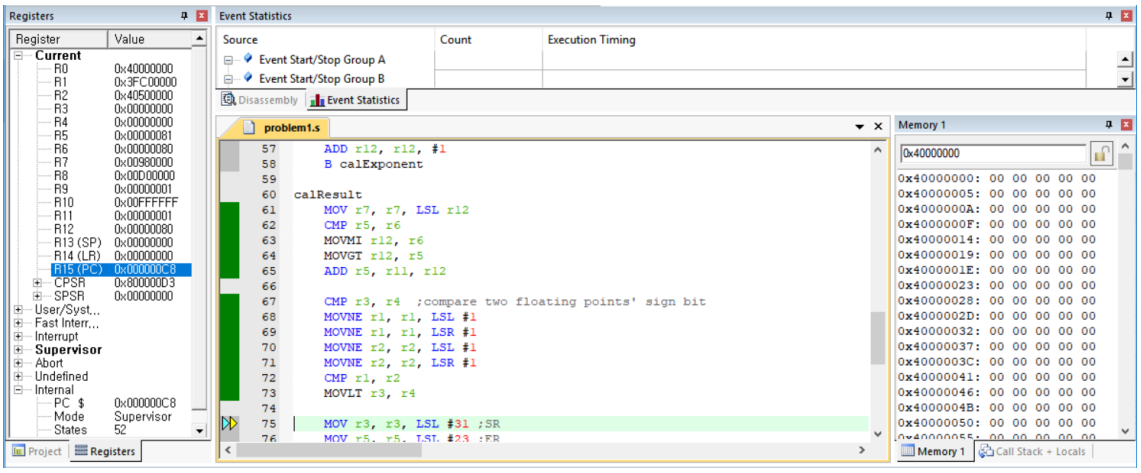


위에서 구한 E_1, E_2, F_1, F_2의 값을 이용하여 shift num을 구하고 부호를 확인한다. 이

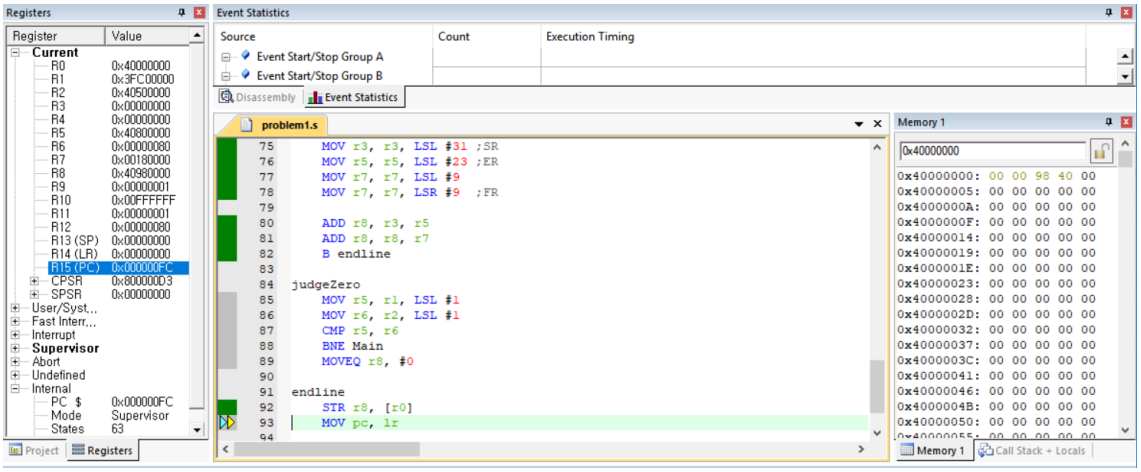
때, 부호가 같으므로 eqaulSign으로 이동한다.



equalSign구문을 통해 mantissa값을 구하고 calExponent로 이동한다. 이곳에서 normalize를 위한 변수들을 올바른 값으로 설정해준다.

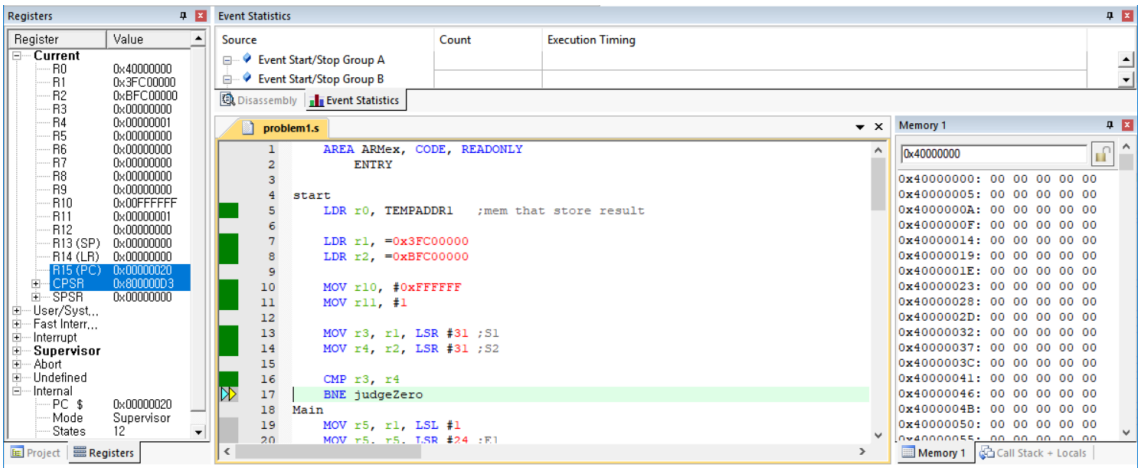


calResult 구문을 통해 result를 구할 수 있게 register를 초기화 해주었다.

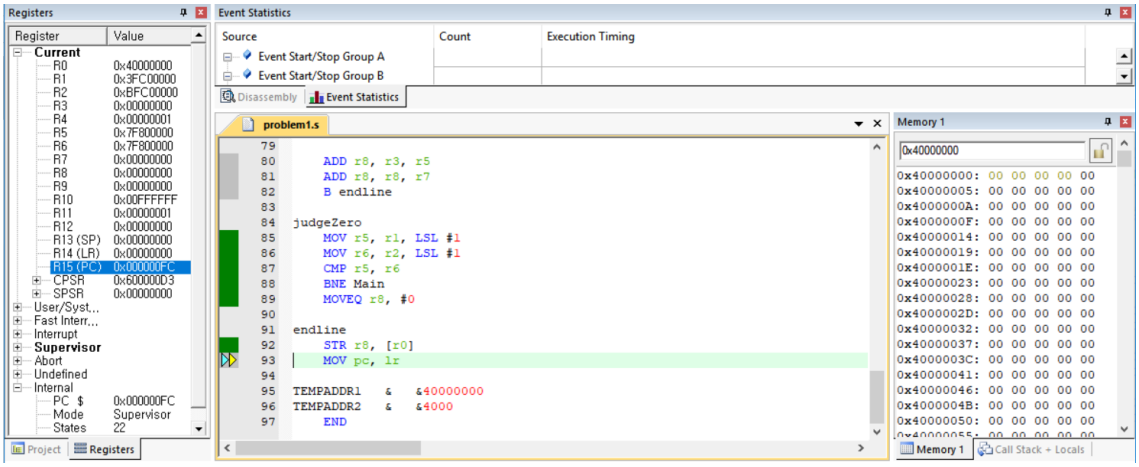


결과적으로 잘 더해져 r8에 0x40980000이 들어갔고, 이를 endline에서 r0에 저장되어있는 0x40000000번지 mem에 store했기 때문에 위와 같은 결과화면을 얻었다.

2) 0x3FC00000 + 0xBFC00000 = 0

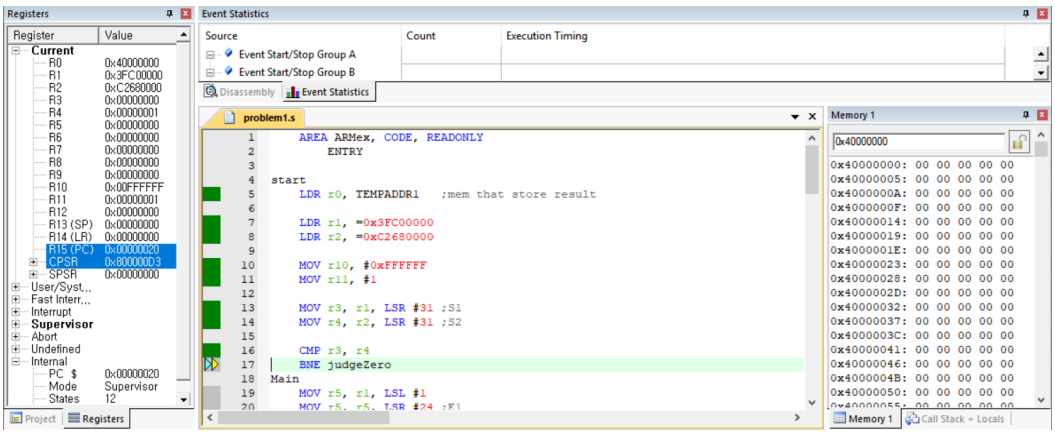


start구문을 통해 계산에 필요한 변수들을 초기화시켜준다.

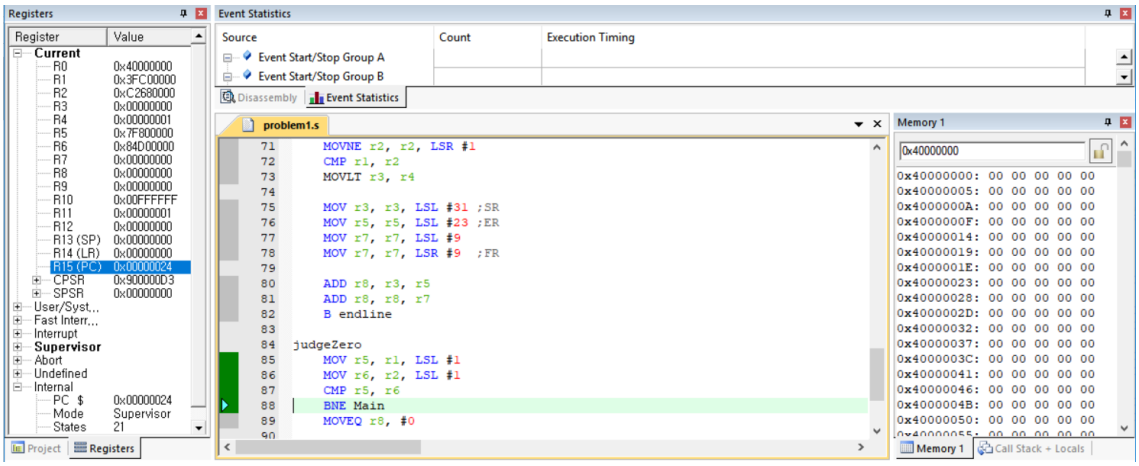


Sign bit가 다르므로 judgeZero로 이동하게 되고 sign bit을 제외한 모든 bit들이 일치하므로 r8에 0을 저장하였고 endline으로 이동하여 0x40000000번째 메모리에 result값인 0을 저장하게 된다.

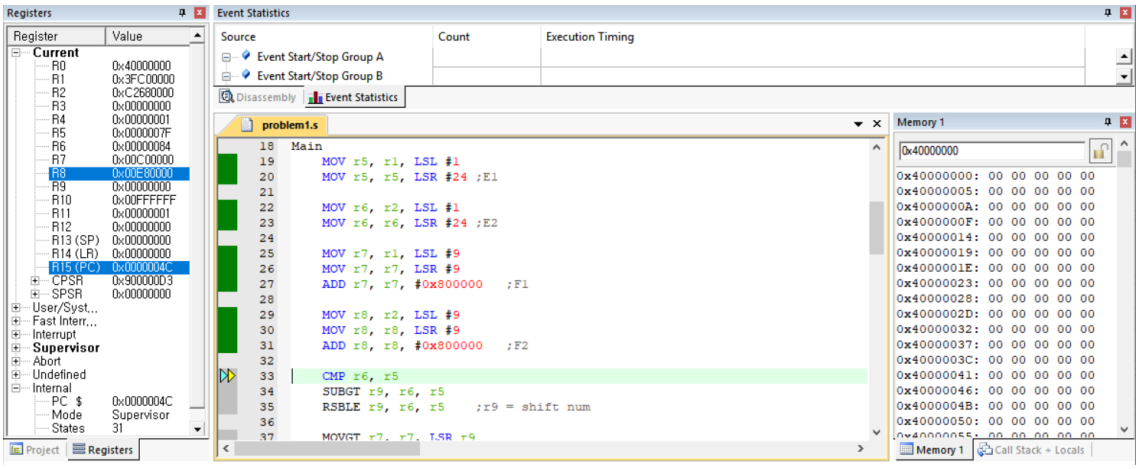
3) 0x3FC00000 + 0xC2680000 = 0xC2620000



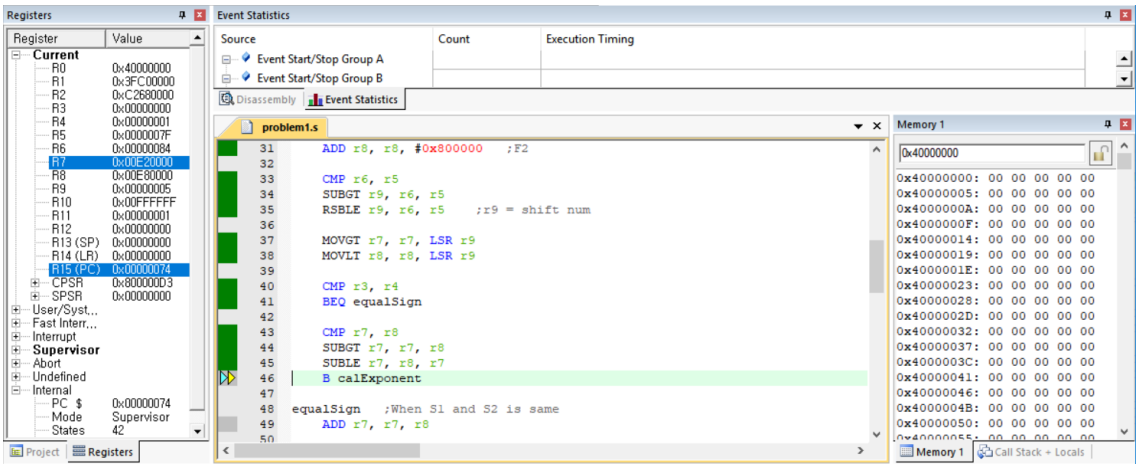
start구문을 통해 초기화를 한 상태를 나타내고 있다.



Signal bit가 다르나 signal bit를 제외한 다른 bit들 또한 다르기 때문에 judgeZero를 들어갔다가 Main구문으로 돌아오는 것을 확인할 수 있다.

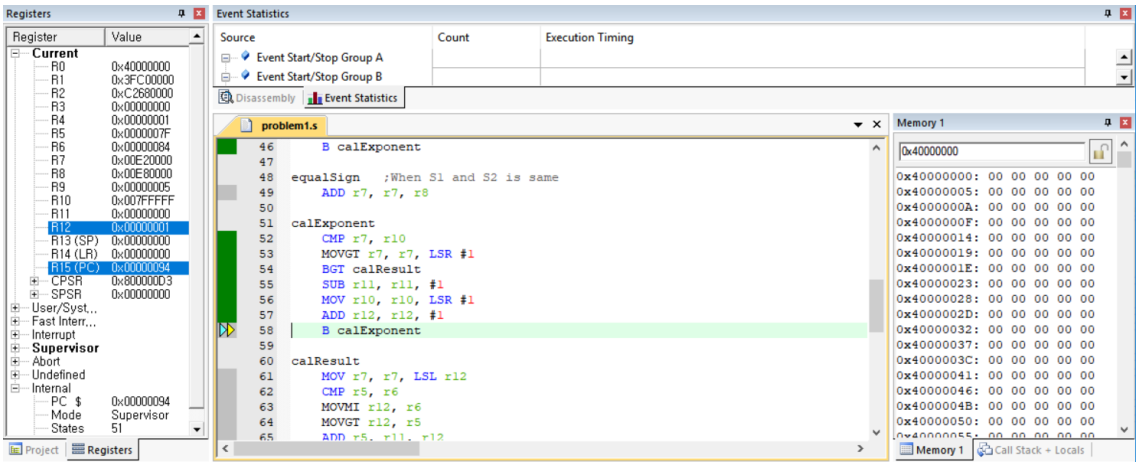


E_1, E_2, F_1, F_2 값을 r1, r2에서 얻어내어 register로 저장했다.

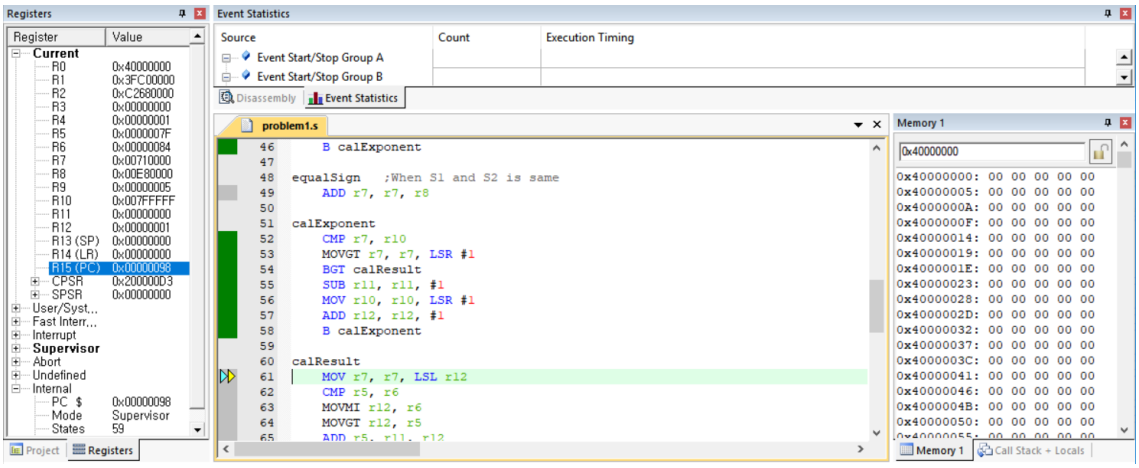


E_1, E_2, F_1, F_2 를 통해 shift num을 구하고 이를 이용하여 올바르게 바뀌어 연산이 가능하도록 값이 바뀐 register들을 확인할 수 있다. 또한 sign bit가 다르기 때문에

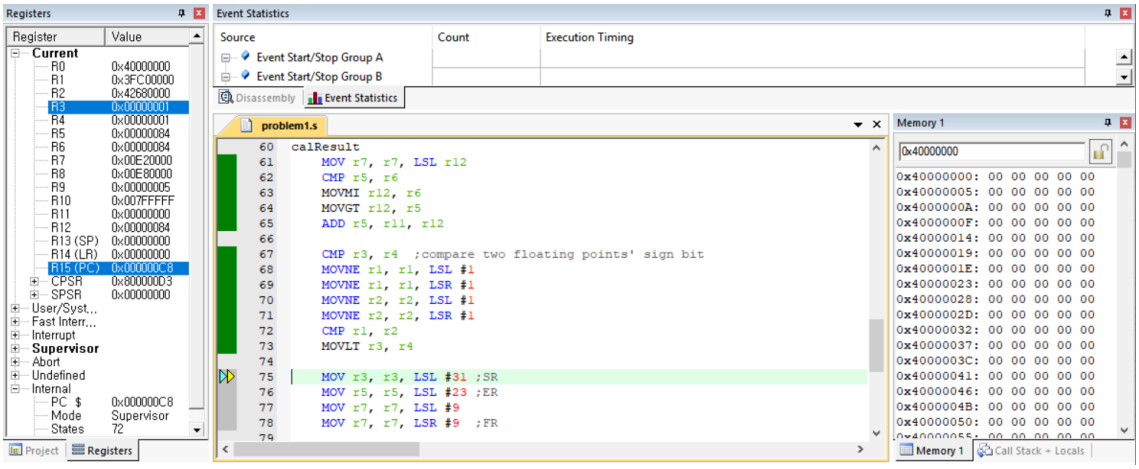
equalSign구문으로 들어가지 않은 것을 확인할 수 있다.



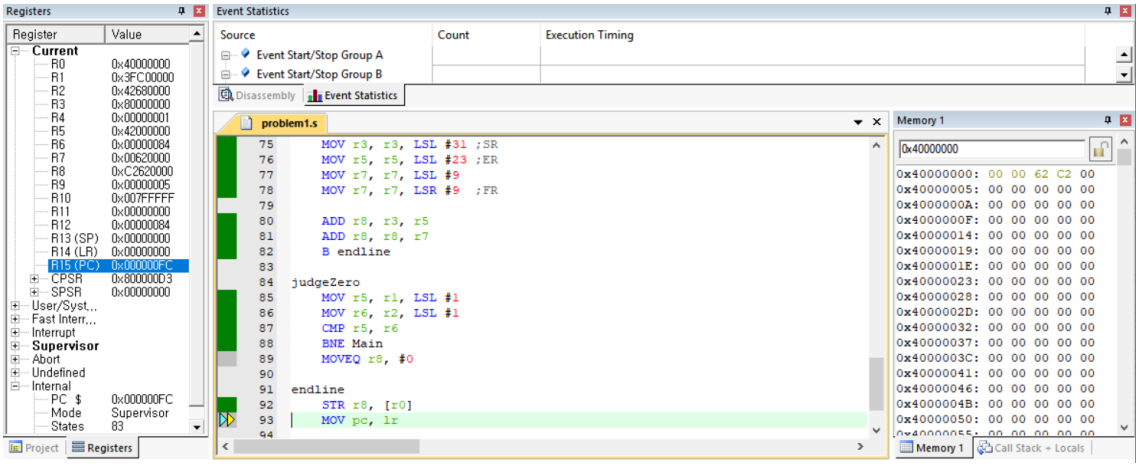
Exponent를 계산하는 중이며 r7보다 r10이 크기 때문에 calResult로 넘어가지 않고 loop문을 도는 모습을 확인할 수 있다.



loop문을 탈출



최종적으로 연산을 위해 필요한 S_R, E_R, F_R의 값이 register에 저장되었다.



최종적으로 연산을 위해 필요한 S_R, E_R, F_R의 값이 자릿수까지 맞춰 register에 저장되었다.

3. 고찰 및 결론

A. 고찰

1) Floating-Point addition

이번 과제는 다른 과제와는 조금 달랐다고 생각한다. 원래는 강의자료에 예제코드가 있어 그 코드를 보며 이해를 했던 부분이 컸는데 이번에는 강의자료에 어떤 코드도 명시되어 있지 않았다. 이에 예시없이 완벽히 이론으로만 이해를 해야 하는 부분이 힘들었다. 따라서 floating point에 대한 강의자료를 다른 때보다 두배 더 많이 읽고 인터넷을 통해 floating point의 덧셈에 대해 공부했다. 그러자 조금씩 이해가 가기 시작했고 코드를 짤 수 있었다.

코드를 짜는 부분에서 고찰했던 것들이 많았다. 2진수로 나타내는 함수가 따로 없었던 것이었다. 이에 어떻게 16진수 혹은 10진수로 나타낼 수 있는 수들을 2진수 연산을 할 것인가에 대한 고민을 하게 되었다. 이전 강의자료들을 살펴보다 보니 shift연산을 통하면 가능할 것이라고 판단하였고 실제로 적용해보니 올바르게 사용되는 것을 확인할 수 있었다.

다음으로 막혔던 부분은 floating point로 나타낸 2진수의 10진수 해석법이었다. 코드를 짜고 틀린 것이 있는지 확인하기 위해서 숫자들을 넣어보는 과정에서 문제가 발생하였다. 손과 머리를 통해서 계산해낸 결과와 구현한 프로그램을 통한 연산결과가 달랐기 때문이었다. 이에 코드의 문제인 줄 알고 디버깅을 하게 되었다. 이에 곧 2진수 floating point를 잘못 해석한 내 머리 문제라는 것을 알게 되었다. 작은 실수로 많은 시간을 버린 것 같아 허무했지만 다음부터 이런 실수를 하지 않고 꼼꼼히 보기로 다짐했다.

마지막으로 예외처리에 관한 부분이었다. 0으로 값을 넣어보지 않고 다 잘 된다고 생각했었다. 하지만 마지막에 하나의 값을 0으로 바꾼 결과 계산했던 값과는 다른 값을 가지게 된다는 것을 알게 되었다. 이에 예외처리를 진행하게 되었다. 이런 상황들을 보아서 항상 코드에서는 많은 예외가 항상 존재하기 때문에 TEST를 많이 진행해야 한다는 생각을 하게 되었다.

B. 결론

1) Floating-point addition

두 floating point의 부호가 같은 경우	두 floating point의 부호가 다른 경우
1.아래의 bit data를 각 숫자에 대해서 추출 ● Sign bits ● Exponent bits ● Mantissa(Fraction)bits	1.아래의 bit data를 각 숫자에 대해서 추출 ● Sign bits ● Exponent bits Mantissa(Fraction)bits
2.Mantissa 형태 앞에 1을 붙임	2. Mantissa 형태 앞에 1을 붙임
3.Exponent를 비교 ● 두 floating point의 exponent의 차이 값 구함 ● |Exponent1 – Exponent2|=shift num	3. Exponent를 비교 ● 두 floating point의 exponent의 차이 값 구함 ● |Exponent1 – Exponent2|=shift num
4.shift num이 0이 아닌 경우 ● Exponent값이 작은 값의 mantissa 값을 shift num만큼 오른쪽으로 shift ● Mantissa의 앞에 1.이 있음을 주의할 것	4. shift num이 0이 아닌 경우 ● Exponent값이 작은 값의 mantissa 값을 shift num만큼 오른쪽으로 shift ● Mantissa의 앞에 1.이 있음을 주의할 것
5.각 mantissa값을 더해 줌	5. 각 mantissa값을 더해 줌
6.Mantissa값을 Normalize 해줌 ● Exponent값 수정	6. Mantissa값을 Normalize 해줌 Exponent값 수정
7.수정이 완료된 mantissa값과 exponent값을 사용하여 결과를 도출 ● S = 두 floating point의 sign bit ● Exponent = Normalize result	7. 수정이 완료된 mantissa값과 exponent값을 사용하여 결과를 도출 ● S = 두 floating point의 sign bit ● Exponent = Normalize result

exponent + 127 ● Fraction = mantissa	exponent + 127 ● Fraction = mantissa

8.Performance

9. 참고문헌

이형근/ 어셈블리프로그래밍/ 광운대학교/ 2018