Assignment 4

1. Problem & Purpose

i. Floating point 값 2개를 메모리로부터 load하여 덧셈 연산을 수행한다.

ii. 양수, 음수 및 0값만이 연산이 가능하다.

iii. 결과값은 다음 메모리 주소에 저장한다.

2. Used Instruction

I. 5 : LDR // STR // MOV // MOVEQ // MOVGE // MOVLT // CMP // B // BEQ // BNE // BLT // SUBHI // MUL // END

i. LDR Rd, operand1 : operand1의 메모리 위치의 값을 word 크기만큼 Rd에 불러온다.

ii. STR Rd, [R0, offset] : R0으로부터 offset만큼 이동한 위치에 R0의 값을 word 크기만큼 저장한다.

iii. MOV Rd operand1 : operand1에 있는 값을 Rd에 저장한다.

iv. MOVEQ Rd operand1 : CMP로 비교하였을 때 = 일 경우, operand1에 있는 값을 Rd에 저장한다.

v. MOVGE Rd operand1 : CMP로 비교하였을 때 >= 일 경우, operand1에 있는 값을 Rd에 저장한다.

vi. MOVLT Rd operand1 : CMP로 비교하였을 때 < 일 경우, operand1에 있는 값을 Rd에 저장한다.

vii. CMP Rd, operand1 : Rd – operand1을 한 state를 cpsr에 업데이트한다.

viii. B operand : operand의 위치로 pc를 이동하여 작업을 수행한다.

ix. BEQ operand : CMP로 비교하였을 때 =일 경우 operand의 위치로 pc를 이동하여 작업을 수행한다.

x. BNE operand : CMP로 비교하였을 때 !=일 경우 operand의 위치로 pc를 이동하여 작업을 수행한다.

xi. BLT operand : CMP로 비교하였을 때 <일 경우 operand의 위치로 pc를 이동하여 작업을 수행한다.

xii. ADD Rd, R0(, R1) : Rd에 R0와 R1을 더한 값을 저장한다. R1이 없을 경우 Rd = Rd + R0로 저장한다.

xiii. ADDEQ Rd, R0(, R1) : CMP로 비교하였을 때 = 일 경우 Rd에 R0와 R1을 더한 값을 저장한다. R1이 없을 경우 Rd = Rd + R0로 저장한다.

xiv. ADDGE Rd, R0(, R1) : CMP로 비교하였을 때 >= 일 경우 Rd에 R0와 R1을 더한 값을 저장한다. R1이 없을 경우 Rd = Rd + R0로 저장한다.

xv. SUBGE Rd, R0 : CMP로 비교하였을 때 >=일 경우 Rd – R0를 Rd에 저장한다.

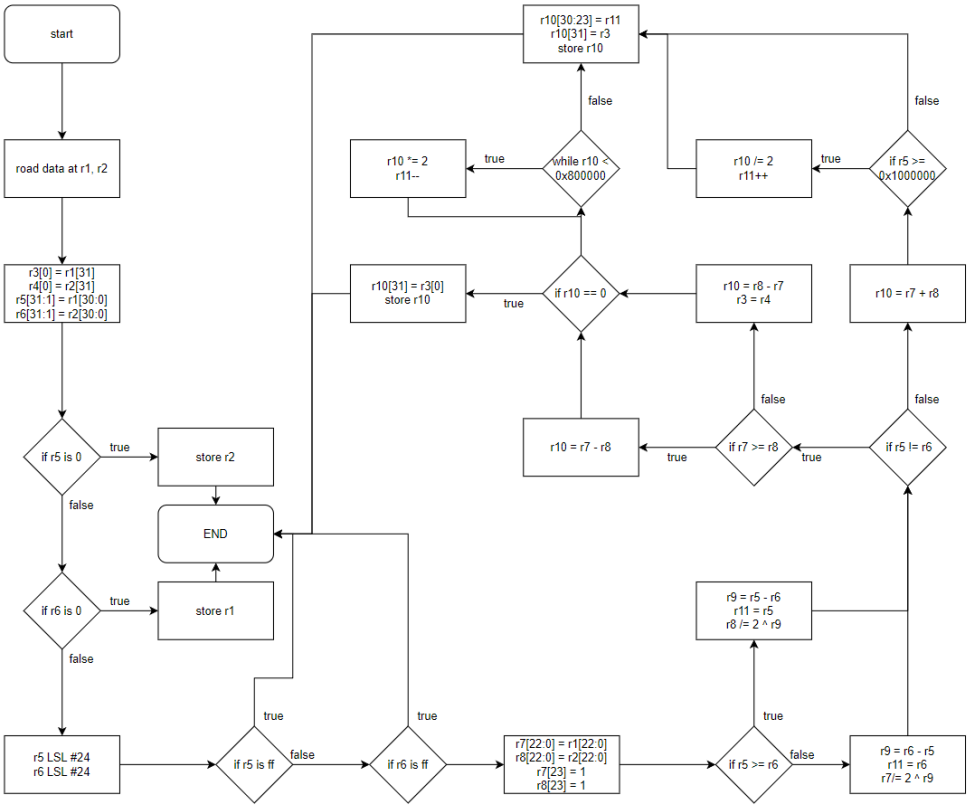
xvi. SUBLT Rd, R0 : CMP로 비교하였을 때 <일 경우 Rd – R0를 Rd에 저장한다.

xvii. BIC Rd, operand1, operand2 : operand1 & !(operand2)의 결과를 Rd에 저장한다.

xviii. END : Assembly code가 끝났음을 의미하는 Instruction

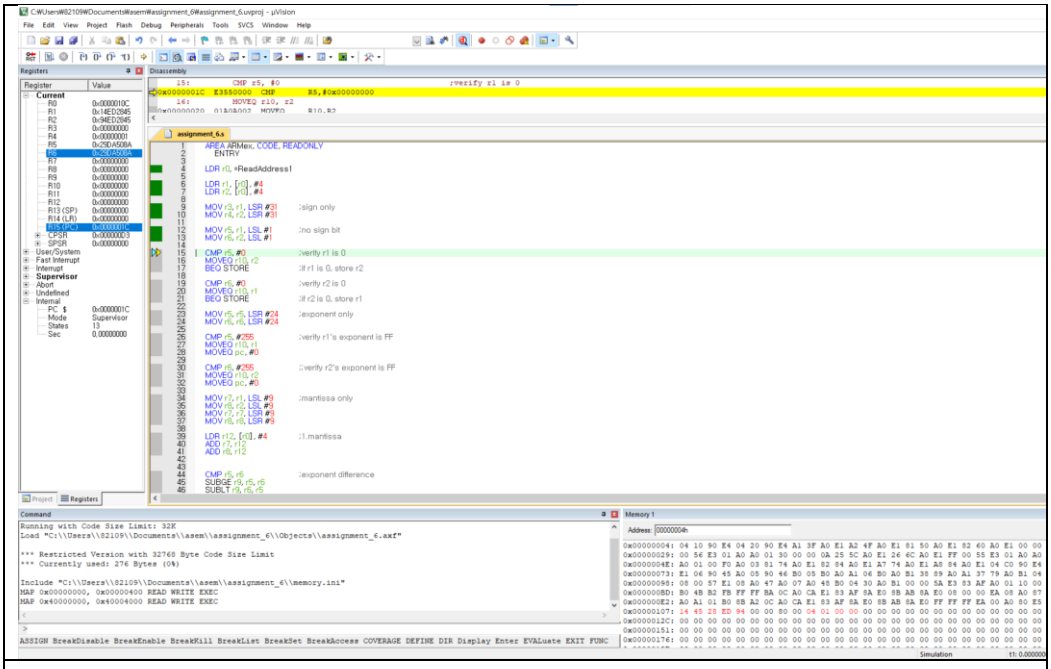
3. Design(Flow chart)

i. 5 flow chart

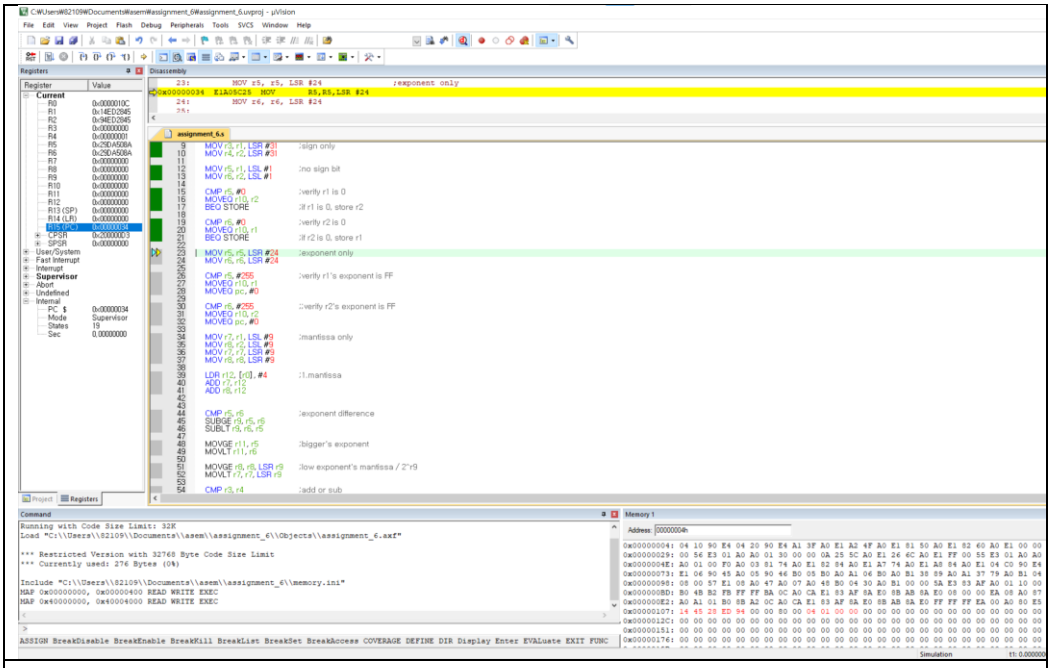


4. Conclusion

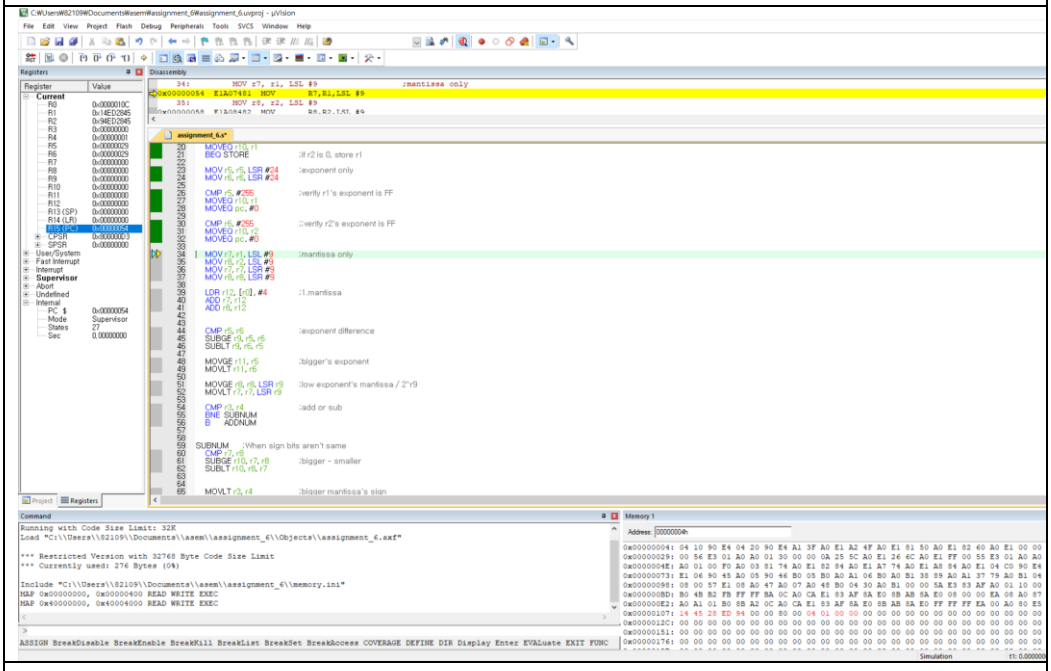
i. 5 result



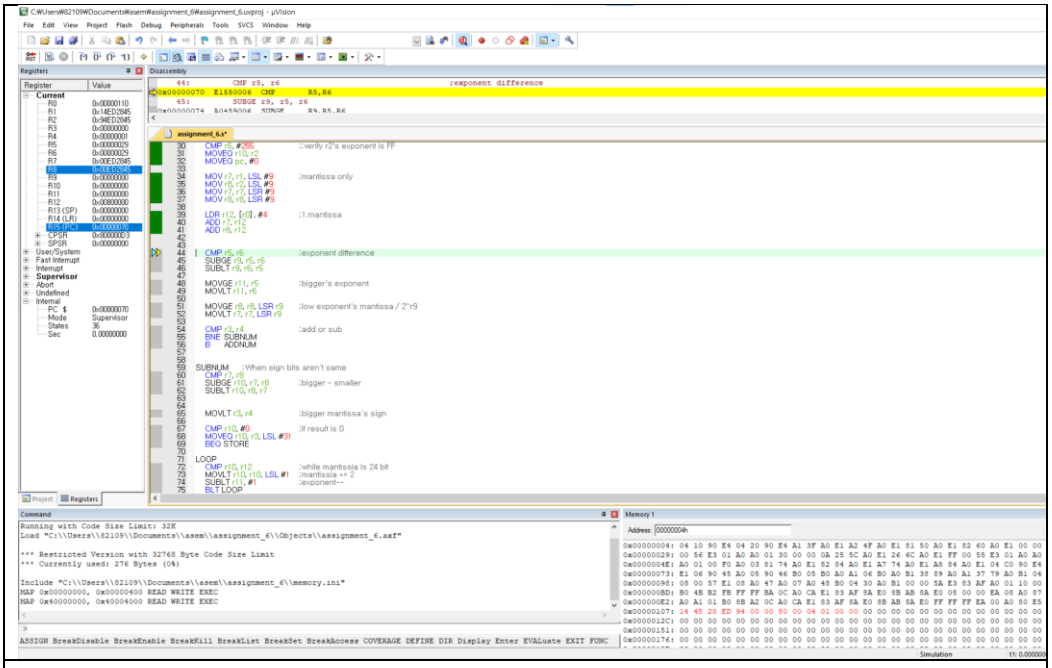
비교할 두 수가 r1, r2에 저장된다. r3와 r4에는 이들의 sign bit만 저장된다. r5, r6에는 sign bit을 제외한 데이터가 저장된다.



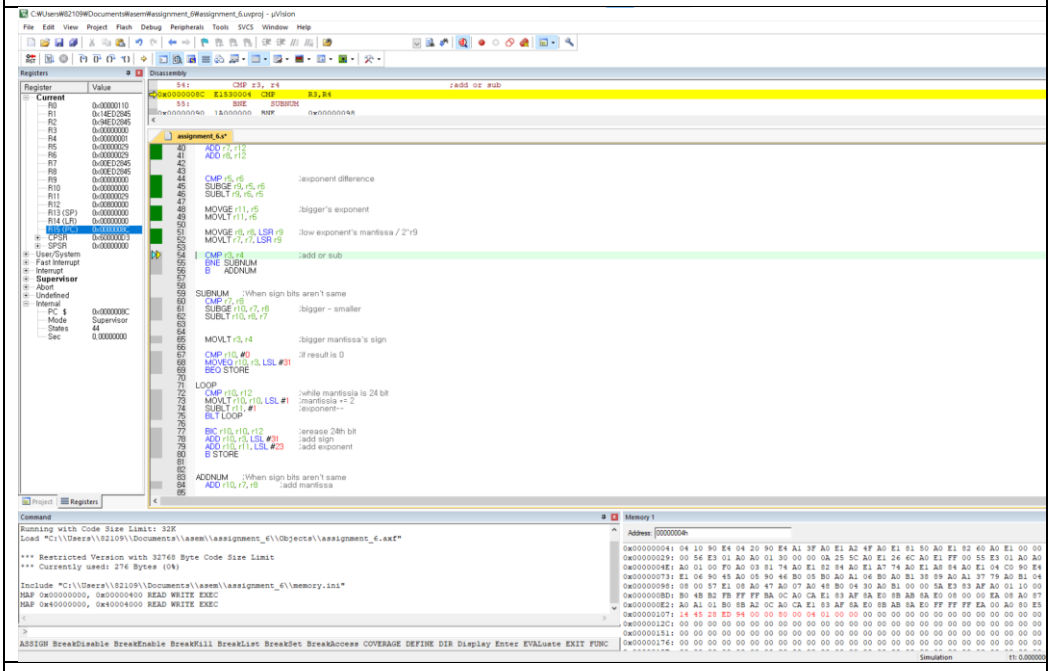
r5나 r6가 0일 경우 더하는 값이 0이라는 의미로 반대 값을 저장할 위치에 저장 후 프로그램을 종료한다.



r5와 r6를 exponent 값만 남겨둔다. 그 후 exponent의 값이 ff인 경우 infinite거나 Nan이기 때문에 연산을 수행하지 않고 함수를 종료한다.

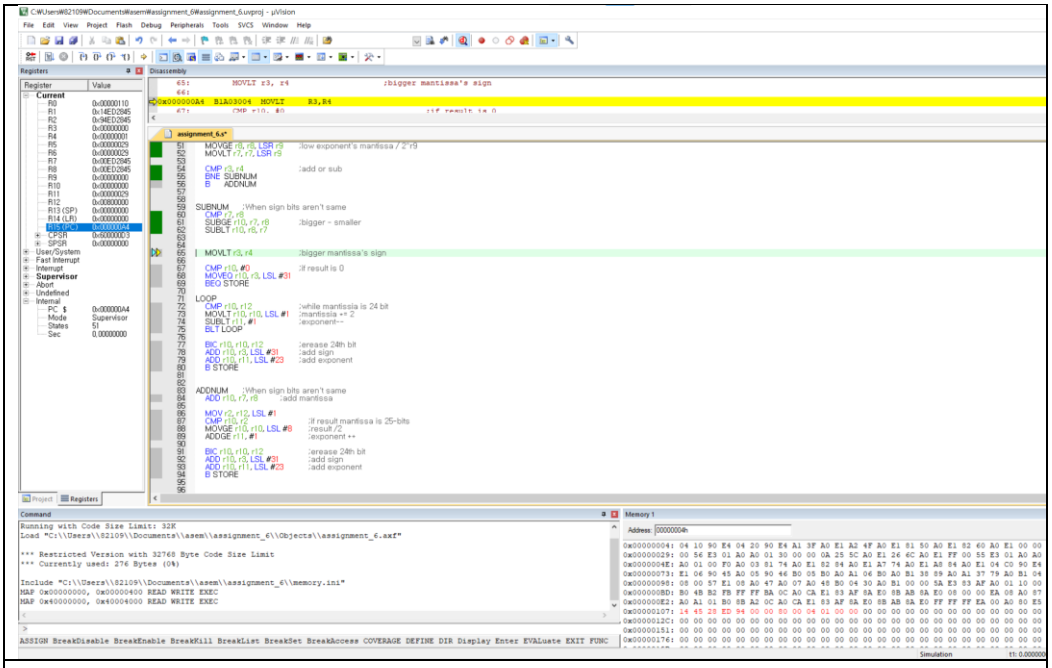


r7과 r8에는 1bit의 1값과 23-bits r1, r2의 mantissa 값을 연결하여 저장시킨다.

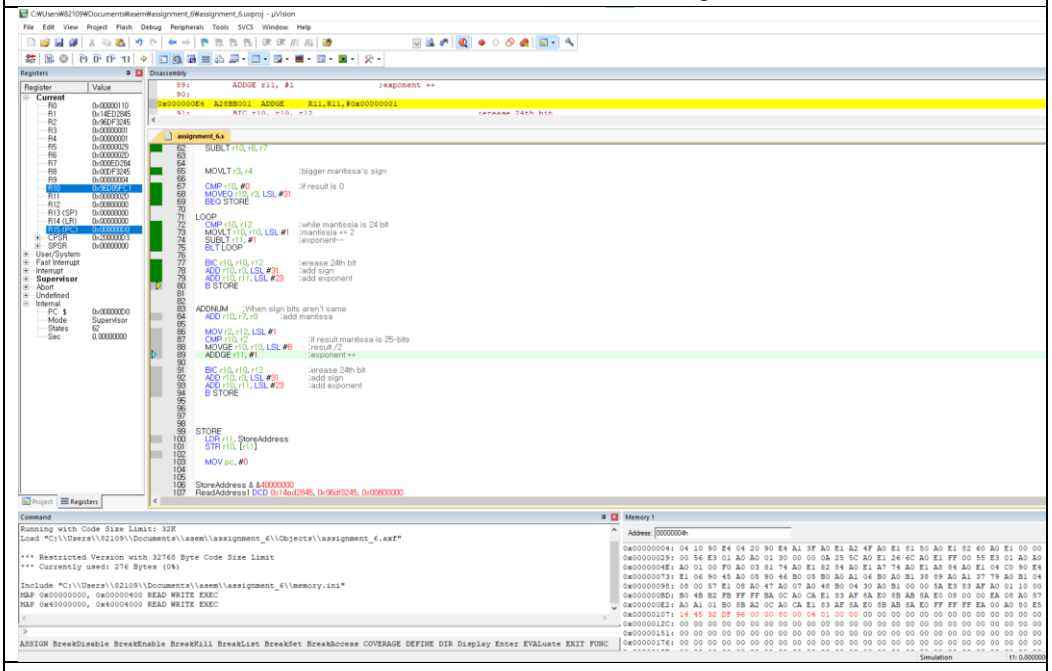


두 exponent를 비교하여 두 값의 차의 절대값을 r9에 저장하고, r11에는 큰 쪽의 exponent를 저장하며, 작은 쪽의 mentissa를 2^r9만큼 나눈다.
그 후 sign bit가 같은 경우 덧셈 branch를, 다른 경우 뺄셈 branch를 수행한다.

1. 뺄셈(a-b = 0인 경우)

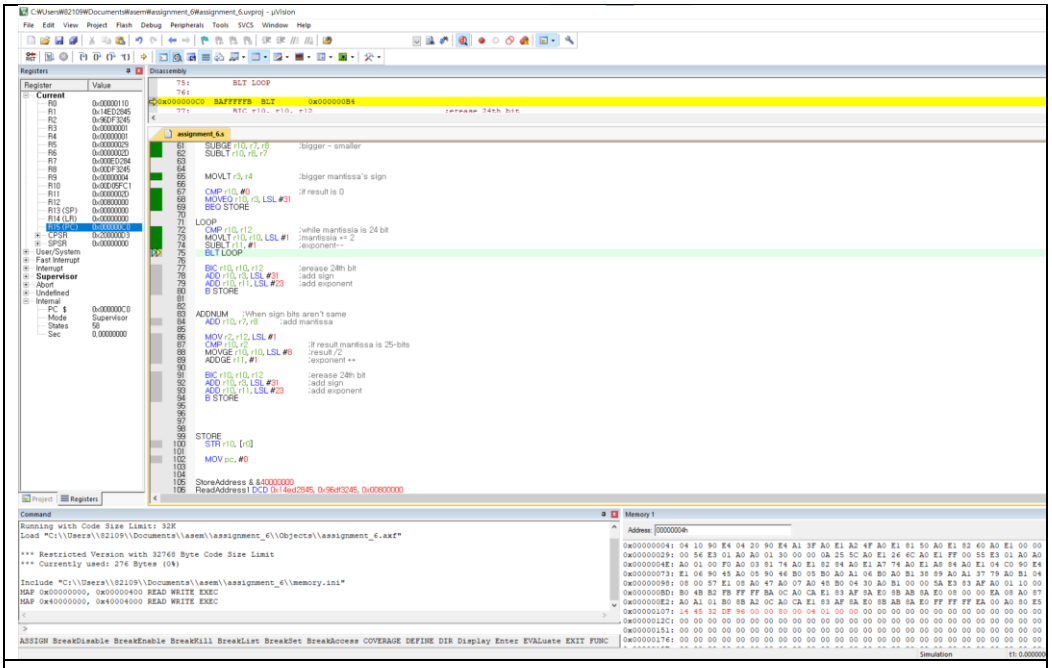


mentissa가 큰 쪽에서 작은 쪽을 빼고, r3에는 큰 쪽의 sign을 저장한다.

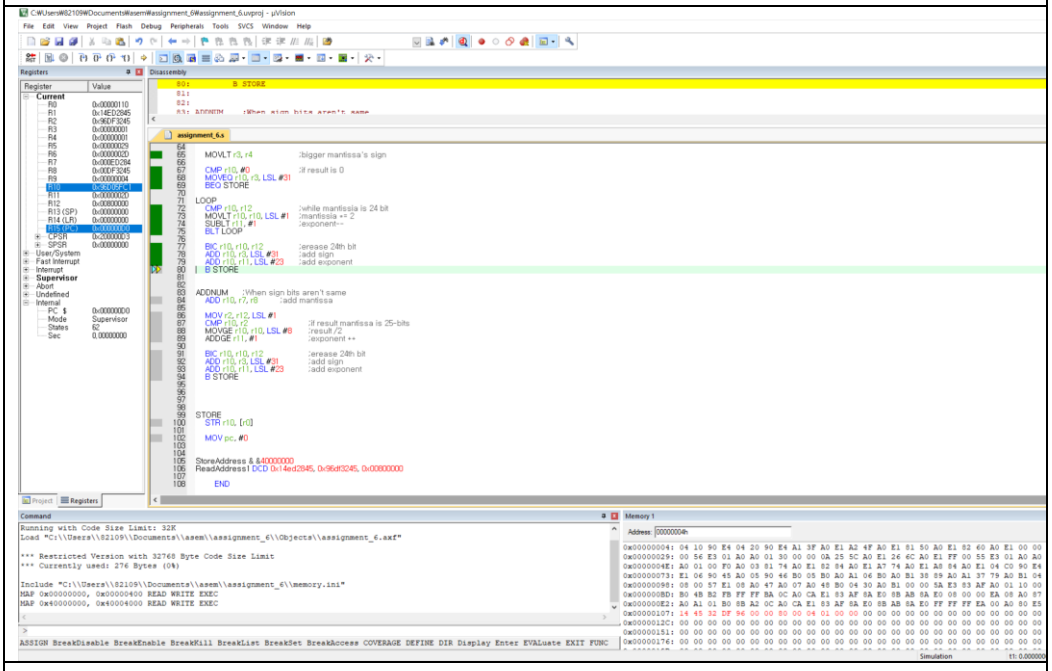


만약 뺄셈의 결과가 0인 경우 결과로 0을 저장한 후 프로그램을 종료한다.

2. 뺄셈(a-b != 0인 경우)

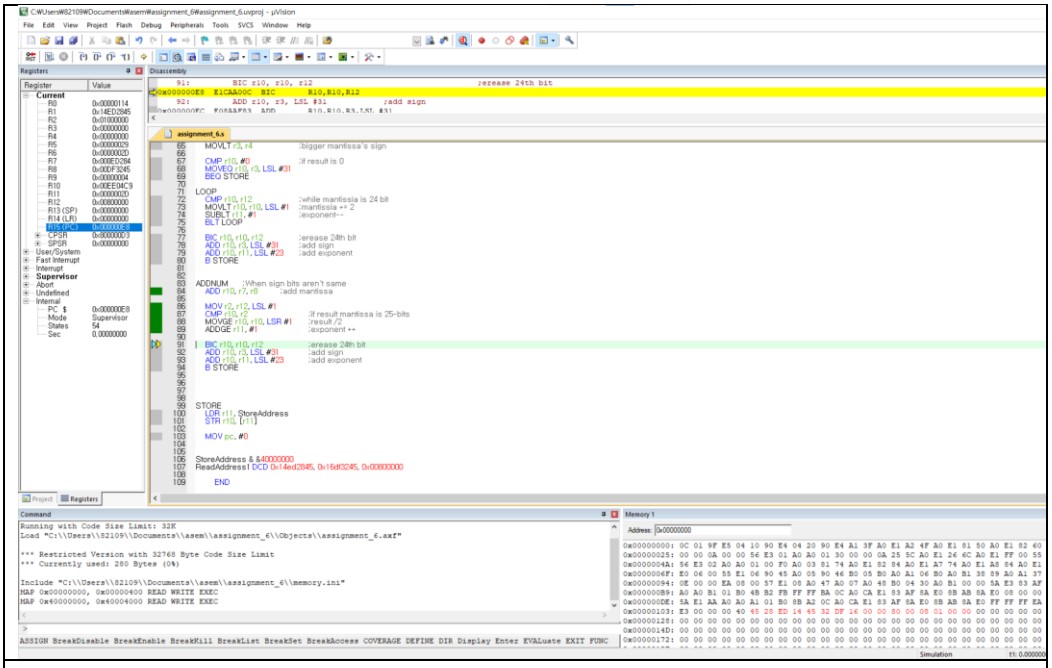


뺄셈을 한 값이 24bit보다 클 때까지 mentissa에 LSL 1을 수행시키고 exponent는 1을 빼준다.

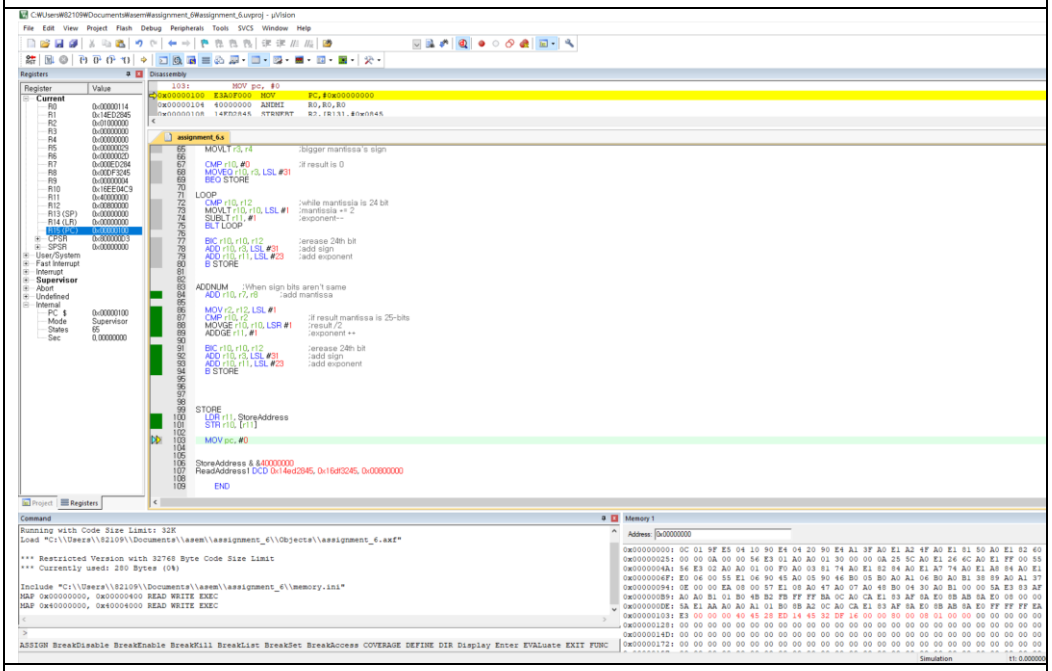


mentissa의 24번째 비트를 제거하고, r10에 sign bit와 exponent를 붙어준 뒤 이를 저장한다.

덧셈의 경우



덧셈의 경우 두 mentissa를 더한 값을 r10에 저장하고 해당 값이 25비트인지 확인한다. 만약 25비트일 경우 LSL1을 수행한다.



r10에 24번째 비트를 삭제하고 sign 과 exponent를 연결한다. 이후 이 값을 저장할 위치에 저장한다.

5. Consideration

- 위의 해당 과제를 수행하면서 처음으로 이정도로 긴 코드를 만든다는 것이 익숙하지 않았다. 우선 각 값이 0이거나 무한대, Nan인지 확인을 하고 연산이 필요하

며, sign에 따라 다른 연산 과정을 거쳐야하였고, 특히 뺄셈의 경우 0으로 나오는 경우까지 고려해주어야 했다. 하지만 또한 연산하는 줄이 길기 때문에 결과를 검증하는데에도 시간이 소요되었다.

6. Reference

i. 이준환 교수님/어셈블리프로그램설계및실습/광운대학교(컴퓨터정보공학부)/2021