어셈블리프로그램설계및실습 보고서

과제 주차: 6주차

학 과: 컴퓨터정보공학부

담당교수: 이형근 교수님

실습분반: 화 6,7 목 5

학 번: 2019202021

성 명: 정성엽

제 출 일: 2022.11.07(월)

1. Problem Statement

ARM 어셈블리에서 floating point number 관련 instructino이 존재하지 않으므로 해당 연산 수행하는 어셈블리 코드를 작성하고 floating point number adder를 통해 덧셈과 뺄셈을 진행하여 연산 원리를 익힌다.

2. Design

A. FLOATING POINT

해당 과제를 이해하기 위해서는 Floating point number에 대한 이해가 필요하다.

IEEE 754 표준 floating point의 표현 방법은 1bit는 sign bit, 다음 8bit는 exponent로 승수를 표현, 나머지 23bit는 mantissa로 표현하여 승이 곱해질 소수점 부분을 가리킨다. 이는 32bit 기준으로 했을 때의 범위이며 64bit로도 가능하다.

1bit (Sign)	8bit (exponent)	23bit mantissa

위의 표를 참고하여 각 자리수를 고려하여 식으로 나타내면 아래와 같이 식으로 표현할 수 있다.

$$FLOAT\ NUMBER = (-1)^{Sign} \times 2^{exponent-127} \times (1.mantissa)$$

B. Floating point의 덧셈 뺄셈 연산

Floating point의 덧셈 또는 뺄셈 연산에서 뺄셈은 덧셈 연산에서 뒤의 수의 부호를 바꾸는 방법을 사용하여 결국 덧셈 연산만을 이용해서도 뺄셈이 가능하다. 만약 두 수의 부호가 같은 경우 그대로 덧셈을 진행하고 다른 경우 절대값이 더 큰 부호를 따라 간다.

i. 두 Floating point에서 Sign, Exponent, Mantissa bit를 추출한다. 해당 bit는 앞서 설명한 표를 참고하여 각 1bit, 8bit, 23bit로 나눈다.
   1. Exponent bit는 둘 중 큰 것을 따로 다른 register에 저장한다
   2. mantissa bit에서는 자릿수를 유지하기 위해 처음 bit에 1을 더한다.

ii. 두 Exponent 값의 차의 절대값을 구하고 이 차만큼 shift num 할 수 있도록 저장한다.
   1. shift num이 0인 경우에는 그대로 넘어간다.
   2. 0이 아니면 exponent 값이 작은 곳의 mantissa를 shift num에 저장된 만큼 LSR(right shift)를 진행한다.

iii. sign 비트만을 추출했던 것을 비교한다.
   1. 같은 경우 그대로 mantissa 값을 더한다. 그리고 그 sign bit를 저장한다.

2. 다른 경우 두 mantissa의 차의 절대값을 구하고 더 큰 절대값을 가지고 있는 값의 sign을 가져와 저장한다.

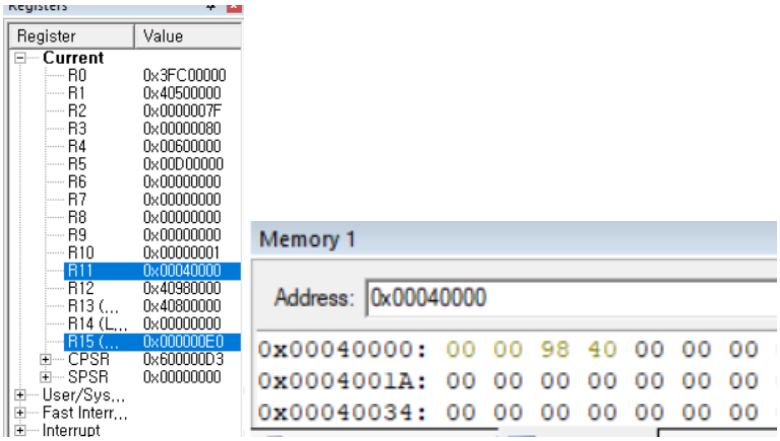
iv. mantissa 값의 normalize 를 진행한다.

1. 조건 식에 따라 Mantissa의 자릿수를 shift 연산을 통해 이동하여 맞춘다.
2. 자릿수를 맞춰가며 최종 Exponent 값을 수정한다.

v. 구한 sign bit, Exponent bit, mantissa bit를 각 자리 수에 맞게 더하여 floating point add 연산을 마무리하고, 값이 0인 경우에 대한 검사를 반드시 진행한다.

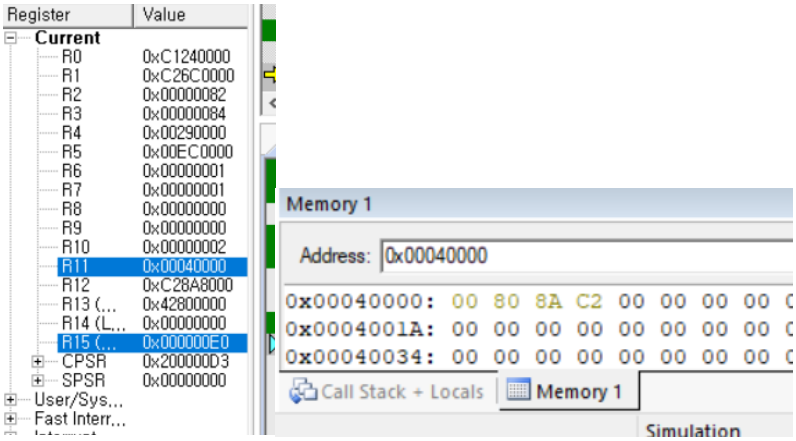
3. Conclusion

A. 양수 더하기 양수



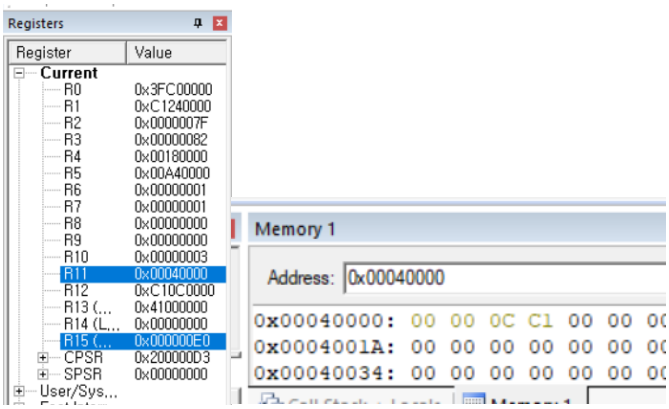
0x3FC00000(1.5)와 0x40500000(3.25)를 더했을 때 결과이다. 정상적으로 0x40980000(4.75)가 출력되었다.

B. 음수 더하기 음수



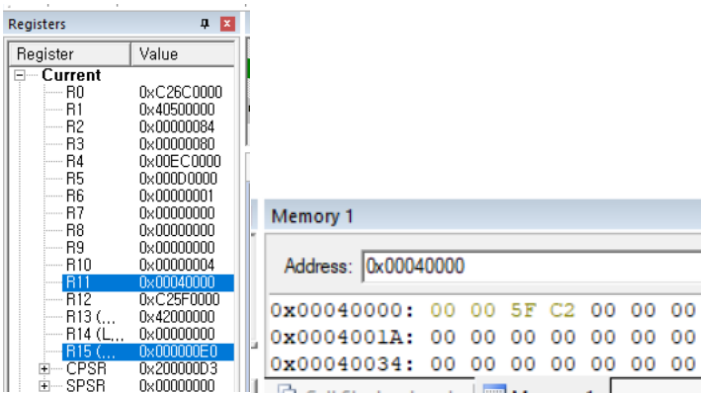
0xC1240000(-10.25)와 0xC26C0000(-59)를 더했을 때 결과이다. 정상적으로 0xC29A8000(-69.25)이 출력되었다.

C. 양수 더하기 음수



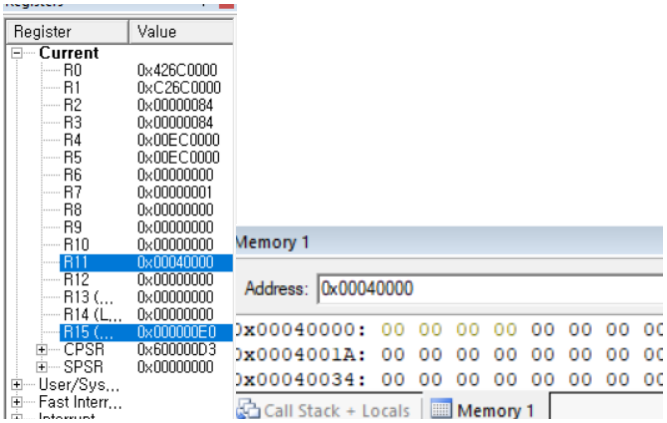
0x3FC00000(1.5)와 0xC1240000(-10.25)를 더했을 때 결과이다. 정상적으로 0xC10C0000(-8.75)이 출력되었다.

D. 음수 더하기 양수



0xC26C0000(-59)와 0x40500000(3.25)를 더했을 때 결과이다. 정상적으로 0xC25F0000(-55.75)가 출력되었다.

E. 부호만 다른 경우



0x426C0000(59)와 0xC26C0000(-59)를 더했을 때 결과이다. 정상적으로 0x00000000(0)이 출력되었다.

4. Consideration

이번 과제에서 float point의 표현 방식에 대해 배우고 이 float point의 덧셈 연산을 직접 구현하였다. 기존에 배운 lsl과 lsr 명령어를 사용하여 자유롭게 자릿수를 오가며 bit를 추출하였다. 또한 branch와 label을 이용하여 각 필요한 연산들을 적재적소에 사용할 수 있도록 하였다.

이번 과제에서 중요했던 Mantissa Normalize할 때 처음 bit 자리에 추가한 1을 제외하고 24bit 이하에 이어질 수 있도록 0x00F00000 즉 15728640보다 큰 경우 lsr 명령어를 통해 오른쪽으로 이동하고 0x00800000 즉 8388608보다 작은 경우 lsl 명령어를 통해 왼쪽으로 이동할 수 있도록 하였다.

이번 실험에서는 해당 알고리즘만 가지고 그에 따라 구현하는 것으로 이전에 C++에서 간단하게 구현하는 느낌과 동시에 수학적으로 해당 비트를 추출하는 것에 아이디어가 필요했다. 또한 Branch를 이용하여 예외의 경우 또한 처리하여 적재적소에 사용할 수 있도록 하였다.

5. Reference

1) 이형근 교수님 / 컴퓨터공학기초실험2/광운대학교 컴퓨터정보공학부/2022

2) Floating Point / https://ko.wikipedia.org/wiki/%EB%B6%80%EB%8F%99%EC%86%8C%EC%88%98%EC%A0%90