어셈블리프로그램 설계 및 실습
과제2 보고서

담당교수	이형근 교수님
실습분반	화요일 (2분반)
소속	컴퓨터정보공학부
학번·성명	2021202085 전한아솔
제출일	2024년 11월 19일

1. Problem Statement

이번 과제의 목적은 IEEE 754 부동소수점 표준의 구조와 원리를 이해하고 메모리와 레지스터의 조작을 익힌다. 이를 이용하여 부동소수점의 덧셈, 뺄셈을 어셈블리어를 이용하여 구현하는 것이 이번 과제의 목표이다.

2. Method

먼저, 부동소수점은 sign bit(1-bit), exp(지수부, 8-bits), Mantissa(가수부, 23-bits)로 구성되어, 총 32-bits로 표현된다. 먼저 DCI를 통해 레지스터에 저장한 2개의 값을 sign bit, exp, mantissa로 나누어 각 레지스터에 저장한다. 먼저 LSR 연산을 이용하여 오른쪽으로 31-bits를 밀어서 LSB에 sign bit를 저장한다. Exp는 저장된 소수를 LSL 연산을 이용하여 왼쪽으로 1 bit를 밀어서 sign bit를 제거한 후, LSR 연산을 이용해 오른쪽으로 24-bits를 밀어서 레지스터에 하위 8비트에 저장되도록 했다. Mantissa는 LSL 연산을 이용하여 왼쪽으로 8-bits를 밀어서 저장했다. 8-bits만 shift한 이유는 Mantissa의 맨 앞을 1로 만들어줘야 하기 때문에 8-bits만 LSL 연산을 수행하고 ORR 연산으로 MSB를 1로 만들었다.
이 방식으로 2개의 소수를 sign bit, exp, mantissa로 구분하여 저장한 후, 정규화를 진행한다.

정규화는 덧셈이나 뺄셈을 진행하기 위해서 지수부를 동일한 수로 맞추는 과정이다. 두 수의 지수부의 차이만큼 지수부가 작은 쪽의 exp에 더해주어 두 지수부를 같게 만들고, 지수부가 작은 쪽의 mantissa를 LSR 연산을 이용하여 두 지수부의 차이만큼 shift한다. 그리고 두 sign bit를 비교하여 덧셈, 뺄셈의 연산을 결정한다.

먼저 덧셈(두 sign bit가 같은 경우)은 먼저 한쪽의 sign bit를 저장하고 두 mantissa 값을 더한다. 이때, Carry가 발생하면 아까 정규화로 동일해진 exp에 1을 더해서 저장하고 Carry가 없다면 그대로 저장한다. Carry가 발생하지 않았다면 두 mantissa의 합 값의 MSB가 1이 되도록 해야한다. 따라서 1 bit를 LSL 연산하여 Mantissa의 맨 앞이 1이 되도록 했다.

뺄셈(두 sign bit가 다른 경우)은 정규화를 진행한 두 mantissa를 비교하여 큰 mantissa를 가진 소수의 부호를 저장하고 큰 값에서 작은 값을 뺀 mantissa를 저장한다. 여기서 만약 두 mantissa의 값이 같다면 뺄셈의 결과는 0이므로 exp와 sign bit를 모두 0으로 만든다.

그리고 결과 mantissa의 MSB를 확인하여 1이라면 LSL을 통하여 1.xx의 결과로 만들어주고 exp 값은 그대로 유지한다. 만약 0이라면 처음 bit인 0은 LSL을 이용하여 제외하고 1.xx의 결과가 될때까지 exp의 값을 1 감소하면서 LSL 연산을 진행한다.

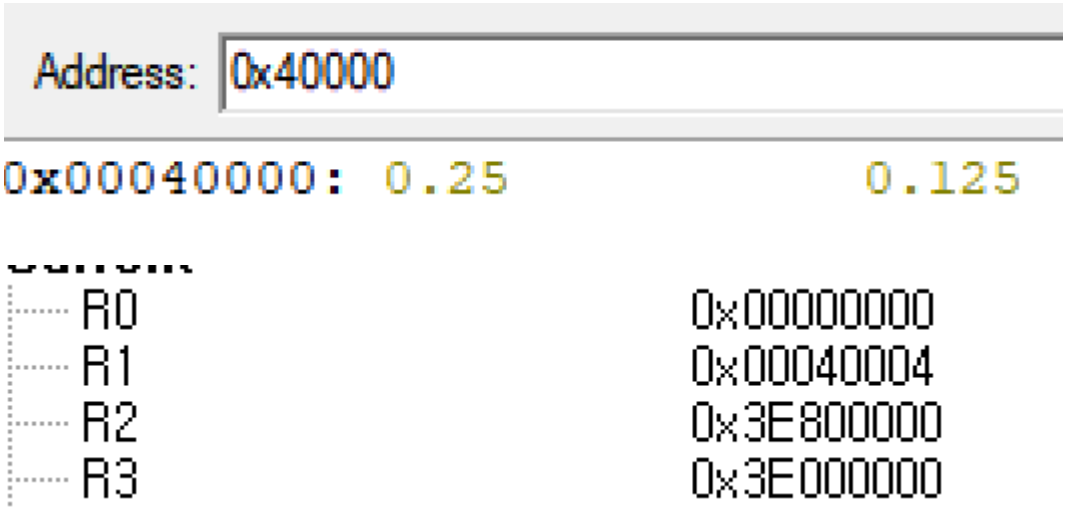
이렇게 결과 값의 sign bit, exp, mantissa가 저장되었는데, sign bit와 exp는 LSB쪽에 존재하고 mantissa는 MSB쪽에 존재한다. 따라서 하나의 값으로 저장하기 위해서 sign bit를 저장하고 LSL을 8 bits 연산한다. 그리고 ORR을 통하여 하위 8 bits에 저장된 exp값을 더해주고 LSL을 23 bits 연산하여 상위 9 bits를 sign bit와 exp로 만든다. 이제 하위 23 bits에 ORR을 통하여 mantissa를 저장하고, 0x40000000의 주소에 STR을 이용하여 결과 값을 저장했다.

3. Result

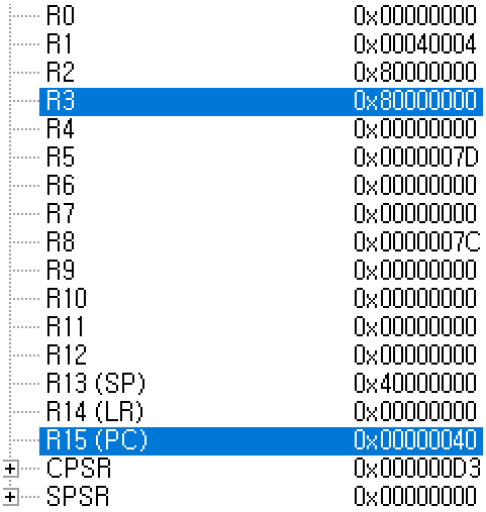
작성한 과제 코드의 예상 결과, 실행 결과 등을 figure와 함께 작성합니다. 어셈블리프로그래밍 수업에서는 muVision에 있는 debugging 기능을 활용하여 레지스터, 메모리의 값의 변화를 첨부합니다.

<Testcase A>

먼저 과제에 명시된 a 테스트케이스인 0.25(0x3E800000)+0.125(0x3E000000)를 확인하겠다.

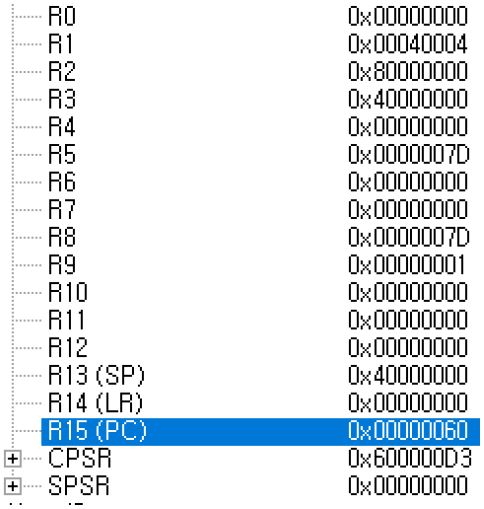


각 값은 LDR을 이용하여 r2와 r3에 저장했다.

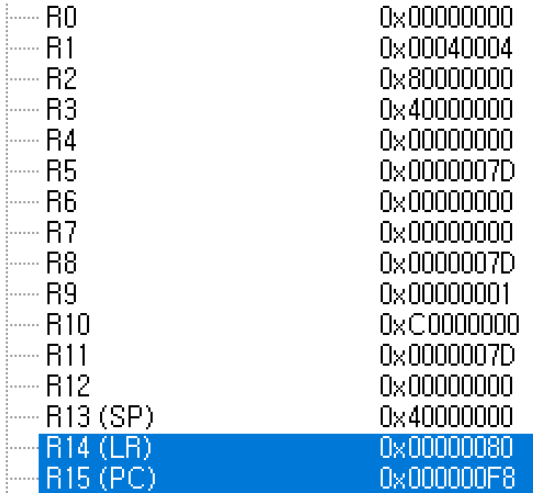


위에서 설명했던 방법으로

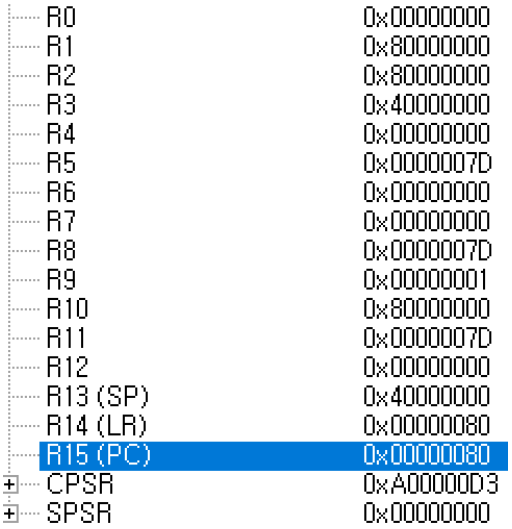
R4에는 0.25의 sign bit를, r5에는 exp, r2에 mantissa를 저장했고, r7에 0.125의 sign bit, r8에 exp, r3에 mantissa를 저장했다. 이때, 각 manstissa는 1.xx의 형태를 만들어 저장했다.



이제 두 exp를 비교하여 더 작은 exp를 가진 소수의 exp를 차이만큼 증가시키고 그 차이만큼 mantissa를 LSR 연산한 결과이다. Exp의 차이가 1이므로 r3가 8000000에서 40000000이 된 결과를 볼 수 있다. 두 sign bit는 같으므로 덧셈 연산이 진행된다.



두 mantissa를 더해서 r10에 저장한 결과, 값은 C0000000이 되었고 carry가 발생하지 않아서 exp는 r11에 그대로 7D가 저장되었다.



Carry가 발생하지 않았으므로 mantissa를 1.xx의 결과로 만들기 위해서 LSL 연산을 시행한 결과, r10은 80000000이 되었다. 이제 결과의 sign bit, exp, mantissa를 모두 저장했으므로 위에서 설명한 방법대로 레지스터에 저장하면 된다.

R0	0x00000000
R1	0x80000000
R2	0x80000000
R3	0x40000000
R4	0x00000000
R5	0x0000007D
R6	0x00000000
R7	0x00000000
R8	0x0000007D
R9	0x00000001
R10	0x00400000
R11	0x0000007D
R12	0x3EC00000
R13 (SP)	0x40000000
R14 (LR)	0x00000080
R15 (PC)	0x000000F4
CPSR	0xA00000D3
SPSR	0x00000000

R12에 3EC00000의 결과를 저장했다.

Address: 0x40000000

0x40000000:	0.375	0	0	0
0x40000010:	0	0	0	0
0x40000020:	0	0	0	0
0x40000030:	0	0	0	0
0x40000040:	0	0	0	0
0x40000050:	0	0	0	0

마지막으로 r12를 0x40000000의 메모리에 저장한 결과, 0.25+0.125의 결과인 0.375가 저장됨을 볼 수 있다.

<Testcase B>

0.5(0x3F000000)-1.75(0xBFE00000)의 케이스를 설명하겠다.

Address: 0x40000

0x00040000:	0.5	-1.75	0	0
0x00040010:	0	0	0	0
0x00040020:	0	0	0	0
0x00040030:	0	0	0	0
0x00040040:	0	0	0	0
0x00040050:	0	0	0	0

R0	0x00000000
R1	0x00040004
R2	0x80000000
R3	0xE0000000
R4	0x00000000
R5	0x0000007E
R6	0x00000000
R7	0x00000001
R8	0x0000007F
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x40000000
R14 (LR)	0x00000000
R15 (PC)	0x00000040
CPSR	0x000000D3
SPSR	0x00000000

testcase A와 동일하게 각 레지스터에 sign bit, exp를 저장하고 mantissa를 또한 1.xx의 형태로 만들어 저장했다.

R0	0x00000000
R1	0x00040004
R2	0x40000000
R3	0xE0000000
R4	0x00000000
R5	0x0000007F
R6	0x00000000
R7	0x00000001
R8	0x0000007F
R9	0x00000001
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x40000000
R14 (LR)	0x00000000
R15 (PC)	0x0000005C
CPSR	0x600000D3
SPSR	0x00000000

-1.75의 mantissa값이 1 더 컸으므로 r5의 값도 1 증가하여 7F가 되고 r2도 1 bit LSR 연산이 진행되어 80000000에서 40000000이 되었음을 볼 수 있다. 두 sign bit는 다르므로 뺄셈 연산이 수행된다.

R0	0x00000001
R1	0x00040004
R2	0x40000000
R3	0xE0000000
R4	0x00000000
R5	0x0000007F
R6	0x00000000
R7	0x00000001
R8	0x0000007F
R9	0x00000001
R10	0xA0000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x40000000
R14 (LR)	0x00000000
R15 (PC)	0x00000098
CPSR	0x000000D3
SPSR	0x00000000

-1.75의 mantissa 값에서 0.5의 mantissa 값을 뺀 결과 r10에 A0000000이 저장됨을 볼 수 있다. 또한 -1.75의 mantissa 값이 더 크므로 sign bit는 음수를 의미하는 1이 r0에 저장됨을 볼 수 있다.

R0	0x00000000
R1	0x80000000
R2	0x40000000
R3	0xE0000000
R4	0x00000000
R5	0x0000007F
R6	0x00000000
R7	0x00000001
R8	0x0000007F
R9	0x00000001
R10	0x40000000
R11	0x0000007F
R12	0x00000000
R13 (SP)	0x40000000
R14 (LR)	0x00000000
R15 (PC)	0x000000D8
CPSR	0xA00000D3
SPSR	0x00000000

결과 mantissa(이전 r10에 저장된 값)의 MSB는 1이므로, 1.xx의 형태로 만들기 위해서 1 bit만 LSR 연산한 결과, r10은 A0000000에서 40000000이 되었다.

R0	0x00000001
R1	0x80000000
R2	0x40000000
R3	0xE0000000
R4	0x00000000
R5	0x0000007F
R6	0x00000000
R7	0x00000001
R8	0x0000007F
R9	0x00000001
R10	0x00200000
R11	0x0000007F
R12	0xBFA00000
R13 (SP)	0x40000000
R14 (LR)	0x00000000
R15 (PC)	0x000000F4
CPSR	0xA00000D3
SPSR	0x00000000

```
0x40000000: -1.25           0           0           0
0x40000010: 0               0           0           0
0x40000020: 0               0           0           0
0x40000030: 0               0           0           0
0x40000040: 0               0           0           0
0x40000050: 0               0           0           0
```

이제 결과의 sign bit, exp, mantissa까지 모두 저장했으므로 위에서 설명한 방법으로 r12에 결과를 저장하고 0x40000000 메모리에 저장한 결과 0.5-1.75의 결과인 -1.25가 저장됨을 볼 수 있다.

4. Discussion and Conclusion

Mantissa를 1.xx의 형태로 만들어서 계산한다는게 어떤 의미인지 몰라서 이를 어셈블리로 계산하기 위해 실습 강의자료를 계속 읽으며 이해하려고 노력했다. 그러던 중, MSB가 1일 때, LSR 연산을 이용하여 앞에 1 을 버리면 이게 곧 1.xx의 형태를 만들어, 0.xx의 값만 사용할 수 있다는 생각으로 과제에 적용하여 이번 과제를 수행했다.

또한 덧셈 연산에서 두 mantissa의 합을 구하고 carry가 발생하지 않으면 맨 앞 bit를 LSL 했는데, 처음 각 소수의 mantissa를 1.xx의 형태로 만들고 합을 구하기 때문에 carry가 발생하지 않아도 MSB는 1이라고 생각하여 한번만 LSL을 하면 1.xx의 형태로 만들 수 있다고 생각했으나, MSB가 0인 경우가 발생하는지의 여부는 정확히 파악하지 못했다.

Reference

실습 강의자료 - 6.Floating_Point
IEEE 754 부동소수점 변환기(계산기) - hi098123 Tools
진법변환 계산기 - hi098123 Tools

명예서약서 (Honor code)

본인은 [어셈블리프로그램설계및실습] 수강에 있어 다음의 명예서약을 준수할 것을 서약합니다.

1. 기본 원칙

- 본인은 공학도로서 개인의 품위와 전문성을 지키며 행동하겠습니다.
- 본인은 안전, 건강, 공평성을 수호하는 책임감 있는 공학도가 되겠습니다.
- 본인은 우리대학의 명예를 지키고 신뢰받는 구성원이 되겠습니다.

2. 학업 윤리

- 모든 과제와 시험에서 정직하고 성실한 태도로 임하겠습니다.
- 타인의 지적 재산권을 존중하고, 표절을 하지 않겠습니다.
- 모든 과제는 처음부터 직접 작성하며, 타인의 결과물을 무단으로 사용하지 않겠습니다.
- 과제 수행 중 받은 도움이나 협력 사항을 보고서에 명시적으로 기재하겠습니다.

3. 서약 이행

본인은 위 명예서약의 모든 내용을 이해하였으며, 이를 성실히 이행할 것을 서약합니다.

학번 | 2021202085 | **이름** | 전한아솔 | **서명** |