어셈블리프로그래밍설계 및 실습 보고서

실험제목: Second_Operand_&_Multiplication /Floating_Point

실험일자: 2023년 10월 17일 (화)

제출일자: 2023년 10월 31일 (화)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 화 6,7 목 5

학 번: 2020202037

성 명: 염정호

1. 제목 및 목적 (3%)

A. 제목

Second_Operand_&_Multiplication/Floating_Point

목적

곱셈 과정에서 Multiplication operation을 사용하는 것과 Second operand를 사용한 코드를 구현해 보고 두 방식의 성능 차이를 비교해 본다. floating포인트 방식을 이용한 값을 메모리에 저장하고 해당 값을 이용한 adder를 구현해 floating포인트의 덧셈과 뺄셈 과정을 이해한다.

2. 설계 (Design) (50%)

A. Pseudo code

Mul problem1

```
LDR r1,save_result

mov r0,#0
mov r2,#1
STR r2,[r1],#4

add r2,r2,r2
STR r2,[r1],#4

mov r3,r2
add r2,r3,r2,LSL#1
STR	r2,[r1],#4

add r2,r0,r2,LSL#2
STR	r2,[r1],#4

mov r3,r2
add r2,r2,r3,LSL#2
STR r2,[r1],#4
```

```
        mov r3,r2,LSL#1
        add r2,r3,r2,LSL#2
        STR r2,[r1],#4

        mov r3,r2
        add r2,r0,r2,LSL#2
        add r2,r2,r3,LSL#1
        add r2,r2,r3
        STR r2,[r1],#4

        mov r2,r2,LSL#3
        STR r2,[r1],#4

        mov r3,r2
        add r2,r2,r3,LSL#3
        STR r2,[r1],#4

        mov r3,r2,LSL#1
        add r2,r3,r2,LSL#3

        STR r2,[r1],#4

save_result & &40000000
```

problem2

```
LDR r1,save_result
        mov r3,#1

    mov r2,#1
    STR r2,[r1],#4

    add r3,r3,#1
    MUL r2,r3,r2
    STR r2,[r1],#4
```

```
add r3,r3,#1
MUL r2,r3,r2
STR r2,[r1],#4

add r3,r3,#1
MUL r2,r3,r2
STR   r2,[r1],#4

add r3,r3,#1
MUL r2,r3,r2
STR r2,[r1],#4

add r3,r3,#1
MUL r2,r3,r2
STR r2,[r1],#4

add r3,r3,#1
MUL r2,r3,r2
STR r2,[r1],#4

add r3,r3,#1
MUL r2,r3,r2
STR r2,[r1],#4

add r3,r3,#1
MUL r2,r3,r2
STR r2,[r1],#4

add r3,r3,#1
MUL r2,r3,r2
STR r2,[r1],#4
```

Problem 3 -1

```
ldr r0,save_result
    mov r1,#17
    mov r2,#3

    mul r4,r1,r2
    str r4,[r0]

save_result & &40000000
```

Problem 3 -2

```
ldr r0,save_result
    mov r1,#17
    mov r2,#3

    mul r4,r2,r1
    str r4,[r0]

save_result & &40000000
```

Problem_float

Input r1, r2

R1, r2 부호 비교

If 같은 경우 -> 덧셈

부호 = r1 msb

Else 다른 경우 -> 뺄샘

부호 => 더 큰수

If 지수가 다른 경우

작은 지수 부분의 sign을

차이 만큼 LSR

Sign 연산

Else

Sign연산

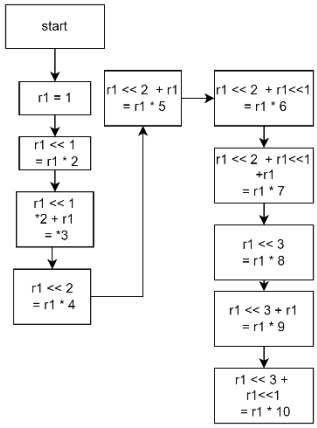
연산 후 sign 위치 재 조정

조정 한 만큼 EX에서 가산 or 감산

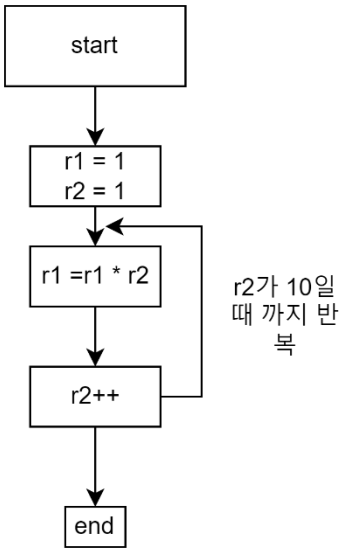
메모리 저장

B. Flow chart 작성

Mul problem1

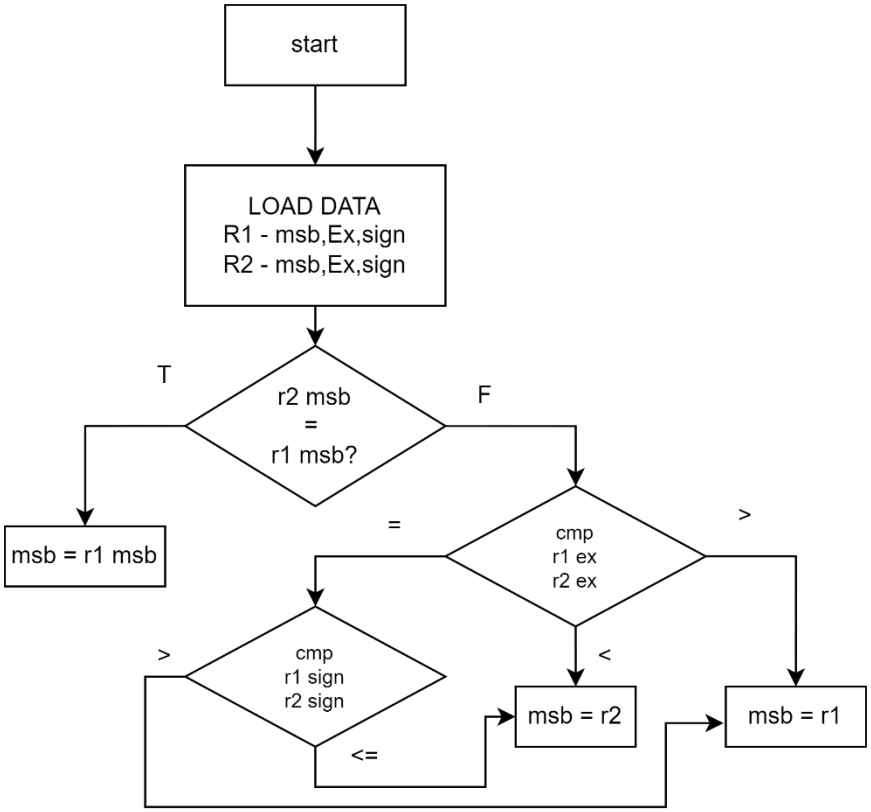


Mul problem2

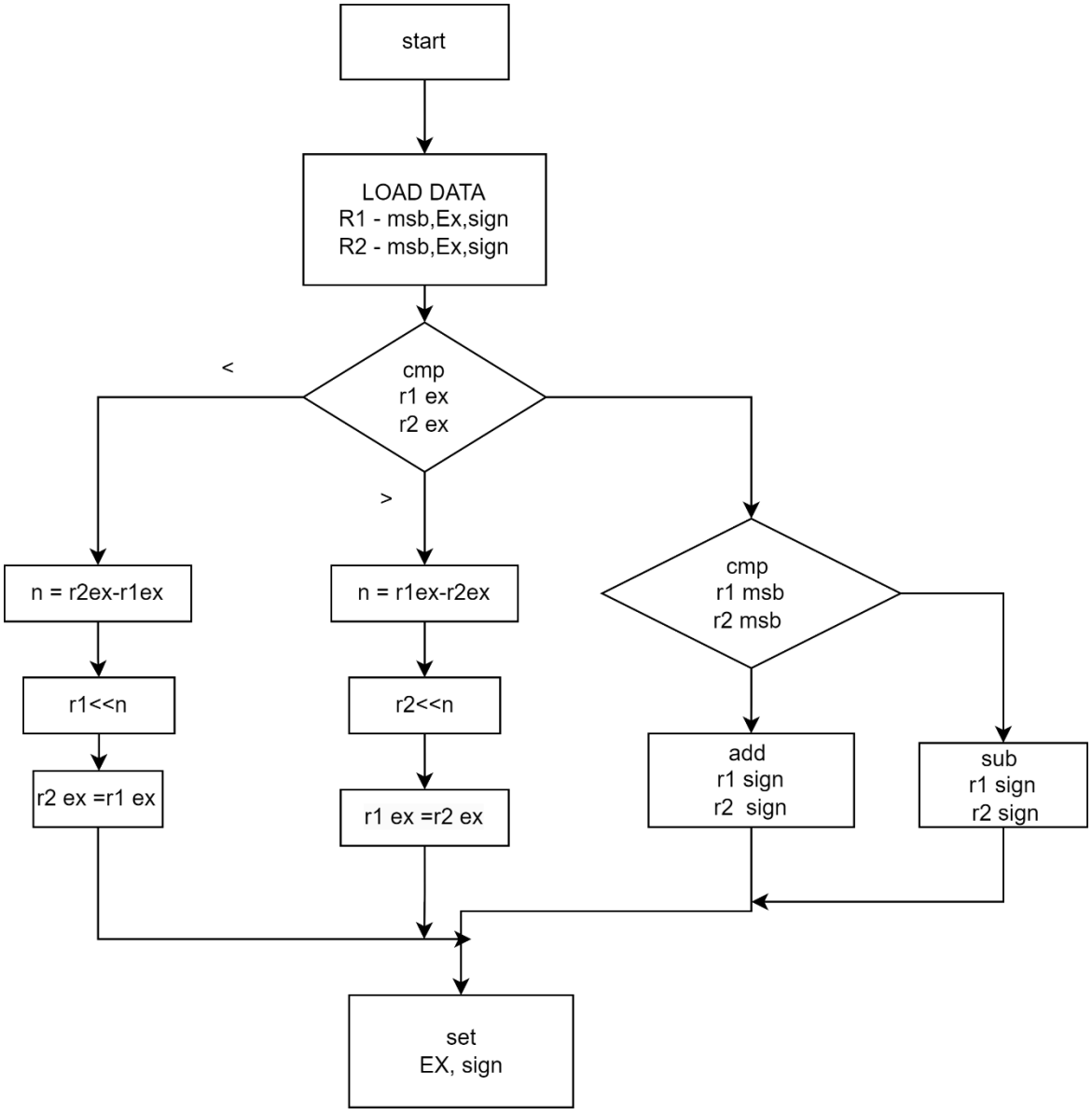


Problem_float

msb

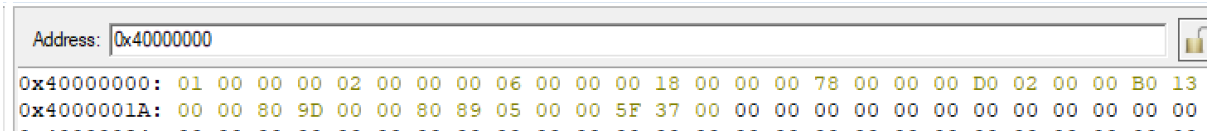

start
LOAD DATA
R1 - msb,Ex,sign
R2 - msb,Ex,sign
r2 msb
=
r1 msb?
T
F
msb = r1 msb
cmp
r1 ex
r2 ex
=
>
cmp
r1 sign
r2 sign
>
<
<=
msb = r2
msb = r1

Ex, sign

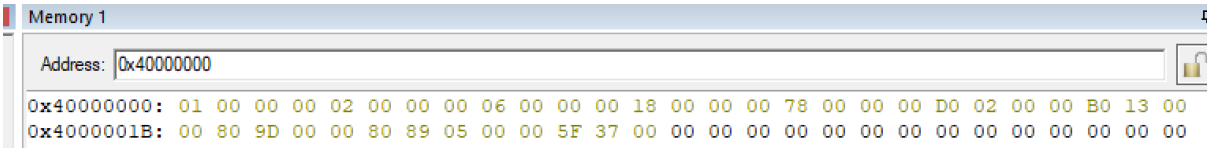


C. Result

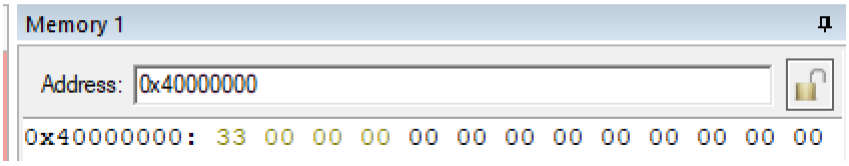
Mul problem1



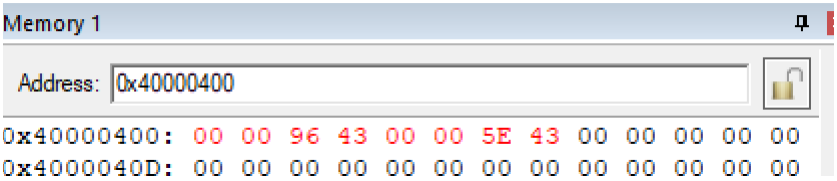
Mul problem2



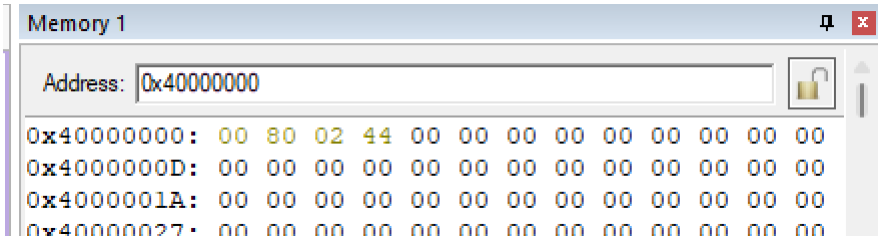
Mul problem3



Problem-float



메모리 번지에 임의로 값을 넣어줌



결과값 확인.

D. Performance

Mul problem1

State : 43

Code size : 124

Mul problem2

State : 53

Code size : 128

Problem3

17 * 3 : state 10 , code size : 24

3 * 17 : state 10 , code size : 24

Problem - float

State : 109

Code size : 316

3. 고찰 및 결론

A. 고찰 (35%)

floating포인터 구현 부분에서 문제가 많았다. 부호 비트만 하더라도 msb가 같을 때는 msb를 이어오지만 부호가 다른 경우 감산을 진행하고 해당 결과 값에 따라 부호를 변경 해주어야 했다. Significand를 계산 할 때 처음에 1을 더해주고 나중에 1을 빼지 않아 계속 이상한 값이 나왔다. 또한 계산하면서 ex가 같을 때 작거나 클 때를 비교해 주어야 해서 여러 경우의 수를 비교해서 해야 했기 때문에 구현에 어려움이 있었다. 또한 구현 과정중에 그때 그때 필요한 함수를 끼워 넣다 보니 코드 내에 중복된 함수가 많아 최적화를 한다면 state수와 size를 줄일 수 있을 것이다.

B. 결론 (10%)

곱셈구현 결과 mul명령어를 이용한 것 보다 shift와 add명령어를 이용해 구현한 경우 저장공간과, 속도 면에서 더 좋다는 것을 확인했다. 17 * 3과 3 * 17의 비교의 경우 원래 알고 있던데로 라면 17 * 3이 더 좋은 연산이지만 결과 확인 결과는 두 경우 모두 같았다. 암 어셈블리 프로그램에서 연산 결과를 특별한 방식으로 처리해 두 연산 결과의 성능을 같게 한 것 같다. 플로팅 포인터 구현을 통해 큰 수를 표현 하는 방법을 알게 되었고 여러 어셈블리 코드를 혼합해 사용하면서 어셈블리 프로그램에 익숙해 질 수 있었다.

4. 참고문헌 (2%)

이준환/어셈블리프로그래밍실습/광운대학교/2023

부동소수점연산/https://jesus-never-fail.tistory.com/20