어셈블리 프로그래밍 설계 및 실습

실험제목: Floating point

실험일자: 2020 년 10 월 19 일 (화)

제출일자: 2020 년 11 월 2 일 (월)

학　　과: 컴퓨터공학과
담당교수: 이준환 교수님
학　　번: 2019202052
성　　명: 김 호 성

1. 제목및목적
   A. 제목
      Floating point
   B. 목적
      - Floating point 의 adder 를 구현하여 floating point 의 덧셈과 뺄셈을 실행한다.
      - Floating point 의 special case 를 고려한다.

2. 설계 (Design)

A. Pseudo code

Main

```
LDR r0 ← Address1
LDR r10 ← =float1
LDR r11 ← =float2
MOV r1 ← #0x3F800000
ADD r1 = r1 + #0x40000000

LDR r2 ← [r10]
CMP r2, #0x80000000
MOVHI r3 ← #1     ;Negative Number
MOVLS r3 ← #0     ;Positive Number

LDREQ ← r2, [r11]  ;Special case N1 = -0
BEQ ENDLENE2

CMP r2, #0x00000000
LDREQ r2, [r11]      ;Special case N1 = +0
BEQ ENDLINE2

CMP r2, r1           ;Special case N1 = Positive Infinite Num
BEQ N1_is_Positive_Infinite
```

```
CMP r2, #0xFF800000
                    ;Special case N1 = Negative Infinite Num
BEQ N1_is_Negative_Infinite

MOV r2 ← r2, ROR#23     ;Rotation for Bit Clear
BIC r4 ← r2, #0xFFFFFF00 ;Bit Clear for Exponent

MOV r5 ← r2, LSR#9
        ;Shift by 9 to the right to obtain the Mantissa value
MOV r2 ← r2, ROR#9      ;Restore original Values


LDR r6 ← [r11]
CMP r6, #0x80000000
MOVH1 r7 ← #1    ;Negative Number
MOVLS r7 ← #0    ;Positive Number

LDREQ r2 ← [r10]  ;Special case N2 = -0
BEQ ENDLINE2      ;go to ENDLINE2

CMP r6, #0x00000000
LDREQ r2 ← [r10]  ;Special case N2 = +0
BEQ ENDLINE2

CMP r6, r1            ;Special case N2 = Positive Infinite Num
BEQ N2_is_Postive_Infinite

CMP r6, #0xFF800000
                    ;Special case N2 = Negative Infinite Num
BEQ N2_is_Negative_Infinite

MOV r6 ← r6, ROR#23     ;Rotation for Bit Clear
BIC r8 ← r6, #0xFFFFFF00 ;Bit Clear for Exponent
```

```
        MOV r9 ← r6 LSR#9
                ;Shift by 9 to the right to obatain the Mantissa value
        MOV r6 ← r6 ROR#9       ;Restore original values

        CMP r3, r7
        BEQ EQUAL
        BNE NOTEQUAL

EQUAL
        ADD r5 = r5 + #0x00800000
        ADD r5 = r9 + #0x00800000
                        ;Attaching 1 in front of Mantissa type

        CMP r4, r8              ;Comparing Exponent
        SUBGT r10 = r3 – r8
        SUBGT r13 = r4 - #0x0000007F

        SUBLT r10 = r8 – r4
                ;Subtract small Exponent value from large Exponent
        SUBLT r13 = r8 - #0x0000007F
                ;Subtract 127 from Exponent value

        MOVGT r11 ← r9, LSR r10
        MOVLT r11 ← r5, LSR r10
;Shift Mantissa to the right by the difference of Exponent value

        ADDGT r12 = r5 + r11
        ADDGT r12 = r9 + r11
                ;Add two Mantissa values with equal Exponent values

        CMP r12, #0x01000000            ;Normalize
        MOVHS r12 ← r12, LSR #1
```

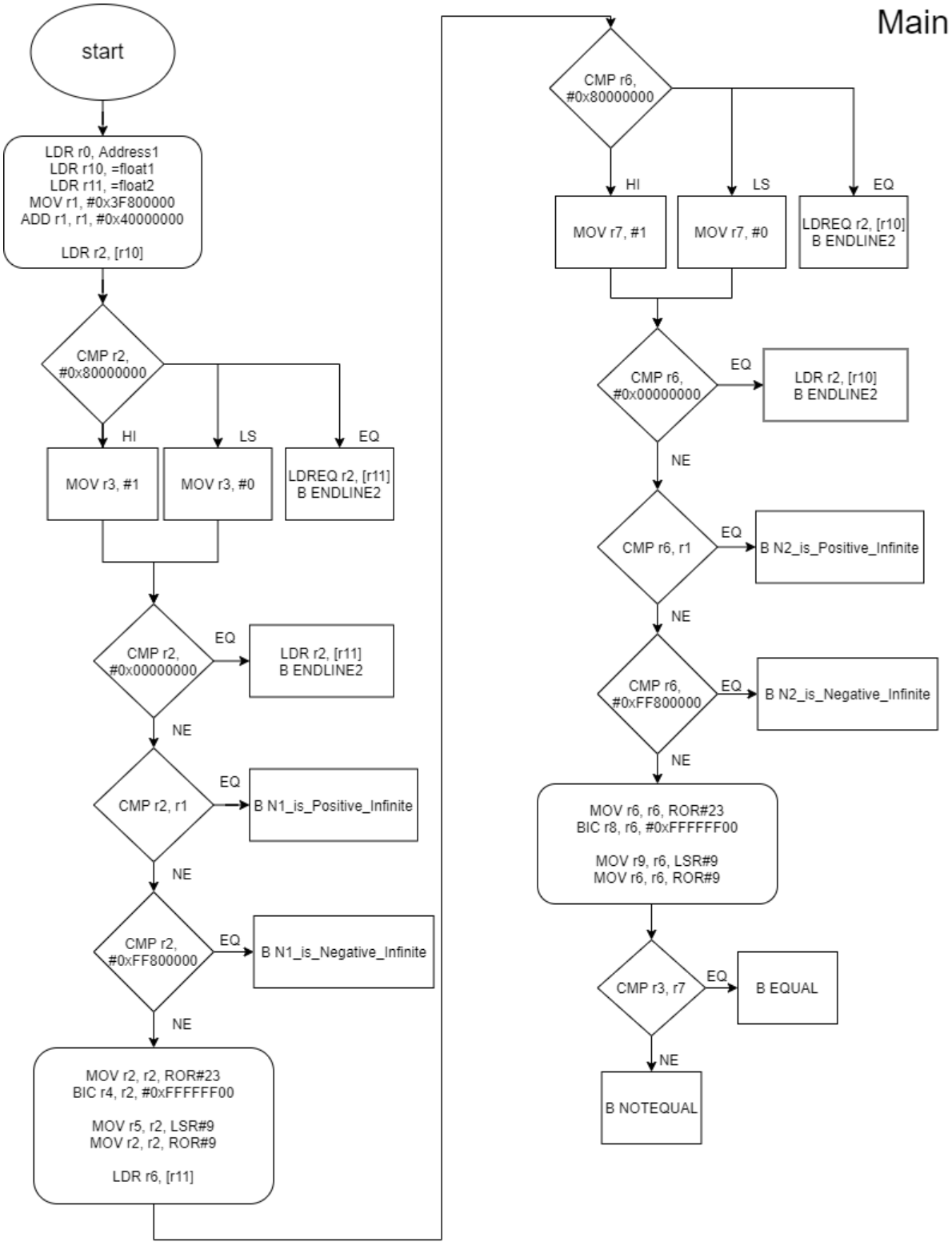
```
        ADDHS r13 ← r13, #1

        MOV r2 ← #0                ;Initialize r2
        MOV r3 ← r3, ROR #1
        ADD r2 = r2 + r3
                                   ;Addition after rotation to obtain sign bit
        MOV r12 ← r12, LSL#9
        MOV r12 ← r12, LSR#9
        ADD r2 = r2 + r12  ;Extract Mantissa value using Shift
        ADD r13 = r13 + #0x0000007F
        MOV r13 ← r13, ROR #9
        ADD r2 = r2 + r13
                        ;Add 127 to get Exponent value and extract

        B ENDLINE2

NOTEQUAL
        CMP r5, r9
                ;Compares absolute values and stores 0 or 1 in r1
        MOVHI r1 ← #1
        MOVLS r1 ← #9

        ADD r5 = r5 + #0x00800000
        ADD r9 = r9 + #0x00800000
                ;Attaching 1 in front of Mantissa type

        CMP r5, r9          ;Comparing Exponent
        CMPEQ r4, r8        ;Exponent comparison if equal
        MOVEQ r2 ← #0       ;Insert 0 to r2, if equal
        BEQ ENDLINE2        ;Go to ENDLINE2, if equal

        CMP r4, r8          ;Comparing Exponent
        SUBGT r10 = r4 – r8
```

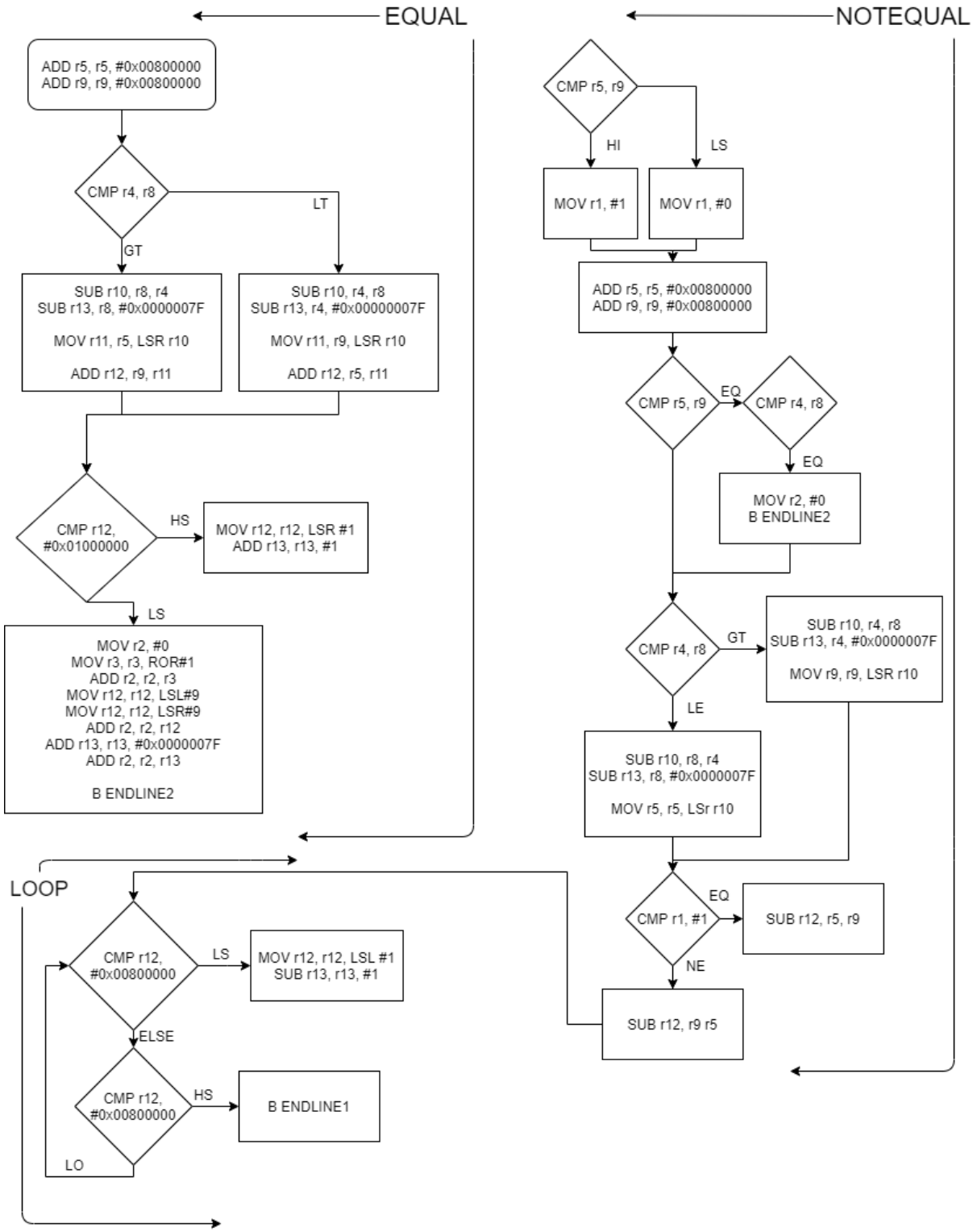
```
        SUBGT r13 = r4 - #0x0000007F
        SUBLE r10 = r8 – r4
        ;Subtract small Exponent value from large Exponent
        SUBLE r13 = r8 - #0x0000007F
        ;Subtract 127 from Exponent value

        MOVGT r9 ← r9, LSR r10
        MOVLE r5 ← r5, LSR r10
;Shift Mantissa to the right by the difference of Exponent value

        CMP r1, #1
        SUBEQ r12 = r5 – r9
        SUBNE r12 = r0 – r5
;The absolute value flag is used to subtract a small value from a
large value

LOOP
        CMP r12, #0x00800000
                ;Condition for Normalizing Mantissa value
        MOVLS r12 ← r12, LSL#1
                ;Shift the Mantissa value to the left by 1
        SUBLS r13 = r13 - #1
                ;Subtract 1 to the exponent value
        CMP r12, #0x00800000 ; r12 – 0x00800000
        BLO LOOP            ;(if r12 < #0x00800000)
        BHS ENDLINE1        ;(if r12 >= #0x00800000)

ENDLINE1
        MOV r2 ← #0         ;Initialize r2
        CMP r1, #1          ;Sign bit Checking
        MOVEQ r3 ← r3, ROR#1
        MOVNE r7 ← r7, ROR#1
        ADDEQ r2 = r2 + r3
```

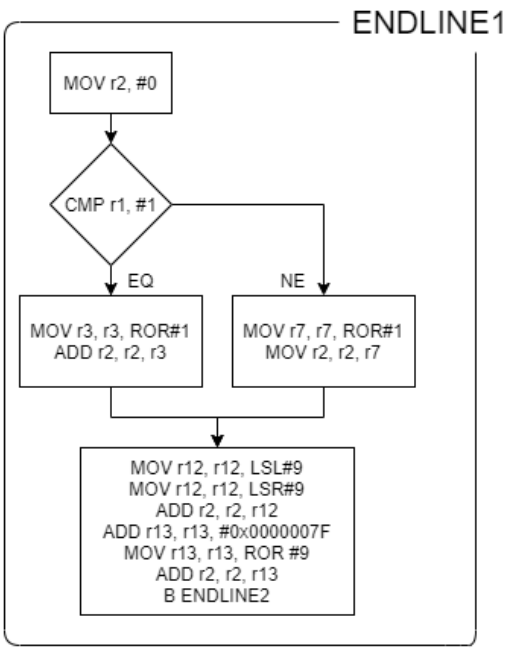
```
        ;Addition after rotation to obtain sign bit
    ADDNE r2 = r2 + r7
    MOV r12 ← r12, LSL#9
    MOV r12 ← r12, LSR#9
    ADD r2 = r2 + r12
    ADD r13 = r13 + #0x0000007F
        ;Extract Mantissa value using Shift
    MOV r13 ← r13, ROR#9
    ADD r2 = r2 + r13
    B ENDLINE2 ;Add 127 to get Exponent value and extract

N1_is_Pisitive_Infinite
    LDR r6 ← [r11]      ;Load float2
    CMP r6, #0xFF800000
            ;special case N2_is_Positive_Infinite number
    MOVEQ r2 ← #0x00000000
            ;-infinite +infinite = 0
    B ENDLINE2 ;go to ENDLINE2

N1_is_Negative_Infinite
    LDR r6 ← [r11]      ;Load float2
    CMP r6, r1          ;r1 = #0x7F800000
    MOVEQ r2 ← #0x00000000
                ;-infinite +infinite = 0
    B ENDLINE2
            ;else case: -infinite +- RealNumber = -infinite
    ;and current value of r2 = +infinite => skip other conditions.

N2_is_Positive_Infinite
    CMP r2, #0xFF800000
        ;special case N1_is_Negetive_Infinite number
    MOVEQ r2 ← #0x00000000
        ;-infinite +infinite = 0
```

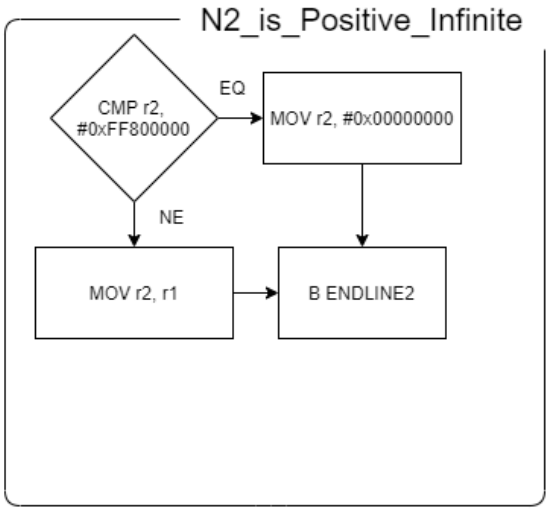
```
        MOVNE r2 ← r1
                ;else case: +-Real number + infinite = +infinite
        B ENDLINE2 ;go to ENDLINE2

N2_is_Negative_Infinite
        CMP r2, r1
                ;special case N1_is_Posititive_Infinite number
        MOV EQ ← r2, #0x00000000
                ;+infinite +(-infinite) = 0
        MOVNE r2 ← r6
                ;else case: +-Real number +(-infinite) = -infinite
                ;skip B ENDLINE2 (No need)

ENDLINE2
        STR r2 → [r0]
                ;Store r2 value in the memory address value of r0
        MOV pc, lr

float1 & 0x3FC00000
float2 & 0x40500000
Address1 DCD &40000
        END
```

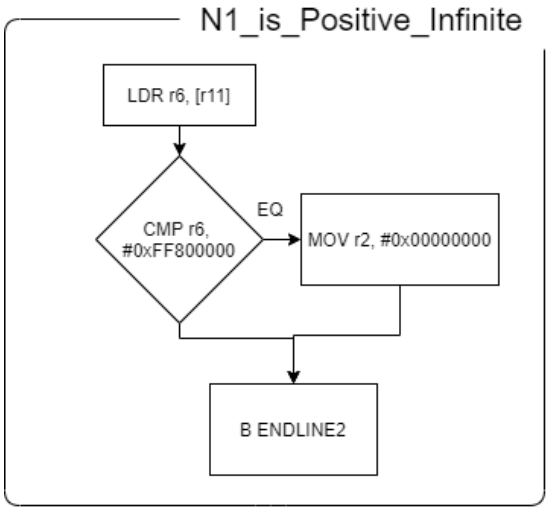
B. Flow chart 작성

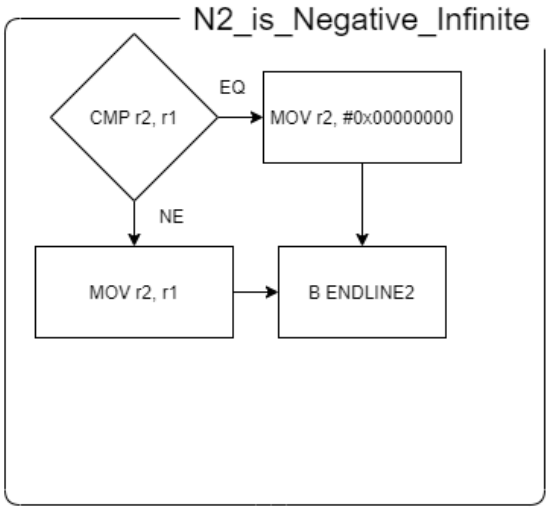


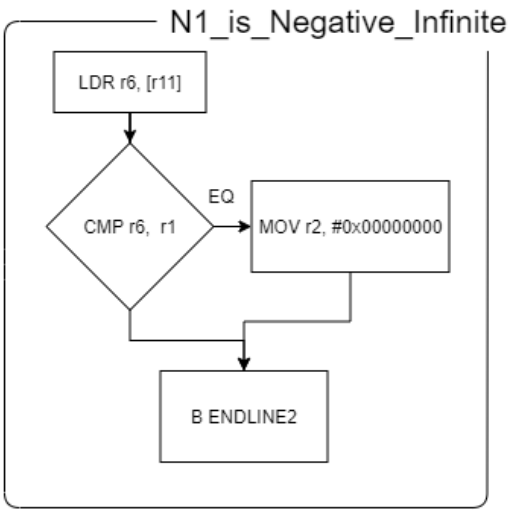

EQUAL
NOTEQUAL
ADD r5, r5, #0x00800000
ADD r9, r9, #0x00800000
CMP r4, r8
LT
GT
SUB r10, r8, r4
SUB r13, r8, #0x0000007F
MOV r11, r5, LSR r10
ADD r12, r9, r11
SUB r10, r4, r8
SUB r13, r4, #0x00000007F
MOV r11, r9, LSR r10
ADD r12, r5, r11
CMP r12, #0x01000000
HS
MOV r12, r12, LSR #1
ADD r13, r13, #1
LS
MOV r2, #0
MOV r3, r3, ROR#1
ADD r2, r2, r3
MOV r12, r12, LSL#9
MOV r12, r12, LSR#9
ADD r2, r2, r12
ADD r13, r13, #0x0000007F
ADD r2, r2, r13
B ENDLINE2
CMP r5, r9
HI
LS
MOV r1, #1
MOV r1, #0
ADD r5, r5, #0x00800000
ADD r9, r9, #0x00800000
CMP r5, r9
EQ
CMP r4, r8
EQ
MOV r2, #0
B ENDLINE2
CMP r4, r8
GT
SUB r10, r4, r8
SUB r13, r4, #0x0000007F
MOV r9, r9, LSR r10
LE
SUB r10, r8, r4
SUB r13, r8, #0x0000007F
MOV r5, r5, LSr r10
LOOP
CMP r12, #0x00800000
LS
MOV r12, r12, LSL #1
SUB r13, r13, #1
ELSE
CMP r12, #0x00800000
HS
B ENDLINE1
LO
CMP r1, #1
EQ
SUB r12, r5, r9
NE
SUB r12, r9 r5

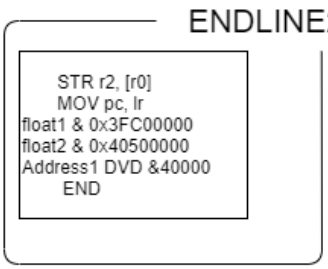

ENDLINE1
MOV r2, #0
CMP r1, #1
EQ
NE
MOV r3, r3, ROR#1
ADD r2, r2, r3
MOV r7, r7, ROR#1
MOV r2, r2, r7
MOV r12, r12, LSL#9
MOV r12, r12, LSR#9
ADD r2, r2, r12
ADD r13, r13, #0x0000007F
MOV r13, r13, ROR #9
ADD r2, r2, r13
B ENDLINE2


N2_is_Positive_Infinite
CMP r2, #0xFF800000
EQ
MOV r2, #0x00000000
NE
MOV r2, r1
B ENDLINE2

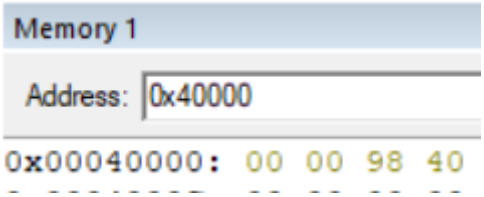

N1_is_Positive_Infinite
LDR r6, [r11]
CMP r6, #0xFF800000
EQ
MOV r2, #0x00000000
B ENDLINE2


N2_is_Negative_Infinite
CMP r2, r1
EQ
MOV r2, #0x00000000
NE
MOV r2, r1
B ENDLINE2


N1_is_Negative_Infinite
LDR r6, [r11]
CMP r6, r1
EQ
MOV r2, #0x00000000
B ENDLINE2


ENDLINE2
STR r2, [r0]
MOV pc, lr
float1 & 0x3FC00000
float2 & 0x40500000
Address1 DVD &40000
END

C. Result



Little endian 기준이므로 0x04980000 이라는 값이 나옴.
(이론 값과 일치)

D. Performance

Performance = Code Size * States
= 500 * 82
= 41000

3. 고찰및결론

A. 고찰

양의 무한대의 16 진수 값: 0x7F800000
음의 무한대의 16 진수 값: 0xFF800000
+0 의 16 진수 값: 0x00000000
-0 의 16 진수 값: 0x80000000
NaN(무리수)의 경우 문제의 조건에서 메모리에서 불러오는 값은 실수라고 가정했기 때문에 제외하고 코드를 구현했다.

예외 처리의 경우로는 다음과 같다.
1. 0 + Real Number
2. Real Number + 0
3. 0 +0
4. 양의 무한대 + Real Number(음의 무한대 제외)
5. 양의 무한대 + 음의 무한대
6. 음의 무한대 + Real Number(양의 무한대 제외)
7. 음의 무한대 + 양의 무한대
8. Real Number(음의 무한대 제외) + 양의 무한대
9. 음의 무한대 + 양의 무한대
10. Real Number(양의 무한대 제외) + 음의 무한대
11. 양의 무한대 + 음의 무한대

1. = N2 의 값을 저장하게 한다.
2. = N1 의 값을 저장하게 한다.
3. = N2 의 값을 저장하게 한다. (예외처리 1 번에서 같이 걸리기 때문이다.)
4. 양의 무한대 값을 저장한다.
5. 0 을 저장한다. (차수가 같은 무한대를 합할 때는 0 이라고 가정한다.)
6. 음의 무한대 값을 저장한다.

7. 0 을 저장한다. (5 번과 이유가 같다.)
8. 양의 무한대 값을 저장한다.
9. 0 을 저장한다. (5 번과 이유가 같다.)
10. 음의 무한대 값을 저장한다.
11. 0 을 저장한다. (5 번과 이유가 같다.)
사실 11 번과 9 번의 경우 각각 7 번과 9 번에서 걸리긴 하지만, 코드의 이해를 쉽게 하기위해 일부러 넣은 경향이 있다. (성능이 떨어지기는 하더라도, 배우는 단계에서 이해하기 쉽게 하기 위함이다.)

또한, 예외처리에서 걸리지 않은(0, 양의 무한대, 음의 무한대를 제외한 실수) 값들은 다음과 같은 과정을 거쳐 값을 연산하게 된다.
1. 두 수의 Sign Bit 비교
2. 두 수에 1 씩 더함(mantissa 앞에 + 1)
3. exponent 비교 후 맞춰 줌.
4. 두 수의 mantissa 값을 더함.
5. 더한 값을 일반화
6. 위의 과정에서 구한 Sign, Exponent, Mentissa 값을 초기화한 r2 에 저장.
만약, Sign bit 가 같지 않다면, NotEqual label 로 넘어가 Sign bit 지정 이후 과정 같음.

B. 결론

- 0, 양의 무한대, 음의 무한대의 값이 들어왔을 경우 예외처리를 실행한다.
- Floating Point 기법을 사용하여 실수의 덧셈 및 뺄셈을 구현할 수 있다.

4. 참고문헌

이준환교수님/디지털논리회로 2/광운대학교(컴퓨터정보공학부)/2020

공영호교수님/디지털논리회로 2/광운대학교(컴퓨터정보공학부)/2020