어셈블리 프로그래밍 설계 및 실습

보고서

실험제목: Floating-Point

실험일자: 2018년 10월 25일 (목)

제출일자: 2018년 11월 01일 (목)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 화 5, 수 6,7

학 번: 2015722025

성 명: 정용훈

1. 제목 및 목적

A. 제목

Floating-Point

B. 목적

지금까지 배운 명령어로는 정수의 합이 가능한 ADD를 사용했다. 이번 실습을 통하여 소수의 값을 계산하는 ADD 함수를 구현한다. 또한 IEEE에서 표준화한 Floating Point의 구조를 이해하며 계산한다.

2. 설계 (Design)

A. Pseudo code

The same command

(1) R0에 최종적으로 값을 저장할 메모리 할당, r1에 float1값, r2에 float2값 r10에 mantissa에 저장할 1값 저장

(2) float값을 비교하여 값이 같으면 CMPNUM함수로 이동, 다른 경우 값의 exponent와 mantissa의 값을 비교, 두 값이 같으면 Savevalue함수로 이동

(3) 레지스터 r3와 r4에 각 float값의 sign비트 저장

(4) 레지스터 r5와 r6에 각 float값의 mantissa 저장

(5) 레지스터 r7와 r8에 각 float값의 exponent의 값 저장

(6) mantissa값에 1을 더함

(7) 두 값의 exponent값을 비교하여 큰 값에서 작은 값을 뺌, 결과를 r9에 저장

(8) exponent값을 비교하여 큰 값의 mantissa값을 r9만큼 오른쪽으로 shift함

(9) exponent값을 비교하여 큰 값을 레지스터 r13에 저장

(10) sing bit를 비교하여 같으면 EQADD함수, 다르면 NEADD함수로 이동

When sign bit is equal (EQADD)

(1) 두 mantissa의 값을 더함, 비교를 위하여 r10에 2*r10의 값을 저장

(2) 더한 mantissa r5와 r10을 비교하여 r5가 더 크면 오른쪽으로 한번 shift

(3) 위에서 비교한 값이 크면 r13에 1을 더함 (normalize를 위한 연산)

(4) Finish 함수로 이동

When sign bit is different (NEADD)

(1) 두 mantissa값을 비교하여 큰 값에서 작은 값을 빼 r5에 저장해줌

(2) Float값 r1, r2를 비교하여 큰 값의 sign bit 값을 r3에 저장(공통 연산에서 왼쪽으로 1 shift했으므로 절대값에 대한 비교임)

(3) R9에 0을 저장

(4) R10d의 값과 r5의 값을 비교하여 r5가 크면 r5를 왼쪽으로 1 shift하고 r9에 1을 더함, (r5의 값이 r10의 값과 같거나 작을 때 까지 (4)항목을 반복)

(5) 레지스터 r13에 기존 r13에서 normalize한 count(r9)만큼의 값을 뺌

(6) Finish 함수로 이동

Finish

(1) R10에 mantissa주소에 저장된 값을 load

(2) R5에 r5에서 r10의 값을 뺌 (mantissa에 1이 더해져 있으므로 빼 준다.)

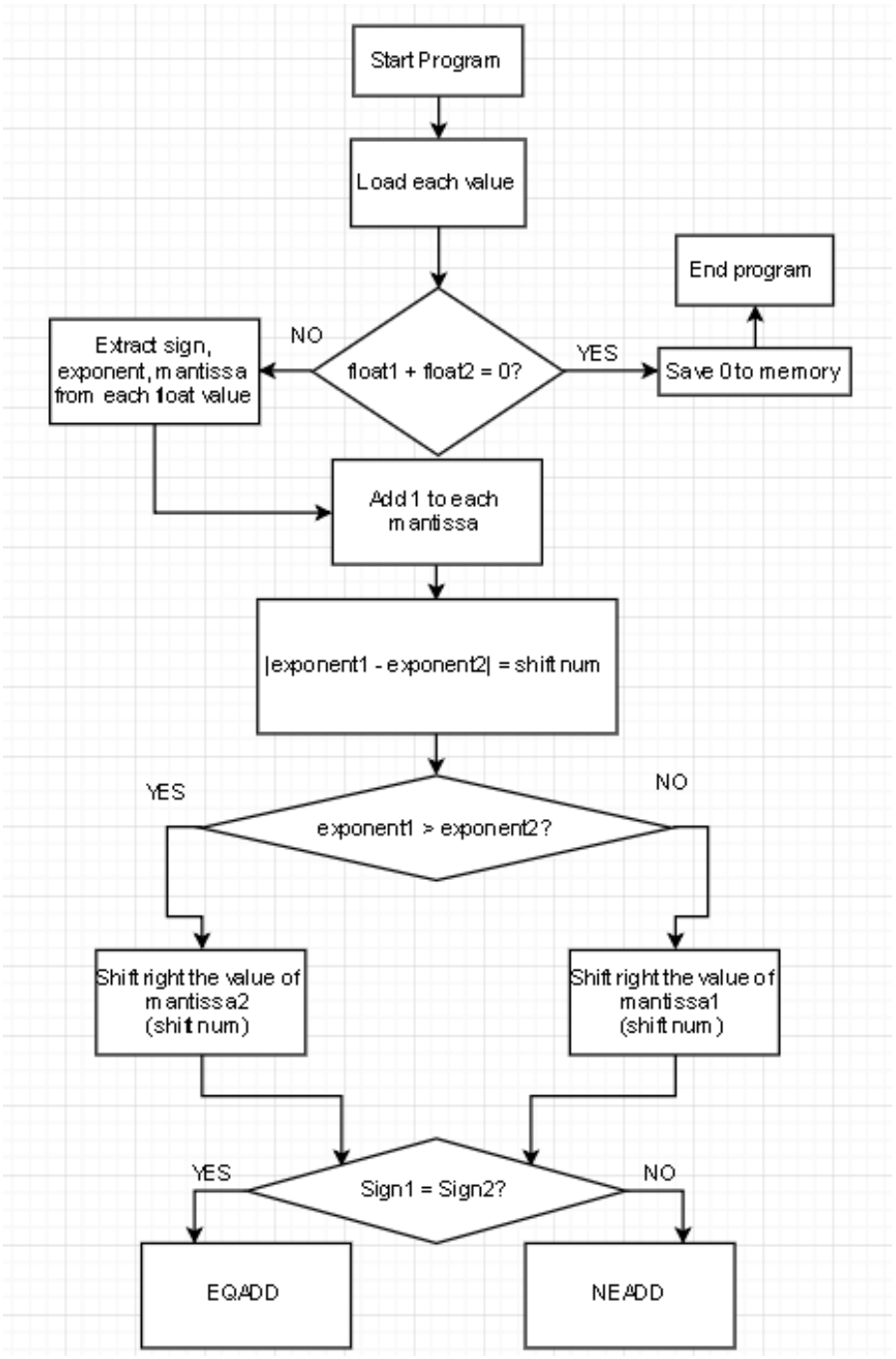
(3) R3의 값을 왼쪽으로 31만큼 shift (sign bit)

(4) R13의 값을 왼쪽으로 23만큼 shift (exponent)

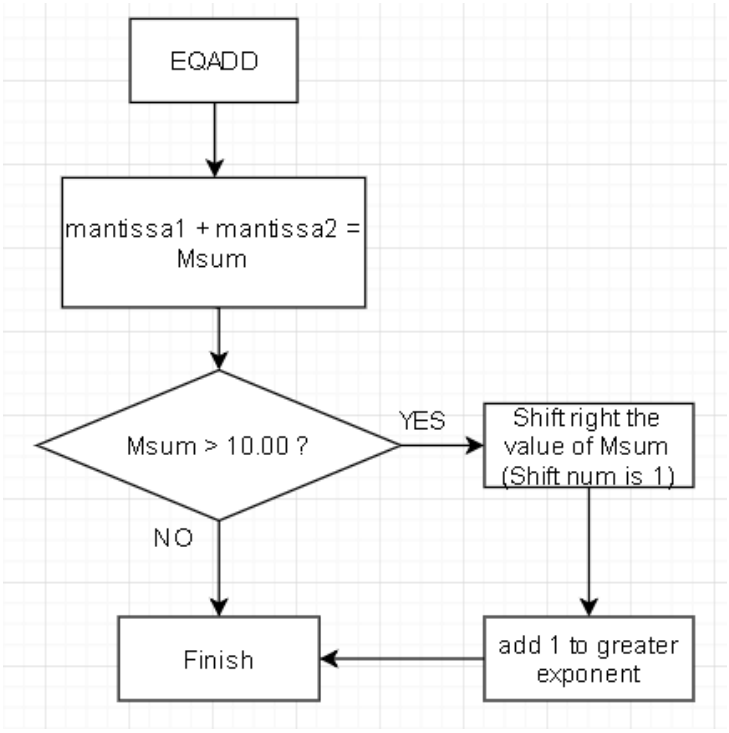
(5) R11에 r1, r13, r5의 값을 더함

(6) R11의 값을 메모리에 저장 (r0)

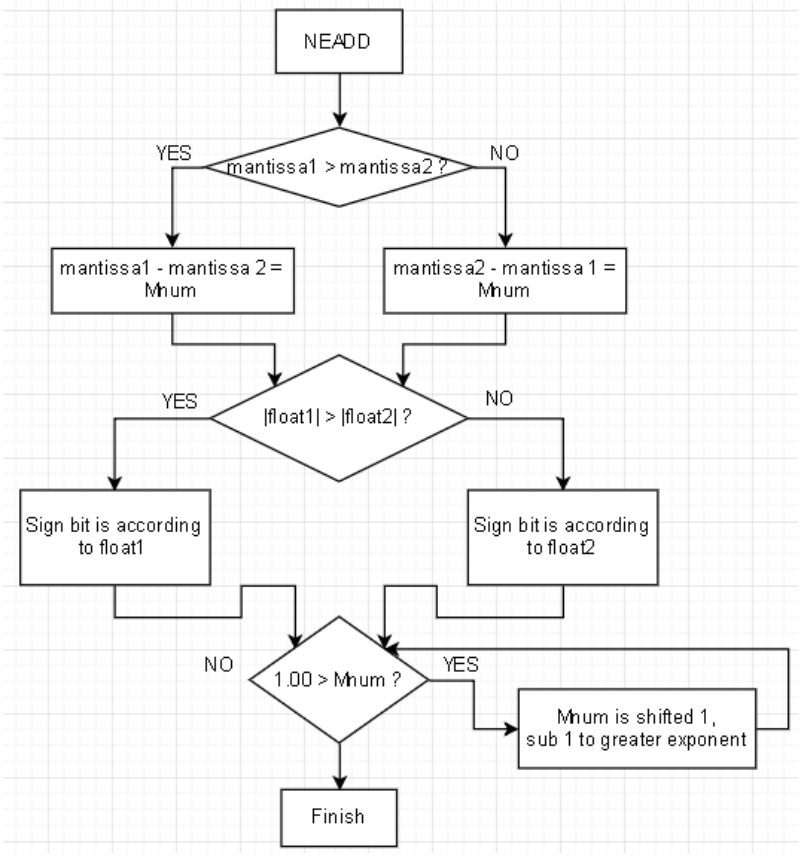
B. Flow chart 작성



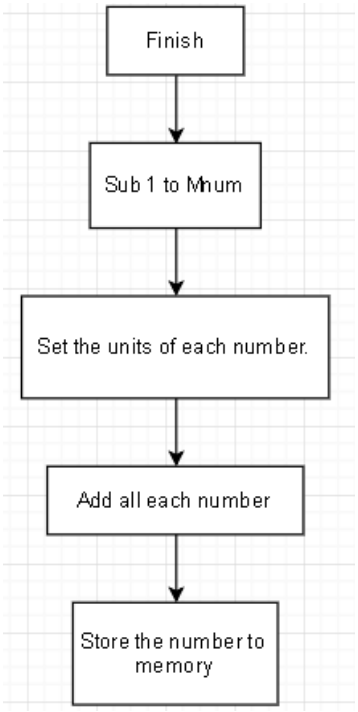
EQADD


EQADD
mantissa1 + mantissa2 = Msum
Msum > 10.00 ?
YES
Shift right the value of Msum (Shift num is 1)
NO
Finish
add 1 to greater exponent

NEADD


NEADD
YES
mantissa1 > mantissa2 ?
NO
mantissa1 - mantissa 2 = Mnum
mantissa2 - mantissa 1 = Mnum
YES
|float1| > |float2| ?
NO
Sign bit is according to float1
Sign bit is according to float2
NO
1.00 > Mnum ?
YES
Mnum is shifted 1, sub 1 to greater exponent
Finish

Finish


Finish
Sub 1 to Mnum
Set the units of each number.
Add all each number
Store the number to memory

C.Result

When sign is equal

```
0x00040000: 1.5             3.25
```

해당 값은 float1과 float2의 값을 확인하는 과정입니다. 값 1.5와 3.25를 더하는 것으로 각 float값의 계산에 필요한 요소를 뽑으면 아래와 같습니다.

```
R3          0x00000000
R4          0x00000000
R5          0x00400000
R6          0x00500000
R7          0x0000007F
R8          0x00000080
```

해당 레지스터의 값은 두 수의 sign bit, mantissa, exponent의 값을 차례데로 넣어 놓은 값입니다. 해당 값을 바탕으로 연산을 진행하면 최종적인 값이 도출됩니다.

```
0x00040000: 4.75            0
```

1.5 + 3.25로 값이 정상적으로 계산되는 것을 확인 할 수 있습니다.

When sign is different

```
0x00040000: -58             3.25
```

float의 값

```
R3          0x00000001
R4          0x00000000
R5          0x00680000
R6          0x00500000
R7          0x00000084
R8          0x00000080
```

```
0x00040000: -54.75          0
```

최종 값

When the sum is zero

```
0x00040000: -58             58
```

float의 값

(Sign bit의 값이 다르고 mantissa의 값과 exponent의 값이 같은 경우 계산 과정을 거치지 않고 메모리에 0을 저장할 수 있도록 예외처리)

```
0x00040000: 0               0
```

최종 값

3. 고찰 및 결론

A. 고찰

이번 과제를 진행하면서 다른 과제에 비해 특히 시간이 많이 걸리게 되었습니다. IEEE에서 표준화한 Floating Point에 대하여 처음 접해 봤기 때문에 이해하고 계산하는 시간이 좀 걸렸으며 sign값과 mantissa, exponent를 각각 구하는 방법을 구상하고 이해하는데 시간이 걸렸습니다. 실습을 진행하는데 있어 결과가 계속 이상하게 나오는 문제가 발생하였고 해결방법으로는 프로그램을 돌리는 동시에 직접 계산하며 순차적으로 문제를 집으며 넘어갔습니다. 문제로는 비교적 도출 방법이 까다로운 exponent의 normalize에서 문제가 발생하였습니다. Normalize를 한 수만큼 값을 더해야 하는데 shift연산을 하며 값이 다르게 나온 것이 문제였습니다. 또한 절대 값을 비교하는 방법에 있어 sign bit까지 포함하여 생각하면서 코드사이즈가 비교적 크게 나오는 문제가 있었습니다. 표준을 보았을 때 two's complement로 생각하는 것이 아니라 sign bit는 온전히 sign만 뜻하기 때문에 exponent값과 mantissa 값만 비교하면 절대값으로 값을 도출할 수 있었습니다. 마지막으로는 결과값 0에 대한 예외 처리입니다. 결과 값을 비교하기 위하여 sign bit의 값이 다른 절대값이 같은 수를 넣어 결과를 보면 0의 값이 아니라 해당하는 절대값의 값이 나오는 게 확인되었습니다. 해당 문제는 sign bit가 다르고 exponent와 mantissa의 값이 같은 경우 메모리에 0이 저장될 수 있도록 처리하며 해결하게 되었습니다.

B. 결론

High level에서는 아무렇지 않게 쓰던 소수의 계산을 직접 구현하는 계기가 되는 실험이었습니다. 컴퓨터가 계산하는 값들이 단순히 알고 있던 덧셈 뺄셈으로 이루어지는 것이 아니라 표준을 정하여 그 표준에 따라 이진수로 이루어진 수로 덧셈과 뺄셈이 이루어지는 것을 실습하며 수 체계에 대하여 보다 이해하며 컴퓨터의 계산 동작이 어떤 식으로 이루어지는지 더 쉽게 이해할 수 있는 계기가 되었습니다. IEEE에 대한 표준을 바탕으로 특정 계산을 하는 실습은 이번이 처음이었습니다. 이런 이유로 결과 값을 도출하는데 실수도 많이 하고 이해가 안가는 부분이 많았지만 후에는 오히려 이런 표준이 이해가 쉽고 편리하다는 것을 알게 되었습니다. 앞으로 수체계에 대한 표준이 나와도 해당 실습을 바탕으로 보다 쉽게 사용할 수 있을 것이라고 예상됩니다. 또한 해당 실습 내용을 응용하면 앞으로 프로그램을 구현하는데 있어 좀더 폭 넓은 수를 표현 할 수 있다고 예상됩니다.

4. 참고문헌

이준환/Floating point number & addition/광운대학교/2018