어셈블리 설계 및 실습 프로젝트 제안서

Bilinear Interpolation

학　　과: 컴퓨터공학과

담당교수: 이준환 교수님

학　　번: 2015722025

성　　명: 정용훈

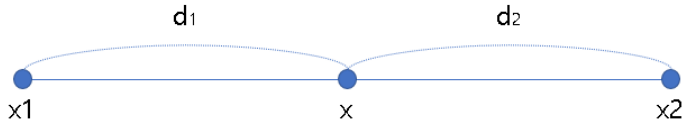
제안서 목차

1. 과제 목표

해당 과제의 목표는 임의의 부동소수점 데이터로 이루어진 20X20 정방행렬에 대한 Bilinear Interpolation을 수행할 수 있는 어셈블리 코드를 작성하는 것이 목표입니다. 특히 단순 구현만이 아니라 state와 code size를 신경 쓰면서 최대한 performance가 좋은 프로그램을 작성하는 것이 중요하며 프로젝트의 평가도 프로그램의 동작은 물론 performance에 맞춰져 있기 때문에 지금까지 실습해왔던 개념들을 응용하여 최대한 performance가 좋은 프로그램을 설계하는데 목적을 두고 있습니다.

2. Project관련 기본 용어

Project제목으로 쓰인 Bilinear Interpolation이란 "2중 선형 보간 법"이라 불리며 사각형의 네 끝점의 값만 알고 있을 경우, 사각형 내부의 임의의 위치에서의 값을 알아내는데 사용되는 계산 방법을 뜻한다. 2중 선형 보간 법은 선형 보간 법을 2중으로 사용하여 계산하는 방법이므로, 이해를 위해서는 먼저 선형 보간 법을 알아야한다. 선형 보간 법의 경우는 단순히 직선상의 두 점 사이의 임의 위치에서의 값을 계산해내는 것이라면, 2중 선형 보간 법은 평면을 이루는 네 꼭지점안에 있는 임의의 위치에서의 값을 계산해내는 방법이다.

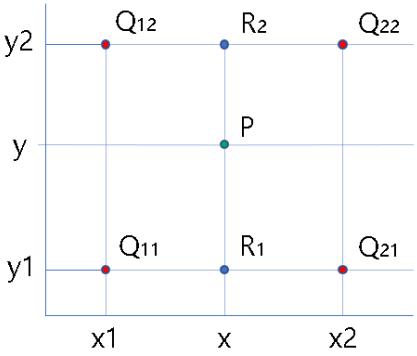


$$f(x) = Bf(x1) + a\ f(x2)$$

$$a = \frac{d1}{d1 + d2} \qquad a + B = 1.0$$

$$B = \frac{d2}{d1 + d2}$$

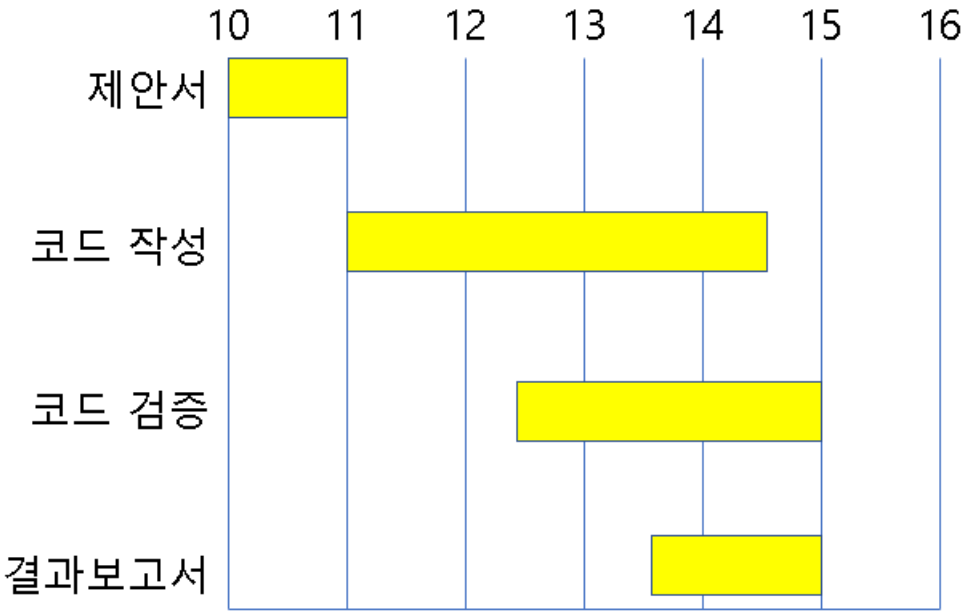
위 그림은 선형 보간 법을 수식으로 나타낸 식이다. 이를 응용하여 2중으로 사용한 것이 2중 선형 보간 법이다.



$$f(x,y) \approx \frac{f(Q11)}{(x2-x1)(y2-y1)}(x2 - x)(y2 - y)$$
$$+\frac{f(Q21)}{(x2-x1)(y2-y1)}(x - x1)(y2 - y)$$
$$+\frac{f(Q12)}{(x2-x1)(y2-y1)}(x2 - x)(y - y1)$$
$$+\frac{f(Q22)}{(x2-x1)(y2-y1)}(x - x1)(y - y1)$$

해당 이미지는 2중 선형 보간 법을 수식으로 나타낸 것이다. 해당 수식을 바탕으로 프로그램 설계를 하면 된다.

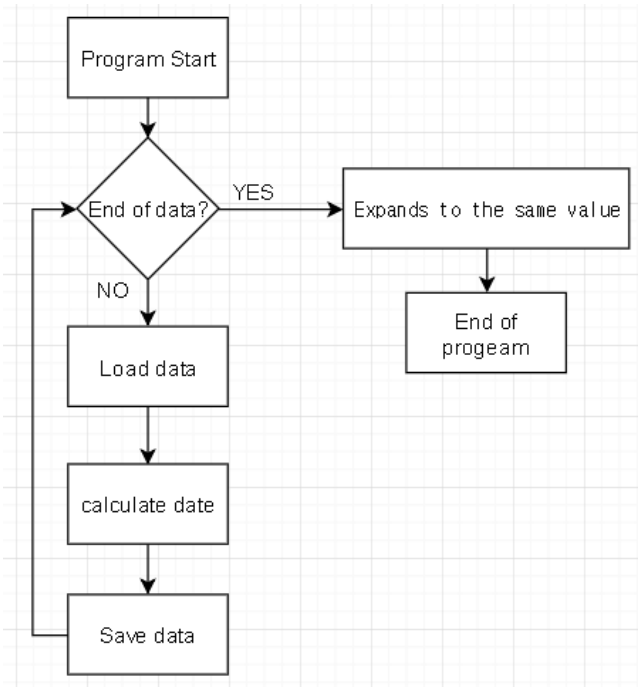
3. 일정



프로그램 설계 일정은 디지털 논리회로 Project와 동일하며 제출기간이 비슷하기 때문에 하루 일정을 잘 조정하여 시간을 분배할 계획이다. 코드작성에 기간 비중을 높게 배치한 이유는 Performance를 최대한 높이기 위한 배치이며 그만큼 코드 검증 또한 빠르게 시작하여 제출기간 전까지 프로그램을 설계할 계획이다.

4. 각 function 별 알고리즘

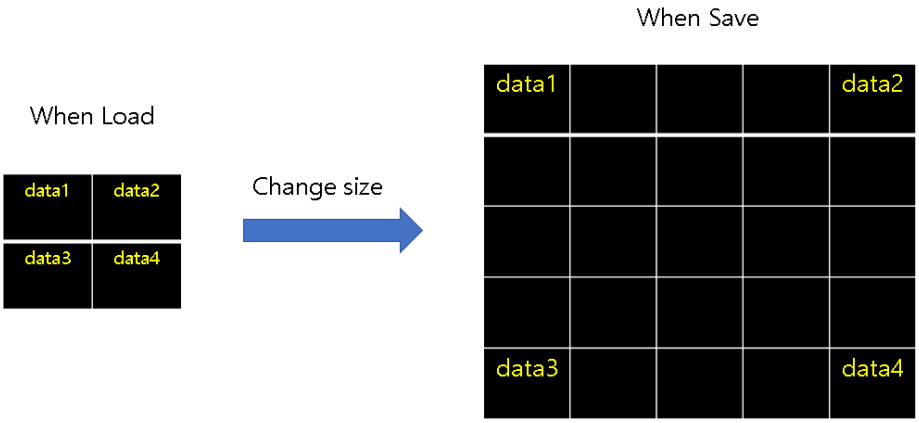
해당 프로젝트에서 필요한 함수로는 순차적으로 데이터를 받아오고, 계산이 된 값을 저장할 수 있는 함수와, 받아온 데이터를 계산하는 함수로 실질적으로 필요한 큰 기능은 두가지가 전부라고 생각됩니다. 다음 그림은 전체적인 알고리즘을 Flow chart로 나타낸 것입니다.



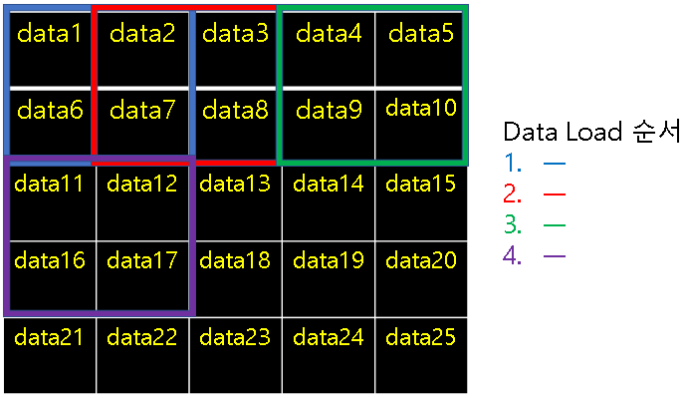
주의할 점으로는 데이터를 계산하는 과정이 굉장히 세부적인 함수로 나뉠 것으로 예상됩니다. 아래 함수는 Project를 진행하는데 사용하게 될 함수를 나열한 것입니다.

(1) Load data & Save data

Load할때의 메모리 사이즈와 Save를 실행할 때의 메모리 사이즈가 차이가 나기 때문에 Load와 Save Function을 잘 구분해주어야 합니다. 그렇지 않으면 엉뚱한 데이터가 들어갈 수 있고 데이터의 계산이 정확하지 않을 수 있습니다. 해당 함수를 검증하는 방법으로는 데이터를 Load하고 나서의 데이터 개수와 계산이 다 되고 데이터가 저장된 개수를 확인하면 쉽게 해결할 수 있습니다.



데이터를 불러오고 저장하는 알고리즘은 다음과 같습니다.



다음 그림과 같은 순서로 데이터를 불러오게 되며 Save를 하게 될 때는 첫번째 그림과 같이 확장된 상태로 들어가게 됩니다. 해당 작업은 데이터가 끝날 때까지 반복되며 마지막 데이터는 끝 값을 가지고 확장되며 계산이 끝나게 됩니다. 해당 function은 아래 항목에서 다루겠습니다.

(2) Padding

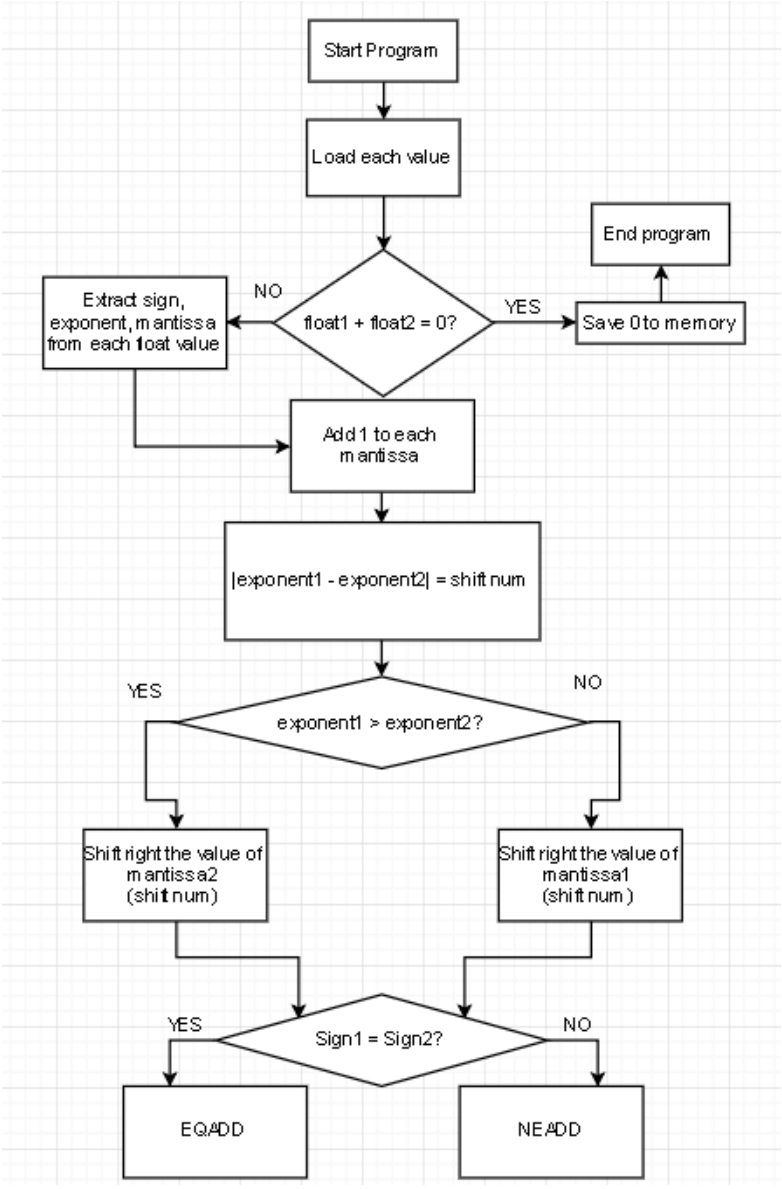
예외처리로 데이터가 끝났을 때 끝 값을 바탕으로 한번 확장하며 프로그램을 종료하게 되어있습니다. Data가 끝나는 것을 Flag로 삼아 flag가 반응하면 저장된 끝 값을 아래 표와 같이 확장하여 프로그램을 종료하면 됩니다.

2	2.5	3	3.5	4	4	4	4
3	3.5	4	4.5	5	5	5	5
4	4.5	5	5.5	6	6	6	6
5	5.5	6	6.6	7	7	7	7
6	6.5	7	7.5	8	8	8	8
6	6.5	7	7.5	8	8	8	8
6	6.5	7	7.5	8	8	8	8
6	6.5	7	7.5	8	8	8	8

메모리를 확장하는 것 자체는 data를 Save할 때 실행시켜주면 되며 해당하는 padding함수는 예외처리로 한번 실행될 함수입니다.

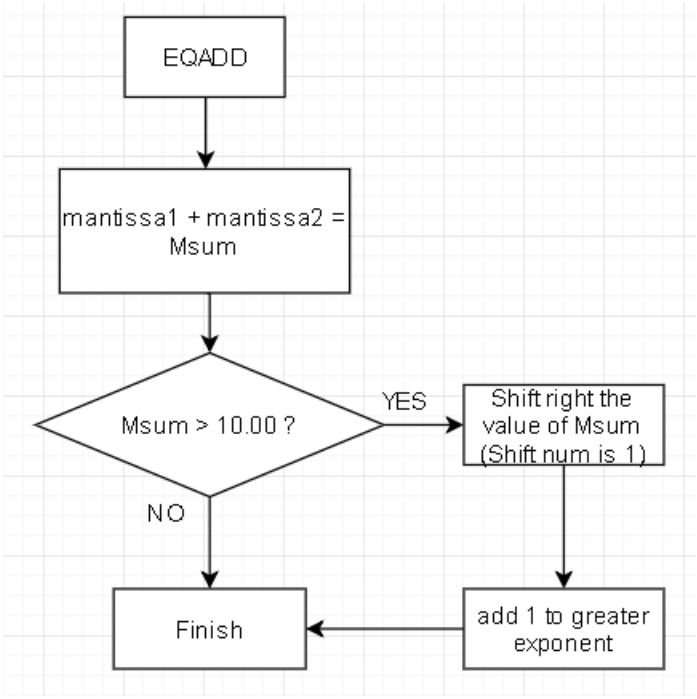
(3) Addition and Subtraction of floating point

다음은 계산을 위한 floating point의 addition과 subtraction 계산입니다. 프로젝트를 진행하며 필수적으로 필요한 계산 방법입니다. 실습을 통해 구현한적이 있으며 해당 계산의 알고리즘은 다음 Flow chart를 따르게 됩니다. 특히 뺄셈을 하게 될 때는 알맞은 값의 부호를 바꾸고 해당 알고리즘을 실행 합니다.

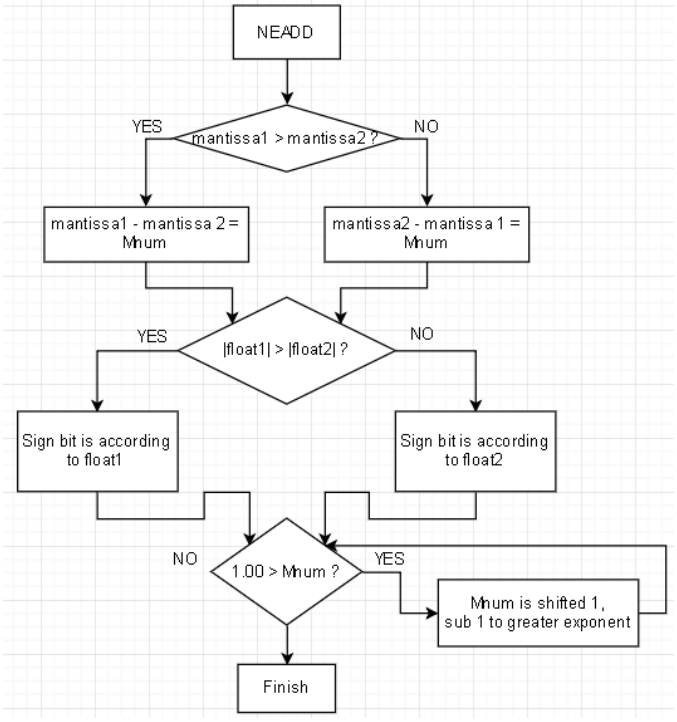


덧셈과 뺄셈의 공통적인 부분이며 값의 크기와 부호를 비교하여 덧셈 연산을 실행할지 뺄셈 연산을 실행할지 결정합니다.

EQADD



NEADD



이렇게 계산을 완료한 최종적인 값을 레지스터에 저장 해놓고 다음 연산을 실행하면 됩니다. 다음은 Floating Point의 곱셈 연산입니다.

(4) Multiplication of floating point

Floating point의 대한 곱셈 함수입니다. 곱셈함수는 normalize과정이 없기 때문에 loop가 필요한 알고리즘이 없어서 Flow chart는 생략하게 되었습니다. 제공된 PDF를 따라 다음 과정을 진행하면 구할 수 있습니다.

1) Sign bit, Exponent bits, Mantissa bits를 추출합니다.

2) 다음으로 Mantissa형태에 1을 붙여줍니다.

3) 각각 두개의 Value를 다음과 같은 형태로 맞춰줍니다.

$$\text{Value}= (-1)^{S} * (1.\text{mantissa}) * 2^{exponent-127}$$

4) 두개의 Value를 추출했다면 Result의 값은 아래 그림과 같습니다.

$$\text{Result}= (-1)^{S1\text{^}S2} * (1.\text{mantissa1}) * (1.\text{mantissa2}) * 2^{exponent1-127+exponent2-127}$$

곱셈의 대한 알고리즘은 해당 항목을 따라가면 값을 도출할 수 있습니다.

5. 예상되는 문제점

계산의 대한 문제와 performance와 code size에 대한 문제가 많이 발생할 것 같습니다. 우선 실습을 한 Floating point에 대한 add와 subtract연산을 구현하는데 있어 code size가 길어지면서 중간 code해석과 register의 중복 사용으로 많은 실수가 있었습니다. 이번 project또한 사용할 함수가 많고 loop가 많을 것으로 예상되기 때문에 register의 사용을 체계적으로 하여 code를 구현하지 않으면 조그만 실수가 많을 것으로 예상됩니다. 또한 보간법에 대하여 어느정도 숙지하고 이해하는 상태지만 코드로 구현하는데 있어 고급언어보다는 계산과정이 복잡하고 많이 생각해야 하기 때문에 이 부분에 있어서도 많은 시간이 소요될 것으로 생각됩니다. 데이터의 불러옴과 저장 그리고 계산의 문제가 해결된다면 반복적으로 같은 함수를 실행시켜 과제의 목적을 달성할 수 있을 것이라고 예상되지만 project를 설계하는데 있어 중점으로 생각해야할 부분이 performance입니다. Performance란 프로그램의 효율성을 따져볼 수 있는 지표로 쓰이는데 Branch를 사용하면 performance가 더 좋을 수 있지만 아닌 경우도 존재하고 단순히 unrolling 방법을 통해서 code를 작성하면 효율이 좋은 경우가 많습니다.
Performance를 고려하여 프로그램을 작성하려면 평소 사용하지 않았던 명령어와 프로그램이 구동하며 중복되는 부분을 없애 주며 state를 줄일 수 있는데 state를 위해 같은 동작을 할 수 있는 다른 명령어도 공부해야 하기 때문에 이에 대한 어려움도 있을 것으로 예상됩니다. 마지막으로 전체적인 코드구현과 계산은 어렵지 크게 문제가 되지 않을 것 같지만 state와 code size가 project를 완성하는데 어려움이 있고 중점 항목이 될 것 같습니다.

6. 검증 전략

Project에 대한 검증 전략은 함수에 대해 part별로 나눠 part별로 검증 후 함수를 연결하여 정확히 돌아가는지 확인하는 것이 가장 효율적이라고 생각됩니다. 우선 가장 먼저 검증할 항목으로는 계산할 data를 주소에서 불러오는 과정이 정확하게 되는지 검증하며 전체적인 계산이 되기 위해 부분적으로 되는 각각의 작은 계산들을 각각 구현하여 그때마다 검증합니다. 검증하는 계산으로는 부동소수의 곱셈과 덧셈이 있으며 또 계산에 대한 예외처리를 검증해야 하며 각각의 계산이 정확히 맞는다면 데이터를 저장하는 부분으로 넘어갑니다. Data의 개수가 많기 때문에 결과적으로 나오는 계산은 data의 개수가 매우 방대하기 때문에 일일이 확인하기 어려울 수 있습니다. 그렇기에 개발자가 임의로 축소된 데이터를 몇 가지 만들어 각각의 case별로 프로그램에 돌려 검증이 완료된다면 제공받은 data도 잘 돌아가는 것으로 예상되기 때문에 이렇게 검증을 끝내면 됩니다. 이렇게 전체적인 프로그램의 설계가 검증을 통해 끝난다면 performance를 좋게 만들기 위하여 code를 깔끔하게 만들며 중복된 부분을 최대한 줄이는 작업을 실행해야 합니다. 이에 대한 검증 방법으로는 같은 동작을 하는 함수나 명령어를 다른 방식으로 작성하여 직접 프로그램을 돌려 state와 code size를 보며 개발자가 직접 눈으로 확인하며 검증하는 방법이 가장 좋다고 생각되며 performance에 대한 검증은 project가 끝날 때까지 진행할 것으로 예상됩니다.