어셈블리프로그램설계및실습 보고서

과제 주차: Project

학 과: 컴퓨터정보공학부

담당교수: 이형근 교수님

실습분반: 화 67, 수 5

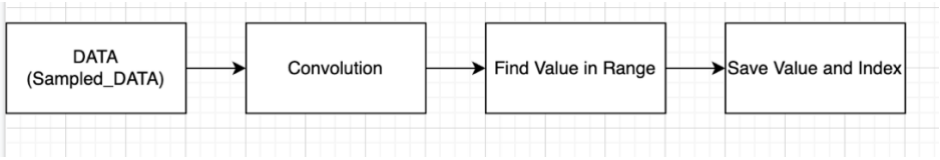
학 번: 2019202021

성 명: 정 성 엽

제 출 일: 2022.12.04(일)

1. Introduction

이번 프로젝트는 임의의 부동 소수점으로 이루어진 이산 신호(discrete signal)와 부동 소수점으로 이루어져 있는 필터와의 합성곱(convolution)을 진행하고 특정 결과를 메모리에 저장하는 코드를 작성한다. 이 프로젝트의 전체 동작은 임의의 부동소수점으로 이루어진 이산 신호와 필터의 합성 곱을 수행한 결과 중 특정 범위 내에 포함되는 값만 추출하여 해당 결과와 결과의 index값을 저장한다. 다만 결과와 index를 같이 저장하는 것이 아닌 결과를 먼저 나열하고 후에 index를 따로 나열한다.



2. Background

A. FLOATING POINT

해당 과제를 이해하기 위해서는 Floating point number에 대한 이해가 필요하다.

IEEE 754 표준 floating point의 표현 방법은 1bit는 sign bit, 다음 8bit는 exponent로 승수를 표현, 나머지 23bit는 mantissa로 표현하여 승이 곱해질 소수점 부분을 가리킨다. 이는 32bit 기준으로 했을 때의 범위이며 64bit로도 가능하다.

1bit	8bit	23bit
(Sign)	(exponent)	mantissa

위의 표를 참고하여 각 자리수를 고려하여 식으로 나타내면 아래와 같이 식으로 표현할 수 있다.

FLOAT NUMBER=(-1)^Sign×2^(exponent-127)×(1.mantissa)

B. Floating point의 덧셈 뺄셈 연산

Floating point의 덧셈 또는 뺄셈 연산에서 뺄셈은 덧셈 연산에서 뒤의 수의 부호를 바꾸는 방법을 사용하여 결국 덧셈 연산만을 이용해서도 뺄셈이 가능하다. 만약 두 수의 부호가 같은 경우 그대로 덧셈을 진행하고 다른 경우 절대값이 더 큰 부호를 따라 간다.

i. 두 Floating point에서 Sign, Exponent, Mantissa bit를 추출한다. 해당 bit는 앞서 설명한 표를 참고하여 각 1bit, 8bit, 23bit로 나눈다.

1. Exponent bit는 둘 중 큰 것을 따로 다른 register에 저장한다
2. mantissa bit에서는 자릿수를 유지하기 위해 처음 bit에 1을 더한다.

ii. 두 Exponent 값의 차의 절대값을 구하고 이 차만큼 shift num 할 수 있도록 저장한다.

1. shift num이 0인 경우에는 그대로 넘어간다.

2. 0이 아니면 exponent 값이 작은 곳의 mantissa를 shift num에 저장된 만큼 LSR(right shift)를 진행한다.

iii. sign 비트만을 추출했던 것을 비교한다.

1. 같은 경우 그대로 mantissa 값을 더한다. 그리고 그 sign bit를 저장한다.

2. 다른 경우 두 mantissa의 차의 절대값을 구하고 더 큰 절대값을 가지고 있는 값의 sign을 가져와 저장한다.

iv. mantissa 값의 normalize 를 진행한다.

1. 조건 식에 따라 Mantissa의 자릿수를 shift 연산을 통해 이동하여 맞춘다.

2. 자릿수를 맞춰가며 최종 Exponent 값을 수정한다.

v. 구한 sign bit, Exponent bit, mantissa bit를 각 자리 수에 맞게 더하여 floating point add 연산을 마무리하고, 값이 0인 경우에 대한 검사를 반드시 진행한다.

C. Floating point의 곱셈 연산

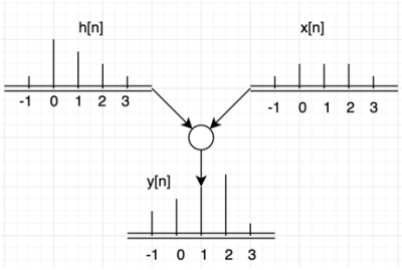
$$Value1 = (-1)^{S1} * (1.mantissa1) * 2^{exponent1-127}$$
$$Value2 = (-1)^{S2} * (1.mantissa1) * 2^{exponent2-127}$$
$$Result = (-1)^{S1+S2} * (1.mantissa1) * (1.mantissa2) * 2^{exponent1+exponent2-127}$$

곱셈 연산 같은 경우 각 sign bit를 더하여 해당 지수로 부호를 나누고, mantissa값에 각 1.0을 더하여 서로 곱하여 값을 다른 값과 곱하며, exponent는 각 자릿수를 나타내므로 더한 후 127을 빼서 결과를 가져온다.

Exponent가 127을 빼는 이유를 더 설명하자면 기존의 exponent는 (S1+127)와 같은 형태를 가지고 있기 때문에 $(S1 + 127) + (S2 + 127) = S1 + S2 + 254$이므로 127만큼 빼서 exponent 형식으로 바꾸는 것이다.

Mantissa 의 곱은 십의 자리 곱과 같은 원리로 작동하지만 이번 과제에서 register는 32bit만 저장가능한데 24bit를 각각 곱하면 최대 48bit의 이진수가 나오므로 어셈블리 명령어 중 UMULL이란 명령어를 사용하여 곱의 결과를 두개의 Register에 나눠서 입력받고 상위 값부터 23개를 mantissa값으로 다시 가져오도록 한다.

C. 이산 합성곱

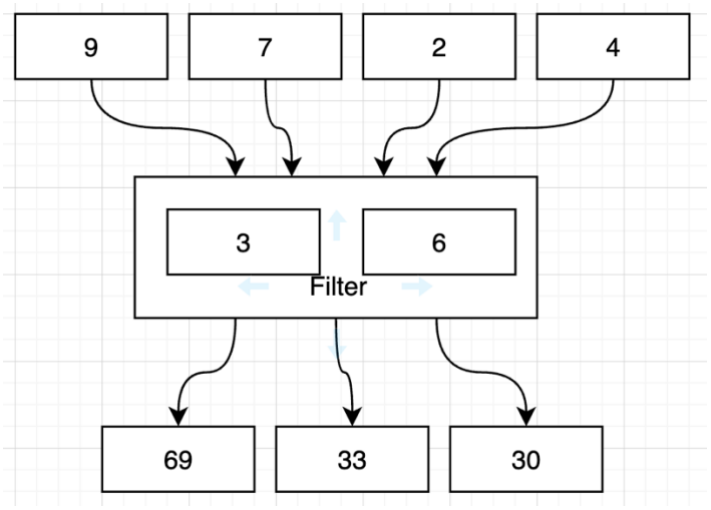


이번 과제에서는 이산 합성곱을 진행하는데 이를 수식으로 나타내면 아래 식과 같다.

$$y[n] = \sum_{k=0}^{M} h[k]x[n-k]$$

해당 수식에서 x는 연속된 데이터를 의미하고 y는 결과 값을, n은 index, h는 filter, 마지막으로 M은 filter의 크기를 의미한다.

간단한 예시를 그림으로 나타내면



같고 필터 개수만큼 위의 임의의 숫자와 곱한 후 더하여 출력한다.

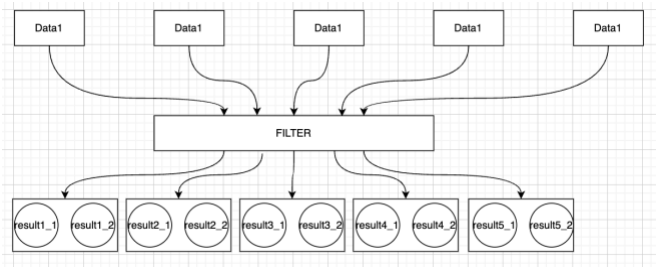
$$9 * 3 + 7 * 6 = 69$$
$$7 * 3 + 2 * 6 = 33$$
$$2 * 3 + 4 * 6 = 30$$

의 연산이 결과로 나오는 것이다.

3. Algorithm

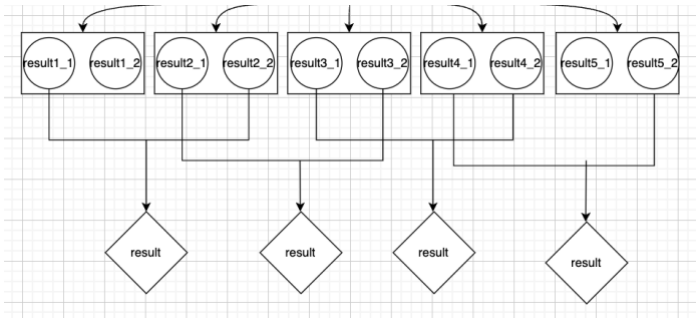
값을 하나 가져오고 필터를 거쳐 다른 값을 다시 가져오고 반복하여 바로바로 result에 저장하는 알고리즘을 구현하려고 하였으나 이를 불가능하게 한 점은 index값을 후에 따로 모두 저장하는 것이었다. 이를 고려하여 먼저 각 숫자와 모든 필터와의 숫자의 곱을 임의의 메모리 TEMPADDR1& &40000000에 저장하고 이 저장한 값을 기반으로 다시 합을 진행하여 TEMPADDR2& &80000000에 저장하였다. 후에 이 저장된 것을 바탕으로 -4.7보다 작은 경우 이때서야 0xF0000000단의 메모리에 저장하고 후에 한번 더 반복하여 index값만 따로 가져오도록 진행하였다.

i. 모든 부동 소수점과 필터 값의 곱 저장



위 그림과 같이 TEMPADDR1에 값이 나열 되어있다. 위의 이미지는 작은 값일 경우 예시이다.

ii. 저장된 값으로 합성 진행



이제 해당 순서처럼 합을 진행하여 합성곱 값을 저장한다. 현재 위의 이미지는 필터가 2개일 때 예시여서 합을 1번만 진행하였지만 6개의 필터인 경우 5번의 합을 진행하고 양 끝의 합은 하지 않는 것을 볼 수 있다.

iii. -4.7 보다 작은 경우

1. 먼저 나온 합성곱의 sign bit가 0인 경우 양수 이므로 값을 저장하지 않고 1인 경우 다음 조건으로 넘어간다.
2. 둘 다 음수일 때 exponent의 값이 더 큰 경우 더 작은 값을 가지므로 각 exponent값을 비교하여 더 큰 경우 바로 값을 저장하고 작은 경우 다음 합성곱 값을 비교한다. 같은 경우 다음 조건으로 넘어간다.
3. 위의 조건을 모두 통과한 경우 마지막으로 mantissa의 값을 비교하여 mantissa의 값이 더 작으면 비교하는 값보다 작으므로 다음 합성곱 값을 비교하도록 넘어가고 아닌 경우 저장한다.

이는 결과값을 저장할 때 그리고 index값을 저장할 때 사용한다.

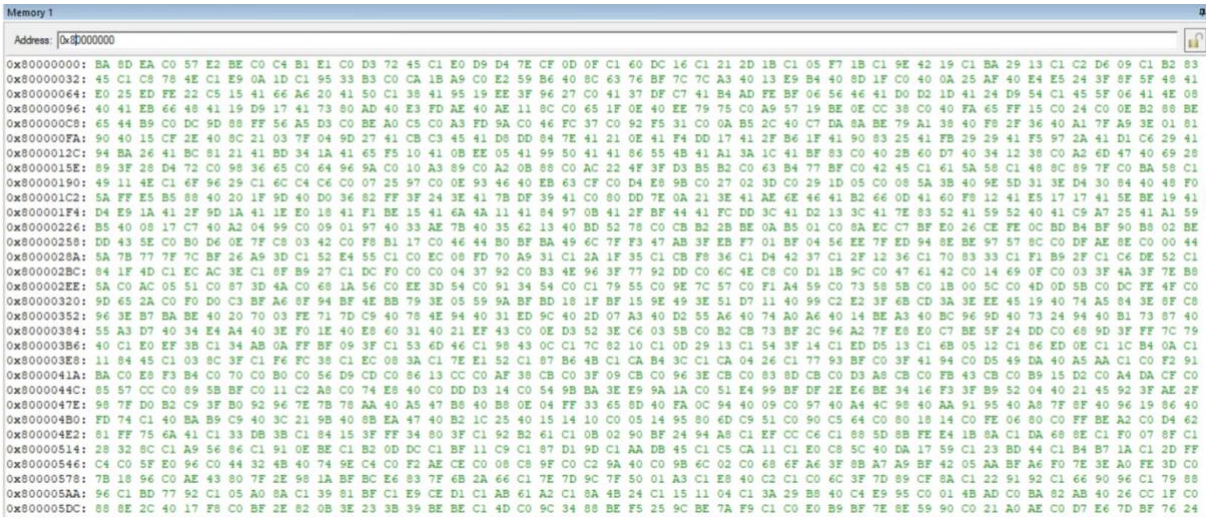
4. Performance & Result

A. 모든 sample data * filter

0x40000000에 모든 sample data와 filter가 곱해진 결과가 저장되있는 것을 볼 수 있고 사용되지 않은 값은 양 끝을 기준으로 5개의 sample data 값은 더하지 않기 때문에 사용하지 않았다.

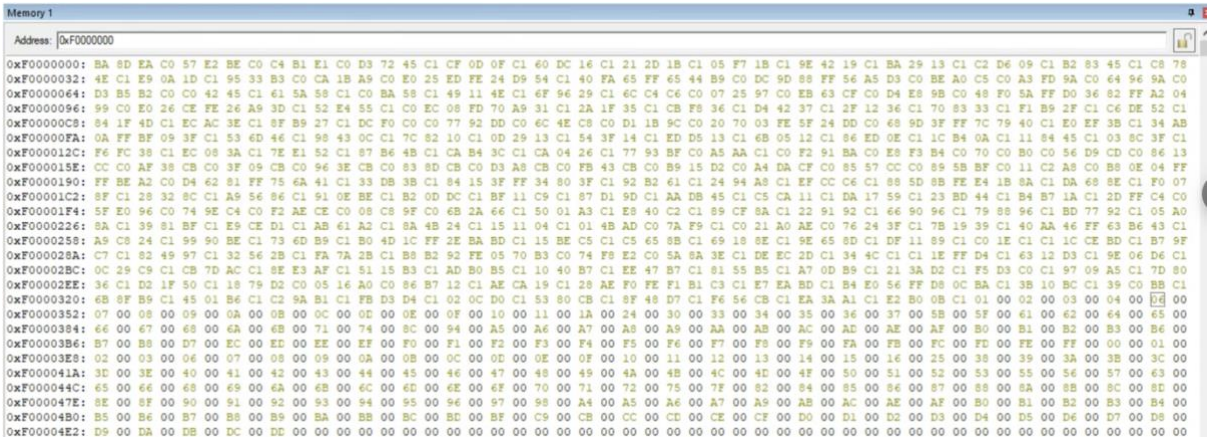
값들은 double word 단위로 저장되어 있고 모두 부동소수점으로 저장되어 있고, 모든 값을 하나하나 직접 계산하기는 어렵기 때문에 (https://t.hi098123.com/IEEE-754)에서 비교해가며 진행하였다. 그리고 소수점을 이진수로 나타낼 때는 소수 아랫자리가 끊기기 때문에 결과값에 오차가 발생하는 것도 확인하였다.

B. 모든 곱 값 합성 후 결과 저장



0x80000000에 모든 곱의 값이 합성된 것이 double word 단위로 저장되어 있는 것을 확인할 수 있다. 후에 모든 값을 -4.7과 비교하기 때문에 모든 값이 사용된 것을 확인할 수 있다. Run to cursor line 동작을 통해 각 동작마다 어떤 곱의 결과를 가져와 합성하는지 확인하고 후에 -4.7과 비교 알고리즘을 설계하였다.

C. -4.7보다 작은 결과 저장



인덱스 개수만 대략 구해봤을 때 200여개가 넘어갔고 이전에 조교님께서 말씀해 주신 결과와 너무 달라서 하나하나 다시 확인해 보니 합성곱에서 무언가 잘못되었고 모든 index를 가져온 것은 확인해 보니 모든 합성곱의 결과는 모두 -4.7 이하만 잘 가져온 것을 확인하였다.

5. Consideration

이번 프로젝트에서는 32bit signed number의 묶음을 floating point로 변환하여 가져오고 filter를 통해 이산 합성곱을 진행하여 해당 합성곱 값이 -4.7보다 작은 경우 저장하도록 프로젝트를 수행하였다. 처음에 dcd의 dataset에 저장된 부동소수점의 16진수를 가져와서 곱셈을 진행하는 것은 각 exponent의 합과 mantissa의 곱을 해야 했고 특히 mantissa의 UMULL 곱에서 얼만큼 값을 가져올 지 정하는데 머리를 좀 써야했다.

그리고 결과를 저장할 때도 각 값을 저장하며 인덱스를 같이 저장하는 것이 아닌 뒤에 따로 저장해야 하기 때문에 값이 오히려 더욱 복잡했다.

마지막으로 처음에는 Register만 활용해서 결과값을 가져올 수 있도록 하려고 하였지만 확실히 제한된 개수의 변수로 만드는 프로그램은 복잡하였고 그래서 대안으로 만든 것이 모든 sample data의 값과 filter의 곱을 먼저 진행해서 memory에 저장하고 합성 결과도 memory에 저장하여 memory로부터 값을 비교하며 저장하도록 진행하였으나 합성곱 과정 중에 일부 잘못된 것이 있는지 마지막에 너무 많은 결과가 저장되었다.

6. Reference

1) 이형근 교수님/ 어셈블리프로그램설계및실습 프로젝트 자료/광운대학교 컴퓨터정보공학부 /2022

2) 이형근 교수님/ 어셈블리프로그램설계및실습 float pointing 자료/광운대학교 컴퓨터정보공학부 /2022