어셈블리프로그램설계및실습 보고서

Project

학 과: 컴퓨터공학과

담당교수: 이형근 교수님

실습분반: 화요일 6, 7

학 번: 2021202058

성 명: 송채영

제 출 일: 2022.12.4(일)

1. Introduction

이번 프로젝트는 임의의 부동 소수점으로 이루어진 이산 신호(discrete signal)와 부동 소수점으로 이루어진 필터와의 합성 곱(convolution)을 수행한 후 특정 범위 내에 포함되는 값만 추출해 해당 결과와 해당 결과의 index 값을 메모리에 저장한다.

2. Background

부동 소수점은 실수를 컴퓨터상에서 근사하여 표현할 때 소수점의 위치를 고정하지 않고 그 위치를 나타내는 수를 따로 적는 것으로, 유효숫자를 나타내는 가수와 소수점의 위치를 풀이하는 지수로 나누어 표현한다. 고정 소수점 방식보다 연산 속도가 느리다는 단점이 있다. 프로젝트에서는 32 bit를 사용한다. 부호부(1 bit)와 지수부(8 bit)와 가수부(23 bit)로 실수를 나타낸다. 부호부는 양수일 때는 0, 음수일 때는 1로 나타내며 가수부는 제일 앞의 비트는 정규화 되었으므로 1로 나타낸다.

다음으로 이산 신호, discrete signal은 일정 시점에서 만 정의되는 신호 또는 수열을 말한다. 이산시간 마다 연속된 값이나 이산 값을 가지며 각 이산시간 사이에는 값이 존재하지 않는다.

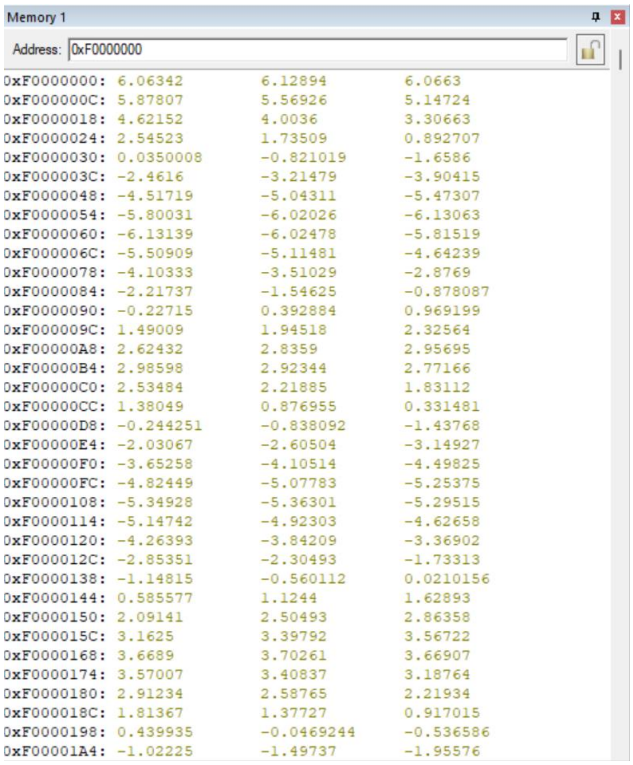
마지막으로 합성 곱, convolution은 하나의 함수와 또 다른 함수를 반전 이동한 값을 곱한 후 구간에 대해 적분하여 새로운 함수를 구하는 것을 말한다. 프로젝트에서는 1D-합성곱을 사용하는데, 합성 곱은 이동하는 방향의 수와 출력 형태에 따라 1D, 2D, 3D로 나눌 수 있다. 1D-합성곱은 필터가 시간을 축으로 좌우로만 이동할 수 있는 것을 말한다. 1D-합성곱은 그래프 곡선을 완화할 때 많이 사용한다.

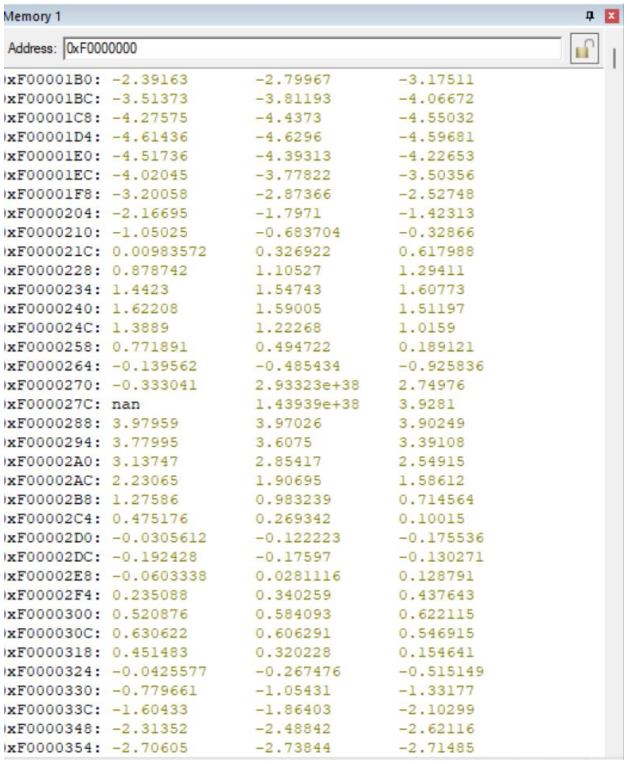
3. Algorithm

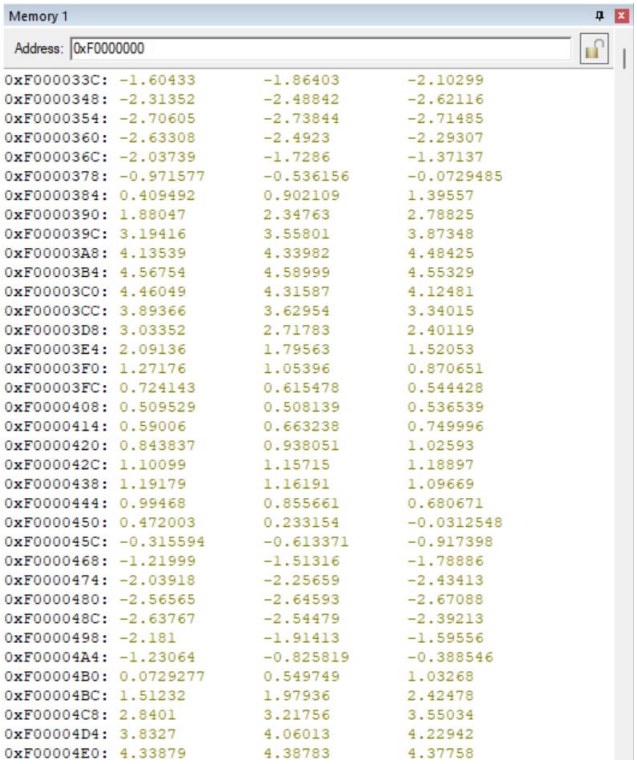
먼저 main함수에서 sampled data와 save result address, h 즉 filter를 각각 register r0, r1, r2에 저장한다. 또한 r0에 1920을 더한 값을 r13에 저장해준다. 이는 loop문에서 multiplication의 끝을 비교하기 위해서 선언해주었다. 이어서 loop를 도는데, sampled data에 +1 해준 것과 filter에 +1 해준 것을 각각 register r3과 r4에 저장하고 multiplication을 진행한다. R0와 r13을 비교하여 같을 경우 stop으로 가 연산을 종료하고, 다를 경우 sampled data와 filter를 각각 20, 24만큼 빼 loop를 돈다. 이때 20과 24만큼 빼는 이유는 처음으로 돌아가 연산을 진행하기 위해서이다. 이 과정은 filter의 개수만큼 진행해야 하므로, 총 6번 진행한다. 그리고 나서 Result에 값을 저장한다. 이후 multiplication에 대해 설명하겠다. Multiplication은

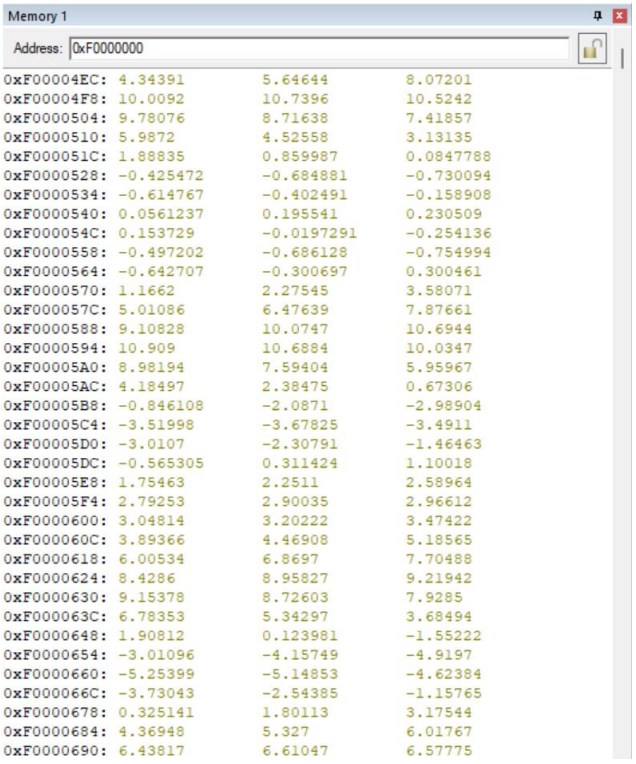
곱셈 연산을 진행한다. Register r5, r6, r7에 첫 번째 값의 sign bit와 exponent 그리고 mantissa를 저장해주었다. Register r8, r9, r10에는 두 번째 값의 sign bit와 exponent 그리고 mantissa를 저장해주었다. 예를 들어1.2와 1.3을 곱셈 연산을 하면 1.56이 되어 총 47 bit가 된다. 하지면 연산 결과가 11.xxx 혹은 10.xxx 의 경우 48 bit가 되기 때문에 이를 위해 두 가지 경우로 나누어 연산을 진행하였다. 첫번째 예시의 경우 상위 비트에 15 비트가 저장되어 있기 때문에 9비트를 가져와 24 비트를 차지하게끔 만들고 하위 비트에서 23에서 9를 뺀 23 비트를 오른쪽으로 shift 해서 9비트를 제외한 나머지를 없앤 후 앞에 비트와 더해주어야 한다. 이 과정을 코드로 구현하였다. 이어서 덧셈 부분은 지난 과제에서 구현하였던 부분과 동일하다. 지난 과제에서는 r12 register에 mantissa 값을 넣어주어 r12로 사용하였지만, 이번 프로젝트에서는 사용할 수 있는 register가 부족하여 값을 직접 넣어 주었다. 또한 비트 수를 맞춰주기 위해 mantissa 부분의 코드를 일부 수정하였다. Sign bit와 exponent 그리고 mantissa를 저장하는 코드는 multiplication 부분과 동일하다. Exponent와 sign 비트를 비교해준다. 이때 사인 비트가 같으면 equal로 다르면 notequal로 가게끔 하였다. equal에서는 mantissa 값을 더해준 후 normalize로 간다. notequal에서는 두 mantissa 값을 비교하여 같다면 endline으로, 다르다면 큰 값에서 작은 값을 뺀다. Normalize 부분에서는 mantissa를 normalize 해주는데 normalize 한 후 exponent 값을 1 뺀다. Finish로 가서 연산을 끝내며 프로그램을 종료한다. result에는 -4.7 보다 낮은 곱셈을 저장하여 index로 나타내는 부분이지만 이 부분은 구현하지 못하였다.

4. Performance & Result

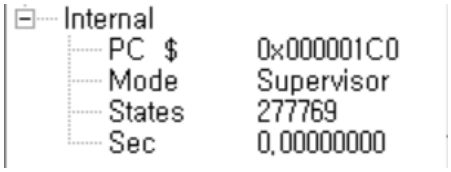

Memory 1
Address: 0xF0000000
0xF0000000: 6.06342 6.12894 6.0663
0xF000000C: 5.87807 5.56926 5.14724
0xF0000018: 4.62152 4.0036 3.30663
0xF0000024: 2.54523 1.73509 0.892707
0xF0000030: 0.0350008 -0.821019 -1.6586
0xF000003C: -2.4616 -3.21479 -3.90415
0xF0000048: -4.51719 -5.04311 -5.47307
0xF0000054: -5.80031 -6.02026 -6.13063
0xF0000060: -6.13139 -6.02478 -5.81519
0xF000006C: -5.50909 -5.11481 -4.64239
0xF0000078: -4.10333 -3.51029 -2.8769
0xF0000084: -2.21737 -1.54625 -0.878087
0xF0000090: -0.22715 0.392884 0.969199
0xF000009C: 1.49009 1.94518 2.32564
0xF00000A8: 2.62432 2.8359 2.95695
0xF00000B4: 2.98598 2.92344 2.77166
0xF00000C0: 2.53484 2.21885 1.83112
0xF00000CC: 1.38049 0.876955 0.331481
0xF00000D8: -0.244251 -0.838092 -1.43768
0xF00000E4: -2.03067 -2.60504 -3.14927
0xF00000F0: -3.65258 -4.10514 -4.49825
0xF00000FC: -4.82449 -5.07783 -5.25375
0xF0000108: -5.34928 -5.36301 -5.29515
0xF0000114: -5.14742 -4.92303 -4.62658
0xF0000120: -4.26393 -3.84209 -3.36902
0xF000012C: -2.85351 -2.30493 -1.73313
0xF0000138: -1.14815 -0.560112 0.0210156
0xF0000144: 0.585577 1.1244 1.62893
0xF0000150: 2.09141 2.50493 2.86358
0xF000015C: 3.1625 3.39792 3.56722
0xF0000168: 3.6689 3.70261 3.66907
0xF0000174: 3.57007 3.40837 3.18764
0xF0000180: 2.91234 2.58765 2.21934
0xF000018C: 1.81367 1.37727 0.917015
0xF0000198: 0.439935 -0.0469244 -0.536586
0xF00001A4: -1.02225 -1.49737 -1.95576


Memory 1
Address: 0xF0000000
xF00001B0: -2.39163 -2.79967 -3.17511
xF00001BC: -3.51373 -3.81193 -4.06672
xF00001C8: -4.27575 -4.4373 -4.55032
xF00001D4: -4.61436 -4.6296 -4.59681
xF00001E0: -4.51736 -4.39313 -4.22653
xF00001EC: -4.02045 -3.77822 -3.50356
xF00001F8: -3.20058 -2.87366 -2.52748
xF0000204: -2.16695 -1.7971 -1.42313
xF0000210: -1.05025 -0.683704 -0.32866
xF000021C: 0.00983572 0.326922 0.617988
xF0000228: 0.878742 1.10527 1.29411
xF0000234: 1.4423 1.54743 1.60773
xF0000240: 1.62208 1.59005 1.51197
xF000024C: 1.3889 1.22268 1.0159
xF0000258: 0.771891 0.494722 0.189121
xF0000264: -0.139562 -0.485434 -0.925836
xF0000270: -0.333041 2.93323e+38 2.74976
xF000027C: nan 1.43939e+38 3.9281
xF0000288: 3.97959 3.97026 3.90249
xF0000294: 3.77995 3.6075 3.39108
xF00002A0: 3.13747 2.85417 2.54915
xF00002AC: 2.23065 1.90695 1.58612
xF00002B8: 1.27586 0.983239 0.714564
xF00002C4: 0.475176 0.269342 0.10015
xF00002D0: -0.0305612 -0.122223 -0.175536
xF00002DC: -0.192428 -0.17597 -0.130271
xF00002E8: -0.0603338 0.0281116 0.128791
xF00002F4: 0.235088 0.340259 0.437643
xF0000300: 0.520876 0.584093 0.622115
xF000030C: 0.630622 0.606291 0.546915
xF0000318: 0.451483 0.320228 0.154641
xF0000324: -0.0425577 -0.267476 -0.515149
xF0000330: -0.779661 -1.05431 -1.33177
xF000033C: -1.60433 -1.86403 -2.10299
xF0000348: -2.31352 -2.48842 -2.62116
xF0000354: -2.70605 -2.73844 -2.71485


Memory 1
Address: 0xF0000000
0xF000033C: -1.60433 -1.86403 -2.10299
0xF0000348: -2.31352 -2.48842 -2.62116
0xF0000354: -2.70605 -2.73844 -2.71485
0xF0000360: -2.63308 -2.4923 -2.29307
0xF000036C: -2.03739 -1.7286 -1.37137
0xF0000378: -0.971577 -0.536156 -0.0729485
0xF0000384: 0.409492 0.902109 1.39557
0xF0000390: 1.88047 2.34763 2.78825
0xF000039C: 3.19416 3.55801 3.87348
0xF00003A8: 4.13539 4.33982 4.48425
0xF00003B4: 4.56754 4.58999 4.55329
0xF00003C0: 4.46049 4.31587 4.12481
0xF00003CC: 3.89366 3.62954 3.34015
0xF00003D8: 3.03352 2.71783 2.40119
0xF00003E4: 2.09136 1.79563 1.52053
0xF00003F0: 1.27176 1.05396 0.870651
0xF00003FC: 0.724143 0.615478 0.544428
0xF0000408: 0.509529 0.508139 0.536539
0xF0000414: 0.59006 0.663238 0.749996
0xF0000420: 0.843837 0.938051 1.02593
0xF000042C: 1.10099 1.15715 1.18897
0xF0000438: 1.19179 1.16191 1.09669
0xF0000444: 0.99468 0.855661 0.680671
0xF0000450: 0.472003 0.233154 -0.0312548
0xF000045C: -0.315594 -0.613371 -0.917398
0xF0000468: -1.21999 -1.51316 -1.78886
0xF0000474: -2.03918 -2.25659 -2.43413
0xF0000480: -2.56565 -2.64593 -2.67088
0xF000048C: -2.63767 -2.54479 -2.39213
0xF0000498: -2.181 -1.91413 -1.59556
0xF00004A4: -1.23064 -0.825819 -0.388546
0xF00004B0: 0.0729277 0.549749 1.03268
0xF00004BC: 1.51232 1.97936 2.42478
0xF00004C8: 2.8401 3.21756 3.55034
0xF00004D4: 3.8327 4.06013 4.22942
0xF00004E0: 4.33879 4.38783 4.37758


Memory 1
Address: 0xF0000000
0xF00004EC: 4.34391 5.64644 8.07201
0xF00004F8: 10.0092 10.7396 10.5242
0xF0000504: 9.78076 8.71638 7.41857
0xF0000510: 5.9872 4.52558 3.13135
0xF000051C: 1.88835 0.859987 0.0847788
0xF0000528: -0.425472 -0.684881 -0.730094
0xF0000534: -0.614767 -0.402491 -0.158908
0xF0000540: 0.0561237 0.195541 0.230509
0xF000054C: 0.153729 -0.0197291 -0.254136
0xF0000558: -0.497202 -0.686128 -0.754994
0xF0000564: -0.642707 -0.300697 0.300461
0xF0000570: 1.1662 2.27545 3.58071
0xF000057C: 5.01086 6.47639 7.87661
0xF0000588: 9.10828 10.0747 10.6944
0xF0000594: 10.909 10.6884 10.0347
0xF00005A0: 8.98194 7.59404 5.95967
0xF00005AC: 4.18497 2.38475 0.67306
0xF00005B8: -0.846108 -2.0871 -2.98904
0xF00005C4: -3.51998 -3.67825 -3.4911
0xF00005D0: -3.0107 -2.30791 -1.46463
0xF00005DC: -0.565305 0.311424 1.10018
0xF00005E8: 1.75463 2.2511 2.58964
0xF00005F4: 2.79253 2.90035 2.96612
0xF0000600: 3.04814 3.20222 3.47422
0xF000060C: 3.89366 4.46908 5.18565
0xF0000618: 6.00534 6.8697 7.70488
0xF0000624: 8.4286 8.95827 9.21942
0xF0000630: 9.15378 8.72603 7.9285
0xF000063C: 6.78353 5.34297 3.68494
0xF0000648: 1.90812 0.123981 -1.55222
0xF0000654: -3.01096 -4.15749 -4.9197
0xF0000660: -5.25399 -5.14853 -4.62384
0xF000066C: -3.73043 -2.54385 -1.15765
0xF0000678: 0.325141 1.80113 3.17544
0xF0000684: 4.36948 5.327 6.01767
0xF0000690: 6.43817 6.61047 6.57775

```
0xF000069C: 6.39828         6.13796         5.8624
0xF00006A8: 5.62915         5.48114         5.44184
0xF00006B4: 5.51262         5.67277         5.88188
0xF00006C0: 6.08466         6.21739         6.2156
0xF00006CC: 6.02196         5.59375         4.90892
0xF00006D8: 3.97027         2.80719         1.47463
0xF00006E4: 0.0495982       -1.3748         -2.69707
0xF00006F0: -3.81615        -4.64045        -5.09632
0xF00006FC: -5.13507        -4.7379         -3.91817
0xF0000708: -2.72095        -1.2196         0.490033
0xF0000714: 2.29749         4.08679         5.74596
0xF0000720: 7.17605         8.29897         9.0632
0xF000072C: 9.44706         9.45922         9.13636
0xF0000738: 8.53846         7.74197         6.83174
0xF0000744: 5.8925          5.0006          4.21704
0xF0000750: 3.58224         3.11324         2.80336
0xF000075C: 2.62454         2.53186         2.46993
```



최종 실행된 결과 화면 사진이며, 총 25개의 값이 저장된 것을 알 수 있다. Convolution의 결과 중 –4.7 이하의 모든 수와 해당 수의 인덱스를 저장하는 부분은 구현하지 못하였다. 또한 중간에 nan이 결과값으로 나온 것으로 보아 연산과정에서 문제가 있는 것 같다. 하나하나 들어가는 값을 확인해 본 결과 loop에서는 문제가 없었으며 덧셈을 해주는 addition부분에서 normalize가 제대로 되지 않아 생기는 문제라고 생각되었으나 해결하지 못하였다. 두 번째 사진에서 볼 수 있듯이 277769개의 state가 실행되었다.

5. Consideration

실습과 과제를 진행하며 배웠던 내용들을 프로젝트에 적용하여 구현하였다. 특히 덧셈 부분은 과제에서 수행하였던 덧셈 부분을 가져왔는데 지난번 과제에서는 오류가 발생하지 않았고 연산 결과가 바르게 나왔다. 하지만 이번 과제에서 덧셈 과정이나 곱셈 과정이 제대로 실행되지 않아 오류가 난 것 같다. 오류를 해결하기 위해 register에 들어가는 값도 확인해보고 코드를 여러 번 읽어보며 잘못된 부분을 찾아보려고 노력했지만, 어느 부분이 잘못된 것인지 찾지 못하였다. 처음에는 e^-38 같은 값이 나왔지만, register를 잘못 넣어주어 생긴 결과였다. 계속해서 들어가는 값을 확인하고 오류를 고치는 과정을 반복하였다. 덧셈 부분의 코드를 이것 저것 수정해보며 실행도 해보았지만, 여전히 결과값이 잘 나오지 않았다. 다른 친구들은 -4.7이하의 수가 총 27개가 나왔다고 하였지만, 나는 25개밖에 나오지 않았고, nan이라는 알 수 없는 값이 나오기도 해 연산 과정에 문제가 있다는 것을 알았다. 하나하나 들어가는 값을 확인하면서 loop와 다른 곳에는 문제가 없음을 알았으며, 덧셈을 해주는 addition 부분에서 normalize가 제대로 되지 않아 생기는 문제라고 생각했다. 결국 코드를 끝까지, 구현하지 못하고, 오류도 잡아내지 못하였지만, 지난 실습을 다

시 복습해 볼 수 있었고 이번 프로젝트가 끝나더라도 끝까지 구현해보고 싶다.

6. Reference

이형근 교수님/어셈블리프로그램설계및실습/광운대학교(컴퓨터정보공학부)/2022

부동소수점/ https://ko.wikipedia.org/wiki/%EB%B6%80%EB%8F%99%EC%86%8C%EC%88%98%EC%A0%90

합성곱/https://ko.wikipedia.org/wiki/%ED%95%A9%EC%84%B1%EA%B3%B1