Floating Point Inverse Matrix

2015722031 박태성

Abstract

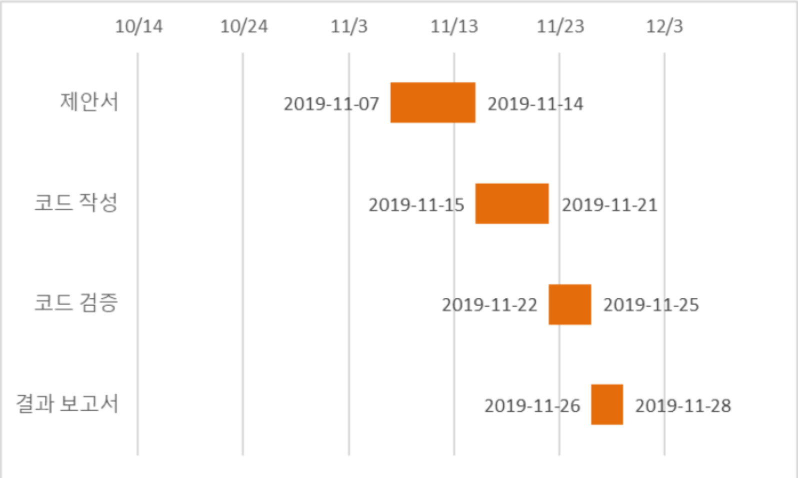
임의의 floating point 데이터로 이루어진 정방 행렬에 대한 역 행렬을 구하는 어셈블리 코드를 작성하며 성능이 가장 좋은 floating point multiplication, division을 구현하는 방법에 대해 생각해보는 데 목적을 둔다.

I. Introduction

A. 프로젝트의 간단한 설명

본 프로젝트는 임의의 부동소수점 데이터로 이루어진 N*N(1<=N<=20) 정방 행렬에 대한 역 행렬을 구하는 어셈블리 코드를 작성하는 것이다. 프로젝트를 구현함에 있어, 성능이 기준이 가장 좋은 코드를 구현하는 데 목적을 둔다. 성능의 기준은 state으로, 값이 작을수록 성능이 좋다. 행렬의 사이즈 N과 N^2개의 부동소수점 데이터는 txt파일을 통해 제공된다.

B. 프로젝트 수행 일정



본 프로젝트의 일정이다. 제안서 작성 시 spec 문서의 이해, 기초 설계, 그리고 flowchart를 작성하였다. 코드 작성은 arithmetic, inverse matrix 구현 순서로 진행하였다. 검증은 주어진 test set으로 하였다. 상대 오차 범위에 부합하지 못하는 결과가 있어, 사칙 연산 mantissa 버림을 반올림으로 수정하였다.

II. Project Specification

A. 역 행렬이 없는 경우는 고려하지 않는다. 그러나, 각 열의 row exchange를 하여 pivot을 구할 수 있는 행렬은 row exchange를 통하여 역 행렬을 구하여야 한다. 따라서, 가우스 조던 소거법을 통한 역 행렬 풀이 도중에 leading one이 0인 상황에는 swap row label로 점프하여 두 행을 바꿔준다.

B. Label: Matrix_data
DCD 명령어를 이용하여 정방 행렬의 크기 N과 임의의 부동소수점 데이터로 구성된다.

C. Label: Result_data
구해진 역 행렬이 저장되는 데이터의 시작 주소 값. 0x60000000 번지에 결과를 1 word 단위로 저장한다.
D. MUL, DIV 명령어는 사용을 금지하므로 floating-point multiplication/ division을 구현하여 활용한다. 따라서, floating-point multiplication/ division을 구현하였다. Multiplication은 radix-4를 기반으로 booth multiplication 방식으로 구현하였다. Division은 이진수 나누기 이진수를 활용하여 구현하였다.
E. 결과의 오차 범위는 상대 오차 2^(-10) 보다 작다.
F. 성능이 가장 좋게(state값이 작게) 구현한다. S suffix, Barrel shift, 그리고 Rule of thumb과 같은 기능을 최대한 활용하도록 한다.
G. Code size는 평가 대상이 아니다. 오로지 state 만 보고 성능을 판단한다.

III. Algorithm

A. Finding Inverse Matrix using Gauss-Jordan elimination method

A. Inverse Matrix

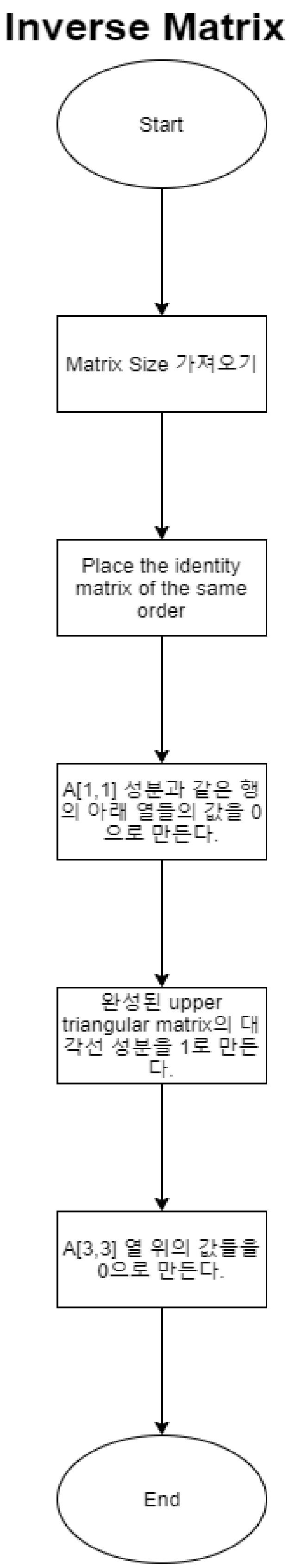
A를 n*n 행렬이라 하고 I를 n*n 정방 행렬이라 하자. AB = BA = I or $AA^{-1} = A^{-1}A = I$를 만족하는 행렬 B를 행렬 A의 역 행렬이라고 한다. 단, 모든 정방 행렬이 역 행렬을 갖지는 않는다. 다음은 역 행렬 구하는 알고리즘의 슈도 코드이다.

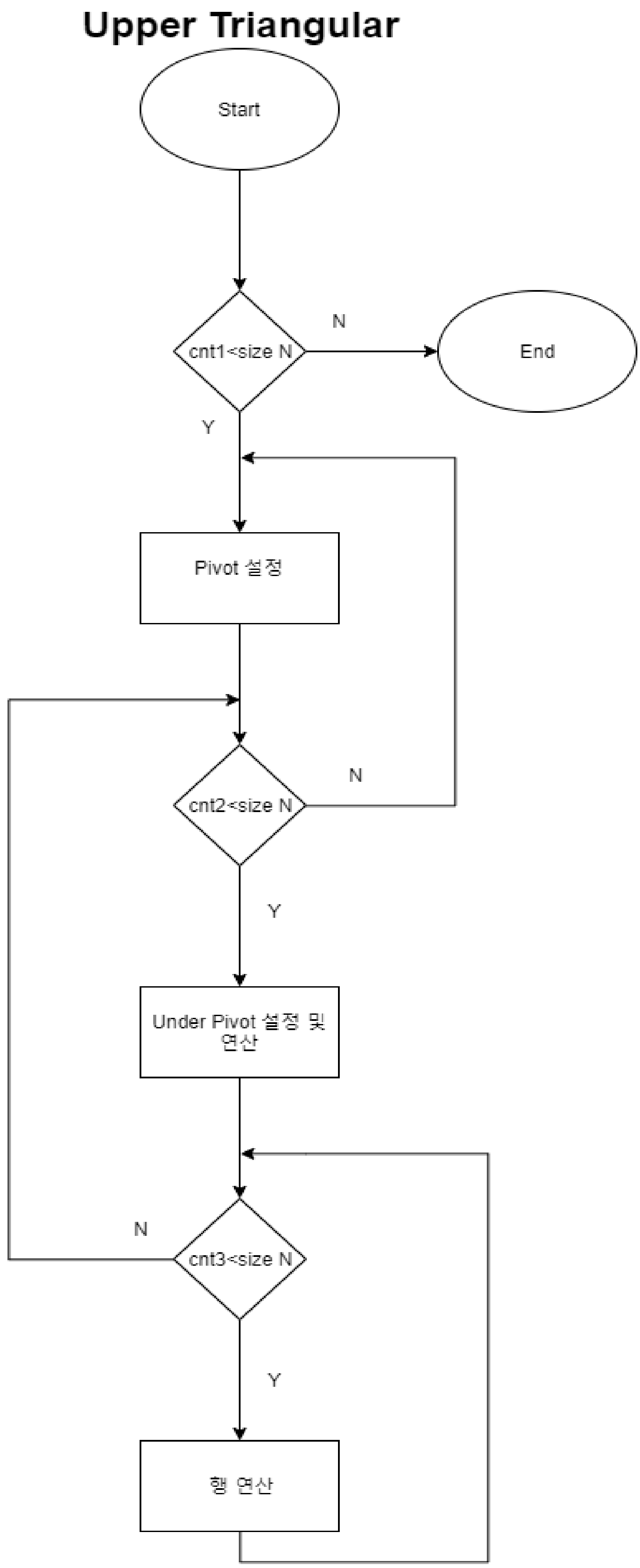
1. Inverse Matrix 만들기
   A. Matrix size 가져오기
   B. 역 행렬 초기화
   C. Matrix_data 상삼각행렬 만들기
   D. Matrix_data 대각선 성분 1 만들기
   E. Matrix_data 하삼각행렬 만들기
2. 상삼각행렬 만들기
   A. Pivot 설정
   B. Under pivot 설정
   C. Under pivot/ pivot 계산.
   D. Matrix data와 Result data의 행 연산.
3. 대각선 성분 1 만들기
   A. Pivot 설정
   B. Matirx_dat와 Result data의 행 연산
4. 하삼각행렬 만들기
   A. Pivot 설정
   B. Above pivot 설정
   C. Above pivot/ pivot 계산.
   D. Matrix data와 Result data의 행 연산.

B. Gauss-Jordan elimination method

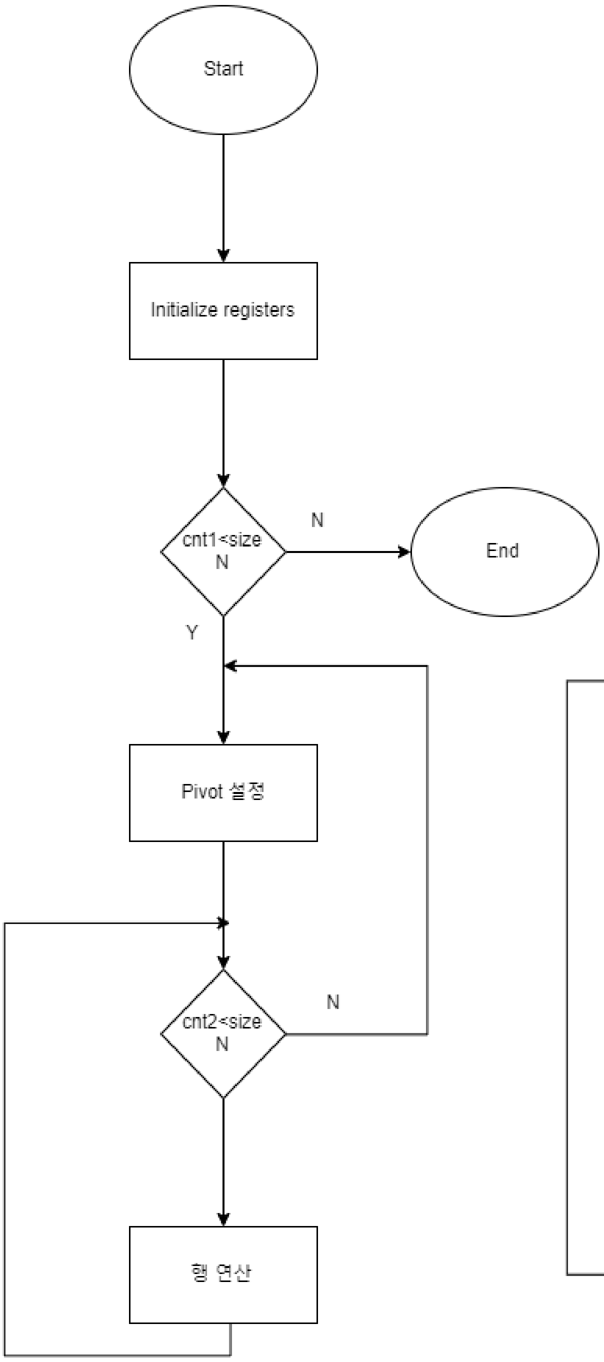
가우스 조던 소거법은 선형대수학에서 선형 방정식의 해를 구하기 위한 알고리즘이다. Elementary row operations를 사용한다. 다음은 세 가지 방식의 elementary row operations이다. 첫 째, 두 행을 교환한다. 둘 째, 행을 0이 아닌 숫자로 곱한다. 셋 째, 어떤 행의 곱을 다른 행에 더한다. 가우스 조던 소거법을 통해 역 행렬을 얻을 수 있다.

C. Block Diagram – Program Overview

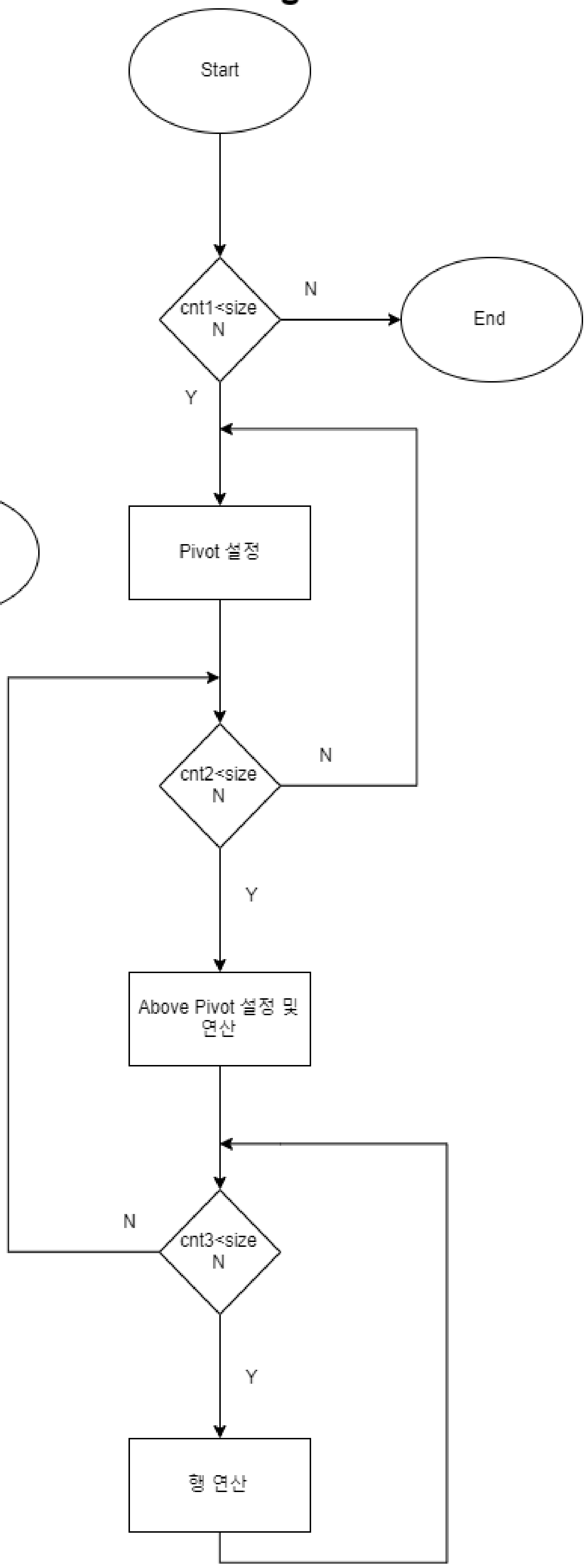

Inverse Matrix
Start
Matrix Size 가져오기
Place the identity matrix of the same order
A[1,1] 성분과 같은 행의 아래 열들의 값을 0으로 만든다.
완성된 upper triangular matrix의 대각선 성분을 1로 만든다.
A[3,3] 열 위의 값들을 0으로 만든다.
End


Upper Triangular
Start
cnt1<size N
N
End
Y
Pivot 설정
cnt2<size N
N
Y
Under Pivot 설정 및 연산
N
cnt3<size N
Y
행 연산

Diagonal



Lower Triangular



B. Arithmetic

A. Floating Point Addition

Sign bit, exponent bits, Mantissa(Fraction) bits에 대해 추출한다.
Mantissa 앞에 1을 붙임.
Exponent를 같게 조정한다. (지수가 작은 쪽을 큰 쪽으로 조정한다.)
덧셈을 진행한다.
계산 결과 mantissa를 정규 화한다.
이에 맞게 exponent값을 조절한다.

B. Floating Point Subtraction

Exponent를 같게 조정한다. (지수가 작은 쪽을 큰 쪽으로 조정한다.)
덧셈을 진행한다.
계산 결과 가수를 정규 화한다.
이에 맞게 지수 값을 조절한다.

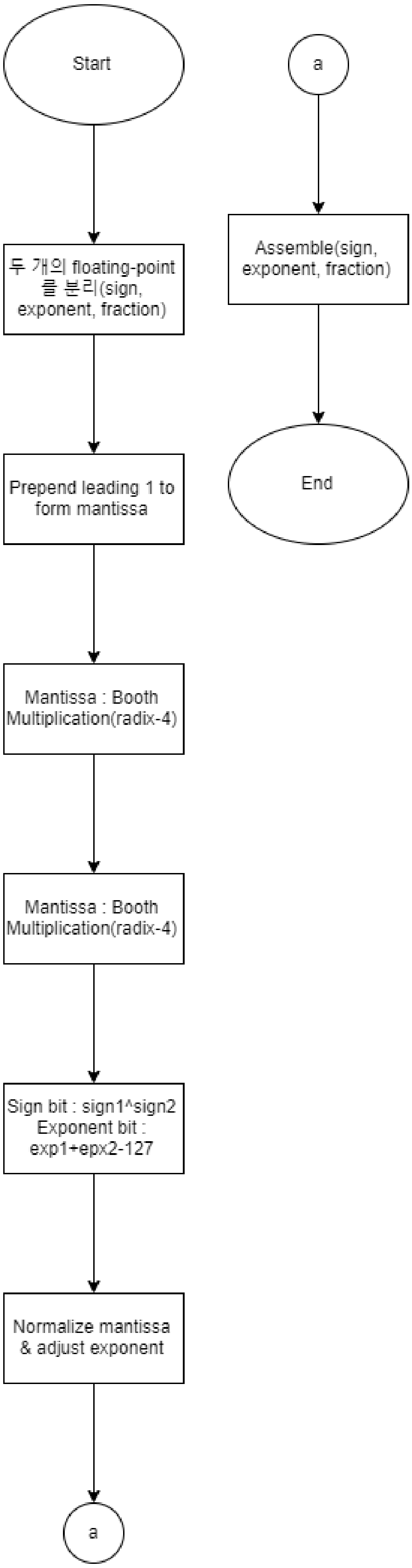
C. Floating Point Multiplication

Exponent를 같게 조정하지 않는다.
가수끼리 곱한다. (Radix-4)
지수끼리 더한다.
소수 이하 자리는 정해진 자리에서 반올림한다.
계산 결과 가수를 정규 화한다.

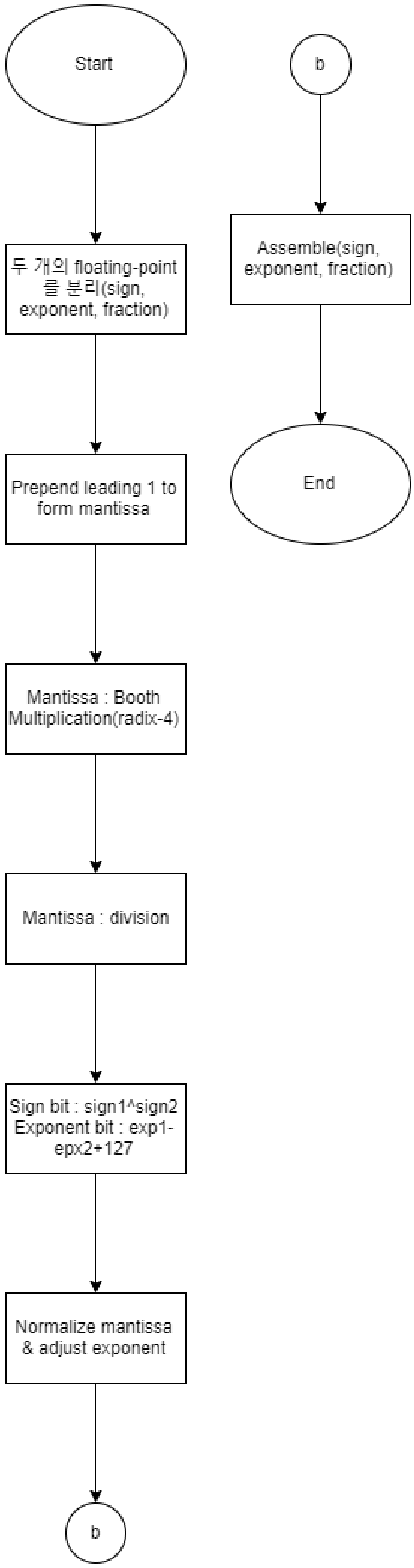
D. Floating Point Division

Q: 몫, R: 나머지, D: 제수
Exponent를 같게 조정하지 않는다.
가수끼리 나눈다. (24bit 2진수 나누기 24bit 이진수)
R < D라면, Q, R을 왼쪽으로 1비트 shift
R >=D라면, Q=Q+1, R=R-D
지수끼리 뺀다.
소수이라 자리는 정해진 자리에서 반올림 한다.
계산 결과 가수를 정규 화한다.

Floating-Point Multiplication

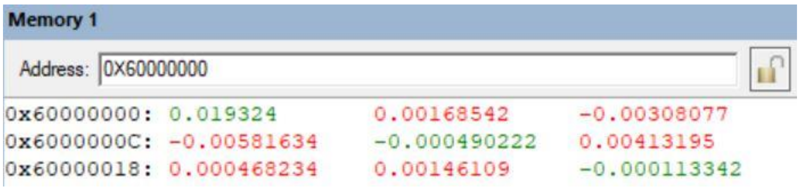


Floating-Point Division



IV. Design Verification Strategy and Results

A. Test Dataset 1 (Size 3)
Inverse Matrix <0x60000000>

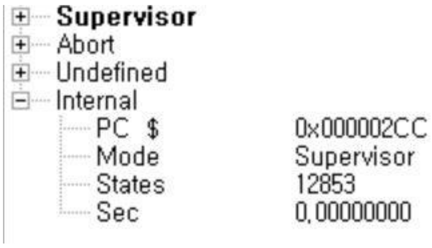


Memory 1

Address: 0X60000000

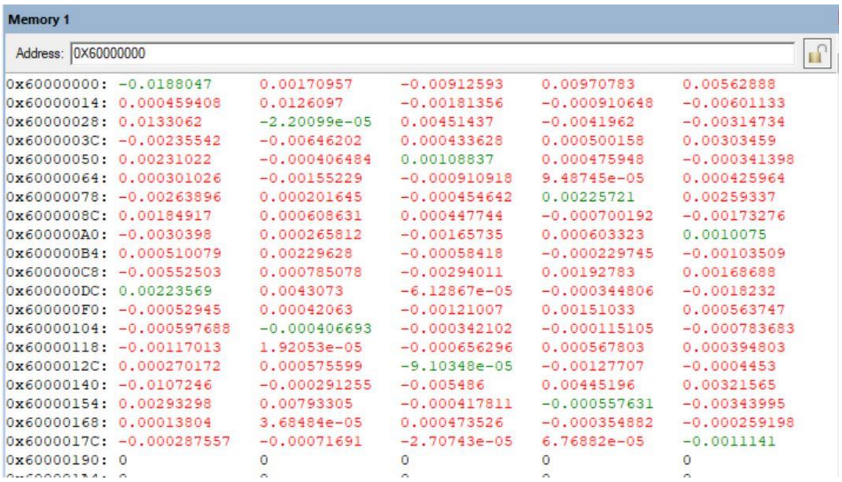
0x60000000:	0.019324	0.00168542	-0.00308077
0x6000000C:	-0.00581634	-0.000490222	0.00413195
0x60000018:	0.000468234	0.00146109	-0.000113342

State: 12853



Supervisor	
Abort	
Undefined	
Internal	
PC $	0x000002CC
Mode	Supervisor
States	12853
Sec	0.00000000

B. Test Dataset 2 (Size 10)
Inverse Matrix <0x60000000>

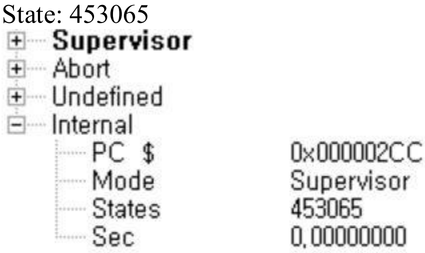


Memory 1

Address: 0X60000000

0x60000000:	-0.0188047	0.00170957	-0.00912593	0.00970783	0.00562888
0x60000014:	0.000459408	0.0126097	-0.00181356	-0.000910648	-0.00601133
0x60000028:	0.0133062	-2.20099e-05	0.00451437	-0.0041962	-0.00314734
0x6000003C:	-0.00235542	-0.00646202	0.000433628	0.000500158	0.00303459
0x60000050:	0.00231022	-0.000406484	0.00108837	0.000475948	-0.000341398
0x60000064:	0.000301026	-0.00155229	-0.000910918	9.48745e-05	0.000425964
0x60000078:	-0.00263896	0.000201645	-0.000454642	0.00225721	0.00259337
0x6000008C:	0.00184917	0.000608631	0.000447744	-0.000700192	-0.00173276
0x600000A0:	-0.0030398	0.000265812	-0.00165735	0.000603323	0.0010075
0x600000B4:	0.000510079	0.00229628	-0.00058418	-0.000229745	-0.00103509
0x600000C8:	-0.00552503	0.000785078	-0.00294011	0.00192783	0.00168688
0x600000DC:	0.00223569	0.0043073	-6.12867e-05	-0.000344806	-0.0018232
0x600000F0:	-0.00052945	0.00042063	-0.00121007	0.00151033	0.000563747
0x60000104:	-0.000597688	-0.000406693	-0.000342102	-0.000115105	-0.000783683
0x60000118:	-0.00117013	1.92053e-05	-0.000656296	0.000567803	0.000394803
0x6000012C:	0.000270172	0.000575599	-9.10348e-05	-0.00127707	-0.0004453
0x60000140:	-0.0107246	-0.000291255	-0.005486	0.00445196	0.00321565
0x60000154:	0.00293298	0.00793305	-0.000417811	-0.000557631	-0.00343995
0x60000168:	0.00013804	3.68484e-05	0.000473526	-0.000354882	-0.000259198
0x6000017C:	-0.000287557	-0.00071691	-2.70743e-05	6.76882e-05	-0.0011141
0x60000190:	0	0	0	0	0

State: 453065



Supervisor	
Abort	
Undefined	
Internal	
PC $	0x000002CC
Mode	Supervisor
States	453065
Sec	0.00000000

C. Test Dataset 3 (Size 20)
Inverse Matrix <0x60000000>

Memory 1

Address: 0X60000000

0x60000000:	0.000132281	-0.00202157	-0.000926676	0.00106078	0.000366045	0.00019877	-0.000517712
0x6000001C:	0.00129308	-0.00096774	0.0010443	-0.000638637	0.000368944	-0.000544795	-0.00189556
0x60000038:	0.000876157	-0.00195883	0.00199767	0.000193326	0.000712781	-0.000129653	0.000312288
0x60000054:	-0.000644922	-0.000682598	-0.000168274	-3.72578e-06	0.000521741	-0.000758311	0.000803008
0x60000070:	0.000718877	-0.000372321	0.000609644	0.000959187	0.00064426	-0.000842802	-0.00153986
0x6000008C:	0.00050148	0.000405155	-0.000356414	-0.000954435	-0.0007042	0.000663769	-0.000385556
0x600000A8:	-0.000878817	-0.000622217	0.000191217	0.00146941	0.000192762	0.000131516	-2.07516e-05
0x600000C4:	-0.000310924	-0.000129807	0.000112919	0.000728671	-4.01752e-05	-0.0016643	-0.000521995
0x600000E0:	-0.000861257	-4.94555e-05	-0.000160438	-0.000799329	0.000281666	0.0012849	0.000229403
0x600000FC:	-0.000991593	0.000500115	0.000311981	0.000877846	-0.000574589	0.000437272	-0.0011339
0x60000118:	0.00117842	0.000275862	0.00111794	0.000919594	-0.00141645	0.00219106	-0.00080445
0x60000134:	-4.30076e-05	-0.00170529	-0.000468209	0.000262801	0.000856774	0.000346933	-0.00144212
0x60000150:	-0.000439208	0.000286643	0.000229955	-0.00138351	0.00153082	-0.00195622	0.00103537
0x6000016C:	3.74721e-05	0.00194311	0.000700228	-0.00201146	0.00330862	-0.0025899	-0.000334736
0x60000188:	-0.000337287	-0.000905124	0.000742194	0.00188432	-0.000306858	-0.00201907	-0.000116064
0x600001A4:	-1.42315e-05	0.000354011	-0.00181035	0.00221177	-0.00224952	0.00179096	0.00093196
0x600001C0:	0.00228577	0.000966015	-0.00253954	0.00525564	-0.0024662	-0.000622747	-0.00108317
0x600001DC:	-0.000952588	0.000176577	-0.000449226	-0.000117995	-0.00118175	0.000358528	0.000457047
0x600001F8:	0.000526645	-0.000294455	0.000196409	-0.00053444	0.000616076	8.55033e-05	0.000696376
0x60000214:	0.000287156	-0.00115932	0.00125826	-0.000980977	-0.000389478	-0.00124164	-0.00132498
0x60000230:	-0.000583625	-0.000138014	-0.000559507	-0.00048949	-0.000241626	0.000403184	-0.000340664
0x6000024C:	0.000424563	-0.000354672	-0.000203982	0.000731089	-0.000198603	0.000680781	-0.000309626
0x60000268:	-0.00057858	0.000678804	0.000721351	9.55441e-05	-0.000234466	-0.000825937	9.14871e-05
0x60000284:	0.0011777	0.00044348	-0.000933072	0.000947933	0.000318829	0.00137255	-7.36881e-05
0x600002A0:	-0.000669889	0.000608402	0.000542563	-0.000191649	0.00062125	0.00213653	-0.000640259
0x600002BC:	-0.000702483	-0.00039117	0.000224438	-0.00244406	-0.000220783	0.000245927	0.000151688
0x600002D8:	-0.000562132	-0.000688808	3.95653e-05	-1.10296e-05	-8.80432e-05	-1.84737e-05	0.000396492
0x600002F4:	-0.000830732	0.0002646	0.000483786	0.000688782	7.1784e-06	-0.000768076	0.000553046
0x60000310:	-0.000252232	-0.000372809	-0.000216844	-0.00089757	0.000897471	0.00274821	0.000352557
0x6000032C:	-0.00236518	3.70559e-05	0.000136987	0.00139338	-0.00209655	0.00176203	-0.00252243
0x60000348:	0.00197607	1.53795e-05	0.0023876	0.00225012	-0.00188834	0.00502535	-0.00455071
0x60000364:	-0.000275026	-0.00235381	-0.000427286	-0.000463684	0.000552818	0.00115781	0.000225218
0x60000380:	0.000163096	-0.000157784	0.00122123	-0.000976886	-0.000852591	0.000688721	-0.000386894
0x6000039C:	-0.00146307	-0.000676753	0.00144188	0.0012753	-0.00192951	-0.0010057	0.000487541
0x600003B8:	-0.00130324	0.000864374	-0.000282212	0.00128115	0.000336896	-0.00173832	0.000206686
0x600003D4:	7.84107e-05	0.000205705	6.15969e-05	0.000134539	0.000229854	0.000466125	0.00018411
0x600003F0:	0.000346695	0.000560086	-0.00085032	0.000249173	-0.000266464	-0.000449916	-0.00118725
0x6000040C:	-0.000437384	0.00071007	0.00180373	-3.22908e-05	-0.00183349	-0.000381269	-3.95689e-05
0x60000428:	0.000528234	-0.00184332	0.00182945	-0.00240927	0.00119035	-7.76956e-06	0.00200319
0x60000444:	0.00110631	-0.00349585	0.00482804	-0.00409721	-0.00202551	-0.000823851	-0.000277407
0x60000460:	0.000114277	-0.00111824	-0.000515786	0.000821726	-9.56243e-05	-0.000105559	-0.000651198
0x6000047C:	0.000695839	-0.000448566	0.000475688	-0.00062858	0.00029257	-0.000640629	-0.000959321
0x60000498:	0.00100433	-0.00151977	0.00135026	0.000287883	0.00103211	0.000901156	0.000695482
0x600004B4:	-0.00084542	-0.000200525	0.000854549	7.71446e-05	-0.000181547	-0.000547332	0.000602268
0x600004D0:	-0.00075422	0.000724174	-0.000608562	0.000556574	-0.000874429	-0.000842667	0.00104702
0x600004EC:	-0.00139199	0.0014091	9.55912e-05	0.000941097	0.000294037	-0.00159487	-0.00392205
0x60000508:	-0.00058016	0.00330673	0.00038215	-2.5213e-05	-0.00180556	0.0037015	-0.00335661
0x60000524:	0.00444999	-0.00282935	7.77096e-06	-0.00360872	-0.00272074	0.00664412	-0.0085296
0x60000540:	0.00767105	0.000126193	0.00250852	0.000464469	-0.000440166	0.00259025	0.00163776
0x6000055C:	-0.00129184	-0.000100191	-0.000485846	0.0020747	-0.00259873	-0.000210632	-0.000790096
0x60000578:	-0.00015844	-0.00206101	0.000859719	0.00307672	0.000844308	0.0017207	-0.0032354
0x60000594:	0.000540053	-0.00290235	0.000630279	-0.000322837	-0.000308592	-0.000324769	8.61346e-05
0x600005B0:	0.000416294	0.000175099	-0.000290854	0.000553932	-0.000318449	0.000218501	0.00022766
0x600005CC:	0.00113313	1.42482e-05	-0.000350533	-0.000215992	-0.000264097	0.000693401	-7.62462e-05
0x600005E8:	-0.000426766	-0.00025886	0.00084544	0.000166457	0.000570993	0.000148727	-0.00111961
0x60000604:	-0.000365395	0.000468145	-0.000535478	0.000626277	6.90727e-06	-0.000690867	0.000233904
0x60000620:	-0.000484759	0.000459234	0.000290604	-0.000222594	-0.00101065	-0.000198121	0.000245066
0x6000063C:	0.00056169	0	0	0	0	0	0

State: 3596361

- Supervisor
- Abort
- Undefined
- Internal
  - PC $ 0x000002CC
  - Mode Supervisor
  - States 3596361
  - Sec 0.00000000

D. Test Dataset 4 (Size 4)

Inverse Matrix <0x60000000>

Memory 1

Address: 0X60000000

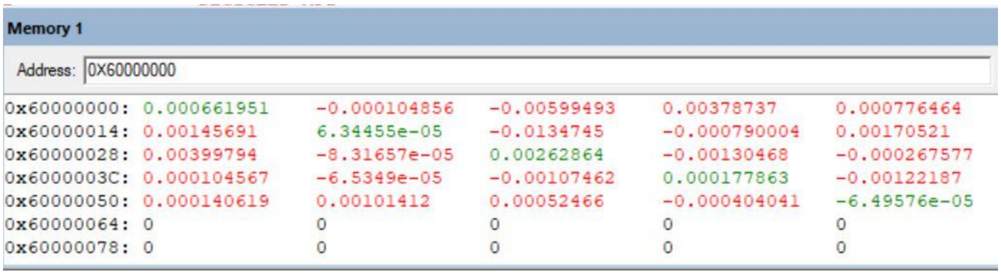
0x60000000:	0.0110382	0.0270086	-0.00211393	0.00400047
0x60000010:	0.00128029	0.00172597	0.000889814	0.00048721
0x60000020:	0.0038238	4.6449e-05	-5.32032e-05	-0.000196736
0x60000030:	0.00837845	-0.00200685	0.000426501	0.00340775
0x60000040:	0	0	0	0
0x60000050:	0	0	0	0
0x60000060:	0	0	0	0

State: 27566

- Supervisor
- Abort
- Undefined
- Internal
  - PC $ 0x000002CC
  - Mode Supervisor
  - States 27566
  - Sec 0.00000000

E. Test Dataset 5 (Size 5)

Inverse Matrix <0x60000000>

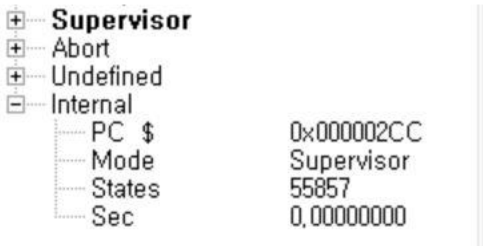


Memory 1

Address: 0X60000000

0x60000000:	0.000661951	-0.000104856	-0.00599493	0.00378737	0.000776464
0x60000014:	0.00145691	6.34455e-05	-0.0134745	-0.000790004	0.00170521
0x60000028:	0.00399794	-8.31657e-05	0.00262864	-0.00130468	-0.000267577
0x6000003C:	0.000104567	-6.5349e-05	-0.00107462	0.000177863	-0.00122187
0x60000050:	0.000140619	0.00101412	0.00052466	-0.000404041	-6.49576e-05
0x60000064:	0	0	0	0	0
0x60000078:	0	0	0	0	0

State: 55857



V. Conclusion

본 프로젝트 구현 시 겪었던 어려움과 그 해결 방안이다. 사용 가능한 레지스터의 개수가 제한되어 있어 역 행렬 구현 시 어려움이 있었다. Block data transfer를 통해 메모리에 data를 써서 해결하였다. 메모리 접근을 하지 않고 register를 유동적으로 사용하면 메모리 접근에 따른 state 소모를 줄 일 수 있지 않을까 생각하였다. 그러나, 코드의 복잡도가 높아지고 register map 관리가 힘들어지는 단점이 있어 block data transfer를 활용하였다. 본 프로젝트 구현 과정에서 MUL과 DIV 명령어 사용은 금지되었다. 따라서, floating-point multiplication/ division을 구현하였다. Multiplication은 radix-4를 기반으로 booth multiplication 방식으로 구현하였다. Radix-4를 사용한 이유는 radix-8, 16을 선택하여 반복 횟수를 줄여도 따져줄 case가 배로 늘어가 state 면에서는 별 차이가 없을 것이라 판단하였다. Division은 이진수 나누기 이진수를 활용하여 구현하였다. Arithmetic 어셈블리 코드 검증 과정에서 data가 16진수, floating-point로 표기됨에 따라, 이를 10진수로 보여주는 별도의 변환기가 필요하였다. 가우스 조던 소거법을 통한 역 행렬 풀이 도중에 leading one이 0인 상황에는 swap row label로 점프하여 두 행을 바꿔준다.

본 프로젝트의 성능을 향상시키기 위하여 사용한 방법이다. Branch는 3개의 state을 소모한다. 반복 문의 기능을 구현할 때 branch를 자주 활용했는데, 반복 횟수가 상수로 정해진 floating point division에서는 반복 문의 operation을 반복 횟수만큼 기술하였다. 반복 문 구현 시 반복 횟수 정보를 레지스터에 저장한다. 그리고 CMP 명령어를 사용하였다. 이를 S suffix를 활용하였더니, N, Z, C, 그리고 V flag를 결과에 따라 업데이트 하여 CMP 명령어대신에 사용하여 state을 줄일 수 있었다. Barrel shift를 많이 응용하였다. 예로, *MOV r1, r1, LSL #2*와 *ADD r9, r9, r8*은 *ADD r9, r9, r8, LSL #2*로 줄일 수 있다. Rule of thumb에 따라 conditional sequence가 3개 또는 그보다 작다면 branch 대신 conditional execution을 사용하였다.

본 프로젝트를 구현하면서 제일 좋은 성능을 어떻게 낼 수 있을까 고민하였다. Code size는 평가 대상이 아니었으므로 반복 문 사용없이 코드를 구현하는 것이 좋은 방법이었다. 그러나, 코드의 양이 너무 방대하여, 코드 관리가 어려웠다. 그래서, 성능과 코드 가독성 둘 다 챙기는 것이 좋은 것이 아닐까 생각했다. 본 프로젝트를 통하여 컴퓨터 구조의 기초적인 지식을 얻고 실질적인 구현 능력을 배양할 수 있었다.

VI. Reference

[1] Binary division(이진 나눗셈): https://blastic.tistory.com/136
[2]부동 소수점의 산술 연산:
https://m.cafe.daum.net/kpucomjjang/D7M4/30?q=D_7_beATEAlUg0&
[3] IEEE-754 Floating Point Converter:
https://www.h-schmidt.net/FloatConverter/IEEE754.html