어셈블리프로그램 설계 및 실습

프로젝트 제안서

Floating Point Inverse Matrix

학 과: 컴퓨터정보공학부

담당교수: 이준환 교수님

실습분반: 목요일 6, 7

학 번: 2015722031

성 명: 박태성

1. Title & Object

A. Title

Term project – Floating Point Inverse Matrix

B. Object

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

2. Algorithm

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. (프로젝트 구현에서는 역 행렬이 없는 경우는 고려하지 않는다.)

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. Floating Point Addition

Sign bit, exponent bits, Mantissa(Fraction) bits에 대해 추출한다.

Mantissa 앞에 1을 붙임.

Exponent를 같게 조정한다. (Exponent가 작은 쪽을 큰 쪽으로 조정한다.)

Mantissa끼리의 덧셈을 진행한다.

계산 결과 mantissa를 normalize한다.

이에 맞게 exponent값을 조절한다.

예로, 1.1100*2^4 + 1.1000*2^2

= 1.1100*2^4 + 0.0110*2^4

= 10.0010*2^4

= 0.1000*2^6 or 1.0001*2^5

D. Floating Point Subtraction

Sign bit, exponent bits, Mantissa(Fraction) bits에 대해 추출한다.

Mantissa 앞에 1을 붙임.

Exponent를 같게 조정한다. (Exponent가 작은 쪽을 큰 쪽으로 조정한다.)

뺄셈을 진행한다.

계산 결과 mantissa를 normalize한다.

이에 맞게 exponent값을 조절한다.

E. Floating Point Multiplication

Exponent를 같게 조정하지 않는다.

Mantissa끼리 곱한다.

Exponent끼리 더한다.

소수점 이하 자리는 정해진 자리에서 반올림한다.

계산 결과 Mantissa를 normalize한다.

Sign bit는 두 operand의 sign bit의 exclusive or한다.

x = mx*2^a, y = my*2^b

X * y = (mx*my)*2^ (a + b)

예로, x = 1.000*2^(-2), y = -1.010*2^(-3)

1.000 * -1.010 = -1.010000

(-2) + (-3) = -5

2^ (-2) + 2^ (-3) = 2^ (-5)

-1.0100 * 2 ^ (-5)

F. Floating Point Division

Exponent를 같게 조정하지 않는다.

Mantissa끼리 나눈다.

Exponent끼리 뺀다.

소수점 이하 자리는 정해진 자리에서 반올림 한다.

계산 결과 Mantissa를 normalize한다.

Sign bit는 두 operand의 sign bit의 exclusive or한다.

x = mx*2^a, y = my*2^b

X/y = (mx/my)*2^ (a-b)

예로, x = 1.000*2^(-2), y = -1.010*2^(-1)

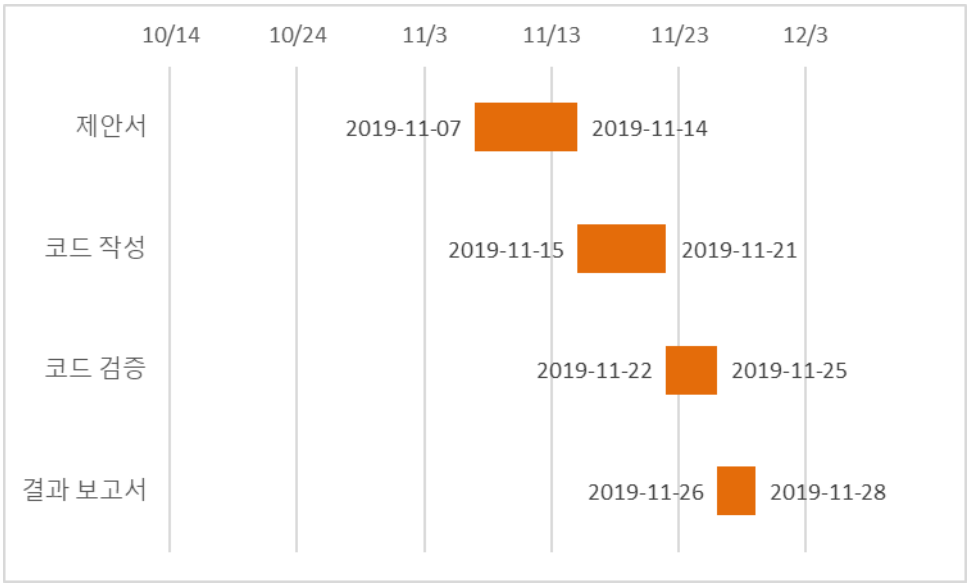
1.000 / -1.010 = -0.1101

(-2) – (-1) = -1

2^ (-2) + 2^ (-1) = 2^ (-1)

-0.1101 * 2 ^ (-1)

3. Schedule



4. Expected Problem

사용 가능한 레지스터의 개수가 제한되어 있다. 행렬의 크기가 커지면 register의 개수가 부족할 것이다. 따라서, 메모리에 data를 쓰기 시작해야 한다. 그러면 state가 증폭할 것으로 예상한다. 가능한 메모리 접근을 피하기 위하여 multiple data transfer를 응용하겠다. LDR 또는 STR pre- or post-indexed를 사용하는 것도 2가지 task를 한 개의 instruction으로 수행할 수 있다. 또한, 어떤 register가 data를 오래 저장하고 있다면 나중에 다신 계산하여 저장하는 것이 cycle 측면에서 효율적이다.

MUL, DIV 명령어는 속도가 느리고 소모되는 state 크다. 따라서, sequence of arithmetic instructions을 대신 사용하여 속도를 빠르게 한다. Barrel Shift를 많이 응용하겠다.

가우스 조던 소거법을 통한 역 행렬 풀이 도중에 leading one이 1이여야하는데 0인 상황이면 다른 열과 바꾼다.

5. Verification strategy

Code size와 state 수를 보면서 optimal한 code를 작성하겠다.

Debugging을 통해 register와 메모리에 정확한 값이 저장/ 이동 되는지 확인하겠다.

6. 참고 문헌

어셈블리프로그램 설계 및 실습/ 이준환/ 광운대학교/ 2019-2학기/ floating point

부동소수점의 산술 연산:

https://m.cafe.daum.net/kpucomjjang/D7M4/30?q=D_7_beATEAlUg0&