

효율적인 로그와 지수 연산을 위한 듀얼 페이즈 명령어 구조

김준서* · 이광엽* · 광재창**

*서경대학교 컴퓨터공학과

**서경대학교 컴퓨터과학과

A Efficient Calculation for log and exponent with A Dual Phase Instruction Architecture

Jun-seo Kim* · Kwang-yeob Lee* · Jae-Chang Kwak**

*Department of Computer Engineering, Seo Kyeong University

**Department of Computer Science, Seokyeong University

E-mail : junseo@skuniv.ac.kr

요 약

본 논문은 작은 사이즈가 요구되는 제한적인 모바일 환경의 프로세서에서 별도의 연산기 없이 제안된 Dual Phase^[1] 명령어 구조를 이용해 효율적인 로그와 지수 연산이 가능한 방법을 제안한다. Floating Point 자료형의 지수부와 실수부를 추출하는 명령어 세트와 테일러 급수 전개를 이용해 로그의 근사치를 계산하여 24비트 단정도 부동 소수점을 연산하고, Dual Phase 명령어 구조를 활용해 명령어 실행 사이클을 줄였다. 제안된 구조는 별도의 연산기를 두는 구조보다 작은 사이즈를 유지하면서 성능저하를 33%까지 최소화 할 수 있는 구조이다.

ABSTRACT

This paper proposes efficient log and exponent calculation methods using a dual phase instruction set without additional ALU unit for a mobile environment. Using the Dual Phase Instruction set, it extracts exponent and mantissa from expression of floating point and calculates 24bit single precision floating point of log approximation using the Taylor series expansion algorithm. And with dual phase instruction set, it reduces instruction execution cycles. The proposed Dual Phase architecture reduces the performance degradation and maintain smaller size.

키워드

로그 연산, 지수 연산, 듀얼 페이즈 명령어 구조.

1. 서 론

최근 수년간 컴퓨터 산업의 성장으로 모바일 기기들이 눈에 띄게 발전하였다. 초기에 비교적 간단한 연산만을 수행하던 모바일 기기들이 이제는 사용자의 요구에 발맞춰 복잡한 연산 능력을 필요로 하게 되었고, 이러한 복잡한 연산을 위한 별도의 연산기는 필수적인 요소가 되었다. 이러한 연산기는 프로세서의 부담을 줄여주며 빠른 응답

속도로 프로세서 전체의 성능을 향상시키는 중요한 요소이다. 그러나 로그와 지수 연산기들은 연산을 빠르게 수행하기 위해서 많은 양의 LUT(Look-up table)메모리와 여러개의 곱셈기, 덧셈기를 필요로 하며,^[2] 별도의 장치들은 작은 사이즈가 요구되는 모바일 환경에서 칩의 사이즈를 증대 시키는 주요 요인 중 하나이다.

본 논문은 작은 사이즈가 요구되는 모바일 환경에서 로그와 지수 연산을 별도의 LUT나 연산

기 없이 테일러 급수 확장 이용하여 로그 연산을 수행하고, 뉴턴-랩슨 반복법을 이용하여 지수 연산을 수행하며, 듀얼 페이즈 명령어 구조를 활용해 연산 사이클을 줄여 성능 저하를 줄이는 방법을 제안한다.

II. 수학적 접근 방법

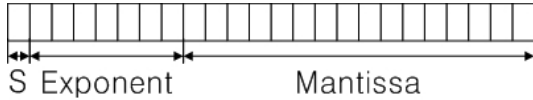


그림 1. 24비트 부동 소수점의 구조

24비트 부동 소수점^[3]은 그림(1)과 같이 Sign 1비트, Exponent 7비트, Mantissa 16비트로 이루어져 있다. 이때 부동소수점 A와 $\log_2 A$ 에 관한 식은 다음과 같이 나타낼 수 있다.

$$A = 2^{\text{exponent}} \times \text{Mantissa} \quad (1)$$

$$\log_2 A = \text{exponent} + \log_2 \text{Mantissa}$$

한편 $\ln x$ 에 관한 테일러 급수의 전개는 아래와 같다.

$$\ln(1+x) = \sum_{n=1}^{\infty} \frac{-(-1)^n}{n} x^n \quad (2)$$

$$= x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots \quad (\text{for } -1 < x \leq 1)$$

$$\log_2(1+x) = (x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots) \times \log_2 e \quad (3)$$

for $(-1 < x \leq 1)$

이때, 식 (1)에서 $1 \leq \text{Mantissa} < 2$ 이므로 테일러 급수를 이용해 $\log_2 \text{Mantissa}$ 를 구한다면 log에 대한 부동소수점의 근사치를 얻을 수 있다.

24비트 부동 소수점의 지수의 관한 식은 다음과 같이 나타 낼 수 있다.

$$a^x = 2^{x \log_2 a} \quad (4)$$

$$= 2^{\text{integer}} \times 2^{\text{fraction}} = 2^{\text{exponent}} \times \text{Mantissa}$$

여기서 Mantissa의 값을 구한다면 24비트 부동 소수점의 exponent값과 mantissa값을 모두 얻을 수 있다. 이 값을 구하기 위하여 다음과 같은 뉴턴-랩슨^[4] 접근방법을 사용하였다.

$$x_{n+1} \approx x_n - ((\log_2 x_n - \text{fraction}) x_n \ln 2) \quad (5)$$

여기서 $2^{\text{fraction}}, 0 \leq \text{fraction} < 1$ 이므로 빠르게 Mantissa에 접근하기 위해서 초기 x값을 fraction+1로 시작하였다.

III. 듀얼 페이즈 명령어 구조

2장에서 살펴본 수학적 접근방법은 보편적이고 정확한 연산방법이지만 근사치를 구하기 위해 많은 반복과 연산이 요구되고 하드웨어 구현시 많은 연산기가 필요해 일반적으로 사용되지 않는 구조이다.^{[5][6]}

듀얼 페이즈 명령어 구조(Dual Phase Instruction Architecture)는 128비트 가변길이의 명령어 구조로써 최대 4개의 소스 오퍼랜드(Source operand)와 최대 2개의 데스티네이션 오퍼랜드(Destination operand)를 갖는 명령어 구조이다. 이러한 구조의 장점으로서는 32비트 길이의 소스 오퍼랜드를 하나의 실행 사이클에 최대 16개까지 연산할 수 있으며 데스티네이션은 32비트 길이의 소스 오퍼랜드를 최대 8개까지 저장할 수 있다. 또한 각각의 페이즈 마다 서로 다른 연산기를 사용할 수 있는 구조이다.

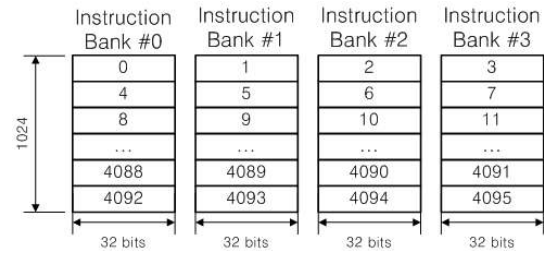


그림 2. 명령어 레지스터 구조

그림 2는 명령어 레지스터 구조로 최대 128비트 가변길이 명령어의 추출(Instruction Fetch)을 위한 뱅크 구조를 나타낸다. 서로 다른 네 개의 뱅크에 저장되어 있는 유닛 명령어들의 집합으로 각 명령어의 End Bit에 따라 최대 128비트까지 동시에 명령어 추출이 가능하다. 또한 이 구조에서 동시에 추출된 명령어를 서로 다른 연산기로 동시에 처리하는 병렬 처리가 가능하다.

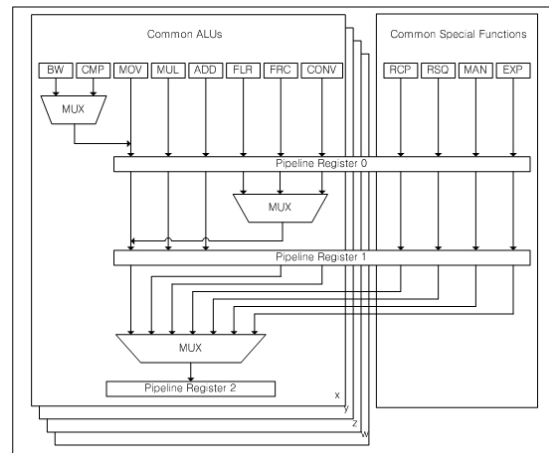


그림 3. 내부 연산기 구조

위의 그림 3 에서 보여지는 것과 같이 각 페이즈 마다 빈번하게 쓰이는 연산들은 XYZW 각 컴퍼넌트 별로 구현되어 있으며 그렇지 않은 RCP(reciprocal), MAN(mantissa), EXP(exponent)와 같이 빈번하게 쓰이지 않는 연산들은 특수 연산(Special Functions)이라는 이름으로 하나씩만 구현되어 있다. 위의 연산기는 2개의 명령어를 페이즈 #0과 페이즈 #1로 나뉘어 서로 다른 연산을

동시에 처리하는 것이 가능하다.

추출된 명령어를 정렬하여 입력된 명령어는 각 페이지당 최대 2개의 유닛 명령어를 할당받아 실행 하게 되며 모든 유닛 명령어는 각 페이지마다 2개씩 최대 4개의 유닛 명령어가 동시에 수행될 수 있다.

이러한 병렬 처리가 가능한 MIMD(Multiple Instruction Multiple Data)구조는 일반적인 SIMD(Single Instruction Multiple Data) 구조에 비해 명령어 실행 사이클을 대폭적으로 줄일 수 있는 구조이다. 이 구조에서 테일러 급수 전개를 이용해 24비트 부동 소수점의 지수와 가수를 추출하는 명령어 세트로 로그 연산을 할 때, 연산 과정에서 여러 연산기를 Dual Phase로 병렬처리 하면 명령어 실행 사이클을 줄여 로그 롬(Log ROM) 및 멱승 롬(Power ROM)의 부재로 야기된 추가 연산으로 인한 성능 저하를 최소화 하는 것이 가능하다.

IV. 구현 및 검증

부동 소수점의 특성을 이용한 로그의 연산은 $1 \leq \text{Mantissa} < 2$ 범위 내에서 이루어지며 지수의 연산은 2^{fraction} , $0 \leq \text{fraction} < 1$ 의 범위에서 이루어진다. 모든 로그와 지수의 연산은 이 범위에서 이루어지며, 로그 롬(Log ROM)이나 멱승 롬(Power ROM)^[7] 없이 동적으로 연산이 이루어진다. 이러한 연산은 별도의 하드웨어 없이 부동 소수점의 지수부(Exponent)와 가수부(Mantissa)를 추출하는 명령어 세트로 컴파일러 단계에서 수행하는 것이 가능하다.

아래의 그림 4와 그림 5는 각각 테일러 급수를 이용해 얻은 로그 값과 그 오차를 보여준다.

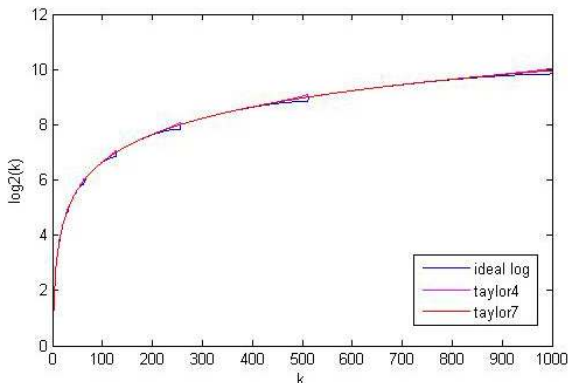


그림 4. 시뮬레이션을 통해 얻은 로그 결과

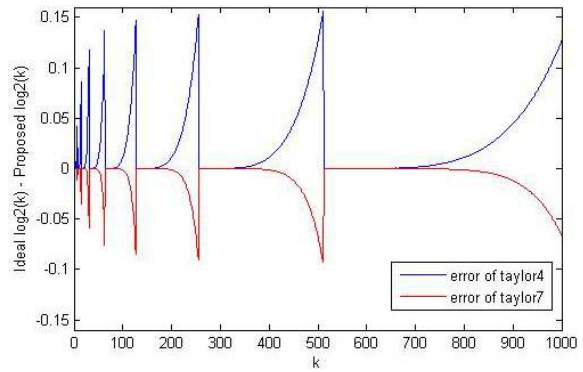


그림 5. 시뮬레이션을 통해 얻은 로그의 오차

로그의 연산은 제안된 듀얼 페이지 구조를 활용하여 로그 연산시 테일러 4행까지는 7 실행 사이클에, 테일러 7행까지는 12 실행 사이클에 결과를 얻을 수 있다. 이는 듀얼 페이지 연산을 하지 않았을 때, 13실행 사이클, 20 실행 사이클에 비해 각각 약 46%, 40% 향상된 속도이다.

아래의 그림 6과 그림 7은 각각 지수 연산 시 사용되는 범위의 수에 대한 로그 값과 로그의 오차를 보여준다. 그림에서 보이는 $0 \leq k < 1$ 의 범위는 지수 연산 시 사용되는 범위이다.

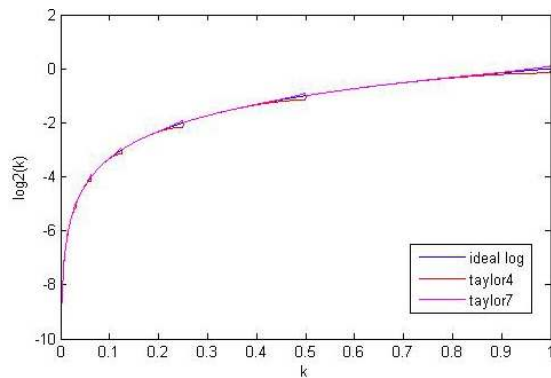


그림 6. 시뮬레이션을 통해 얻은 로그 결과.

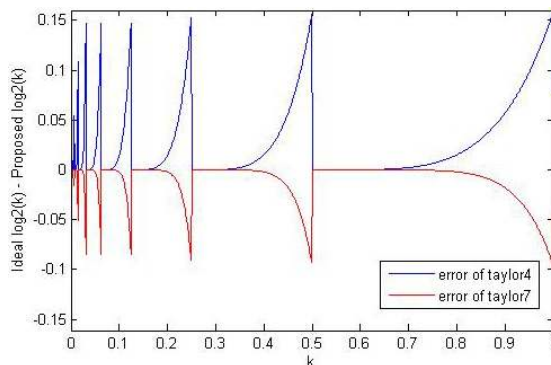


그림 7. 시뮬레이션을 통해 얻은 로그의 오차

아래의 그림8과 그림 9는 뉴턴-랩슨 접근법을 이용해 얻은 지수의 결과와 오차를 보여준다.

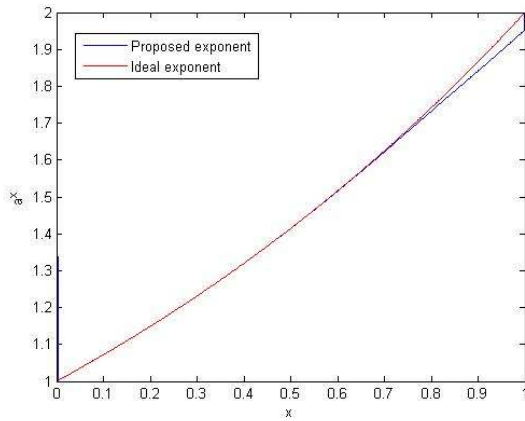


그림 8 시뮬레이션을 통해 얻은 지수 결과

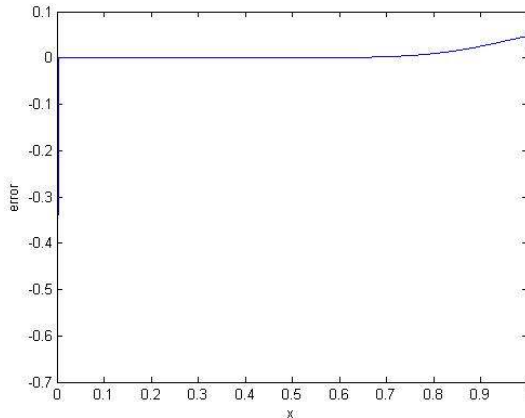


그림 9. 시뮬레이션을 통해 얻은 지수의 오차

그림 8과 그림 9에 나타난 결과는 뉴턴-랩슨 접근법을 이용해 지수 값을 연산한 결과이다. 뉴턴-랩슨 접근법은 한번의 로그 연산을 포함해 총 18 명령어 사이클에 연산이 가능하다. 이 과정에서 결과값에 빠르게 수렴하기 위해 최초의 x 값을 $\text{fraction}+1$ 에서 시작하였다. 지수의 연산과정에서 두 번의 뉴턴-랩슨 접근과 한 번의 로그 연산을 포함해 총 60 실행 사이클에 결과를 얻을 수 있다. 동일한 연산을 일반적인 SIMD 명령어로 하였을 경우에 로그 연산이 20 실행 사이클, 뉴턴-랩슨 접근이 한 번의 로그 연산을 포함하여 28 실행 사이클에 실행되어 지수 연산 시 두 번의 뉴턴-랩슨 접근과 한 번의 로그 연산을 포함하여 90 실행 사이클에 결과를 얻을 수 있다. 듀얼 페이즈 명령어 구조는 일반적인 SIMD 연산에 비해 약 33% 향상된 속도를 보인다.

V. 결 론

로그 롬(Log ROM)과 멱승 롬(Power ROM)을 사용하는 기존의 연산기는 빠른 응답시간을 가지

지만 많은 양의 메모리가 필요하여 칩의 크기를 증가시키기 때문에 작은 크기가 요구되는 모바일 프로세서에서 구현하기에는 크기의 부담이 있다.

본 논문에서는 듀얼 페이즈 명령어 구조에서 효율적으로 로그와 지수값을 연산하는 방법을 연구하였다. 제안된 구조의 로그 및 지수 연산기는 24비트 부동 소수점의 지수부와 실수부를 추출하는 명령어 세트로 칩 크기 증가의 주 요인인 로그 롬과 멱승 롬을 배제하여 크기의 부담을 줄이고, 그로 인한 성능 저하를 듀얼 페이즈 명령어 구조의 특성을 이용해 일반적인 SIMD 명령어 구조에 비하여 로그 연산은 40%, 지수 연산은 33% 적은 명령어 실행 사이클로 연산의 성능 저하를 개선하였음을 보여준다.

감사의 글

본 논문은 서울시 산학연 협력사업(10560)의 지원을 받았으며 IDEC의 장비를 활용하였습니다.

참고문헌

- [1] Woo-Young Kim, A Design of a Shader based on the Variable-Length Instruction for a Mobile GP-GPU
- [2] N. Takagi, "Powering by a table look-up and a multiplication with operand modification," IEEE trans. Comput., vol. C-47, no.2, pp.152-161, Feb.1998
- [3] IEEE Standards Association, <http://standards.ieee.org>
- [4] Richard L. Burden, Numerical Analysis
- [5] Israel Koren, Computer Arithmetic Algorithms, Jogn Wiley & Sons Inc. 1993
- [6] Hideyuki kabuo, et al., "Accurate rounding scheme for the Newton-Raphson method using redundant binary representation," IEEE trans. on Computers vol.43, no.1, pp.43-51, jan.1994
- [7] Ki-Hyun Park, A Design of Accurate High-Speed Reciprocal Square Roots