논문 2004-41CI-5-3

고속 십진 나눗셈을 위한 혼합 알고리즘

(Mixed Algorithm for Fast Decimal Division)

권 순 열*, 최 종 화*, 김 용 대**, 한 선 경**, 유 영 갑**

(Soonyoul Kwon, Jonghwa Choi, Yongdae Kim, Seonkyoung Han, and Younggap You)

요 약

본 논문은 십진 나눗셈에서 연산 속도를 향상시키기 위해 혼합 나눗셈 알고리즘을 제안한다. 이진수 체계에서는 비복원 알고리즘이 복원 알고리즘에 비해 항상 작은 횟수를 갖지만 십진 연산에서는 몫의 값에 따라 연산 횟수가 달라진다. 십진수는 한 자리로 나타낼 수 있는 수의 범위가 0~9 이므로 현재 부분 나머지의 절대 값과 이전 부분 나머지의 절대 값을 비교하여 이전 부분 나머지의 절대 값이 현재 부분 나머지의 절대 값 보다 크면 비복원 알고리즘을 선택하고 작으면 복원 알고리즘을 선택함으로써 연산 횟수를 줄일 수 있다. 몫이 64 자리일 경우 제안한 혼합 알고리즘은 복원 알고리즘에 비해 80.9%의 연산 횟수를 줄였고 비복원 알고리즘에 비해 64.5%의 연산 횟수를 줄였다.

Abstract

In this paper, we proposed a mixed algorithm to improve decimal division speed. In the binary number system, nonrestoring algorithm has a smaller number of operation than restoring algorithm. In decimal number system, however, the number of operations differs with respect to quotient values. Since one digit ranges 0 to 9 in decimal, the proposed mixed algorithm employs both nonrestoring and restoring algorithm considering current partial remainder values. The proposed algorithm chooses either restoring or nonrestoring algorithm based on the remainder values. The proposed algorithm improves computation speed substantially over a single algorithm decreasing the number of operations.

Keywords: decimal division, BCD, restoring, nonrestoring

I. 서 론

실생활에서 사용하는 연산은 일반적으로 십진 기반으로 행하여진다. 그러나 컴퓨터에서의 계산은 십진이 아닌 이진으로 이루어지는데 그 이유는 십진 연산이 이진 연산보다 비트 효율이나 연산 속도 면에서 효과적이지 못하기 때문이다[1]. BCD(binary coded decimal)는 십진수를 표현하는데 가장 많이 사용되는 인코딩 방식이나 비트 효율이 낮기 때문에 이를 향상시킨 인코딩 방식들이 나오고 있다[2]. 연산 속도를 향상시키기 위한 알고리즘은 이진수에서는 SRT가 대표적이나 십진수에서는 찾아보기 힘들다[3,4].

이진 나눗셈 연산은 소수점 이하 연산에서 십진수를 이진수로 변환할 때 발생하는 변환 오차로 인하여 정확한 몫을 얻을 수 없다. 부동 소수점을 이용하여 변환 오차를 상당히 줄일 수 있지만 오차를 완전히 제거할 수는 없다[1,5,6]. 이러한 변환 오차를 없애기 위해서는 십진수 체계를 그대로 이용한 연산을 수행하면 된다. 본 논문에서는 변환 오차가 발생하지 않는 십진 나눗셈을 기반으로 가/감산 연산 횟수를 줄인 혼합 나눗셈 알고리즘을 제시한다. 이 알고리즘은 복원/비복원 알고리즘을 적절하게 선택하여 연산 속도를 높이는 것이다.

본 논문은 다음과 같이 구성된다. Ⅱ장에서 십진 나눗셈 알고리즘인 복원과 비복원 알고리즘을 설명한다. Ⅲ장에서 제안한 십진 혼합 알고리즘과 가/감산 연산 블록을 살펴보고, Ⅳ장에서는 연산 횟수 비교 및 검증을 살펴본다. 마지막으로 Ⅴ장에서 본 논문의 결론을

* 학생회원, ** 정회원, 충북대학교 정보통신공학과
(Department of Information & Communication Engineering, Chungbuk Nat'l University)

※ 본 연구는 산업자원부의 지역혁신 인력양성사업의 일부 지원으로 수행되었음.

접수일자: 2004년6월24일, 수정완료일: 2004년8월9일

맺는다.

Ⅱ. 십진 나눗셈 알고리즘

이 장에서는 십진 복원 및 비복원 알고리즘을 설명한다. 1절에서는 복원 나눗셈 알고리즘을 설명하고 2절에서는 비복원 나눗셈 알고리즘을 설명한다.

1. 복원 나눗셈 알고리즘

복원 나눗셈 알고리즘은 뺄셈과 쉬프트의 반복 연산 과정이다. 피제수(X)에서 제수(D)를 뺀 결과 부분 나머지(p_i)가 양수이면 연속하여 뺄셈을 수행하며 부분 나머지 값이 음수이면 부분 나머지에 제수를 더하는 복원 연산이 필요하다. 몫(q_i)의 값은 뺄셈을 수행할 때마다 1 을 증가시키며 복원 과정인 덧셈 연산을 수행할 때 1 을 감소시킨다. 복원 과정 이후 부분 나머지와 몫을 한 자리(4비트) 좌 쉬프트하여 다음 자리 몫을 결정한다. 십진 복원 나눗셈 알고리즘을 정리하면 표 1과 같다. 초기의 부분 나머지(p_{-1})는 피제수(X)가 되고 초기의 몫(q_{-1})은 0 으로 초기화 된 후 원하는 몫의 값을 얻을 때까지 반복 횟수(m)만큼 반복하게 된다.

십진 복원 알고리즘의 동작 순서를 그림 1에 나타내었다. 초기 입력 값으로 피제수(dividend)와 제수(divi-sor), 그리고 반복 횟수를 입력으로 받는다. 여기서, 초기의 부분 나머지(partial remainder)는 피제수가 된다. 부분 나머지의 부호를 검사하여 양수이면 부분 나머지에서 제수를 빼고 몫의 값을 1 증가시킨다. 뺄셈 결과인 부분 나머지가 음수가 될 때 까지 계속하여 뺄셈을 수행하고 몫의 값은 1 증가시킨다. 만약, 뺄셈 결과인 부분 나머지가 음수이면 부분 나머지에 제수를 더하는 복원 과정을 수행하고 몫의 값을 1 감소시킨다. 이런 복원 과정을 거친 후에는 반드시 몫을 한 자리 좌 쉬프트하고 반복 횟수를 1 증가시킨다. 현재의 반복 횟수가 원하는 반복 횟수와 같을 경우는 현재의 몫과 나머지를 출력하고 종료하게 된다. 현재의 반복 횟수가 원하는 반복 횟수보다 작을 경우는 부분 나머지를 한 자리 좌 쉬프트한 후 다시 반복하게 된다.

2. 비복원 나눗셈 알고리즘

비복원 알고리즘은 복원 알고리즘과 유사하게 뺄셈과 쉬프트 연산의 반복 과정이다. 복원 알고리즘은 부분 나머지(p_i)의 부호가 음수일 경우 부분 나머지에

표 1. 복원 알고리즘 의사 코드
Table 1. Pseudo code of a restoring algorithm.

$p_{-1} = X$
$q_{-1} = 0$
for $i = 0$ to $m-1$
  if $p_i > 0$ then
    $p_i = 2^4 p_{i-1} - D$
    $q_i = q_{i-1} + 1;$
  else
    $p_i = 2^4 p_{i-1} + D;$
    $q_i = q_{i-1} - 1;$
  $i++;$
return $(p, q);$

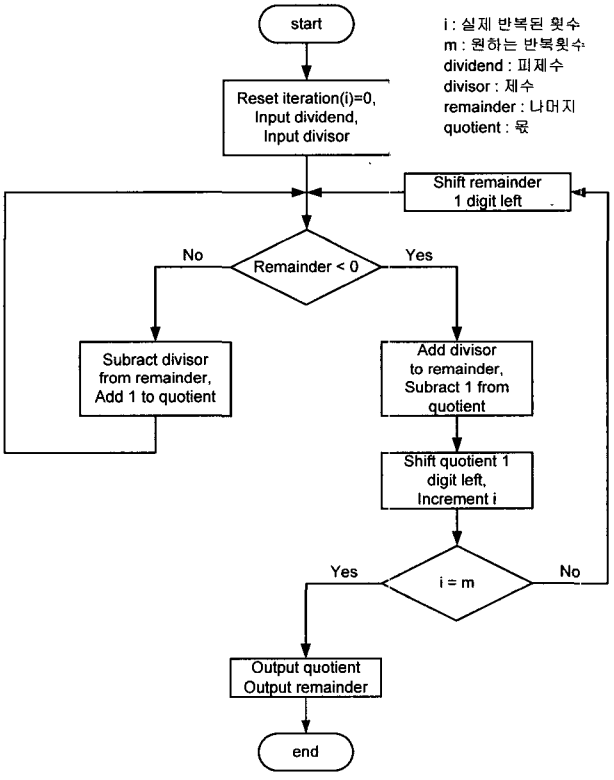

그림 1. 복원 나눗셈 알고리즘
Fig. 1. Restoring division algorithm.

제수(D)를 더하는 복원 과정이 필요하나 비복원 알고리즘은 부분 나머지의 부호가 다시 양수가 될 때 까지 부분 나머지에 제수를 더하게 된다. 십진 비복원 나눗셈 알고리즘을 정리하면 표 2와 같다.

비복원 나눗셈 알고리즘의 연산 순서는 그림 2에 나타내었다. 초기 입력 값으로 피제수와 제수, 그리고 반

표 2. 비복원 알고리즘 의사 코드
Table 2. Pseudo code of a nonrestoring algorithm.

$p_{-1} = X$
$q_{-1} = 0$
$p_0 = 2^4 p_{-1} - D$
for $i = 0$ to $m-1$
  if not_change then
    if $p_i > 0$ then
      $p_i = p_i - D;$
      $q_i = q_i + 1;$
    else
      $p_i = p_i + D;$
      $q_i = q_i - 1;$
  else
    $p_i = 2^4 p_{i-1};$
    $q_i = 2^4 q_{i-1} + 1;$
    $i++;$
return $(p, q);$

※ not_change:부분 나머지의 부호가 변화하지 않음

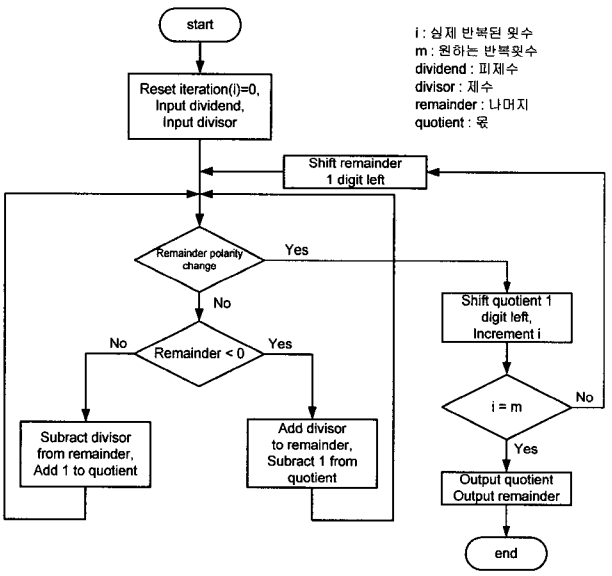

그림 2. 비복원 나눗셈 알고리즘
Fig. 2. Nonrestoring division algorithm.

복 횟수를 입력으로 받는다. 부분 나머지의 부호가 이전의 부분 나머지와 비교하여 같다면 부분 나머지의 부호에 따라 다음과 같이 수행된다. 부분 나머지가 양수이면 부분 나머지에서 제수를 빼고 몫의 값을 1 증가시킨다. 부분 나머지가 음수이면 부분 나머지에서 제수를 더하고 몫의 값을 1 감소시킨다. 부분 나머지의 부호가 바뀔 때까지 위의 과정을 반복한다. 만약 부분 나머지의 부호가 바뀌면 몫을 한 자리 좌 쉬프트하고 반복 횟수를 1 증가 시킨다. 현재의 반복 횟수가 원하는 반복 횟수와 같을 경우는 현재의 몫과 나머지를 출력하고 종료하게 된다. 현재의 반복 횟수가 원하는 반복 횟수보다 작을 경우는 부분 나머지를 한 자리 좌 쉬프트한 후 다시 반복하게 된다.

Ⅲ. 제안한 십진 혼합 알고리즘과 가/감산 연산 블록

이 장에서는 제안한 십진 혼합 알고리즘과 이를 위한 가/감산 연산 블록을 살펴본다. 1절에서는 가/감산 연산 횟수를 줄이기 위해 복원 알고리즘과 비복원 알고리즘을 선택하는 십진 혼합 알고리즘을 설명한다. 본 알고리즘은 십진 가/감산 연산 블록이 필요하므로 이를 위해 단일 회로로 구성된 가/감산 연산 블록을 2절에서 설명한다.

1. 제안한 십진 혼합 알고리즘

이진 나눗셈의 경우 복원 알고리즘과 비복원 알고리즘의 연산 횟수의 차이는 최대 n비트 차이가 난다. 그러나 십진인 경우 기수(radix)가 10 이므로 한 자리가 가질 수 있는 수의 범위는 0~9 이므로 복원과 비복원 알고리즘에서 연산 수행 횟수가 다르다. 각 자리의 몫을 결정할 때 작은 연산 횟수를 가지는 알고리즘을 선택하면 연산 횟수를 줄일 수 있다. 복원과 비복원 알고리즘의 선택은 다음과 같다. 가/감산 연산 수행 후 부분 나머지의 부호가 이전 부분 나머지의 부호와 다를 경우 두 알고리즘 중 하나를 선택하게 된다. 이전 부분 나머지의 절대 값과 현재 부분 나머지의 절대 값을 비교하여 이전 부분 나머지의 절대 값이 크면 비복원 알고리즘을 선택하고 현재 부분 나머지의 절대 값이 크면 복원 알고리즘을 선택한다. 제안한 십진 혼합 알고리즘을 정리하면 표 3과 같다.

제안한 혼합 알고리즘의 연산 순서는 그림 3에 나타내었다. 초기 입력 값으로 피제수와 제수, 그리고 반복 횟수를 입력으로 받는다. 부분 나머지의 부호가 이전의 부분 나머지와 비교하여 같다면 앞서 설명한 비복원 알고리즘에서의 가/감산과 같다. 즉, 부분 나머지의 부호가 양수이면 부분 나머지에서 제수를 빼고 몫의 값을 1 증가 시킨다. 부분 나머지가 음수이면 부분 나머지에서

제수를 더하고 몫의 값을 1 감소시킨다. 부분 나머지의 부호가 바뀔 때까지 위의 과정을 반복한다.

부분 나머지의 부호가 이전 부분 나머지의 부호와 다르면 현재 부분 나머지의 절대 값과 이전 부분 나머지의 절대 값을 비교한다. 현재 부분 나머지의 절대 값이 크면 앞서 설명한 복원 알고리즘에서의 가/감산과 같다. 즉, 부분 나머지의 부호가 양수이면 부분 나머지에서 제수를 빼고 몫의 값을 1 증가 시킨다. 뺄셈 결과인 부분 나머지가 음수가 될 때 까지 계속하여 뺄셈을 수행하고 몫의 값은 1 씩 증가 시킨다. 부분 나머지의 부호가 음수이면 부분 나머지에 제수를 더하고 몫의 값을 1 감소시킨다. 즉, 복원을 시킨 후 몫을 한 자리 좌 쉬프트하고 반복 횟수를 1 증가 시킨다. 현재의 반복 횟수가 원하는 반복 횟수와 같을 경우는 현재의 몫과 나머지를 출력하고 종료하게 된다.

현재 부분 나머지의 절대 값과 이전 부분 나머지의 절대 값을 비교하여 현재 부분 나머지의 절대 값이 작으면 비복원 알고리즘의 연속으로써 몫을 한 자리 좌 쉬프트하고 반복 횟수를 1 증가 시킨다. 현재의 반복 횟수가 원하는 반복 횟수와 같을 경우는 현재의 몫과 나머지를 출력하고 종료하게 된다.

2. 혼합 나눗셈 알고리즘을 위한 가/감산 연산 블록

나눗셈 알고리즘은 뺄셈과 쉬프트 연산의 연속 과정이므로 연산 속도를 향상시키기 위해서는 효율적인 가/감산 연산 블록이 필요하다. 십진 수 체계에서는 이진수와 달리 한 자리가 4 비트로 구성되므로 일정한 크기(weight)를 가지지 않기 때문에 십진수 연산을 위한 가/감산 연산 블록이 별도로 필요하다. 제안한 혼합 알고리즘을 수행하기 위해서는 가산과 감산 연산이 필요하고 이를 위해 가/감산 연산을 하나의 회로로 수행되게 하였다. 가/감산 연산을 하나의 회로로 수행하기 위해 감산 연산시 감수는 9 의 보수 형태를 취하여야 한다. 9 의 보수를 효율적으로 만들기 위한 방법은 Excess -3 코드를 이용하는 것이다[6]. 각 비트 마다 1 의 보수를 취함으로써 9 의 보수를 만들 수 있기 때문에 가/감산 연산시 모든 연산자는 Excess-3 코드를 이용하였다. Excess-3 코드를 이용한 한 자리(4비트) 가/감산 연산 블록을 그림 4에 나타내었다.

선택 신호(sel)는 덧셈과 뺄셈 연산을 선택하는 신호로써 덧셈(sel=0)일 경우 감수를 그대로 연산 입력으로 사용하고 뺄셈(sel=1)일 경우 감수는 9 의 보수 값으로 입력된다. 십진 가/감산 블록은 BCD CLA(Binary Co

표 3. 제안한 혼합 알고리즘 의사 코드
Table 3. Pseudo code of the proposed mixed algorithm.

$p_{-1} = X$
$q_{-1} = 0$
$p_0 = 2^4 p_{-1} - D$
for $i = 0$ to $m-1$
  if change then
    if $p_i > 0$ then
      $p_i = p_i - D;$
      $q_i = q_i + 1;$
    else
      $p_i = p_i + D;$
      $q_i = q_i - 1;$
  else
    if $p_{i-1} > p_i$ then
      $p_i = 2^4 p_{i-1};$
      $q_i = 2^4 q_{i-1} + 1;$
    else
      if $p_i > 0$ then
        $p_i = p_i - D;$
        $q_i = q_i + 1;$
      else
        $p_i = p_i + D;$
        $q_i = q_i - 1;$
        $p_{i+1} = 2^4 p_i;$
        $q_{i+1} = 2^4 q_i + 1;$
        $i++;$
return (p, q);

※ change: 현재 부분 나머지의 부호가 이전 부분 나머지의 부호와 다름

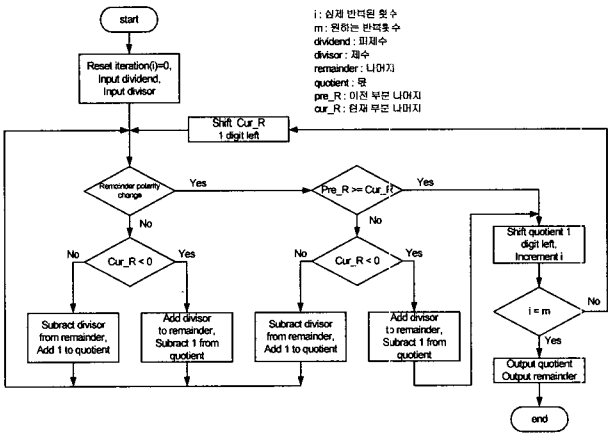

그림 3. 제안한 혼합 나눗셈 알고리즘
Fig. 3. Proposed mixed division algorithm.

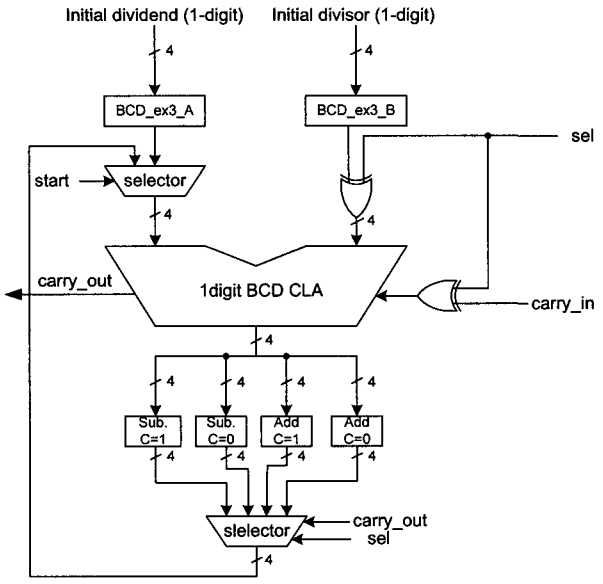

그림 4. 한 자리 가/감산 연산 블록
Fig. 4. One-digit add/subtract block.

-ded Decimal Carry Lookahead)을 이용하여 캐리전파에 따른 지연을 줄였다[5]. 가/감산 연산 결과 값은 Execss-3 코드를 이용하였기 때문에 보정이 필요하다. 보정은 4 가지 경우가 있으며 다음과 같다.

덧셈일 경우 캐리발생 여부에 의해 각각 다른 보정 단계를 거친다. 캐리가 발생하지 않을 경우 연산 결과에서 3 을 빼면 된다. 캐리가 발생할 경우는 연산 결과에 3 을 더하면 된다. Excess-3 코드 연산 결과를 X_i 라고 하고 보정된 결과를 Y_i라고 하면 식 (1)과 같다.

$$Y_i = \begin{cases} X_i - 3 & \text{if } carry_i = 0 \\ X_i + 3 & \text{if } carry_i = 1 \end{cases} \quad (1)$$

뺄셈의 경우 캐리발생의 유무를 확인하여 다음과 같은 보정 단계를 거친다. 뺄셈의 경우는 덧셈과 달리 전단의 캐리에 영향을 받기 때문에 복잡하다. 캐리가 발생하지 않고 전단 캐리가 0 이면 연산 결과에 3 을 빼면 된다. 캐리가 발생하지 않고 전단캐리가 1 이면 2 를 빼면 된다. 캐리가 발생하고 전단 캐리가 0 이면 3 을 더하고 전단 캐리가 1 이면 4 를 더하여 보정을 한다. Excess-3 코드 연산 결과를 X_i라고 하고 보정된 결과를 Y_i라고 하면 식 (2)와 같다.

$$Y_i = \begin{cases} X_i - 3 & \text{if } carry_i = 0 \text{ and } carry_{i-1} = 0 \\ X_i - 2 & \text{if } carry_i = 0 \text{ and } carry_{i-1} = 1 \\ X_i + 3 & \text{if } carry_i = 1 \text{ and } carry_{i-1} = 0 \\ X_i + 4 & \text{if } carry_i = 1 \text{ and } carry_{i-1} = 1 \end{cases} \quad (2)$$

Ⅳ. 연산 횟수 비교 및 검증

기존의 복원 알고리즘만을 이용한 나눗셈의 경우 한 자리의 몫을 계산하기 위하여 최대 11회의 가/감산 연산이 필요하며 평균 6회의 가/감산 연산을 수행한다. 비복원 알고리즘을 이용한 나눗셈의 경우 한 자리의 몫을 계산하기 위하여 최대 10 회의 가/감산 연산이 필요하며 평균 5.5 회의 가/감산 연산을 수행한다[3]. 제안한 혼합 알고리즘은 최대 6 회의 가/감산 연산이 필요하며 평균 3.5 회의 가/감산 연산을 수행한다. 단, 몫이 한 자리일 경우는 세 가지 알고리즘 모두 11번의 가/감산 연산 횟수를 가진다. 그 이유는 몫의 첫 자리 값을 결정할 때 세 가지 알고리즘 모두 뺄셈을 수행하여 몫을 얻기 때문이다. 그리고 나머지가 음수일 경우 세 가지 알고리즘 모두 피제수의 부호가 나머지의 부호와 같도록 해주는 보정 단계를 거치기 때문이다. 이후의 몫 결정 비트부터 위의 최대 연산 횟수만큼 몫의 자리에 비례하여 가/감산 연산 횟수가 증가한다.

세 가지 알고리즘의 가/감산 연산 횟수를 검증하기 위해 C 프로그래밍을 이용하였다. 검증 환경은 Pentium 4 1.8GHz, 메모리 512M를 가진 컴퓨터를 이용하였다. 각 알고리즘의 의사코드를 기반으로 하여 정확한 몫의 값을 얻는데 소요된 연산횟수를 구하였다. 입력 값으로는 피제수가 1~9,999,999 사이의 값을 가지며 제수는 피제수 각각에 대하여 1~피제수 사이의 값을 입력하여 검증하였다. 위의 범위를 가지는 수를 검증하는데 하루 정도 소요되었으며 저장한 데이터 값은 4 Gbyte 정도이다.

세 가지 알고리즘에 대한 규칙성을 찾으면 다음과 같다. 복원 알고리즘은 몫의 자리수(n)에 대하여 11×n 번의 가/감산 횟수를 가지며 비복원 알고리즘은 몫의 자리수(n)가 짝수일 경우는 10×n 번의 가/감산 횟수를 가지고 n=4×i-3 (i=1,2,3,...)일 때 10×n+1 번이며, n=4*i-1 (i=1,2,3...)일 때 10×n-1번의 가/감산 횟수를 가진다.

세 가지 알고리즘이 최대 가/감산 연산 횟수를 가지는 몫의 유형을 살펴보면 다음과 같다. 복원 알고리즘은 각 자리의 몫이 9를 가질 때 최대의 연산 횟수를 가진다. 즉, 몫이 99999...의 형태를 취할 때이며 최소 연산 횟수는 몫이 10000...의 형태를 취할 때이다. 비복원 알고리즘은 몫의 각 자리가 9 와 0 의 반복으로 구성될 때 최대의 연산 횟수를 가진다. 즉, 몫이 90909...의 형태를 취할 때이며 최소 연산 횟수는 몫이 19090...의 값을 가질 때이다. 제안한 혼합 알고리즘은 최상위 몫의 값

이 9 이고 이후 자리의 값이 4 를 가질 때 최대의 연산 횟수를 가진다. 즉, 몫이 94444...의 형태를 취할 때이며 최소 연산 횟수는 비복원 알고리즘과 같은 형태를 가질 때이다. 몫의 값이 7 자리를 넘는 범위에 대하여는 앞에서 찾은 규칙성에 의한 근사 값을 입력 한 후 검증하였다.

가/감산 연산 횟수를 비교해 보면 몫이 1 자리(1~9)일 경우는 세 가지 알고리즘 모두 최대 11 회이고 최소는 3 회이다. 이는 첫 자리 몫 결정은 피제수에서 제수를 빼어야 하기 때문에 모두 같은 연산 횟수를 가진다. 몫이 2 자리일 경우 즉, 10~99 사이일 경우는 복원 알고리즘이 22 회로 가장 많은 가/감산 연산 횟수를 가지며 제안한 혼합 알고리즘이 17 회로 가장 작은 가/감산 연산 횟수를 가진다. 몫이 3 자리일 경우 즉, 100~999 사이일 경우 최대 연산 횟수는 복원 알고리즘이 33 회, 비복원 알고리즘이 29 회, 제안한 알고리즘이 23 회이다. 몫의 자리 수에 대한 세 가지 알고리즘의 최대 가/감산 연산 횟수를 표 4와 그림 5에 나타내었다. 표 4에서 알 수 있듯이 몫의 자리수가 증가할수록 제안한 혼합 알고리즘의 가/감산 연산 횟수와 단일 알고리즘의 가/감산 연산 횟수의 차이는 증가한다.

Ⅳ. 결 론

기존의 이진수 체계는 소수점 이하에서 변환 오차가 발생하게 된다. 이러한 변환오차를 없애기 위하여 십진수 기반의 사칙 연산기가 필요하다. 본 논문에서는 십진 가/감산블록을 이용한 고속 십진 나눗셈 알고리즘을 제안하였다. 기존의 이진 기반 나눗셈 연산에서 복원과 비복원 나눗셈 알고리즘의 가/감산 연산 횟수의 차이는 몫의 비트 크기(n)에 따라 최대 n-1 번이다. 그러나 십진 나눗셈 연산에서는 몫의 범위가 0~9 이므로 비복원 알고리즘과 복원 알고리즘에서의 가/감산 연산 횟수의 차이는 평균 1 회이지만 제수와 부분 나머지의 차이에 따라 최대 9 번이다. 부분 나머지의 부호가 바뀌면 이전 부분 나머지의 절대 값과 현재 부분 나머지의 절대 값을 비교하여 복원과 비복원 알고리즘을 선택함으로써 단일 알고리즘을 사용할 때 보다 가/감산 연산 횟수가 감소함을 확인하였다. 제안한 혼합 알고리즘의 가/감산 연산 횟수는 몫의 자리 수에 비례하여 단일 알고리즘의 가/감산 연산 횟수보다 줄어든다. 몫의 자리가 64 자리일 경우 복원 알고리즘에 비해 80.9%, 비복원 알고리즘에 비해 64.5%의 가/감산 연산 횟수가 줄어들었다.

표 4. 각 알고리즘에 대한 최대 가/감산 연산 횟수
Table 4. The maximum number of add/subtraction operation on each algorithm.

몫 자리 수	복원 알고리즘	비복원 알고리즘	제안한 알고리즘	제안한 알고즘과의 연산 횟수 비교(%)	
				복원	비복원
1	11	11	11	0.0	0.0
2	22	20	17	29.4	17.6
3	33	29	23	43.4	26.1
4	44	40	29	51.7	37.9
5	55	51	35	57.1	45.7
6	66	60	41	60.9	46.3
7	77	69	47	63.8	46.8
8	88	80	53	66.0	50.9
9	99	91	59	67.8	54.2
10	110	100	65	69.2	53.8
16	176	160	101	74.3	58.4
32	352	320	197	78.7	62.4
64	704	640	389	80.9	64.5

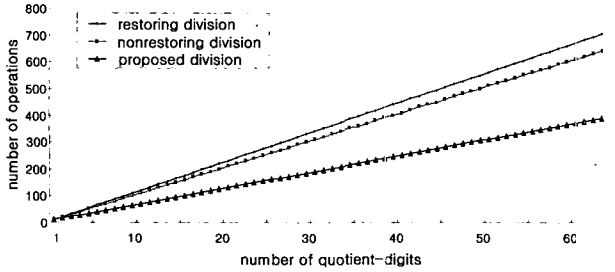

그림 5. 최대 가/감산 연산 횟수 비교
Fig. 5. Comparison of add/subtraction operation.

하드웨어 구현에서 가장 큰 지연을 갖는 블록이 가/감산 블록이므로 가/감산 연산 횟수의 감소는 곧 ALU의 성능을 향상시키게 된다. 제안한 혼합 알고리즘을 십진 ALU에 적용하면 절단 오차를 없앨 수 있고 빠른 십진 나눗셈 연산을 수행 할 수 있다.

참 고 문 헌

[1] M. F. Cowlish, et al., "A decimal floating-point specification," Proc. 15th IEEE Symposium on Computer Arithmetic, pp. 147-154, June 2001.
[2] M. Cowlishaw, "Densely packed decimal encoding," IEE Proc. Comput. Digit. Tech., vol. 149, no. 3, pp. 102-104, May 2002.
[3] S. Hermann, Decimal Computation, Wiley-Interscience Publication, 1974.
[4] P. Behrooz, Computer Arithmetic, Oxford Univer

-sity Press, 2000.
[5] 최종화, 유영갑, "고속 십진 가산을 위한 3초과 코드 carry lookahead설계," 대한전자공학회 논문집, 제40권 CI편 제5호, 241-249쪽, 2003. 9.
[6] 최종화, 한선경, 유영갑, "십진수 계산을 위한 3초과 부호 가감산기 설계," *대한전자공학회 논문집*, 제 40권 CI편 제6호, 348-354쪽, 2003. 11.
[7] M. S. Schmookler and A. Weinberger, "High speed decimal addition," *IEEE Transactions on Computers*, vol. C-20, no. 8, pp. 862-866, August 1971.
[8] K. Israel, *Computer Arithmetic Algorithms*, 2nd Ed, A K Peters, Ltd. 2002.
[9] 권순열, 최종화, 이선영, 유영갑, "Decimal divider using nonrestoring and restoring with shift processing and CLA subtraction processing," *대한전자공학회 2004년도 제 11회 한국반도체 학술대회 논문집*, 제2권, 171-172쪽, 2004.

저 자 소 개

권 순 열(학생회원)
2003년 충북대학교 정보통신 공학과 공학사.
2003년~현재 충북대학교 정보통신 공학과 석사과정 재학중.
<주관심분야: Computer arithmetic, ASIC 설계, Communication system>

최 종 화(학생회원)
2002년 충북대학교 반도체공학과 공학사.
2004년 충북대학교 정보통신 공학과 공학석사.
<주관심분야: VLSI 설계, 연산회로 설계>

김 용 대(정회원)
1990년 충북대학교 정보통신 공학과 공학사.
1993년 충북대학교 컴퓨터공학과 공학석사.
1989년~1998년 신흥기술연구소 팀장.
2000년~현재 충북대학교 정보통신공학과 박사과정 재학 중.
<주관심분야: Computer arithmetic, ASIC 설계, 암호 시스템>

한 선 경(정회원)
1991년 충북대학교 정보통신 공학과 공학사.
1993년 충북대학교 정보통신 공학과 공학석사.
2004년 충북대학교 정보통신 공학과 공학박사.
<주관심분야: Computer arithmetic, Cryptographic system, ASIC 설계>

유 영 갑(정회원)
1975년 서강대학교 전자공학과 공학사.
1975년~1979년 국방과학연구소 연구원.
1981년 Univ.of Michigan, Ann Arbor 전기전산학과 공학석사
1986년 Univ.of Michigan, Ann Arbor 전기전산학과 공학박사
1986년~1988년 금성반도체 (주) 책임 연구원
1993년~1994년 아리조나 대학교 객원교수
1998년~2000년 오레곤 주립대학교 교환교수
1988년~현재 충북대학교 정보통신공학과 교수
<주관심분야: VLSI 설계 및 Test, 고속 인쇄 회로 설계, Cryptography>