논문 2004-41CI-4-4

# 저전력 ARM7 TDMI의 정수 나눗셈 명령어 설계

## (A Design of Interger division instruction of Low Power ARM7 TDMI Microprocessor)

오 민 석*, 김 재 우*, 김 영 훈*, 남 기 훈*, 이 광 엽**

( Min Seok Oh, Young Hoon Kim, Jae Woo Kim, Ki Hoon Nam, and Kwang Youb Lee)

요 약

현재 ARM7 TDMI 마이크로프로세서는 소프트웨어 루틴들의 반복 알고리듬을 사용하여 정수 나눗셈 연산을 처리하고 있어 많은 명령어 수와 긴 수행 시간을 갖는다. 본 논문은 ARM7 TDMI 마이크로프로세서의 연산기능 중 구현되지 않은 정수 나눗셈 연산 기능을 제안하였다. 이를 위해 부호 없는 정수 나눗셈 명령어인 'UDIV' 명령어와 부호 있는 정수 나눗셈 명령어인 'SDIV' 명령어를 새로 정의하였으며, 명령어들의 수행하기 위해 ARM7 TDMI 마이크로프로세서의 데이터 패스에 나눗셈 알고리듬을 적용하였다. 적용한 나눗셈 알고리듬은 비복원 알고리듬이며, 기존의 데이터 패스를 최대한 이용하여 추가되는 하드웨어 유닛을 최대한 줄였다. 제안된 방법을 검증하기 위하여 HDL(Hardware Description Language)을 이용하여 RTL(Register Transfer Level)에서 설계하여 시뮬레이션 하였으며, 현재 ARM7 TDMI 마이크로프로세서의 정수 나눗셈 연산 처리 방법과 제안된 구조에서의 정수 나눗셈 연산 처리 방법을 수행 시간과 수행 명령어 수 측면에서 비교하였으며, 기존의 논문에서 제안한 정수 나눗셈기와 수행 시간과 추가되는 하드웨어 면적을 비교하였다.

Abstract

The ARM7 TDMI microprocessor employ a software routine iteration method in order to handle integer division operation, but this method has long execution time and many execution instruction. In this paper, we proposed ARM7 TDMI microprocessor with integer division instruction. To make this, we additionally defined UDIV instruction for unsigned integer division operation and SDIV instruction for signed integer division operation, and proposed ARM7 TDMI microprocessor data path to apply division algorithm. Applied division algorithm is nonrestoring division algorithm, and additive hardware is reduced using existent ARM data path. To verify the proposed method, we designed proposed method on RTL level using HDL, and conducted logic simulation. we estimated the number of execution cycles and the number of execution instructions as compared proposed method with a software routine iteration method, and compared with other published integer divider from the number of execution cycles and hardware size.

**Keywords :** ARM7 TDMI microprocessor, software routine iteration, nonrestoring division algorithm

## I. 서 론

최근 IP(Intellectual Property)를 이용한 SoC(System On a Chip) 설계가 반도체 설계의 새로운 방안으로 대두되면서 핵심적인 Core IP 개발 및 이를 이용한 SoC 설계가 새로운 산업으로 등장하고 있으며, 정보 통신 분야의 급속한 발전으로 SoC를 통한 임베디드 시스템(Embedded System) 설계 시 보다 복잡한 기능과 다양한 연산, 빠른 처리 속도가 요구되는 Core IP의 사용이 필수적이다. 현재 SoC 설계에 있어 핵심적인 Core IP인 ARM7 TDMI 마이크로프로세서는 정수 처리 코어(Integer Core)로 다양한 정수 산술, 논리연산 기능을 갖고 있으나, 사칙연산 중 하나인 나눗셈 연산을 처리하기 위한 별도의 명령어와 데이터 패스를 갖고 있지 않다. 이는 대부분의 알고리듬은 나눗셈보다는 덧셈, 뺄셈, 또는 곱셈을 더 빈번하게 사용하고, 32bit 나눗셈을 수행하는데 요구되는 하드웨어 유닛은 복잡도와 면적으

* 학생회원, ** 정회원, 서경대학교 컴퓨터공학과
(Department of Computer Engineering, Seokyeong University)

※ 본 연구는 IT_SoC와 IDEC 사업단 지원으로 수행되었습니다.
접수일자: 2003년7월2일, 수정완료일: 2004년6월10일

로 인해 높은 비용을 필요하기 때문이다. 그래서 현재 ARM7 TDMI 마이크로프로세서에서는 소프트웨어 루틴들이 반복 알고리듬을 사용하여 정수 나눗셈을 수행한다. 그러나 정수 나눗셈 연산은 임베디드 시스템에서 데이터 처리와 데이터 구조 구현 시 빈번히 사용되며, 이를 처리위해 소프트웨어 루틴의 수행 시 많은 반복 연산으로 인해 긴 수행 시간을 갖으며, 이는 병목 현상을 초래할 수 있다.

본 논문에서는 ARM7 TDMI 마이크로프로세서의 연산기능 중 구현되지 않은 정수 나눗셈 연산기능을 제안하였다. 이를 위해 이를 위해 부호 없는 정수 나눗셈 명령어인 'UDIV' 명령어와 부호 있는 정수 나눗셈 명령어인 'SDIV' 명령어를 새로 정의하였으며, 나눗셈 연산 명령어의 수행을 위하여 ARM7 TDMI 마이크로프로세서의 데이터 패스에 나눗셈 알고리듬을 적용하였다. 적용한 나눗셈 알고리듬은 비복원 알고리듬이며, 기존의 데이터 패스를 최대한 이용하여 추가되는 하드웨어 유닛을 최대한 줄였다. 현재 ARM7 TDMI 마이크로프로세서의 정수 나눗셈 처리 방법인 소프트웨어 루틴 반복 알고리듬 lib1funcs를 구성하는 명령어의 개수와 총 수행 시간을 본 논문이 제안한 나눗셈 연산 명령어를 통한 정수 나눗셈 처리 방법과 비교하였으며, 기존의 정수 나눗셈기와 수행 시간, 추가되는 하드웨어 면적을 비교하였다. 본 논문의 구성은 서론에 이어 Ⅱ장에서는 현재 ARM7 TDMI 마이크로프로세서의 나눗셈 처리방법인 소프트웨어 루틴 반복에 대해서 자세히 알아보고, Ⅲ장에서는 새로운 정수 나눗셈 연산 명령어 정의 및 기존 ARM7 TDMI 마이크로프로세서 Data Path에 정수 나눗셈 연산 알고리즘의 적용에 대해 기술하였으며, Ⅳ장에서는 제안된 정수 나눗셈 처리 방식과 소프트웨어 루틴 반복 방식, 타 논문에서 제안된 정수 나눗셈기와의 수행시간, 명령어 개수, 면적 측면에서 비교 분석하여, Ⅴ장에서 결론을 맺었다.

## Ⅱ. ARM7 TDMI 마이크로프로세서의 정수 나눗셈 처리 방법

ARM7 TDMI 마이크로프로세서는 정수 나눗셈 연산을 수행하기 위한 별도의 명령어가 정의 되어 있지 않다. 현재 ARM7 TDMI 마이크로프로세서는 쉬프트 연산과 뺄셈 연산, 비교 연산으로 이루어진 소프트웨어 루틴을 반복하여 정수 나눗셈 연산을 수행한다. ARM GCC Complier의 정수 나눗셈 소프트웨어 루틴 라이브

표 1. ARM GCC 컴파일러의 정수 나눗셈 연산 소프트웨어 라이브러리 lib1funcs의 구성: (a) 기본 구성; (b) 라벨 SYM의 구성; (c) 라벨 Lgot_result의 구성; (d) 라벨 Loop3의 구성

Table 1. Integer division software library of ARM GCC complier: (a) The basic scheme; (b) The sheme of label SYM; (c) The scheme of label Lgot_result; (d) The scheme of label Loop3.

```
dividend .req    r0      -- 레지스터 할당
divisor  .req    r1
result   .req    r2
curbit   .req    r3
ip       .req    r12
sp       .req    r13
lr       .req    r14
pc       .req    r15

SYM:             -- 수행 조건 판단
Loop1:           -- 제수, 피제수 초기화
Lbignum:          -- 제수, 피제수 초기화
Loop3:           -- 나눗셈 연산 주 처리
Lgot_result:      -- 결과 저장
Ldiv0:           -- 제수가 0일 때 처리
```

(a)

```
SYM:
        eor     ip, dividend, divisor    *
        mov     curbit, #1
        mov     result, #0
        cmp     divisor, #0
        rsbmi   divisor, divisor, #0     *
        beq     Ldiv0
        cmp     dividend, #0
        rsbmi   dividend, dividend, #0   *
        cmp     dividend, divisor
        bccLgot_result
```

(b)

```
Lgot_result:
        mov     r0, result
        cmp     ip, #0                   *
        rsbmi   r0, r0, #0               *
        RET     pc, lr
```

(c)

```
Loop3:
        cmp     dividend, divisor
        subcs   dividend, dividend, divisor
        orrcs   result, result, curbit
        cmp     dividend, divisor, lsr #1
        subcs   dividend, dividend, divisor, lsr #1
        orrcs   result, result, curbit, lsr #1
        cmp     dividend, divisor, lsr #2
        subcs   dividend, dividend, divisor, lsr #2
        orrcs   result, result, curbit, lsr #2
        cmp     dividend, divisor, lsr #3
        subcs   dividend, dividend, divisor, lsr #3
        orrcs   result, result, curbit, lsr #3
        cmp     dividend, #0
        movnes  curbit, curbit, lsr #4
        movne   divisor, divisor, lsr #4
        bne     Loop3
```

(d)

표 2. 정수 나눗셈 연산 L_udivsi3, L_umodsi3, L_divsi3, L_modsi3의 각 라벨에 해당하는 수행 명령어수, 최대 반복 횟수, 수행 사이클 횟수

Table 2. The number of execution instructions, maximum repetitions, execution cycles each label.

| 라벨 \ 연산 | | L_udivsi3 | L_umodsi3 | L_divsi3 | L_modsi3 |
|---|---|---|---|---|---|
| SYM | 명령어수 | 6 | 5 | 10 | 9 |
| | 최대반복 | 1 | 1 | 1 | 1 |
| | 수행시간 (cycle) | 4~8 | 4~7 | 7~12 | 6~11 |
| Loop1 | 명령어수 | 5 | 5 | 5 | 5 |
| | 최대반복 | 7 | 7 | 7 | 7 |
| | 수행시간 (cycle) | 5~47 | 5~47 | 5~47 | 5~47 |
| Lbig_num | 명령어수 | 5 | 5 | 5 | 5 |
| | 최대반복 | 3 | 3 | 3 | 3 |
| | 수행시간 (cycle) | 5~19 | 5~19 | 5~19 | 5~19 |
| Loop3 | 명령어수 | 16 | 17 | 16 | 17 |
| | 최대반복 | 8 | 8 | 8 | 8 |
| | 수행시간 (cycle) | 16~142 | 17~150 | 16~142 | 17~150 |
| Lgot_result | 명령어수 | 2 | 9 | 4 | 12 |
| | 최대반복 | 1 | 1 | 1 | 1 |
| | 수행시간 (cycle) | 4 | 4~11 | 6 | 11~15 |

러리인 lib1funcs[1]은 ARM 명령어로 이루어진 4개의 루틴들로 구성되어있다. 이 4개의 루틴은 부호 없는 나눗셈 연산 처리 루틴인 l_udivsi3과 부호 없는 모듈러 연산 처리 루틴인 L_umodsi3, 부호 있는 나눗셈 연산 처리 루틴인 L_divsi3, 부호 있는 모듈러 연산 처리 루틴인 L_modsi3으로 구성된다.

l_udivsi3, L_divsi3, L_umodsi3, L_modsi3, 4개의 루틴들은 기본적으로 표 1-(a)의 형태로 구성된다. 정수 나눗셈 연산에 사용할 레지스터들을 할당하며, ARM 명령어로 구성된 SYM, Loop1, Lbignum, Loop3, Lgot_result, Ldiv0, 6개의 루틴을 갖는다. 라벨 SYM에 해당하는 루틴은 제수가 '0' 이거나 피제수가 제수보다 큰 경우를 확인한다. 라벨 Loop1과 Lbignum은 정수 나눗셈 연산 처리의 핵심적인 루틴인 Loop3을 위한 제수, 피제수 레지스터 초기화 루틴이다. 라벨 Lgot_result에 해당하는 루틴은 Loop3의 연산 결과를 저장하고 정수 나눗셈 루틴을 벗어나 주 프로그램으로 복귀하는 역할을 하며, 라벨 Ldiv0에 해당하는 루틴은 예외 상황을 처리하는 역할로 제수가 '0'일 경우 수행하여, 소프트웨어 인터럽트를 발생한다. 라벨 SYM, Lgot_result에 해당하는 루틴의 명령어 구성은 정수 나눗셈 연산이 부호 고려 여부에 따라 다르다. 표 1-(b)은 부호 있는 나눗셈 연산 루틴(L_divsi3, L_modis3)의 라벨 SYM의 명령어 구성이다. * 마크가 있는 명령어가 부호 있는 정수 나눗셈 처리 루틴에서 추가된 명령어들로 2의 보수 표현된 부호 있는 제수, 피제수를 부호 없는 수로 바꾸는 역할을 한다. 표 1-(c)은 부호 있는 연산 처리 루틴의 라벨 Lgot_result의 명령어 구성으로 * 마크가 있는 명령어가 부호 있는 정수 나눗셈 처리 루틴에서 추가된 명령어들로 부호 없는 연산 결과를 2의 보수 표현된 부호 있는 수로 바꾸는 역할을 한다. 정수 나눗셈 연산의 핵심적인 처리 루틴인 Loop3은 연산이 나눗셈, 모듈러 연산 경우에 따라 명령어 구성이 다르다. 표 1-(d)은 나눗셈 연산 루틴의 Loop3으로 부분 나머지와 제수의 크기를 비교하는 명령어와 부분 나머지를 오른쪽으로 2회 쉬프트 시켜서 제수를 빼는 명령어들의 루프로 구성되어있다. 표 2는 ARM7 TDMI 마이크로프로세서에서 4

표 3. 정수 나눗셈 연산 루틴 L_udivsi3, L_umodsi3, L_divsi3, L_modsi3의 수행 명령어 수, 수행 싸이클 횟수

Table 3. The number of execution instructions, execution cycles for integer division routine L_udivsi3, L_umodsi3, L_divsi3, L_modsi3N.

| | case | 수행 시간 (cycle) | 수행 명령어 개수 |
|---|---|---|---|
| L_udivsi3 | divisor = 0 | software interrupt | |
| | divisor > dividend | 12+(10) | 8+(4) |
| | divisor ≦ dividend and divisor ≠ 0 | 36+(10) ~ 218+(10) | 34+(4) |
| L_umodsi3 | divisor = 0 | software interrupt | |
| | divisor > dividend | 7+(10) | 5+(4) |
| | divisor ≦ dividend and divisor ≠ 0 | 36+(10) ~ 232+(10) | 41+(4) |
| L_divsi3 | divisor = 0 | software interrupt | |
| | divisor > dividend | 17+(10) | 14+(4) |
| | divisor ≦ dividend and divisor ≠ 0 | 42+(10) ~ 224+(10) | 40+(4) |
| L_modsi3 | divisor = 0 | software interrupt | |
| | divisor > dividend | 19+(10) | 13+(4) |
| | divisor ≦ dividend and divisor ≠ 0 | 47+(10) ~ 240+(10) | 48 + (4) |

개의 정수 나눗셈 연산 루틴을 수행 시 각 라벨에 해당하는 루틴들의 수행 명령어 개수와 최대 반복 횟수, 수행 사이클 횟수를 나타내었다[1].

L_udivsi3, L_umodsi3, L_divsi3, L_modsi3, 4개의 나눗셈 연산 처리 루틴은 주 프로그램에서 호출되는 서브루틴(Sub-routine)이므로 호출 시 서브루틴에 진입하는 과정과 서브루틴 종료 시 복귀 과정이 필요하다. 서브루틴 진입과정은 서브루틴에서 사용할 레지스터를 스택에 저장하고 제수와 피제수를 서브루틴에서 사용할 레지스터로 이동시킨 후 서브루틴으로 분기하는 동작으로 구성되며, 서브루틴 종료 시 복귀 과정은 주 프로그램으로 복귀한 후 서브루틴에서 할당하여 사용된 레지스터를 스택으로부터 이전 상태로 복구하는 동작으로 구성된다. 표 3은 4개의 나눗셈 연산 루틴의 진입 및 복귀를 포함한 총 명령어 개수와 총 수행 시간을 나타내었다. 괄호는 서브루틴의 진입 및 복귀에 사용되는 명령어수와 수행 시간을 나타낸다[2-4].

## Ⅲ. 정수 나눗셈 명령어 정의 및 나눗셈 알고리듬 데이터 패스 설계

Ⅱ장에서 살펴본 현재 ARM7 TDMI 마이크로프로세서의 정수 나눗셈 연산 처리 방법인 소프트웨어 루틴 방법은 작은 면적과 저전력이 요구되는 임베디드 시스템에서 전용 나눗셈 처리 유닛을 갖는 것보다 비용 측면에서 유리할 수 있으나, 많은 수행 명령어로 인해 프로그램 코드 사이즈가 크게 증가되며, 긴 수행 시간으로 인해 병목 현상을 초래할 수 있다. 본 논문의 중요한 목표는 정수 나눗셈 연산 처리 시 소프트웨어 루틴 방법보다 적은 수행 명령어와 짧은 수행 기간 갖으면서 전용 나눗셈 처리 유닛보다 작은 하드웨어 면적을 갖는 것이다. 이를 위해 정수 나눗셈 연산을 단일 명령어로 처리할 수 있는 정수 나눗셈 명령어를 제안함으로써 프로그램 코드 사이즈를 줄일 수 있으며, 현재 ARM7 TDMI 마이크로프로세서의 데이터 패스를 최대한 이용하여 나눗셈 알고리즘을 적용함으로써 추가되는 하드웨어를 줄일 수 있다.

### 1. 정수 나눗셈 명령어의 정의

제안되는 정수 나눗셈 연산 명령어는 현재 ARM 명령어들과 오퍼레이션 코드 필드(OP-code field), 소스 레지스터 필드(Source register field), 목적지 레지스터 필드(Destination register field)의 위치를 동일하게 하

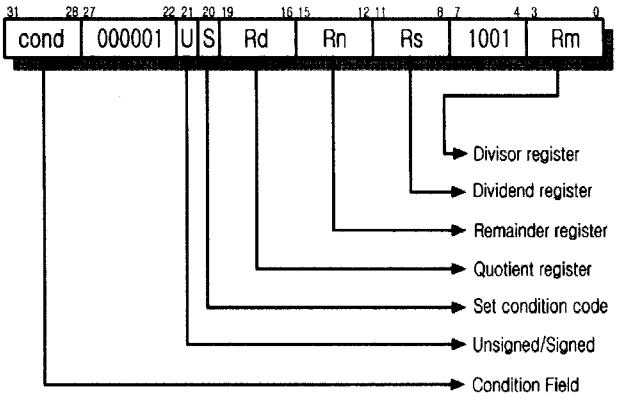



그림 1. 제안된 정수 나눗셈 명령어 형식
Fig. 1. Proposed integer division instruction format.

표 4. 제안된 정수 나눗셈 명령어의 동작
Table 4. The operation of proposed integer division instruction

| U-bit[21] | 명령어 | 의미 | 효과 |
|---|---|---|---|
| 0 | UDIV | Unsigned division, modular | Rd := Rs/Rm<br>Rn := Rs mod Rm |
| 1 | SDIV | Signed division, modular | Rd := Rs/Rm<br>Rn := Rs mod Rm |

여 새롭게 정의되는 명령어로 인해 명령어 해석 및 레지스터 오퍼랜드 인출 시 추가되는 하드웨어가 없게 하였다[2-4]. 새로 정의된 정수 나눗셈 연산 명령어 UDIV는 부호 없는 정수 나눗셈 연산과 부호 없는 모듈러 연산을 동시에 수행하고, SDIV는 부호 있는 나눗셈 연산과 부호 있는 모듈러 연산을 동시에 수행하며, 수행 동작은 Rd = Rs/Rm, Rn = Rs mod Rm 이다. S bit의 상태에 따라 연산 결과의 상태(Z flag : 연산 결과가 0, N flag: 연산 결과가 음수)를 상태 레지스터인 CPSR(Current Processor Status Register)에 저장할 수 있다. 제안된 정수 나눗셈 연산 명령어의 형식은 그림 1과 같으며, 수행 동작은 표 4와 같다.

### 2. 나눗셈 알고리듬 데이터 패스 설계

위의 1에서 제안한 정수 나눗셈 명령어의 수행을 위한 데이터 패스는 radix-2 비복원 알고리듬을 적용하여 설계하였다. 적용된 비복원 알고리듬은 몫 결정 시 BSD(Binary Signed Digit) 방식을 적용하여 몫은 -1 또는 1을 갖는다. 이는 나눗셈 연산 종료 시 표준 2 진수로 변환과정을 거쳐야하는 단점을 갖는다. 또 비복원 알고리듬 특징상 나눗셈 종료 시 최종 나머지의 부호와 피제수의 부호가 다를 수 있다. 이 경우 최종 나머지를 보정해야 된다. 그러나 비복원 알고리듬은 2의 보수 표현된 부호 있는 나눗셈 연산에 바로 적용된다. 복원 알

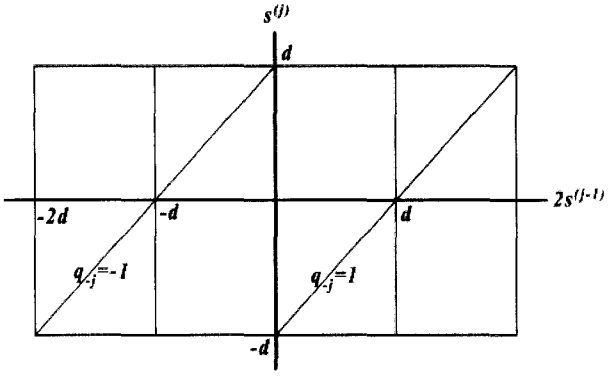



그림 2. 비복원 나눗셈 알고리듬의 Z-Z plot
Fig. 2. The Z-Z plot of nonrestoring division algorithm.

고리즘은 양수 값에 대해서만 처리가 가능하기 때문에 부호 있는 나눗셈 연산 시 제수와 피제수를 반드시 양수로 변환해야 하며, 연산 결과인 몫과 나머지 역시 제수와 피제수의 부호에 따라 부호 변환이 필요하다[5][6]. 이는 부호 있는 나눗셈 연산 시 적지 않은 사이클을 요구한다. 그림 2는 Radix-2 비복원 나눗셈 알고리듬의 새로운 부분 나머지(New partial remainder)와 이전의 부분 나머지(Old partial remainder)의 관계를 나타낸 Z-Z plot이다[5][6]. 제수 d는 k-bit이며, 피제수 z는 2k-bit이다. 이전의 부분 나머지 $S^{(j-1)}$는 좌측 쉬프트 연산 1회를 통해 $2S^{(j-1)}$이 된다. 이때 몫 $q_{-j}$의 결정 방법은 식 (1)과 같으며 몫은 BSD 방식으로 -1, 1로 표현된다. 결정된 몫을 이용하여 식 (2)과 같은 방법으로 새로운 부분 나머지 $S^{(j)}$를 구한다. 비복원 방식은 반복 방식으로 식 (1)과 식 (2)의 과정을 제수의 bit수인 k회 반복해야 한다.

$$\text{if } 2s^{(j-1)} \geq 0 \text{ then } q_{-j} = 1$$
$$\text{if } 2s^{(j-1)} < 0 \text{ then } q_{-j} = -1 \quad (1)$$

$$s^{(j)} = 2s^{(j-1)} - q_{k-j}(2^k d), \quad s^{(0)} = z \text{ and } s^{(k)} = 2^k s \quad (2)$$

k-bit BSD로 표현된 몫, 식 (3)을 k-bit 이진수로 변환하는 방법은 다음과 같다. -1에는 0을 할당하고, 1에는 그대로 1을 할당하여 k-bit 식 (4)을 만든다. 식 (4)에서 $p_{k-1}$을 반전한 뒤 레프트 쉬프트 1회 후 LSB(Least Significant Bit)에 1을 삽입하면 식 (5)의 2의 보수 표현된 표준 이진수로 변환된다[5][6].

$$q = (q_{k-1}q_{k-2}\cdots q_0)_{BSD},\ q_i \in \{-1, 1\} \quad (3)$$

$$p = p_{k-1}p_{k-2}\cdots p_0,\ p_i \in \{0, 1\} \quad (4)$$

$$q = (\overline{p_{k-1}}p_{k-2}\cdots p_0 1)_{2's-compl.} \quad (5)$$

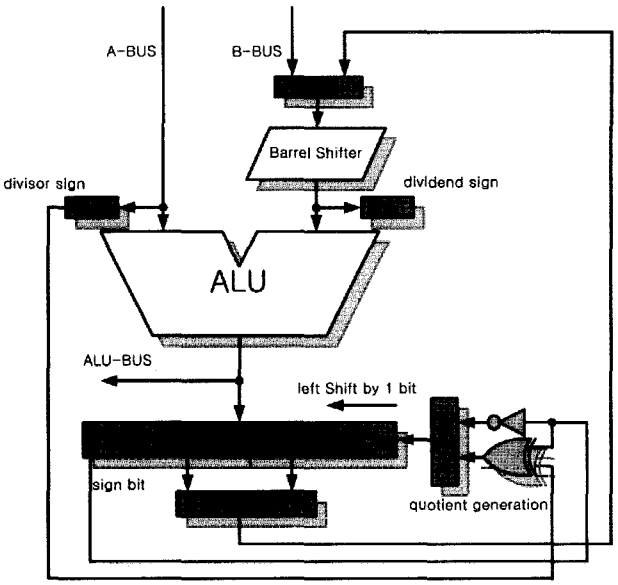



그림 3. 비복원 나눗셈 알고리듬을 적용한 데이터 패스
Fig. 3. The data path applied nonrestoring division algorithm.

위에서 살펴본 비복원 나눗셈 알고리듬은 좌측 쉬프트 연산과 비교, 덧셈, 뺄셈 연산으로 구성되며, BSD를 표준 이진수 방법으로 변환하는 방법 역시 복잡해 보이나 좌측 쉬프트 연산과 1을 증가하는 연산으로 간단히 구현할 수 있다. 비복원 나눗셈 알고리듬을 수행하기 위해 필요한 연산 기능은 현재 ARM7 TDMI 마이크로프로세서의 데이터 패스가 대부분 갖고 있다. 식 (1)의 몫 결정 방법은 부분 나머지의 부호와 제수의 부호를 이용하여 몫을 결정할 수 있으며, 식 (2)의 새로운 부분 나머지를 구하는 방법과 BSD 변환 역시 ALU의 덧셈, 뺄셈 연산과 이용할 수 있다. 그림 3은 비복원 나눗셈 알고리듬 수행을 위해 제안된 ALU 주변의 데이터 패스이다. 어둡게 표시된 부분이 추가된 하드웨어 유닛이며, 32bit 멀티플렉서 3개, 1bit 래치 2개, 64bit 쉬프트 레지스터 1개로 구성된다.

64bit 쉬프트 레지스터는 몫과 부분 나머지를 저장하는 역할을 하며, 2개의 래치는 제수와 피제수의 부호를 저장하는 역할을 한다. 몫을 결정하는 회로는 부호 있는 나눗셈 연산과 부호 없는 나눗셈 연산에 따라 구성이 다르다. 그림 4는 제안된 정수 나눗셈 연산 데이터 패스에서 정수 나눗셈 명령어 수행 흐름을 나타내는 순서도이다. 먼저 제수는 0이 아닌 것으로 가정한다. 나눗셈 연산은 제수가 0일 경우 연산을 할 수 없으므로 이를 처리해주는 예외 상황을 발생해야 한다. 이는 제수가 0 일 때를 체크하여 이에 해당하는 소프트웨어 인터럽트 명령어로 처리한다.

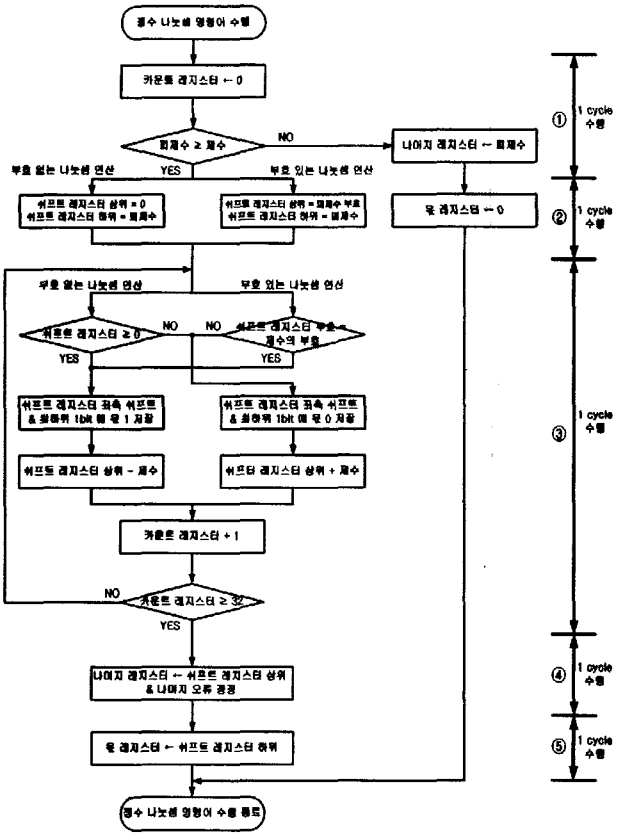



그림 4. 제안된 정수 나눗셈 명령어 수행 순서도
Fig. 4. The flow chart of proposed integer division instruction.

그림 4의 순서도에서 ①에서는 카운트 레지스터를 0으로 초기화 시키며, 피제수와 제수의 크기를 비교한다. 정수 나눗셈 특징상 제수가 피제수 보다 클 경우 몫은 0이 되므로 나머지 레지스터에 피제수를 저장하고, 몫 레지스터에 0을 저장하여 나눗셈 연산을 간단히 수행할 수 있다. 피제수와 제수의 크기를 비교하는 방법은 피제수에서 제수를 ALU를 이용하여 빼는 연산을 수행하여 피제수와 제수의 부호와 연산 결과를 이용한다. ②에서는 64bit 시프트 레지스터를 초기화하며, 부호 없는 나눗셈 연산일 경우 상위에 0을 확장하여 저장하고, 하위에는 B-BUS로 전달된 피제수를 ALU를 통하여 저장한다. 부호 있는 나눗셈 연산을 경우 상위에는 피제수의 부호를 확장하여 저장하고 하위에는 피제수를 저장한다. ③에서는 몫을 결정하여 저장하고, 결정된 몫을 바탕으로 부분 나머지를 발생하는 수행을 한다. 몫 결정 방법은 부호 없는 나눗셈 연산 경우 쉬프트 레지스터에 저장된 피제수의 부호가 양수일 경우 1을 할당하며, 음수일 경우 0을 할당한다. 부호 있는 나눗셈 연산 경우 쉬프트 레지스터에 저장된 피제수의 부호와 제수의 부호가 같을 경우 1을 할당하며, 부호가 틀릴 경우 0

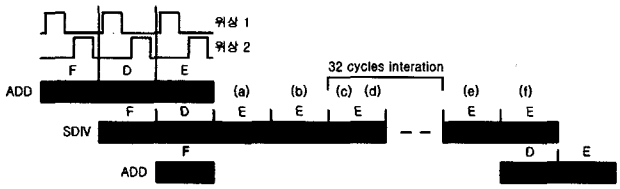



그림 5. SDIV 명령어 수행 파이프라인
Fig. 5. The pipeline of SDIV instruction execution.

을 할당한다. 여기서 0은 의미상 -1과 같으며, 식 (4)의 BSD 변환 과정에서 -1은 0으로 변환 되므로 몫 발생시 미리 -1을 0으로 할당하였다. 발생한 몫은 쉬프트 레지스터가 왼쪽으로 1회 쉬프트 시 최하위 비트에 저장되며, 1일 경우 ALU를 이용하여 왼쪽으로 1회 쉬프트 된 쉬프트 레지스터의 상위 32bit과 제수를 뺄셈 연산을 통해 부분 나머지를 발생하여서 쉬프트 레지스터 상위 32bit에 저장하고 0일 경우 덧셈 연산을 통해 부분 나머지를 발생하여 쉬프트 레지스터 상위 32bit에 저장한다. 세 번째 사이클 과정은 32회 반복하며, 32회 반복 종료 시 쉬프트 레지스터 상위 32bit에는 나머지가 저장되어 있고, 하위 32bit에는 몫이 저장되어 있다. 64bit 쉬프트 레지스터에 저장된 나머지와 몫은 ALU를 통하여 제안된 정수 나눗셈 명령어에서 명시한 목적지 레지스터로 저장된다. 이 때 나머지의 부호가 피제수의 부호와 다를 경우 나머지를 보정해야 한다. 나머지 보정 방법은 제수와 피제수의 부호가 같을 경우 나머지와 제수를 ALU에서 덧셈 연산하여 ALU-BUS를 통해 나머지 목적지 레지스터에 저장하며, 제수와 피제수의 부호가 다를 경우 나머지에서 제수를 ALU에서 뺄셈 연산하여 ALU-BUS를 통해 나머지 목적지 레지스터에 저장한다. 64bit 쉬프트 레지스터 하위에 저장된 몫은 배럴 쉬프터를 통해 왼쪽으로 1bit 쉬프트 한 뒤 ALU에서 1을 증가 시킨 뒤 몫 목적지 레지스터에 저장한다. 이 과정은 식 (5)의 BSD 변환 과정과 같다.

ARM7 TDMI 마이크로프로세서는 3단계 파이프라인 구조를 갖으며 F(fetch), D(decode), E(execution)로 단계로 구성된다. 각 단계는 두 개의 위상(2-phase)을 갖으며, 데이터 패스와 제어 회로 구성하는 회로들은 위상들의 상위 레벨에 동기(high-level sensitive)하여 동작한다. 그림 5는 파이프라인에서 부호 있는 나눗셈 명령어 SDIV의 수행을 나타낸다.

SDIV 명령어는 다중 수행 명령어로 수행 시 그 다음 명령어는 SDIV 명령어 수행 종료까지 대기한다. 그림 5의 SDIV 명령어의 다중 수행 중 첫 번째 수행 사이클인 (a)은 그림 4의 순서도에서 ①과 같이 제수와 피제

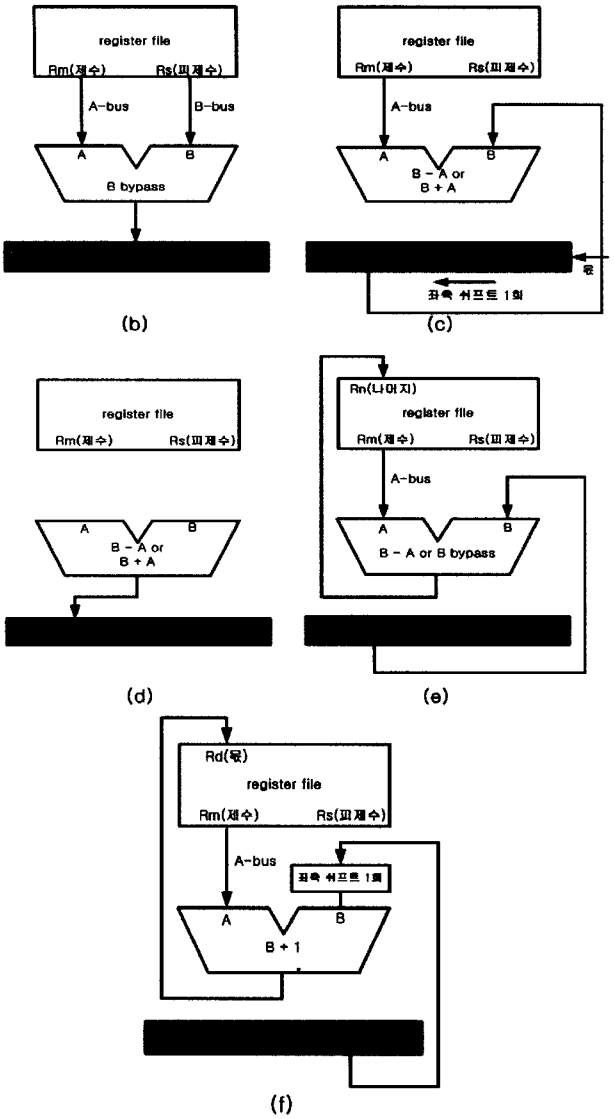



그림 6. SDIV 명령어 수행 시 데이터 패스의 동작
Fig. 6. The data path operation of SDIV instruction execution.

수의 크기를 비교하는 과정이다. 두 번째 수행 사이클인 (b)은 64bit 쉬프트 레지스터를 피제수로 초기화하는 과정으로 그림 4의 순서도의 ②와 같으며, 데이터 패스의 동작은 그림 6의 (b)와 같다. 그 다음 수행 사이클은 몫과 부분 나머지를 구하는 과정으로 32회 반복되며, 그림 4의 순서도에서 ③과 같다. 이 수행 사이클의 데이터 패스의 동작은 그림 6의 (c), (d)와 같으며, (c)는 위상 1에 동기하여 64bit 쉬프트 레지스터를 왼쪽으로 1회 쉬프트 동작한다. 이 과정에서 쉬프트 레지스터의 최하위 비트에는 몫이 저장되며, 쉬프트 된 상위 32bit은 ALU의 B를 통해 전달되어 뺄셈 또는 덧셈 연산을 수행하는 동작을 나타낸다. (d)는 위상 2에 동기하여 ALU에서 계산된 부분 나머지를 쉬프트 레지스터 상위

표 5. 소프트웨어 루틴 반복 방법과의 비교
Table 5. Comparisons with software routine iteration.

| | | 경우 | 수행 사이클 | 수행 명령어 |
|---|---|---|---|---|
| 소프트웨어 루틴 반복 방법 | 부호 없는 나눗셈 | 제수 = 0 | software interrupt | |
| | | 제수 > 피제수 | 12+(10) | 8+(4) |
| | | 제수 ≤ 피제수<br>제수 ≠ 0 | 36+(10)<br>~ 218+(10) | 34+(4) |
| | 부호 없는 모듈러 | 제수 = 0 | software interrupt | |
| | | 제수 > 피제수 | 7+(10) | 5+(4) |
| | | 제수 ≤ 피제수<br>제수 ≠ 0 | 36+(10)<br>~ 232+(10) | 41+(4) |
| | 부호 있는 나눗셈 | 제수 = 0 | software interrupt | |
| | | 제수 > 피제수 | 17+(10) | 14+(4) |
| | | 제수 ≤ 피제수<br>제수 ≠ 0 | 42+(10)<br>~ 224+(10) | 6 + (4) |
| | 부호 있는 모듈러 | 제수 = 0 | software interrupt | |
| | | 제수 > 피제수 | 19+(10) | 40+(4) |
| | | 제수 ≤ 피제수<br>제수 ≠ 0 | 47+(10)<br>~ 240+(10) | 48+(4) |
| 제안하는 방법 | SDIV, UDIV 명령어 | 제수 = 0 | software interrupt | |
| | | 제수 > 피제수 | 3 | 1 |
| | | 제수 ≤ 피제수<br>제수 ≠ 0 | 36 | 1 |

32bit에 저장되는 동작을 나타낸다. 그림 5의 (e), (f)는 SDIV 명령어의 최종 수행 단계로 (e)단계에서는 나머지를 저장하고, (f)단계에서는 몫을 저장한다. 이는 그림 4의 순서도에서 ④, ⑤ 과정에 해당하며, 그림 6의 (e), (f)는 몫과 나머지를 저장하는 데이터 패스의 동작을 나타낸다[7].

## Ⅳ. 성능 비교

본 논문이 제안한 ARM7 TDMI 마이크로프로세서에서의 정수 나눗셈 연산을 위한 정수 나눗셈 명령어와 나눗셈 데이터 패스는 기존의 정수 나눗셈 처리 방법인 소프트웨어 루틴 반복 방식과 비교해서 정수 나눗셈 연산 시 적은 수행 사이클 수와 수행 명령어 수를 갖는다. 표 5는 기존의 정수 나눗셈 수행 방법인 소프트웨어 루틴 방식과 제안하는 방법의 수행 사이클 수 및 수행 명령어수를 비교한 표이다. 비교하는 정수 나눗셈 연산은 부호 없는 나눗셈과 모듈러 연산, 부호 있는 나눗셈과

표 6. 기존 논문의 정수 나눗셈기와의 비교
Table 6. Comparisons with other published integer divider.

| | 경우 | 수행 사이클 | 하드웨어 면적 |
|---|---|---|---|
| bit-serial [8] | 제수 = 0 | 34 + α | 32bit register 64bit shift register 32bit sub/adder 32bit 2to1 MUX |
| | 제수 > 피제수 | | |
| | 제수 ≤ 피제수 제수 ≠ 0 | | |
| mixed radix[9] | 제수 = 0 | 3 | 17947 gate 0.6 um cell library |
| | 제수 > 피제수 | 5 | |
| | 제수 ≤ 피제수 제수 ≠ 0 | 23 | |
| 제안하는 방법 | 제수 = 0 | software interrupt | 64bit shift register 32bit 2:1 MUX * 3 |
| | 제수 > 피제수 | 3 | |
| | 제수 ≤ 피제수 제수 ≠ 0 | 36 | |

모듈러 연산이며, 각 연산에서 제수가 0일 때, 제수가 피제수 보다 클 때, 제수가 0이 아니며 피제수가 제수보다 크거나 같을 때로 3가지 경우를 나누었다. 제안하는 정수 나눗셈 수행 방법은 제수가 0일 경우 기존의 방법과 동일하게 소프트웨어 인터럽트를 발생하며, 제수가 피제수보다 클 경우 기존의 방법에 비해 평균적으로 1/4의 수행 사이클 수를 갖으며, 제수가 0이 아니고 피제수가 제수보다 크거나 같을 경우 기존의 방법에 비해 평균적으로 1/5의 수행 사이클 수를 갖는다. 수행 명령어 수는 제안하는 정수 나눗셈 명령어 SDIV, UDIV 단일 수행으로 인해 많은 명령어로 구성된 기존의 소프트웨어 루틴 방법 보다 현저하게 적으며, 명령어 수행 시 나눗셈 연산 결과인 몫과 모듈러 연산 결과인 나머지를 동시에 발생하므로 몫을 얻는 루틴과 나머지를 얻는 루틴이 다른 기존 방식에 비해 수행 사이클 수 및 수행 명령어 수 측면에서 유리하다.

표 6은 제안하는 방법과 기존의 논문에서 제안되었던 정수 나눗셈기와 수행 사이클 수 및 하드웨어 면적 측면에서 비교한 결과이다. bit-serial 나눗셈기는 radix-2 복원 나눗셈 알고리듬을 적용한 정수 나눗셈기로 본 논문이 제안하는 데이터 패스에 적용한 비복원 나눗셈 알고리듬과는 차이가 있다. 수행 사이클에서 추가되는 α는 bit-serial 나눗셈기가 적용한 복원 알고리듬이 부호 있는 연산을 수행하지 못하므로 부호 있는 나눗셈 연산 시 제수와 피제수를 양수로 초기화하는데 필요한 사이클 수와 연산 종료 후 몫과 나머지의 부호 보정에 필요한 사이클 수이다[8]. 이 과정은 목표로 하는 마이크로프로세서에 의존하여 사이클 수가 결정되나, 제수와 피제수의 부호를 판단하고 음수 일 경우 양수로 변환하는 과정과 제수와 피제수의 부호를 저장한 후 연산 종료 시 저장된 부호를 바탕으로 몫과 나머지를 음수로 변환하는 과정은 최소 6~7 사이클의 수행 시간이 필요하다. mixed radix 나눗셈기는 64bit/32bit 고속 정수 나눗셈기로 radix를 4와 2를 혼용하여 몫을 결정하는 방식이다[9]. 나눗셈 수행 사이클 수는 본 논문이 제안하는 방법에 비해 적으나, 하드웨어 부담이 매우 크다. 표 6에 제시한 게이트 카운트는 0.6 um 셀 라이브러리로 합성한 결과이며, 본 논문의 제안을 적용하는 ARM7 TDMI 마이크로프로세서가 0.5 um 셀 라이브러리로 합성한 게이트 카운트(45.000 gate)의 약 1/3에 해당한다[1].

## V. 결 론

본 논문에서는 ARM7 TDMI 마이크로프로세서의 연산기능 중 구현되지 않은 정수 나눗셈 연산기능을 제안하였다. 이를 위해 부호 없는 정수 나눗셈 명령어인 'UDIV' 명령어와 부호 있는 정수 나눗셈 명령어인 'SDIV' 명령어를 새로 정의하였으며, 제안된 명령어의 수행을 위해 비복원 나눗셈 알고리듬을 기존의 데이터 패스를 이용하여 적용함으로써 추가되는 하드웨어를 최소화 시켰다. 제안된 정수 나눗셈 연산 방법은 HDL을 이용해 RTL 레벨로 설계하여 기능 시뮬레이션을 하였으며, 기존의 ARM7 TDMI 마이크로프로세서의 정수 나눗셈 연산 방법인 소프트웨어 루틴 반복 방법에 비해 1/5에 해당하는 수행 사이클 수를 갖으며, 나눗셈 명령어 단일 수행으로 인해 수행 명령어 수가 현저히 적다. 그리고 기존 논문의 정수 나눗셈기인 bit-serial 나눗셈기에 비하여 적은 수행 사이클 수를 갖으며, mixed radix 나눗셈기에 비하여 많은 수행 사이클 수를 갖으나, 하드웨어 부담이 현저하게 작다. 제안된 정수 나눗셈 수행 방법은 ARM7 TDMI 마이크로프로세서에서뿐만 아니라 정수 나눗셈 수행 명령어와 나눗셈 데이터 패스가 없으며, 유사한 데이터 패스 구조를 갖는 ARM9 마이크로프로세서에서도 큰 하드웨어 부담 없이 적용할 수 있을 것으로 생각된다.

## 참 고 문 헌


[1] Programming Techniques (ARM DUI0021A), Advanced RISC Machines Ltd (ARM), 1995.

[2] Steve Furber, ARM System on a chip Architec -ture, Addison-Wesley, pp. 75-76, 1996.

[3] ARM7 TDMI Data Sheet (ARM DDI0029E), Advanced RISC Machines Ltd(ARM), 1995

[4] Dave Jagger, ARM Architecture Reference Manual, Prentice Hall, 1996.

[5] Behrooz Parhami, Computer Arithmetic, Algorith -ms and Hardwarw Design, Oxford University Press, pp. 128-211, 2000.

[6] Israel Koren, Computer Arithmetic Algorithms, John Wiley & Sons. Inc, pp. 35-40, 1978.

[7] Patterson and Hennessy, Computer Organization & Design, Morgan Kaufmann. Inc, pp. 265-274, 1998.

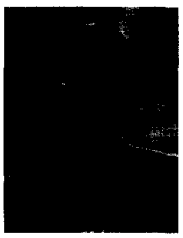
[8] 옹 수환, 선우 명훈, "새로운 Bit-serial 방식의 곱셈기 및 나눗셈기 아키텍쳐 설계", 전자공학회논문지, 제36권, C편, 제3호, pp 17-24, 1999, 03.

[9] C.-C. Wang, C.-J. Huang, and G.-C. Lin, "Cell -based implementation of radix-4/2 64b dividend 32b divosor signed integer divider using then COMPASS cell library", IEE Proc.-comput., Vol147, No2, pp 109-115, march 2000.


## 저 자 소 개


**오 민 석**(학생회원)
2003년 서경대학교 컴퓨터공학과 학사 졸업
2003년~현재 서경대학교 컴퓨터 공학과 석사 과정.
<주관심분야: 저전력 마이크로프로세서, Computer arithmetic>

**김 영 훈**(학생회원)
2002년 서경대학교 컴퓨터공학과 학사 졸업
2002년~현재 서경대학교 컴퓨터 공학과 석사 과정.
<주관심분야 : 저전력 마이크로프로세서, 암호프로세서>



**이 광 엽**(정회원)
1985년 8월 서강대학교 전자공학과 학사.
1987년 8월 연세대학교 전자공학과 석사.
1994년 2월 연세대학교 전자공학과 박사.
1989~1995년 현대전자 선임연구원,
1995년~현재 서경대학교 컴퓨터공학과 조교수.
<주관심분야: 마이크로프로세서, 암호프로세서>

**김 재 우**(학생회원)
2003년 서경대학교 컴퓨터공학과 학사 졸업
2003년~현재 서경대학교 컴퓨터 공학과 석사 과정.
<주관심분야: 3D 그래픽 가속기, 마이크로프로세서>

**남 기 훈**(학생회원)
1999년 서경대학교 컴퓨터과학과 학사 졸업
2001년 서경대학교 컴퓨터과학과 석사 졸업
2004년 서경대학교 컴퓨터과학과 박사 수료
<주관심분야: 마이크로프로세서, 임베디드 시스템>