Signed Integer Division 명령어를 추가한 ARM7 Core 설계

오 민 석, 조 태 헌, 남 기 훈, 이 광 엽
서경대학교 컴퓨터공학과
전화 : 02-940-7240 / 핸드폰 : 011-294-3619

Design of an ARM7 Core with a Singed Integer Division Instruction

Min-Seok Oh, Tae-Heon Cho, Ki-Hoon Nam, Kwang-Youb Lee
Dept. of Computer Engineering, SeoKyeong University
E-mail : omani21@dreamwiz.com

Abstract

본 논문은 ARM7 TDMI 마이크로프로세서의 연산 기능 중 구현되지 않은 나눗셈 연산 기능을 추가로 구현하였다. 이를 위해 ARM ISA(Instruction Set Architecture)에 부호를 고려한 나눗셈 명령어인 'SDIV' 명령어를 추가로 정의하였으며, 나눗셈 알고리즘 Signed Nonrestoring Division을 수행할 수 있도록 ARM7 TDMI 마이크로프로세서의 Data Path를 재설계하였다.

제안된 방법의 타당성을 검증하기 위하여 현재 ARM7 TDMI 마이크로프로세서의 정수 나눗셈 연산 처리 방법과 제안된 구조에서의 정수 나눗셈 연산 처리 방법을 비교하였으며, 그 결과 수행 cycle의 수가 40%로 감소되는 것을 확인하였다.

I. 서론

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. 나눗셈 연산은 사칙연산 중 가장 하드웨어 설계가 복잡하고 구현 시 하드웨어 면적이 가장 크다. 그러나 정수 나눗셈 연산은 임베디드 시스템에서 데이터 처리와 데이터 구조 구현 시 빈번히 사용된다. 그래서 현재 ARM7 TDMI 마이크로프로세서에서는 쉬프트 연산과 뺄셈 연산의 루프인 Programmed Division으로 정수 나눗셈 연산을 처리하거나, 보조 프로세서(Coprocessor)로 사용하는 지수 연산기(Floating Point Unit)에서 이를 처리한다. 그러나 Programmed Division은 많은 루프로 인해 긴 수행 시간을 갖으며, 지수 연산기에서 정수 나눗셈 연산을 처리하게 될 경우 데이터전송, 데이터변환 등 긴 수행시간을 갖는다.

본 논문에서는 ARM7 TDMI 마이크로프로세서의 연산기능 중 구현되지 않은 정수 나눗셈 연산기능을 추가로 구현하였다. 이를 위해 ARM ISA(Instruction Set Architecture)에 부호를 고려한 나눗셈 명령어인 'SDIV' 명령어를 추가로 정의하였으며, 나눗셈 알고리즘인 부호를 고려한 Nonrestoring Division을 수행할 수 있도록 ARM7 TDMI 마이크로프로세서의 Data Path 재설계하였다.

본 논문은 IDEC 사업의 지원으로 작성되었습니다.

II. ARM7 TDMI 마이크로프로세서의 정수 나눗셈 처리 방법과 구조

2. 1 ARM7 TDMI 마이크로프로세서의 나눗셈 연산 처리 방법

ARM7 TDMI 마이크로프로세서는 정수 나눗셈 연산을 처리하기 위한 별도의 명령어가 정의 되어 있지 않다. 현재 ARM7 TDMI 마이크로프로세서는 지수 연산기 또는 다른 여러 연산 명령어를 이용하여 Programmed Division 방식으로 나눗셈 연산을 처리한다. 표 1.은 정수 나눗셈 연산을 처리하는 루틴의 ARM 어셈블리 코드로 쉬프트(Shift) 연산과 뺄셈 연산의 루프로 이루어져 있다.

```
#ifdef L_divsi3
assignment:                          레지스터 할당
evaluation:                          나눗셈 수행 조건 판단
Loop1:                               dividend, dividend initialize
Loop2:                               dividend, dividend initialize
Loop3:                               main division loop
        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
Lgot_result:                         saving result
Ldiv0:                               for divide by '0'
#endif /* L_divsi3 */
```

표 1. ARM GCC Cross complier의 Signed integer division 루틴인 L_divsi3의 어셈블리 코드

Signed integer 나눗셈 루틴인 L_divsi은 3개의 루프와 4개의 레이블로 총 44개의 명령어로 구성되어 있다. 이중 4개의 명령어는 젯수가 0일 경우 에러 메시지를 위한 레이블 Ldiv0에 해당하므로 정상의 나눗셈 처리 과정은 40개의 명령어로 이루어진다. L_divsi의 Loop3은 나눗셈 연산 처리의 Main loop로 비교 명령어인 cmp, 뺄셈 명령어인 sub, 분기 명령어인 b로 총 16개의 명령어로 이루어져 있으며, 최고 4번 반복될 수 있다.[1][2][3]

ARM7 TDMI 마이크로프로세서의 정수 나눗셈 처리 방법인 L_divsi3 programmed division 루틴의 정상적인 나눗셈의 경우 나눗셈 루틴으로 분기하는 명령어 수행 시간까지 포함해서 51 cycle에서 105 cycle이 소요된다.

2. 2 ARM7 TDMI 마이크로프로세서의 구조

ARM7 TDMI 마이크로프로세서의 기본 구조는 그림 1.과 같다.[1] ARM7 TDMI 마이크로프로세서는 3단 파이프라인 구조로 32bit RISC 프로세서이다.[1][2][3]

ARM7 TDMI 마이크로프로세서의 가장 큰 특징 중 하나인 32bit ALU와 직렬로 연결된 32bit 베럴 쉬프터(Barrel shifter)는 데이터 연산과 쉬프트 동작을 동시에 수행할 수 있게 해준다. 이는 쉬프트 연산과 뺄셈 연산이 주를 이루는 programmed division에서 두 연산을 한 명령어로 빠르게 처리할 수 있는 매우 큰 이점을 갖는다.

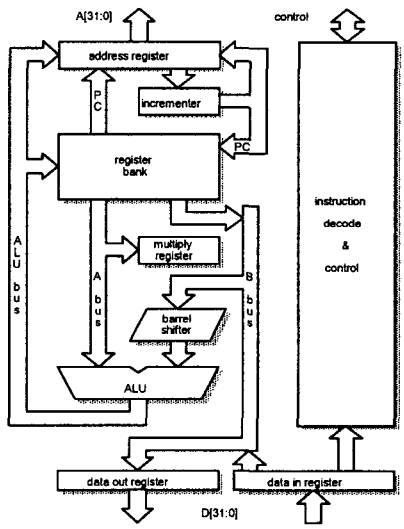

그림 2. ARM7 TDMI의 구조

ARM7 TDMI 마이크로프로세서의 또 다른 특징인 명령어 조건 수행은(Conditional execution) 모든 명령어가 수행조건을 만족해야 수행하도록 되어 있다. 이 역시 젯수와 피젯수의 크기를 계속 비교하여 쉬프트 연산과 뺄셈 연산의 수행 여부를 판단하는 programmed division에서 큰 이점을 갖는다. 그림 2.는 ARM ISA의 명령어 형식 중 Condition field를 나타낸 그림이며, 표 2.는 Condition code에 대한 설명이다.[1]

그림 4. The ARM condition code field

Opcode [31:28]	Mnemonic extension	Interpretation	Status flag state for execution
0000	EQ	Equal / equals zero	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set / unsigned higher or same	C set
0011	CC/LO	Carry clear / unsigned lower	C clear
0100	MI	Minus / negative	N set
0101	PL	Plus / positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N equals V
1011	LT	Signed less than	N is not equal to V
1100	GT	Signed greater than	Z clear and N equals V
1101	LE	Signed less than or equal	Z set or N is not equal to V
1110	AL	Always	any
1111	NV	Never (do not use!)	none

표 2. ARM condition code

III. 정수 나눗셈 명령어 정의 및 데이터 패스 설계

3. 1 Radix-2 Nonrestoring Division 알고리즘

본 논문이 제안한 Signed Division 명령어 수행을 위해 사용한 나눗셈 알고리즘은 Radix-2 비복원 나눗셈(Nonrestoring Division) 방법으로 2's-Complement 수의 나눗셈에 바로 적용되며, 많은 하드웨어 추가 없이 ARM의 Data path에 적용할 수 있다. 그림 3.은 비복원 나눗셈의 새로운 부분 나머지(Partial remainder)와 전의 부분 나머지의 관계를 나타낸 Z-Z plot이다.[4]

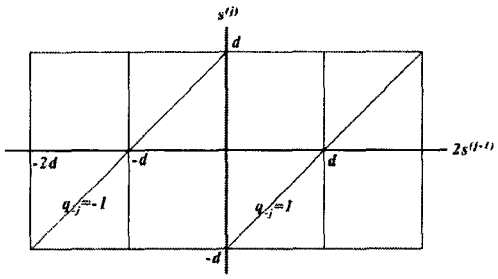

그림 3. Radix-2 비복원 나눗셈의 Z-Z plot

Radix-2 나눗셈 알고리즘의 기본 식은 식(1)과 같으며, 몫의 결정 방법은 식(2)와 같다.[4][5]

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

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

3. 2 Signed Division Instruction의 정의

나눗셈 연산 명령어 정의는 ARM ISA 명령어들의 소스 레지스터 필드(Source register field)와 목적지 레지스터 필드(Destination register field)의 위치를 동일하게 정의하여 새로운 명령어로 인해 레지스터 디코딩에 추가되는 하드웨어가 없게 하였다. 새로 정의한 나눗셈 연산 명령어 SDIV는 Signed Integer Division을 수행하며, 수행 동작은 Rd = Rs/Rm, Rn = Rs mod Rm 로 나눗셈 연산과 모듈러 연산의 결과를 갖고, S bit의 Setting에 따라 Z(Zero flag), N(Negative flag)를 상태 레지스터에 저장할 수 있다. SDIV 명령어의 Encoding은 그림 4.와 같다.

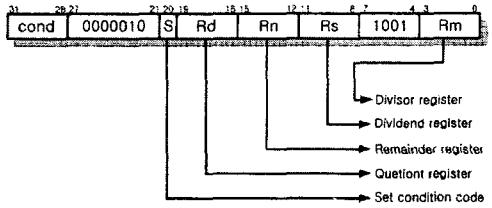

그림 4. SDIV Instruction Encoding

3. 3 Signed Division Data Path 설계

나눗셈 연산이 수행되기 위한 데이터 패스는 64bit shift register가 있어야 하며, 연산 결과는 값과 나머지, Z(Zero flag), N(Negative flag)이다.[6] 나눗셈 명령어 SDIV는 34번의 수행 Cycle을 가지며, Divide by ±1 조건 시 2번의 수행 Cycle을 갖는다. 그림 5.는 나눗셈 연산이 수행되기 위한 데이터 패스 블록도이다.

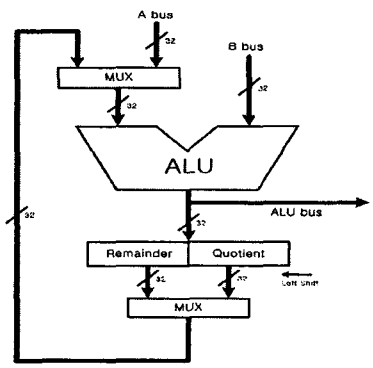

그림 5. 나눗셈 명령어 수행을 위한 데이터 패스

IV. 시뮬레이션 결과 및 비교

4. 1 SDIV 명령어 시뮬레이션 결과

그림 6.은 SDIV 명령어의 시뮬레이션 결과로

-140 / 30을 예로 하였으며, 결과로 몫 -4, 나머지 -20을 얻었다. 수행 시간은 34번 Cycle이 소요되며, 1S(Sequential cycle) + 33I(Internal cycle)로 메모리 Cycle 타입으로 구성된다.

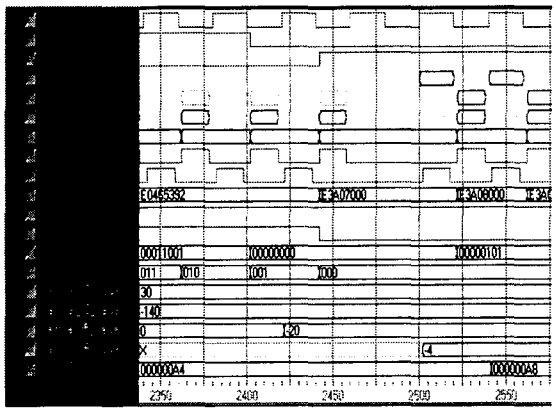

그림 6. SDIV 명령어의 수행 시뮬레이션

그림 7.은 SDIV 명령어 수행 시 Divide by +/-1 조건으로 수행한 시뮬레이션 결과로 20 / 1을 예로 하였으며, 결과로 몫 20, 나머지 0을 얻었다. 수행 시간은 2Cycle을 소요하며, 1S(Sequential cycle) + 1I(Internal cycle)로 메모리 Cycle 타입으로 구성된다.

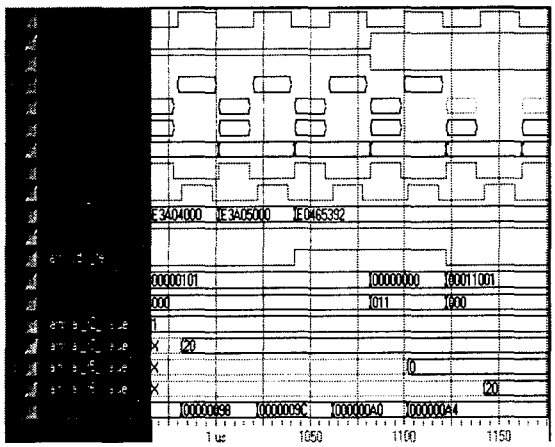

그림 7. Divisor가 1일 경우의 SDIV 명령어 수행 시뮬레이션

4. 2 수행 시간 및 추가 하드웨어

표 3.은 SDIV 명령어의 수행 시간과 2. 3장에서 살펴본 나눗셈 처리 루틴의 수행 시간을 비교하였다.()는 나눗셈 루틴으로 분기하는 과정의 명령어의 수치이다.

	case	수행 시간(cycle)	명령어의 개수
SDIV	divisor = ±1	2	1
	divisor ≠ ±1	34	
나눗셈 처리 루틴	divisor = 0	exception	9 + (3)
	divisor > dividend	20 + (9)	13 + (3)
	divisor ≦ dividend and divisor ≠ 0	47 + (9) ~101 + (9)	44 + (3)

표 3. 수행시간 및 명령어 개수의 비교

SDIV 명령어를 수행을 위한 Data path 설계 시 하드웨어는 64bit shift register 1개와 1bit register 2개가 추가되었다.

V. 결론

본 논문이 제안한 SDIV 명령어를 추가한 ARM7 코어는 나눗셈 연산 수행 시간이 34Cycle로 보통의 ARM 명령어 보다 매우 길다. 그러나 나눗셈 처리 루틴인 Programmed division 보다 약 40% 정도의 적은 수행 시간을 가지며, 명령어 개수에서도 매우 적으므로 코드 밀도(Code density)에서도 이점을 갖는다. 또 SDIV 명령어를 수행하기 위한 나눗셈 알고리즘은 비교적 간단한 radix-2 비환원 나눗셈 알고리즘을 사용하여 추가되는 하드웨어를 최소화하였다.

Reference

[1] Steve Furber, ARM System Architecture: Addison-Wesley, 1996.

[2] ARM7TDMI Data Sheet (ARM DDI0029E): Advanced RISC Machines Ltd(ARM), 1995.

[3] Dabe Jagger, ARM Architecture Reference Manual : Prentice Hall, 1996.

[4] Behrooz Parhami, "Computer Arithmetic : Algorithms and Hardwarw Design", Oxford University Press, 2000.

[5] Israel Koren, Computer Arithmetic Algorithms : John Wiley & Sons. Inc, 1978.

[6] Patterson and Hennessy, Computer Organization & Design : Morgan Kaufmann. Inc, 1998.