ARM9 호환 32bit RISC Microprocessor의 설계

황 보 식, 남 형 진
선문대학교 전자공학과
E-mail : bsHwang36@yahoo.co.kr

Design of an ARM9 Compatible 32bit RISC Microprocessor

Bo Sik Hwang and Hyoung Gin Nam
Dept. of Electronic Engineering, Sun Moon University

Abstract

In this study, we designed an ARM9 compatible RISC microprocessor using VHDL. The microprocessor was designed to support Harvard architecture with separate instruction cache and data cache. The state machine was optimized for multi-cycle instructions. In addition, a data forwarding mechanism was adopted to reduce the stall cycles due to data hazards. Assembly programs were up-loaded into a ROM block for system-level simulation. Proper operation of the designed microprocessor was confirmed by investigating the contents of the internal registers as well as the RAM block. Futhermore, the simulation results clearly indicated that the operation speed of the processor designed in this study is enhanced by reducing the execution cycles required for multiplication related instructions.

Ⅰ. 서 론

최근 Embedded System 등 많은 디지털 시스템들은 일반적으로 Microprocessor를 내장하여 시스템의 기능 중 상당 부분을 소프트웨어적으로 처리하는 추세여서, 독창적인 시스템의 서비스 개발을 위해서는 Microprocessor를 자체적으로 설계 개발할 수 있는 기술 확보가 매우 중요하다. 한편, ARM의 핵심기술을 채용한 반도체칩은 다른 제품에 비해 전력소모량이 적고 속도는 빠르면서 크기가 작다는 장점을 갖고 있어 ARM의 기술 확산이 빠르게 진행되고 있다. ARM은 일반 칩으로 제공되지 않고 지적 재산 즉, IP 형태로 공급되어 IP를 공급받은 회사에서 자사에 맞는 회로를 부가적으로 추가하여 다양한 기능을 수행할 수 있도록 하고 있다. 또한 ARM은 현 시대가 요구하는 SoC (System On a Chip) 에 대응할 수 있는 수단이 되어가고 있다. 본 연구는 다양한 시스템의 주요한 Controller로 사용되고 있는 ARM 계열 Micropro- cessor의 회로적인 구조와 Instruction 구조 등의 기본 구조를 연구하여 Microprocessor를 설계하였다. 또한 일부 Multiply에 관련된 Instruction의 Execution Cycle을 줄여 동작 속도를 향상시켰다.

Ⅱ. Microprocessor 구조

1. Instruction 구조

Microprocessor를 설계하기 위해서는 Instruction Set 구조에 따라 내부 구조를 정의하여야 한다. 즉, Instruction Set은 Microprocessor의 내부 구조를 정의하고 설계하는데 중요한 포인트가 된다. 본 연구에서는 ARM9 호환 Microprocessor를 설계하기 위하여 ARM9TDMI와 동일한 V4 Instruction Set을 사용하였다. V4의 Instruction Set은 Data Processing Instruction, Single/Multiple Data Transfer Instruction, Branch Instruction, Multiply Instruction, Software Interrupt Instruction 등으로 구성되어 있으며 Coprocessor를 사용하기 위한 Coprocessor

Instruction들을 포함하고 있다.

ARM 계열 Microprocessor는 높은 Instruction 밀도(High Code Density) 를 가지고 있다. ARM 모드인 경우 32 Bit의 Instruction 크기를 가지고 있으며 32 Bit의 Instruction 중 일부 Bit는 조건적인 실행을 위한 Condition Code를 포함하고 있다. 이 Condition Code는 ARM 계열 Microprocessor가 갖는 큰 특징 중 하나로서 조건 비교, 조건 분기 Instruction을 별도로 마련할 필요를 없애준다. 즉, 모든 Instruction마다 Condition Code를 검사하여 조건에 맞는 경우에 한해 Instruction을 수행하도록 한다. 따라서 별도의 조건 판단, 조건부 실행 및 분기에 관련된 Instruction이 필요 없으며 다른 RISC Microprocessor보다 짧은 Code로 같은 역할을 수행할 수 있다.

2. Pipeline

ARM9에서는 Pipeline 수행 간 Stall이 발생하는 횟수를 줄이기 위해 Memory Access Stage를 분리해 내고 Write-Back Stage를 독립시켜 총 5단의 Pipeline 구조를 채택하였다. 각 Pipeline은 Instruction Fetch, Instruction Decode, Instruction Execute, Memory Access, Write-Back으로 구성되며 각 Pipeline Register를 통해서 Control Signal과 Operand가 전달되도록 되어있다. 이렇게 Memory Access Stage가 분리된 5단 Pipeline 구조를 수행하기 위해서 Memory 구조도 Von Neumann 구조에서 Harvard 구조로의 변경이 필수적이다. 따라서 본 설계에서는 5단 Pipeline 구조와 함께 Harvard 구조의 Memory Access 방식을 사용하였다.

한편, Pipeline 처리 시 Data Hazard에 의한 Pipeline 지연을 막기 위해 Data Forwarding Circuit을 추가하고 Decode Pipeline Register, Execute Pipeline Register, Memory Access Pipeline Register 사이에 Data Bus를 추가하여 Write-Back 단계 이전에도 미리 갱신된 Data를 사용할 수 있도록 하였다.

3. State Machine

State Machine은 Pipeline구조에 따라 기본 상태들을 정의하였다. 경우에 따라서는 Stall이라는 상태가 발생하는데 State Machine은 Stall을 충분히 고려하여 설계되어야 한다. 본 연구에서 설계한 Processor는 Harvard 구조를 채택하였기 때문에 발생할 수 있는 Stall의 종류는 Multiple Load Stall, Branch Stall, Multiplier Stall, Swap stall등이 있다. 이를 고려하여 모든 Instruction 동작 시에 진행되는 Pipeline의 동작을 명시하여 State Machine을 설계하였다.

1) Top State Machine

전체 State Machine은 그림1과 같다. 모두 17개의 State로 구성되어 있다. Stall이 발생되지 않는 Instruction의 경우 정원이 인가되면 F ⇨ FD ⇨ FDE ⇨ FDEM ⇨ FDEMW ⇨ FDEMW … 로 State가 천이된 다음 FDEMW State에서 머물게 된다. 하지만 Stall이 발생하는 Instruction인 경우에는 Instruction의 종류에 따라 적절한 Control Signal이 발생되어 State가 천이된다. 다시 FDEMW State로 돌아오기 위해서는 Instruction의 종류에 따라 각각 여러 State를 거쳐야한다.

2) Sub State Machine

Sub State Machine은 그림2에서 나타내었다. 그림1에서 WE state와 E state는 Multiply Instruction에 관련된 State로 처음 MUL이라는 제어신호와 함께 WES State로 이동하고 동시에 Sub State Machine도 동작하게 된다. E State로 진행한 후에 Sub State Machine이 계속 동작하게 되고 Sub State Machine이 동작을 끝내면 SUB_END라는 제어신호가 발생하여 FDEM State로 이동하게 된다.

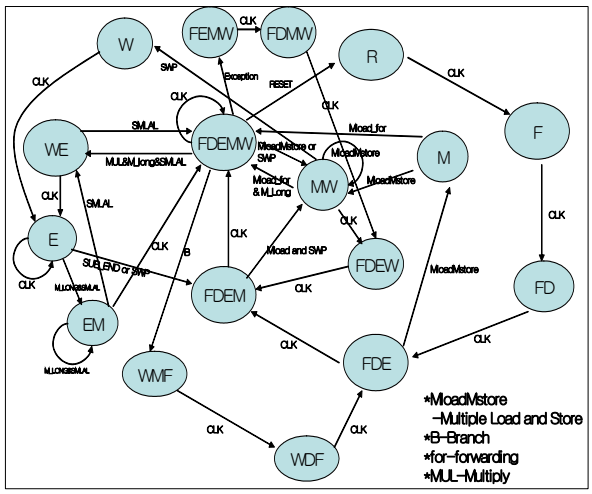

그림 1. Top State Machine

Sub State Machine은 Multiplier의 연산 과정에 대한 State만을 고려하였다. 총 2개의 State로 구성되어 총 4번의 연산을 한다. Booth Algorithm을 채택하여 32 Bit×8 Bit를 1 Clk에 연산하므로 32 Bit×32 Bit를 4번에 나누어 연산하도록 구성하였다. MUL이라는 제어신호를 시작으로 최종 SUB_END라는 제어신호로 Sub State Machine은 종료된다.

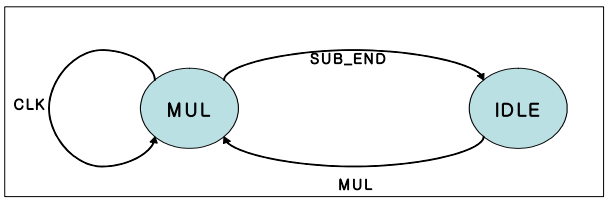

그림 2. Sub State Machine

4. Data Path

ARM 계열 Microprocessor의 구조적인 특징으로는 분리된 연산 구조를 들 수가 있다. ARM 계열 Microprocessor는 Execute Block의 ALU와 별도로 Barrel Shifter, Multiplier를 두고 있어 Shift 연산 Instruction을 별도로 필요로 하지 않으며 즉치 상수에 대한 처리를 일반 RISC Processor보다 효율적으로 처리하고 있다. 또한 구조적으로 Barrel Shifter ⇨ ALU 혹은 Barrel Shifter ⇨ Multiplier ⇨ ALU를 거치는 복잡한 연산을 수행할 수 있도록 되어 있어 복잡한 연산을 하나의 연산으로 수행할 수 있는 능력을 가지고 있다.

5. 시스템 Block Diagram

설계한 Processor의 구조는 그림 3에 제시한 Block Diagram에서 볼 수 있듯이 Fetch, Decode, Execute, Memory Access 및 Write-Back 등 각 Pipeline Register에 따라 나누어져 있다.

Fetch에는 Instruction을 인출하기 위해 Address를 저장하는 Address Reg와 Read한 Instruction을 저장하는 IR (Instruction Register) 이 있고, Branch의 경우 생성된 Address를 선택하기 위한 MUX가 있다. Decode에는 Instruction을 해석하기 위한 Instruction Decode Block이 있고, Execute에는 Data Forwarding Control과 입력되는 Data를 선택하기 위한 Mux, Barrel Shifter, Multiplier, ALU 등이 있다. Memory Access 및 Write-Back에는 Data 정렬과 Data Address 저장을 위한 Data Write Control이 있다.

Ⅲ. Simulation 결과 및 고찰

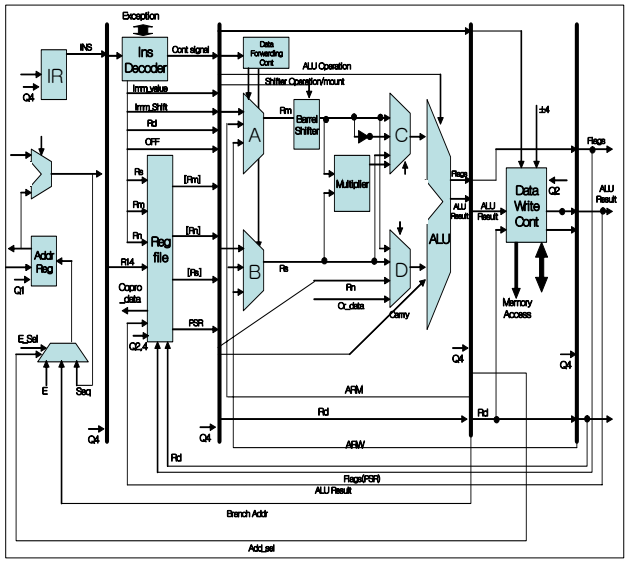

그림 3. Microprocessor의 Block Diagram

1. Simulation 결과

전체 설계는 VHDL을 사용하여 진행되었고 Modelsim을 이용하여 Simulation을 수행하였다. 시스템 차원에서의 동작 검증을 위해서 Microprocessor에 ROM Block과 RAM Block을 연결하여 설계한 Processor 검증을 위한 시스템을 구성하였다. 설계된 Processor가 ROM Block에 저장된 Machine Code를 수행할 때마다 Microprocessor 내의 Register와 RAM Block의 저장된 Data를 확인하여 동작을 검증하였다. 그림 4에는 이러한 시뮬레이션 결과의 일부를 나타내었다. 그림에서 볼 수 있듯이 ROM에 입력된 Instruction이 Micro- processor에 의하여 실행됨에 따라서

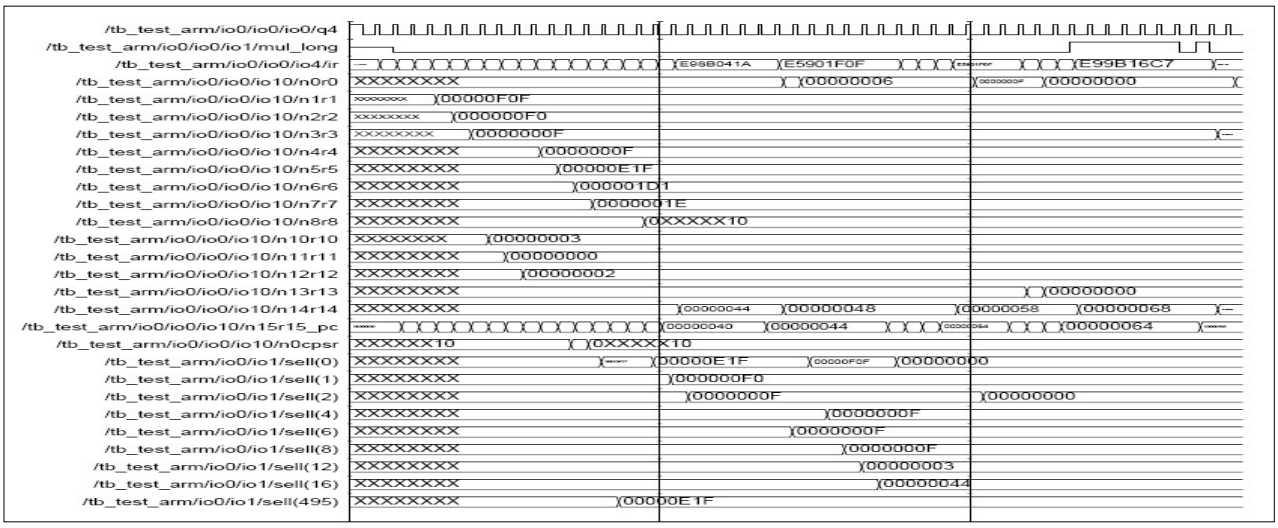

그림 4. System Level simulation 결과

순차적으로 Instruction Address가 갱신되고 r15_pc Register의 Data가 갱신되는 것을 확인 할 수 있다. 하지만 1 Cycle이상의 Execution Cycle을 가지는 Instruction을 실행할 경우 Microprocessor는 Instruction Fetch를 하지 못하므로 Instruction Address도 갱신되지 못하고, 또한 r15_pc Register의 Data도 갱신되지 못하고 유지하게 되는 것을 확인할 수 있다. 그림 4에서 MUL_LONG Signal은 Multiply Long Instruction의 Execution Cycle을 나타낸다. Q4 Signal을 기준으로 보았을 때 Multiply Long Instruction을 수행하는데 총 6 Cycle이 걸리는 것을 알 수 있다.

2. 고찰

ARM사의 ARM9TDMI, ARM7TDMI와 본 연구에서 설계한 Microprocessor의 LDM, SWP, MUL, MLA, MULL, MLAL의 최대 Execution Cycle을 비교해 보았다. 표 1에 제시한 비교 결과에서 알 수 있듯이 본 연구에서 설계된 Microprocessor는 Multiply 관련 Instruction에서 ARM사의 ARM9TDMI나 ARM7TDMI 보다 적은 Execution Cycle을 갖는 것이 확인되었다.

일반적인 Instruction에서는 Execute내에 있는 ALU를 통한 Execution Cycle이 같지만 Multiply 관련 Instruction에서는 Multiplier 내의 ALU와 Execute Block의 ALU를 이용하여 Execution Cycle을 줄일 수 있었다.

표 1. Execution Cycle 비교 결과

Instruction	Execute Cycle		
	ARM7TDMI	ARM9TDMI	본 연구 결과
LDM	17	17	17
SWP	3	3	3
MUL	4	4	4
MLA	5	5	4
MULL	6	6	5
MLAL	7	7	6

Ⅳ. 결론

저전력 Embedded Microprocessor가 요구되는 분야들이 계속적으로 성장하는 추세여서, 이러한 분야에 쓰일 수 있는 Microprocessor를 개발하는 일이 중요한 쟁점이 되고 있다.

본 연구에서는 Instruction의 동작, Pipeline 구조 및 동작, Data Path의 구조 등을 정의하고 이에 적절한 State Machine을 설계한 다음 이들을 토대로 하여 ARM9 호환 Micro- processor를 VHDL을 사용하여 RTL (Register Transfer Level) 차원에서 설계하였다. 검증을 위해서는 Micro- processor에 ROM Block과 RAM Block을 연결하여 테스트 시스템을 구성한 다음 설계된 Processor가 ROM Block에 저장된 Machine Code를 수행함에 따른 Microprocessor 내의 Register와 RAM Block의 저장된 Data 변화를 확인하였다.

설계된 Microprocessor의 경우 Multiply 관련 Instruction에서 Execution Cycle이 감소되었음이 확인되었고, 이로써 Processor의 동작 속도가 향상되었음이 입증되었다.

참 고 문 헌

[1] Steve Furber, "ARM system-on-chip architecture, 2nd Edition", ADDISON WESLEY, March 2000.
[2] Advanced RISC Machines, ARM7TDMI Datasheet, August 1995.
[3] Advanced RISC Machines, ARM9TDMI Technical Reference Manual, Rev 1, 1998.
[4] Advanced RISC Machines, ARM940T Technical Reference Manual, Rev 1, 1999.
[5] 경종민, 박인철, "고성능 마이크로프로세서 구조 및 설계 방법", 대영사, 19-49쪽, 2000.12.
[6] http://www.arm.com