The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 18, No. 5, pp.229-235, Oct. 31, 2018. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2018.18.5.229
JIIBC 2018-5-30

VHDL을 이용한 ARM 프로세서의 설계 및 모의실행

Design and Simulation of ARM Processor using VHDL

이종복*

Jongbok Lee*

요 약 ARM은 2016년 현재 매일 4천만 개씩 선적되고 있으며 860억 개 이상 이동통신, 가전, 기업, 임베디드 시스템 분야에서 탑재되어 널리 이용되고 있다. 국내의 반도체 기술은 메모리 반도체 설계에 있어서는 세계 최고의 수준이나, 프로세서의 설계는 그에 미치지 못하여 메모리와 프로세서의 균형있는 발전을 이루지 못하고 있다. 일반적으로 프로세서를 설계할 때는 고가의 전용 소프트웨어의 필요로 인하여 환경을 갖추기가 어렵다. 그러나 최근에 Altera에서 제공하는 ModelSim은 무료로 누구나 이용할 수 있다. 본 논문에서는 유럽 및 전 세계의 대학 및 연구소에서 ASIC 설계에 널리 쓰이는 VHDL을 이용하여 32 비트 ARMv4 계열의 프로세서를 설계하고, ModelSim으로 모의실행하였다. 그 결과, ARM의 37 개 명령어를 성공적으로 수행할 수 있었다.

Abstract As of in the year of 2016, 40 million ARM processors are being shipped everyday and more than 86 billion ARM processors are mounted in mobile communications, consumer electronics, enterprises, and embedded systems. Nationally, we are capable of designing high-end memory semiconductors, but not in processors, resulting in unbalance. Generally, highly expensive software programs are necessary for designing processors which makes it difficult to set up proper environments. However, ModelSim simulator provided by Altera is free and everybody can use it. In this paper, the VHDL language which is widely used in Europe, universities, and research centers around the world for the ASIC design is selected for designing 32-bit ARM processor and simulated by ModelSim. As a result, 37 instructions of ARMv4 has been successfully executed.

Key Words : ARM, VHDL, ModelSim

Ⅰ. 서 론

ARM 아키텍처는 1980년대에 Acorn Computer Group에서 최초로 개발되었는데, 이 그룹은 Advanced RISC Machine으로 명칭을 바꿔서 오늘의 ARM이 되었다. 현재 100억 개의 ARM 프로세서가 매년 판매되고 있으며, 대부분의 스마트폰과 태블릿에 ARM 프로세서가 탑재된다. 또한 ARM 프로세서는 게임기, 카메라, 로봇, 자동차, 컴퓨터 서버 등 광범위하게 쓰이고 있다. ARM은 특이하게도 직접 프로세서를 판매하지 않으며, 다른 회사로 하여금 라이선스를 통하여 시스템 온 칩의 형태로 공급한다. 예를 들어서, 삼성, Altera, 애플, 퀄컴 등의 회사들이 여기에 속한다. 이와같이 ARM 프로세서는 임베디드 기기에 이용되는 RISC 프로세서로서, 낮은 전력을 사

*정회원, 한성대학교 전자정보공학과
접수일자 : 2018년 9월 1일, 수정완료 : 2018년 10월 1일
게재확정일자 : 2018년 10월 5일

Received: 1 September, 2018 / Revised: 1 October, 2018 / Accepted: 5 October, 2018
*Corresponding Author: jblee@hansung.ac.kr
Dept. of Electronic & Information Eng., Hansung University, Korea

용하도록 설계하여 모바일 시장에서 큰 강세를 보이고 있다. 오늘날, 데스크탑 컴퓨터나 서버 컴퓨터를 제외한 대부분의 스마트폰 및 임베디드 기기에는 전부 ARM 프로세서가 탑재되어 있다고 해도 과언이 아니다.

국내에서는 메모리 반도체 설계 및 공정 기술이 세계 최고의 수준이지만, 프로세서 설계는 아직 그렇지 못한 형편이다. 따라서 프로세서 설계 분야에 대한 연구를 활발히 하고 박차를 가할 필요가 있다. 본 논문에서는 VHDL을 이용하여 5 단계 파이프라인을 갖는 ARM 프로세서를 설계하였으며, 코딩 및 검증을 위하여 ModelSim을 활용하였다. 본 연구에서 채택한 ARM의 아키텍처 계열은 ARMv4로서, ARM8 및 Strong ARM에서 채택한 구조이다.

본 논문은 다음과 같이 구성된다. 2 장에서 ARM 프로세서구조의 특징과 아울러 ARM 프로세서의 명령어 및 파이프라인 단계별 동작을 살펴 고찰한다. 3 장에서 본 논문에서 설계한 ARM 프로세서의 구조를 자세히 살펴보며, 4 장에서 모의실험 환경 및 결과를 보이고 5 장에서 결론을 맺는다.

II. ARM 프로세서의 명령어와 파이프라인

표 1에 본 논문에서 설계한 ARM 명령어 집합을 나타냈다. ARM 명령어는 크게 산술 논리, 곱셈, 메모리, 분기의 4 가지 유형으로 구성된다[1].

표 1. 설계된 ARM 프로세서의 명령어 집합
Table 1. The instruction set of the Designed ARM processor

명령어 유형	명령어
산술 논리	AND EOR SUB RSB ADD ADC SBC RSC TST TEQ CMP CMN ORR MOV LSL LSR ASR RRX ROR BIC MVN
곱셈	MUL MLA UMULL UMLAL SMULL SMLAL
메모리	STR LDR STRB LDRB STRH LDRH LDRSB LDRSH
분기	B BL

산술 논리 명령어는 AND, EOR, SUB 등의 21 개 명령어로 구성된다. 곱셈 명령어는 6 개의 명령어로 이루어지며, 결과가 32 비트인 것과 64 비트인 것으로 나뉜다.

특히 MLA, UMLAL, SMLAL은 곱셈의 결과에 다시 덧셈을 수행하므로, 일반 명령어가 두 개의 피연산자 레지스터를 읽는 것과 달리, 세 개의 피연산자 레지스터를 동시에 읽어야 한다. 메모리 명령어는 STR, LDR 등의 스토어 및 로드 명령어로 구성되며, 워드 단위 이외에 바이트 단위 또는 하프워드 단위로 수행한다. 마지막으로, 분기 명령어는 B와 BL로 구성된다. B 명령어는 24비트의 변위에 2를 곱하고 PC+8의 값을 더한 어드레스로 무조건 분기한다. BL 명령어는 분기 종료 후에 다시 리턴을 하기 위하여 PC+8의 값을 R14 레지스터에 저장하며, 분기하는 어드레스는 B의 경우와 같다. 한편, 다수의 로드 스토어 명령, 인터럽트, 코프로세서와 관련된 잡다한 명령어는 본 설계에서 제외되었다.

그림 1은 ARM 프로세서의 5 단계 파이프라인이다. 이것은 인출(Fetch), 해독(Decode), 메모리 접근(Memory), 실행(Execute), 되쓰기(Write Back)의 5 단계로 나뉜다. 각 단계의 사이마다 파이프라인의 모든 신호를 래치시키는 32비트 D 플립플롭이 설치된다.

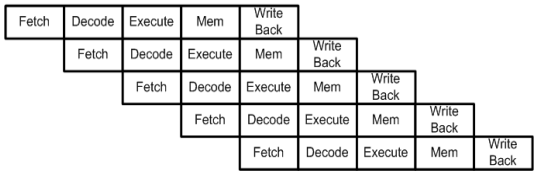

그림 1. ARM 프로세서의 5 단계 파이프라인
Fig. 1. The 5 stage pipeline of ARM processor

명령어 파이프라인에서 명령어 간에 쓰기 후 읽기(RAW) 종속이 발생하는 경우인 해저드 (hazard)는, 해저드 유닛에서 처리한다. 인접한 명령어 간에 발생하는 해저드는 메모리단계의 결과를 포워딩 (forwarding) 시키고, 한 개의 명령어만큼 떨어진 명령어 간에 발생하는 해저드는 되쓰기단계의 결과를 포워딩 시켜서 해결할 수 있다. 마지막으로, 두 개의 명령어만큼 떨어진 명령어 간에 발생하는 해저드는 레지스터 화일의 내부 포워딩에 의하여 해소된다. 레지스터 화일의 내부 포워딩이란, 레지스터의 결과를 기록과 동시에 읽어내는 값으로 전달하는 것이다[2,3].

III. ARM 프로세서의 구조

그림 2에 본 논문에서 설계한 ARM 프로세서의 전체 블럭도를 보였다. 인출단계, 해독단계, 실행단계, 메모리단계, 되쓰기단계의 각 사이마다 파이프라인 레지스터가 삽입되어 명령어 파이프라인 체계를 지원한다. 복잡성을 피하기 위하여, 제어신호는 선을 직접 연결하지 않고 제어신호 이름으로 연결 상태를 알 수 있게 하였다.

1. 인출단계 (Fetch)

인출단계는 명령어 메모리와 2 개의 멀티플렉서, 덧셈기로 구성된다. 명령어 메모리에는 ARM 프로세서가 실행할 명령어들이 기억되어있으며, 프로그램 카운터 (PC) 레지스터에 수록된 명령어 주소를 공급받아 명령어를 출력하는 기능을 수행한다. 명령어들을 순차적으로 인출할 때는 주소를 바이트 단위로 접근하므로 다음 명령어의 주소로 PC+4의 값인 PCPlus4F 신호를 이용하나, R_{14}에 값을 저장하는 BL 분기 명령어의 경우에는 레지스터화일에서 읽어서 ALU를 통과한 후 ALUOutW에서 값이 나오므로, 제 1의 멀티플렉서를 설치하여 PCSrcW의 선택 신호로 PCPlus4F와 ALUOutW의 두 값 중에서 하나를 선택하여 제 2의 멀티플렉서로 보낸다.

분기명령어에 의한 페널티를 최소화하기 위하여 분기가 실제로 일어나는 경우에 분기목적주소를 실행단계에서 계산한 값 ALUResultE와 제 1의 멀티플렉서에서 생성한 값 중에서 BranchTakenE 선택신호로 선택하여 조기에 프로그램 카운터에 공급할 수 있도록 추가의 분기용 제 2의 멀티플렉서가 필요하다.

또한, 인출부와 해독부에서 모두 2 개의 덧셈기가 필요한데, 순차적인 다음 주소를 발생시키기 위하여 PC+4인 PCPlus4F를 실행하는 덧셈기와, BL 명령어의 경우 복귀하는 명령어의 주소로 PC+8인 PCPlus8D를 계산하는 덧셈기로 쓰이기 때문이다. PCPlus8D의 값은 LR 레지스터에 저장되어, BL 명령어의 복귀하는 명령어의 주소로 이용된다.

2. 해독단계 (Decode)

해독단계는 제어부, 다수의 멀티플렉서, 레지스터화일, 부호확장기로 구성된다. 제어부에서는 인출한 명령어의 OP, Funct, Rd 필드를 입력으로 하여 PCSrcD, RegWriteD, MemtoRegD, MemWriteD, ALUControlD, BranchD, ALUSrcD, FlagWriteD, RegSrcD, ImmSrcD, MulControlD의 제어신호를 발생시킨다. 이 제어신호들 중에서 PCSrcD, RegWriteD, MemtoRegD는 명령어 파이프라인의 모든 단계를 거쳐서 최종 단계인 되쓰기단계에서 각각 PCSrcW, RegWriteW, MemtoRegW로 전달된다. ALUControlD, BranchD, ALUSrcD, FlagWriteD, MulControlD 제어신호들은 각각 실행단계까지인 ALUControlE, BranchE, ALUSrcE, FlagWriteE, MulControlE로 전달되며, 마지막으로 RegSrcD와

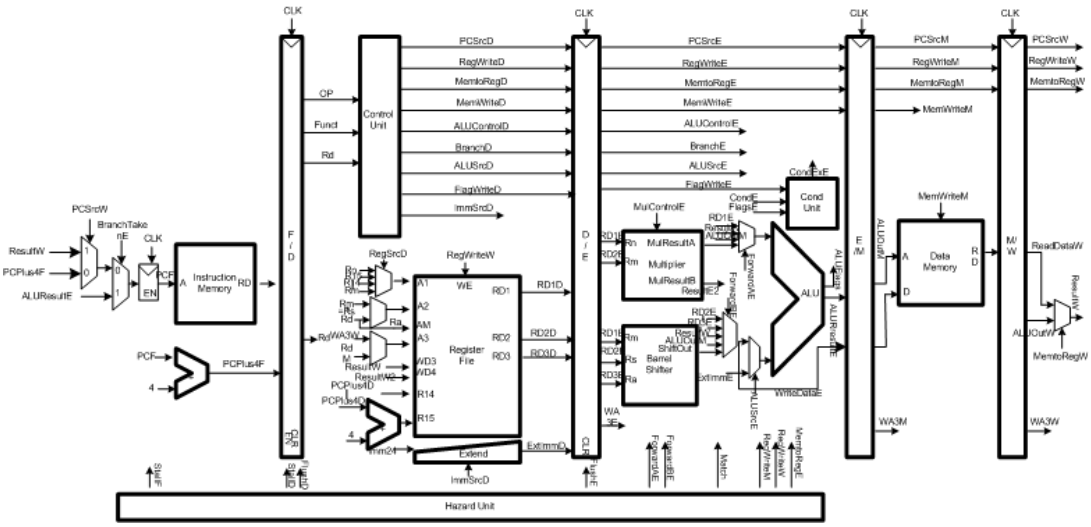

그림 2. ARM 프로세서의 전체 블럭도
Fig. 2. The block diagram of ARM processor

ImmSrcD는 해독단계에서 쓰이는데 그치고 더 이상 전달되지 않는다.

레지스터 화일은 동시에 3 개의 피연산자를 읽고, 2 개의 결과값을 쓸 수 있도록 설계되었다. 레지스터 화일이 동시에 3 개의 피연산자를 읽어야 하는 이유는 ARM의 곱셈 명령어 중에서 MLA, UMLAL, SMLAL은 두 개의 피연산자를 곱한 결과에 다시 세 번째 피연산자를 더해야 하므로, 해독단계에서 3 개의 피연산자를 동시에 읽어야하기 때문이다. 또한, 레지스터 화일에 2 개의 결과값을 써야하는 이유는 곱셈 명령어 중에 UMULL, UMLAL, SMULL, SMLAL은 64 비트 결과를 생산하므로, 32 비트 결과값을 동시에 두 번 써야 하기 때문이다.

레지스터 화일의 제 1 피연산자 입력은 R_n, R_{15}, R_{14}, R_m 중에서 멀티플렉서의 RegSrcD 제어신호로 선택한다. R_n은 일반 데이터처리 명령어에서 제 1 레지스터이며 명령어 필드의 19:16에 해당한다. R_{15}는 프로그램 카운터로 이용되는 레지스터이고, R_{14}는 BL 분기 명령어에서 리턴 어드레스를 저장한다. R_m은 일반 데이터처리 명령어에서 제 2 레지스터이나, 곱셈 명령어에서는 제 1 레지스터에 해당되므로 위 단자로 공급된다.

레지스터 화일의 제 2 피연산자 입력은 R_m과 R_d를 멀티플렉서로 선택한다. R_m은 일반 데이터처리 명령어의 제 2 레지스터이며, 명령어 필드의 3:0에 해당한다. 한편, 쉬프트 명령어의 경우에 쉬프트해야 하는 양을 가리키는 R_s도 같은 필드를 이용한다. R_d는 데이터처리 명령어의 목적지 레지스터로서 명령어 필드의 15:12에 해당하는데, 제 2 피연산자의 입력으로 공급되는 이유는 스토어 명령어의 경우 목적지 레지스터가 아닌 소오스 레지스터로 작용하기 때문이다. 레지스터 화일의 제 3 피연자 입력은 곱셈 및 누산 명령어 MLA의 R_a가 공급되며, 이것은 명령어 필드의 15:12에 해당하므로 일반 데이터 처리 명령어의 R_d와 같다.

레지스터 화일의 제 1 출력은 제 1 피연산자의 값 RD1D, 제 2 출력은 제 2 피연산자의 값 RD2D, 제 3 출력은 곱셈기에서 곱한 다음에 더하는 값 RD3D이다.

다음은 주요 제어신호들에 대한 기술이다. PCSrcD 제어신호는 프로그램 카운터 멀티플렉서에 공급하는 제어신호로서, PC+4의 값을 갖는 PCPlus4F와 ALU의 계산결과 또는 메모리로부터 읽은 값인 ResultW 둘 중에서 선택하는 역할을 수행한다. RegWriteD 제어신호는 레지스터 화일에 결과를 쓸 것인지 여부를 결정한다. 조건부 동작을 위하여, 만일 CondEx 신호가 만족되지 않으면 RegWriteD를 설정하지 않을 수 있다. MemtoRegD 제어신호는 메모리로부터 읽은 값을 레지스터에 쓸 것인지 여부를 결정하며, 이것은 로드 명령어일 때 쓰인다. MemWriteD 제어신호는 데이터를 메모리에 쓰도록 하며, 스토어 명령어인 경우에 해당한다. ALUControlD 제어신호는 ALU에 공급하여 어떤 유형의 연산을 할 것인지를 결정하는데 쓰인다. BranchD 제어신호는 조건플래그들과 함께 실제 분기가 이루어질 경우 인출단의 분기 멀티플렉서에 공급되어 계산한 분기타겟 주소를 PC에 적재한다. ALUSrcD 제어신호는 ALU의 제2입력을 레지

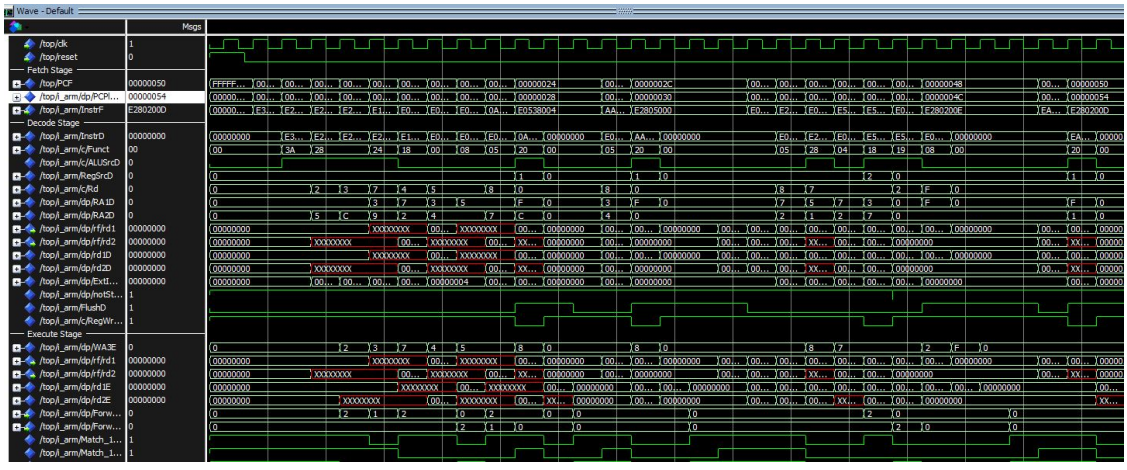

그림 3. ARM 프로세서의 ModelSim 실행 결과
Fig. 3. ModelSim simulation waves of ARM processor

스터화일의 출력 및 포워딩에 의하여 얻은 값과 부호확장기를 통해서 얻은 값 ExtImmE 중에서 선택하는 역할을 한다. FlagWriteD 제어신호는 C, V, Z, N으로 구성되는 플래그를 업데이트할 것인지를 결정한다. 마지막으로 ImmSrcD 제어신호를 이용하여 부호 확장기에서 공급받은 즉각 피연산자를 몇 비트로 확장할 것인지를 지정할 수 있다. 마지막으로, MulControlD 제어신호는 곱셈기에서 어떤 유형의 곱셈을 실행하는지를 결정하는 역할을 한다.

명령어 파이프라인을 타고 전달되는 대부분의 신호들은 주로 제어신호들이지만, 일부 데이터 신호 역시 전파된다. 해독단계에서 검출된 목적지 레지스터 번호 Rd는 실행단계, 메모리 단계, 되쓰기단계에서 각각 WA3E, WA3M, WA3W로 끝까지 전달되어, ALU에서 계산하거나 로드를 통하여 읽은 데이터를 레지스터화일에 올바르게 되쓰기한다.

3. 실행단계 (Execute)

실행단계는 ALU, 곱셈기, 쉬프터, 멀티플렉서, 조건플래그 단위 및 해독단계에서 파이프라인 래치를 통하여 넘어온 각종 제어신호 등으로 구성된다. ALU의 상단에 곱셈기와 쉬프터를 설치해서 산술논리연산 외에 곱셈과 쉬프팅 동작을 지원한다. ALU의 제 1 입력은 레지스터 화일의 출력단자 1에서 공급되는 RD1E, 메모리단계에서 포워딩되는 ALUOutM, 되쓰기 단계에서 포워딩되는 ResultW, 마지막으로 곱셈기에서 출력되는 MulResultA 중에서 멀티플렉서로 선택한다.

ALU의 제 2 입력은 레지스터 화일의 출력단자 2에서

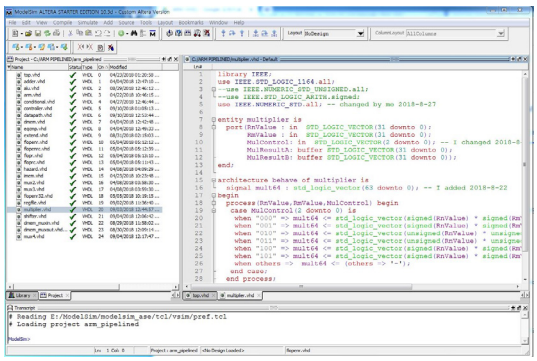

그림 4. ModelSim에서 Multiplier.vhd의 프로그래밍
Fig. 4. The programming of Multiplier.vhd using ModelSim

공급되는 값 RD2E, 메모리단계에서 포워딩되는 값 ALUOutM, 되쓰기 단계에서 포워딩되는 ResultW, 부호확장기로부터 공급되는 ExtImmE, 그리고 배럴 쉬프터(Barrel Shifter)의 출력 중에서 멀티플렉서로 선택한다. 위에서 메모리단계와 되쓰기단계에서 포워딩하는 이유는 명령어 간에 데이터 해저드 (data hazard)가 발생했을 때 작용한다. 부호확장을 제외한 ALU의 제2입력 값은 스토어인 경우 WriteDataE를 거쳐서 메모리단계에서 데이터메모리의 입력데이터로 연결된다.

분기명령어와 PC에 대한 쓰기 때문에 정지(stall)와 삭제(flush) 신호에 대하여 주의를 기울여야한다. 분기가 채택되었을 때, 후속의 두 개 명령어를 파이프라인에서 삭제해야한다. 그리고 PC에 대한 쓰기 작업이 파이프라인 내에 존재하면, 이것이 완료될 때까지 파이프라인을 정지해야한다. 이것은 인출단계를 정지함으로써 구현되며, 특정 단계를 정지시키면, 다음 단계를 삭제해야 명령어가 반복 실행되는 것을 방지할 수 있다.

4. 메모리단계 (Memory Access)

메모리단계는 데이터메모리와 실행단계에서 파이프라인 래치를 통하여 전달된 각종 제어신호들로 구성된다. 데이터 메모리의 주소 입력은 ALU에 의하여 계산되므로, 실행단계의 ALU 출력 ALUResultE가 파이프라인 래치를 거친 ALUOutM으로 연결된다. 데이터 메모리의 데이터 입력은 레지스터 화일의 제 2 출력으로부터 공급되어 배럴 쉬프터를 거친 데이터인 WriteDataE와 연결된다. 로드 명령어에 의한 데이터 메모리의 출력은 ReadDataW의 값으로 되쓰기 단계의 파이프라인 레지스터로 전달된다. 한편, 스토어 명령인 경우 MemWriteM 제어신호가 1이 되어서 데이터메모리에 값을 기록할 수 있다.

5. 되쓰기단계 (Write Back)

되쓰기단계는 명령어 파이프라인 제 2단계인 해독단계에서 데이터를 읽어들이는 것과는 반대로, 레지스터 화일에 데이터를 기록하는 과정이다. 되쓰기단계에서 발생되는 제어신호는 PCSrcW, RegWriteW, MemtoRegW의 3 가지이다. PCSrcW 제어신호는 인출단계로 전달되어 다음의 명령어 인출을 결정하는데 쓰인다. RegWriteW 제어신호는 ALU가 계산한 결과나 로드명령어의 데이터를 레지스터 화일에 데이터되쓰기를 허용

하는 역할을 한다. 따라서, 마이크로 프로세서의 명령어 파이프라인에서 레지스터 화일은 유일하게 읽기와 쓰기를 위하여 두 번 접근되는 장치이다. 로드인 경우에 데이터 메모리로부터 읽은 값인 ReadDataW를 레지스터 화일에 써야 하며, ALU 연산인 경우에는 그 연산결과인 ALUOutW를 레지스터 화일에 역시 써야 한다. 따라서, MemtoRegW 제어신호로 둘 중에 하나를 선택하는 멀티플렉서를 이용한다. 선택된 값은 ResultW를 통하여 레지스터 화일과 인출단계로 각각 전달된다.

IV. 모의실험 환경 및 결과

본 논문의 모의실험은 운영체제 Windows 7에서 3.1GHz로 동작하는 Intel Core i5-2400에서 시행하였다. ModelSim은 Altera Starter Edition 10.3d 버전을, VHDL 컴파일러는 2002 버전을 이용하였다. 그림 4에 ModelSim을 이용하여 ARM 프로세서의 연산유닛으로 쓰이는 곱셈기를 VHDL로 코딩한 화면의 예시를 나타냈다.

총 25 개의 VHDL 프로그램 소오스 모듈이 ARM 프로세서를 설계하는데 이용되었다. 표 2는 모의실험에 사용된 ARM 어셈블리어 코드 프로그램이다[3]. 이 프로그램은 ARM의 일반 데이터 처리 명령어인 MOV, ADD, SUB, ORR, AND, ADD, SUBS와 분기명령어인 BEQ, BGE, B 그리고 메모리 명령어인 STR, LDR 등으로 구성된다.

표 2. 모의실험에 이용된 ARM 어셈블리어 프로그램
Table 2. The ARM assembly program for simulation

```
MOV R0, #0
ADD R2, R0, #5
ADD R3, R0, #12
SUB R7, R3, #9
ORR R4, R7, R2
AND R5, R3, R4
ADD R5, R5, R4
SUBS R8, R5, R7
BEQ END
SUBS R8, R3, R4
BGE AROUND
ADD R5, R0, #0
AROUND SUBS R8, R7, R2
ADDLT R7, R5, #1
SUB R7, [R3, #84]
LDR R2, [R0, #96]
ADD R15, R15, R0
ADD R2, R0, #14
B END
ADD R2, R0, #13
ADD R2, R0, #10
END STR R2, [R0, #100]
```

그림 3에 ARM 프로세서가 실행되는 ModelSim의 결과 파형을 나타냈다. 그 결과, ARM 프로세서가 5 단계의 명령어 파이프라인에 따라서 인출-해독-실행-메모리 접근-레지스터화일 되쓰기 순서대로 진행하는 것을 확인할 수 있었다. 또한 데이터 해저드가 발생했을 때 적절히 포워딩이 일어나고, 분기에 의한 제어 해저드가 발생했을 때 새로운 타겟으로부터 명령어를 인출하는 것을 확인할 수 있었다.

V. 결론

본 논문에서는 ModelSim 환경에서 VHDL을 이용하여 37 개의 명령어를 실행할 수 있는 ARM 프로세서를 설계 및 개발하였다. 주어진 ARM 어셈블리 프로그램을 모의실행시킨 결과, 올바르게 동작하는 것을 확인할 수 있었다.

추후로, ARM에서 제공하나 본 설계에서 누락된 인터럽트와 관련된 기타 명령어를 포함시키고, 설계된 ARM 프로세서를 Altera사의 Quartus II를 이용하여 합성한 후에, 정적 시간 분석 (STA)과 합성 후 모의실험 (Post synthesis simulation)을 거쳐 최종적인 동작을 검증하고 FPGA로 프로그래밍하여 동작을 검증할 예정이다[4-10]. FPGA로 검증을 완료한 후에는, Synopsis로 합성하여 국내 기관인 IDEC을 통하여 ASIC 칩으로 구현할 예정이다.

References

[1] ARM Architecure Reference Manual, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.architecture.reference/index.html

[2] J. L. Hennessy, and D. A. Patterson, “Computer Architecture A Quantitative Approach”, 6th Edition; 2018.

[3] S. L. Harris, and D. M. Harris, "Digital Design and

Computer Architecture ARM Edition", Elsevier Korea LLC, 2016.

[4] F.J. Jurado Carmona, J. Tombs, M.A. Aguirre Echanove and A. Torralba, "Implementation of a fully pipelined ARM compatible microprocessor core," XVII Design on Circuits and Integrated Systems Conference, 2002, pp. 559-563.

[5] J. S. Pastor, I. Gonzalez, J. Lopez, F.G. Arribas, J. Martinez, "A Remote Laboratory for Debugging FPGA-Based Microprocessor Prototypes," Proceedings of the IEEE International Conference on Advanced Learning Technologies, 2004.

[6] A.A. Morgan, M.E. Allam, M.A. Salama, and H.A.K Mansourm "Implementation of an ARM Compatible Processor Core for SOC Designs," 2005 International Conference on Information and Communication Technology, Dec. 2005.

[7] M. M. Kinage and D.G. Khairnar, "Design and Implementation of FPGA Soft Core Processor for Low Power Multicore Embedded System using VHDL," Sep. 2016.

[8] J. Davidson, "FPGA Implementation of a Reconfigurable Microprocessor," IEEE Custom Integrated Circuits Conference, 1993, pp.3.2.1-3.2.4.

[9] E. Alaer, A. Tangel, M. Yakut, "MIB-16 FPGA based Design and Implementation of a 16 Bit Microprocessor for Educational Use," 6th WSEAS International Conference on Circuits, Systems, Electroncs, Control & Signal Processog, 2007. pp.284-288.

[10] H.S. Herman, C. Srihari, M. Matthew, "Pipeline Reconfigurable FPGAs," Journal of VLSI Signal Processing Systems,", 2000, Vol. 24, pp. 129-146

저자 소개

이 종 복(정회원)

- 1964년 8월 20일생
- 1988년 : 서울대 컴퓨터공학과 졸업.
- 1998년 : 동 대학 전기공학부 졸업 (공박)
- 1998 ~ 2000 : LG반도체 선임연구원.
- 2000년 ~ 현재 : 한성대 전자정보공학과 교수

- Tel : 02-760-4497
- Fax : 02-760-4435
- E-mail : jblee@hansung.ac.kr

<주관심분야 : 마이크로 프로세서, 멀티코어 프로세서, GPGPU 프로세서>

※ 본 연구는 한성대학교 교내학술연구비 지원과제임.