The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 20, No. 4, pp.135-141, Aug. 31, 2020. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2020.20.4.135
JIIBC 2020-4-19

토마술로 알고리즘을 이용하는 비순차실행 프로세서의 설계 및 모의실행

The Design and Simulation of Out-of-Order Execution Processor using Tomasulo Algorithm

이종복*

Jongbok Lee*

요 약 오늘날 서버, 데스크탑, 노트북과 같은 범용 컴퓨터뿐만이 아니라, 가전, 임베디드 시스템에서 중앙처리장치는 대부분 멀티코어 프로세서로 구성된다. 멀티코어 프로세서의 성능향상을 위하여, 토마술로 알고리즘을 적용한 비순차실행 프로세서를 각 코어 프로세서로 이용하는 것이 요구된다. 토마술로 알고리즘을 적용한 비순차실행 프로세서는 명령어 간의 종속성이 없고 피연산자가 준비된 명령어를 순서와 관계없이 먼저 실행하고, 분기어 너머로 예측실행을 수행함으로써, 모든 명령어를 순서대로 실행하는 순차실행 프로세서보다 성능을 크게 높일 수가 있다. 본 논문에서는 VHDL의 레코드 데이터형을 이용하여 토마술로 알고리즘을 이용하는 비순차실행 프로세서를 설계하고, GHDL로 검증하였다. 모의실험 결과, ARM 명령어로 구성된 프로그램에 대한 연산을 성공적으로 수행할 수 있었다.

Abstract Today, CPUs in general-purpose computers such as servers, desktops and laptops, as well as home appliances and embedded systems, consist mostly of multicore processors. In order to improve performance, it is required to use an out-of-order execution processor by Tomasulo algorithm as each core processor. An out-of-order execution processor with Tomasulo algorithm can execute the available instructions in any order and perform speculation in order to reduce control dependencies. Therefore, the performance of an out-of-order execution processor can be significantly improved compared to an in-order execution processor. In this paper, an out-of-order execution processor using Tomasulo algorithm and ARM instruction set is designed using VHDL record data types and simulated by GHDL. As a result, it is possible to successfully perform operations on programs written in ARM instructions.

Key Words : Tomasulo Algorithm, Out-of-Order Execition, VHDL

Ⅰ. 서 론

비순차실행(Out-of-Order Execution)은 1970년대와 1980년대 초에 자료흐름(Data Flow) 방식을 이용하는 컴퓨터구조의 주요 연구 분야로 출발하였다. 비순차실행을 사용한 최초의 처리기는 그보다 훨씬 이전인 1964년에 스코어보드(scoreboard)를 사용한 CDC 6600이다. 스코어보드 방식은 명령어 간에 데이터 종속이 없고

*정회원, 한성대학교 기계전자공학부
접수일자 2020년 5월 1일, 수정완료 2020년 8월 3일
게재확정일자 2020년 8월 7일

Received: 1 May, 2020 / Revised: 3 August, 2020 /
Accepted: 7 August, 2020
*Corresponding Author: jblee@hansung.ac.kr
School of ME Engineering, Hansung University, Korea

연산유닛이 이용 가능할 때, 비순차실행을 가능하게 하는 것이다. 그러나 이 방식은 명령어 간의 반종속(anti dependency)과 출력종속(output dependency) 문제를 해결하지 못했고, 중앙집중적 제어로 인하여 성능에 한계가 있었다. 그로부터 약 3년 후인 1966년에 IBM 360/91의 설계에 토마술로 알고리즘을 도입하여 완전한 비순차실행을 구현하였다[1]. 토마술로 알고리즘은 스코어보드 방식과는 달리, 레지스터 재명명(register renaming) 기술을 도입하여 반종속과 출력종속을 일거에 해결했으며, 분기의 방향을 미리 예측하고 실행함으로써 제어종속의 부담을 경감시킬 수 있는 예측실행(speculation)으로까지 확장이 가능하다는 장점이 있다.

그러나, 비순차실행은 그 하드웨어의 높은 복잡성에 의하여 그로부터 30년 후인 1990년대에 이르러서야 비로소 일반화되었다. 1990년에 IBM은 최초의 비순차 마이크로 프로세서 인 POWER1을 도입했으며, 이 때 비순차실행은 부동 소수점 명령으로 제한되었다. 1993년도의 IBM/Motorola PowerPC 601, 1995년도의 Fujitsu/HAL SPARC64, Intel Pentium Pro, 1996년도의 MIPS R10000, HP PA-8000, AMD K5, DEC Alpha 21264가 대표적 비순차실행 프로세서들의 좋은 예이다.

위에서 살펴본 바와 같이 비순차실행 기술은 1990년대 중반까지 주류 CPU로는 자리매김을 하지 못했으며, 이 기술을 채택한 프로세서는 구현에 넓은 실리콘 영역을 필요로 하므로 필연적으로 가격이 높을 수밖에 없다. 따라서, 비용에 민감한 저가형 프로세서 시장에서는 가격 경쟁력이 없으므로 이 기술을 이용하지 않고 있다. 또한, 비순차실행 프로세서는 전력을 많이 필요로 하기 때문에 모바일 환경에서 요구되는 저전력 사용 규격과는 다소 거리가 멀다.

한편, 비순차실행 프로세서는 위에서 살펴본 단점 이외에, 최근에 치명적인 보안상의 하드웨어 취약점이 발견되기도 하였다. 이것은 2017년과 2018년에 일부 마이크로 프로세서 제조업체에서 보고된 Spectre와 Meltdown이다[2]. 향후의 비순차실행 마이크로 프로세서는 고성능을 위하여 비순차실행과 예측실행을 적극 활용하되, 이러한 하드웨어 취약점이 발생하지 않도록 운영체제와의 상호 보완이 절실히 요구된다.

국내의 프로세서 설계 분야는, 세계 최고 수준의 메모리 반도체 설계 및 공정 기술에 비하여 매우 미흡한 형편이다. 지금까지 국내 대학, 연구소 및 기업체에서 단순한 RISC 프로세서를 설계하고 반도체칩으로 구현한 경우는 극히 일부 발견된다. KAIST는 2008년부터 Core-A 프로세서를 야심차게 개발했으나, 비순차실행방식이 아닌 순차실행방식에 그쳤고, 현재로서는 명맥을 잇지 못하고 있다. 삼성전자는 프로세서를 자체 개발하지 않고, 2010년부터 ARM Cortex-A8을 라이센싱하여 Exynos 칩을 직접 생산하였다. 그러나, 이것은 비순차실행 프로세서가 아니었고 경쟁력이 타사에 미치지 못하여, 삼성전자는 그만 2019년에 자체 CPU 코어개발을 중단하기로 하였다.

위에서 살펴본 바와 같이, 국내에서 비순차실행 프로세서를 설계 및 구현한 예는 거의 찾아볼 수가 없으므로, 늦었더라도 이에 대한 연구를 활발히 하고 박차를 가하여 메모리 반도체와 균형을 이룰 필요가 있다. 본 논문에서는 VHDL을 이용하여 토마술로 알고리즘을 지원하는 비순차실행 프로세서를 설계했으며, 명령어세트로 임베디드 마이크로 프로세서로 가장 널리 쓰이는 ARM을 이용하였다[3]. 본 설계의 코딩 및 검증을 위하여 리눅스에서 실행되는 GHDL과 GTKWave를 활용하였다.

본 논문은 다음과 같이 구성된다. 2 장에서 토마술로 알고리즘을 이용한 비순차실행 마이크로 프로세서의 원리에 대하여 살펴보고, 3 장은 본 논문에서 설계한 비순차실행 프로세서의 구조를 VHDL의 관점에서 기술한다. 4 장에서 모의실험 환경과 모의실험 결과를 보이고, 5 장에서 결론을 맺는다.

II. 토마술로 알고리즘을 이용하는 비순차실행 마이크로 프로세서의 원리

마이크로 프로세서에서는 빈번한 분기어에 의한 제어종속이 성능에 큰 영향을 끼치므로, 이것을 타개하기 위하여 비순차실행 이외에 추가의 기능이 요구된다. 따라서, 본 논문에서는 하드웨어를 기반으로 예측실행 기능을 추가한 토마술로 알고리즘을 목표로 설계하였다. 하드웨어 기반의 예측실행의 핵심개념은 어느 명령어를 실행할 것인가를 선택하는 동적 분기 예과, 제어종속이 해결되기 이전의 예측실행, 그리고 이에 따른 동적 스케줄링이다. 이러한 모든 기능을 지원하기 위하여 비순차실행 프로세서에 예약스테이션 (Reservation Station)과 재정렬버퍼(Reorder Buffer)를 설치하였다.

예약스테이션의 기본 개념은 피연산자가 준비된 즉시 예약스테이션에 등록함으로써 프로세서가 레지스터로부

터 피연산자를 가져오는 수고를 경감하고, 아울러 아직 피연산자가 준비되지 않은 명령어는 그것을 공급할 예약스테이션을 가리키도록 함으로써 레지스터 재명명 기능을 간접적으로 수행하는 것이다. 중앙형 레지스터 화일보다 예약스테이션을 이용함으로써 얻는 잇점으로 첫 번째, 해저드(hazard) 감지와 실행제어가 분산되며, 두 번째, 연산 결과들이 레지스터를 통하지 않고 예약스테이션으로부터 연산유닛으로 직접 공급되는 것을 들 수 있다. 이것은 피연산자를 기다리는 모든 유닛들이 공통데이터버스(common data bus)로부터 동시에 그 값들이 적재됨으로써 구현된다.

토마술로 알고리즘에서 명령어들의 비순차실행을 허용하면서도 순차완료 되도록 하여 정밀한 예외처리(precise interrupt)를 가능하게 하려면 재정열버퍼를 필요로 한다. 재정렬버퍼는 실행이 종료했으나 아직 완료하지 않은 명령어들을 유지하는 하드웨어 버퍼이다. 재정렬버퍼는 또한, 예측실행에서 실행을 마쳤으나 아직 완료를 하지 않은 연산결과를 예약스테이션에 임시로 공급하는 역할을 수행함으로써, 예약스테이션과 마찬가지로 추가의 레지스터들을 공급하는데 쓰인다.

그림 1에 재정렬버퍼를 포함하는 토마술로 기반의 프로세서를 나타냈다. 재정렬버퍼는 명령어 유형, 목적지, 값, 준비의 4 가지의 영역 등으로 구성된다. 명령어 유형 영역은 그 명령어가 분기어, 스토어 명령어, ALU 연산 및 로드 명령어 중의 어느 것인지를 나타낸다. 목적지 영역은 ALU 연산 및 로드명령어인 경우 결과가 기록될 목적 레지스터의 번호를 나타내고, 스토어 명령어인 경우에는 데이터가 메모리에 기록되는 주소를 나타낸다. 값 영역에는 명령어가 완료되기 전까지 그 명령어의 연산 결과를 수록하고, 준비 영역은 그 명령어의 실행 여부를 나타낸다. 이 때, 재정렬버퍼는 원래의 토마술로 알고리즘의 스토어버퍼를 포함한다. 스토어는 2 단계로 실행되는데, 마지막 단계는 명령어의 완료단계에서 이루어진다. 비록 초기의 토마술로 알고리즘에서 예약스테이션의 레지스터 재명명 기능이 재정렬버퍼로 대체되었으나, 명령어가 발행되었을 때와 실행을 시작할 때의 시간 차이를 메꿔주는 기능이 필요하며, 이것은 여전히 예약스테이션에 의하여 처리된다.

각 예약스테이션에는 발행된 명령어들이 삽입되고, 그 명령어들은 연산유닛에서 실행되기를 기다린다. 만일에 어느 명령어의 피연산자의 연산 결과가 얻어지는 경우에는 그 결과 역시 예약스테이션의 해당 항목에 삽입된다. 그러나 피연산자가 아직 준비가 되지 않았다면, 그 값을

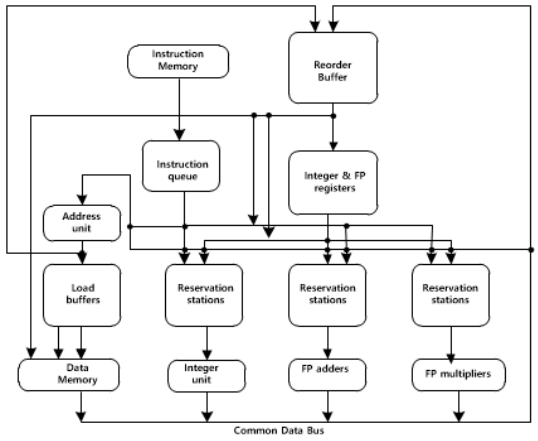

그림 1. 토마술로 알고리즘을 이용하는 비순차실행 프로세서의 블럭도

Fig. 1. The out-of-order execution processor with Tomasulo algorithm

공급하는 예약스테이션을 가리키도록 한다. 다음은 토마술로 알고리즘의 주요 4단계를 기술한다.

1. 발행 (issue)

FIFO 구조로 구성된 명령어큐의 첫 번째 항목에서 다음 명령어를 인출 받는다. 만일에 부합하는 예약스테이션과 재정렬버퍼가 모두 비어있으면, 이 명령어를 예약스테이션과 재정렬버퍼에 동시에 삽입한다. 만일에 예약스테이션 또는 재정렬버퍼가 비어있지 않으면 구조적 해저드가 발생한 것으로서, 재정렬버퍼와 예약스테이션이 모두 준비될 때까지 발행을 멈춘다 (stall).

예약스테이션이 명령어에 대한 연산을 실행하기 위하여 피연산자가 필요하다. 피연산자는 명령어의 유형에 따라 제 1 피연산자만을 이용하는 경우와, 제 1 피연산자와 제 2 피연산자를 모두 필요로 하는 경우로 나뉜다. 이러한 피연산자를 확보하는 것은 토마술로 알고리즘의 핵심으로서, 명령어의 피연산자를 읽기 위하여 3 가지의 경우를 고려해야한다. 첫 번째로 피연산자에 대한 처리가 완료되어 그 값이 레지스터에 있는 경우로서, 예약스테이션은 그 값을 레지스터로부터 정상적으로 읽어온다. 두 번째, 만일 피연산자가 레지스터에 기록되어있지 않으나 재정렬버퍼에는 존재하는 경우이다. 이것은 연산이 실행되어 재정렬버퍼에 임시 등록이 되었으나 아직 레지스터에 완료하지 않은 상태에 해당한다. 이 때는 재정렬버퍼에 있는 피연산자를 예약스테이션으로 전송한다. 세 번째, 필요한 피연산자에 대한 연산이 아직 실행이 끝나지 않아서 레지스터는 물론 재정렬버퍼에도 그 값이 존재하

지 않는 경우이다. 이 때는 피연산자가 준비되어있지 않으므로, 이 값을 생산할 명령어의 재정렬버퍼 상의 위치인 태그를 예약스테이션에 보낸다. 예약스테이션은 태그를 이용하여, 추후에 해당 피연산자가 연산이 실행되고 재정렬버퍼에 등록되면 그 값을 가져올 수 있다. 예약스테이션을 이용하는 이 과정에서 재명명으로 인하여 반종속과 출력종속이 해결된다.

2. 실행 (execute)

발행단계에서 살펴본 바와 같이, 만일 피연산자가 준비되어있지 않으면, 공통데이터버스를 모니터링 하여 필요한 피연산자 값이 연산될 때까지 대기한다. 만일 피연산자가 준비되었으면, 그것을 기다리는 모든 예약스테이션에 공급한다. 모든 피연산자가 준비된 명령어는 순서와 상관없이 각 연산유닛에서 연산을 실행할 수 있으므로 비순차실행이 가능하다. 이 단계에서 대부분의 명령어는 한 싸이클에 실행을 마칠 수가 있으나, 어떤 명령어는 여러 싸이클이 소요될 수도 있다. 이와 같이 피연산자가 준비될 때까지 명령어의 연산을 유예시킴으로 인해서, 실제종속 (true dependency)를 해결할 수 있다.

로드 명령어는 실행을 위하여 2 단계가 필요하다. 첫 번째 단계는 유효주소의 계산이며, 이것은 로드버퍼에 삽입된다. 스토어 명령어는 이미 데이터메모리에 기록할 데이터를 확보한 상태이므로, 유효주소의 계산만 필요하다.

3. 쓰기 (write)

결과가 준비되었을 때, 공통데이터버스를 통하여 레지스터와 그 결과를 기다리는 재정렬버퍼와 예약스테이션에 각각 기록한다. 스토어의 경우 메모리에 저장해야하는 값을 재정렬버퍼의 값 영역에 기록한다. 만일에 저장해야 할 값이 아직 준비가 되지 않았다면, 공통데이터버스가 그 값이 준비가 될 때까지 모니터링을 해야 한다.

4. 완료 (commit)

명령어를 마무리하는 마지막 단계로서, 마무리하는 명령어가 분기어, 스토어, 로드를 포함한 일반 명령어 중에 어느 것 인가에 따라서 처리가 다르다. 일반 명령어가 재정렬버퍼의 헤드에 도달하고 그 결과가 준비되었을 때, 그 결과는 최종적으로 레지스터에 기록되고 명령어를 재정렬버퍼에서 삭제함으로써 정상적으로 완료된다. 스토어 명령어를 완료하는 것은 이것과 유사하지만, 단지 데이터가 레지스터가 아닌 메모리에 기록된다는 점이 다르다. 예측오류가 난 분기어가 재정렬버퍼의 헤드에 도달하는 경우, 이것은 예측이 틀렸다는 것을 의미한다. 이 때는 재정렬버퍼를 일소 (squash) 하고 분기어의 올바른 후속 명령어부터 실행을 재개해야한다. 반면에, 만일 분기어가 올바르게 예측되었다면 정상적으로 완료된다.

본 연구에서 설계한 토마술로 알고리즘을 이용하는 비순차실행 프로세서는 멀티싸이클 방식으로 동작하며, 위에서 기술한 핵심 단계 이외에 명령어가 완료될 때까지 총 6 단계로 설정하였다. 이것은 인출(fetch), 발행(issue), 실행(execute), 메모리 접근(memory), 쓰기(write Back), 완료(commit)로 구성된다.

III. 토마술로 알고리즘을 이용하는 비순차실행 프로세서의 VHDL 설계

디지털 설계를 위하여 1983년과 1984년에 각각 VHDL과 Verilog가 최초로 개발되었다. VHDL은 Pascal과 Ada 프로그래밍 언어를 모델로 했으며, 주로 대학 및 연구소에서 쓰이고 있다. Verilog는 C언어를 기반으로 만들어졌기 때문에 VHDL보다 간결하며 주로 업체에서 많이 쓰이고 있다.

본 연구에서 토마술로 알고리즘을 이용하는 비순차실행 프로세서를 설계하기 위하여 VHDL을 선택하였다. 그 이유는 VHDL이 Verilog보다 다양한 데이터 형식을 표현할 수가 있으므로 상대적으로 높은 수준의 추상화가 가능한 고급 프로그래밍 언어이기 때문이다. VHDL은 특히, Verilog에서 포함되지 않는 레코드 자료구조를 지원하고 있어서 하드웨어를 설계할 때 복잡도를 줄이고 효율성을 높일 수 있다.

VHDL의 레코드 자료구조는 다양한 데이터형을 갖는 원소들을 묶어서 하나의 단위로 표현할 수 있으므로 배열보다 높은 추상도를 가짐과 동시에 합성이 가능하기 때문에, 재정렬버퍼나 예약스테이션을 갖는 토마술로 알고리즘 기반의 비순차실행 프로세서의 설계에 적합하다. 만일에 Verilog와 같이 레코드 자료구조를 이용할 수 없다면, 재정렬버퍼나 예약스테이션을 표현하기 위하여 1 차원 배열을 적절하게 잘라서 레코드의 각 필드로서 구분을 해야하고, 또한 이것에 대한 2 차원 배열로 만들어야 하므로 복잡하고 불편하다.

재정렬버퍼와 예약스테이션을 레코드형 배열로 선언한 후에, 조건과 부합하는 레코드형을 찾을 때는 VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
package poop_pkg is
type TyReserv is record
       Busy : std_logic;
       Opcode : std_logic_vector(1 downto 0);
       V1 : std_logic_vector(31 downto 0);
       V2 : std_logic_vector(31 downto 0);
       Q1 : natural;
       Q2 : natural;
       Tag : natural;
       Addrs : std_logic_vector(31 downto 0);
       Value : std_logic_vector(31 downto 0);
       Executed : std_logic;
end record TyReserv;
type TyArrayReserv is array (9 downto 0) of TyReserv;
type TyROB is record
       Busy : std_logic;
       Opcode : std_logic_vector(1 downto 0);
       Ready : std_logic;
       State : std_logic_vector(2 downto 0);
       Dest : std_logic_vector(3 downto 0);
       Value : std_logic_vector(31 downto 0);
       Addrs : std_logic_vector(31 downto 0);
       Mispredict : std_logic;
       Commit : std_logic;
       Squash : std_logic;
       Speculative : std_logic;
       Link : natural;
end record TyROB;
type TyArrayROB is array (15 downto 0) of TyROB;
```

그림 2. 재정렬버퍼와 예약스테이션의 VHDL 레코드형 자료구조
Fig. 2. The VHDL record data structures of ROBs and Reservation Stations

에서 지원하는 순환 반복문인 For 또는 While 문을 이용함으로써 비순차실행 프로세서를 간편하고 효율적으로 설계할 수 있다.

VHDL에서 레코드를 자료구조로 갖는 신호를 모든 모듈에서 이용하기 위하여, 레코드 자료구조를 패키지문에 선언하고, 모든 VHDL 모듈에서 use문을 이용하여 참조가 가능하게 할 수 있다. 그림 2는 본 연구에서 설계에 이용한 VHDL의 레코드 자료구조를 패키지로 선언한 것이다. 예약스테이션의 단일 레코드형을 TyReserv, 재정렬버퍼의 단일 레코드형을 TyROB로 선언한 다음에, 다시 TyArrayReserv와 TyArrayROB를 각각 16 개와 10 개의 레코드형 배열로 선언하였다. 예약스테이션 레코드형의 주요 구성은 Busy, Opcode, V_1, V_2, Q_1, Q_2, Tag, Addrs, Value 등으로 구성된다. Busy는 예약스테이션의 항목이 현재 비어있는지의 여부를 나타내는데 쓰이고, Opcode는 해당 명령어의 유형을 기록한다. V_1, V_2는 명령어의 준비된 제1 피연산자와 제 2 피연산자를 각각 나타내며, 이 때 Q_1, Q_2의 값은 모두 0이다. 만일 피연산자들이 아직 준비되지 않았을 경우에는, Q_1과 Q_2는 각각 첫 번째와 두 번째 피연산자를 공급해줄 명령어의 재정렬버퍼 항목번호를 나타낸다. Tag는 현 명령어의 목적

```
MAIN : LDR R2, [R1, #0]   E5912000
       MUL R4, R2, R3     E0040392
       STR R4, [R1, #0]   E5814000
       SUB R1, R1, #8     E2411008
       SUBS R5, R1, R3    E0515003
       BNEQ MAIN          1AFFFFF9
       ADD R5,R2,R3       E0825003
```

그림 3. 모의실행의 입력에 이용된 ARM 프로그램과 기계어
Fig. 3. The input ARM program and machine code used for simulation

레지스터 값을 계산하여 공급하는 명령어의 재정렬버퍼 상의 항목번호이고, Addrs는 로드와 스토어의 유효주고 계산에 쓰이며, Values는 명령어의 연산결과를 수록한다.

재정렬버퍼 레코드형은 Busy, Opcode, Ready, Dest, Values, Addrs 등으로 구성된다. Ready는 명령어가 완료되었는지의 여부를 나타내며, Dest는 목적 레지스터의 번호를 기록하며 그 외에는 예약스테이션과 같다.

각 VHDL 모듈에서 엔티티문의 포트로 레코드형 배열을 입력 또는 출력으로 선언하여, 레코드형 배열 단위로 서로 다른 모듈 간에 효율적으로 정보를 주고받을 수 있다. 이 때, 서로 다른 VHDL 모듈에서 재정렬버퍼나 예약스테이션에 쓰기작업을 수행하는 것은 다중 원천 신호의 구동에 해당 되므로, 하드웨어 기술언어에서 금지된다. 따라서, 다중 원천 신호 구동을 해결하기 위한 현명하고 적절한 방안의 모색이 필요하다[4-6].

IV. 모의실험

본 논문의 모의실험은 운영체제 Fedora 30에서 3.1 GHz로 동작하는 Intel Core i5-2400 데스크탑 컴퓨터에서 시행하였다. VHDL 2008 버전을 컴파일하기 위한 도구로 GHDL 0.37 버전을, 모의실험 파형을 관찰하는 도구로 GTKWave 3.3.101 버전을 이용하였다. 총 18개의 독립적인 VHDL 프로그램 모듈이 비순차실행 ARM 프로세서를 설계하는데 이용되었다.

그림 3은 모의실행에서 비순차실행 프로세서의 입력으로 이용한 ARM 어셈블리 프로그램과 기계어를 나란히 표현한 것이다. 이 때, 로드와 스토어는 1 싸이클, 정수형 명령어는 1 싸이클, 곱셈을 실행하는 MUL 명령어는 3 싸이클이 소요된다.

그림 4에 토마술로 알고리즘을 이용하는 비순차실행 프로세서를 GHDL로 컴파일 후에 모의실행되는 것을 GTKWave의 결과 파형으로 나타냈다. 순차실행 프로세서인 경우에는, 입력 프로그램의 두 번째 명령어인 MUL

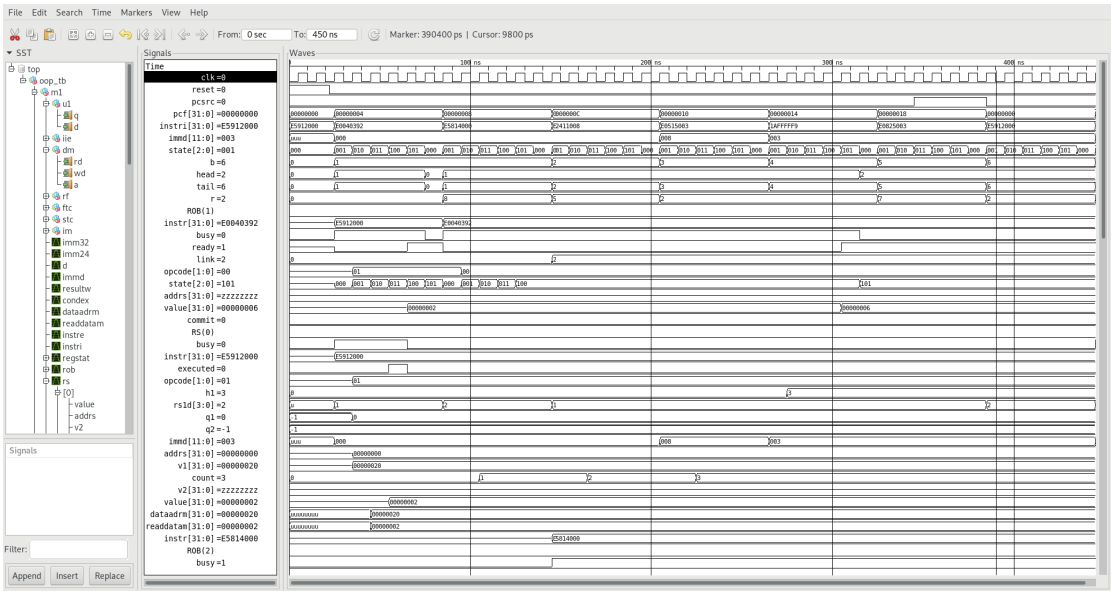

그림 4. 토마술로 알고리즘을 이용하는 비순차실행 프로서세서의 모의실행 결과
Fig. 4. The simulation results of the Out-of-order execution processor using Tomasulo Algorithm

연산이 완료되지 않았을 때, 그 이하의 명령어들은 발행 및 실행을 할 수가 없다. 그러나, 토마술로 알고리즘을 이용하는 비순차실행 프로세서에서는 두 번째와 세 번째인 MUL 명령어와 STR 명령어의 실행이 끝나지 않았더라도, 네 번째 명령어인 SUB의 발행 및 실행이 되었다. 그리고 분기여부가 결정되기 전에 BNEQ 분기어를 예측실행 하였다. 따라서, 비순차실행 프로세서가 토마술로 알고리즘과 예측실행을 기반으로 올바르게 실행하는 것을 확인할 수 있었다.

V. 결 론

본 논문에서는 Fedora 운영체제에서 VHDL을 이용하여, ARM 명령어를 기반으로 하는 비순차실행 프로세서를 설계하였다. 기존의 대다수 연구가 주요 자료구조인 재정렬버퍼와 예약스테이션을 배열을 이용하여 비순차실행 프로세서를 설계한 반면에, 본 논문에서는 레코드형을 이용하여 설계함으로써 프로그램 언어상으로 높은 수준의 추상도와 간결성을 확보하였다. 주어진 ARM 어셈블리 프로그램을 모의실행 시킨 결과, 토마술로 알고리즘에 따라 비순차실행 프로세서가 올바르게 처리되는 것을 확인할 수 있었다.

추후의 연구로, 멀티싸이클 방식으로 설계된 비순차실행 프로세서를 완전한 파이프라인 방식으로 수정하는 것이 첫 번째 목표이다. 두 번째로는 완전한 파이프라인 방식에 추가하여, 한 싸이클에 2개 이상의 명령어를 인출하여 실행하는 수퍼스칼라 프로세서로 개선시키는 것이다.

References

[1] R. M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," IBM Journal of Research and Development, Vol. 11, Issue. 1, Jan 1967, pp. 25-33. DOI:https://doi.org/10.1147/rd.111.0025

[2] A. Prout et al., "Measuring the Impact of Spectre and Meltdown," 2018 IEEE High Performance extreme Computing Conference (HPEC), Waltham, MA, 2018, pp. 1-5. DOI:https://doi.org/ 10.1109/hpec.2018.8547554.

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

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

[5] S. L. Harris, and D. M. Harris, "Digital Design and Computer Architecture ARM Edition", Elsevier Korea LLC, 2016. DOI:https://doi.org/10.1016/C2018-0-14352-8.

[6] J. Lee, “Design and Simulation of ARM Processor using

VHDL," Journal of The Institute of Internet, Broadcasting and Communication, vol. 18, no. 5, pp. 229-235, Oct. 2018.
DOI:https://doi.org/10.7236/JIIBC.2018.18.5.229.

저 자 소 개

이 종 복(정회원)

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

- Tel : 02-760-4497
- Fax : 02-760-4435
- E-mail : jblee@hansung.ac.kr
- 관심분야: 마이크로 프로세서,멀티코어 프로세서, 텐서 프로세서 유닛.

※ 본 연구는 한성대학교 교내학술연구비 지원과제임.