Computer Architecture

Multicycle CPU

날 짜 : 2019. 05.01

교수님 : 이성원 교수님

학 과 : 컴퓨터정보공학부

학 번 : 2015722025

이 름 : 정 용 훈

1. 문제의 해석 및 해결 방향

A. 실험 내용에 대한 설명

해당 프로젝트는 Multiple cycle CPU에서의 명령이 어떻게 동작하며, 이해하고 구현하는데 목적을 가지고 있다. 주요 동작 모듈은 MainFSM과 MyFSM이 있으며, 과제는 MyFSM에 SLTI, ANDI, LWAI, BGE를 구현하는데 목적을 둔다. MainFSM과 MyFSM의 동작은 1차 과제와 마찬가지로 inst.txt파일을 통하여 명령을 받아 MainFSM에 op code를 비교하여 op code가 없으면, MyFSM으로 동작을 넘겨 주어 instruction을 동작하게 하며, My에서 동작하는 instruction은 구현하는 명령들이 된다.

명령어 SLTI, ANDI는 I type이기 때문에 같은 state로 넘겨주어 op code를 비교하여 ALUop명령만 다르게 넣어주면 간단하게 해결되며, Fetch가 되기 전 state도 공통이다.

BGE명령은 처음 구현할 때 sub를 통하여 같은지 아닌지 판단 후 같지 않을 경우 다시 PC를 불러와 Greater인지 판단하는 작업을 하여 state가 2개 더 있었지만 역으로 SLT의 rs rt의 위치가 바뀌어있기 때문에 zero flag를 통하여 GE를 판단할 수 있기 때문에 ALU를 사용하여 state하나로 줄이게 되었다.

마지막으로 LWAI는 우선 Rs와 Rt를 더한 후 해당하는 메모리주소의 값을 추출하여 Rd에 저장하게 하였으며, Rt의 값을 +4 시켜주어 다시 Rt에 update시켜주어야 해야 하므로 각각의 write가 한번씩 되어야 하기 때문에 state가 4개로 정의 되어있다.

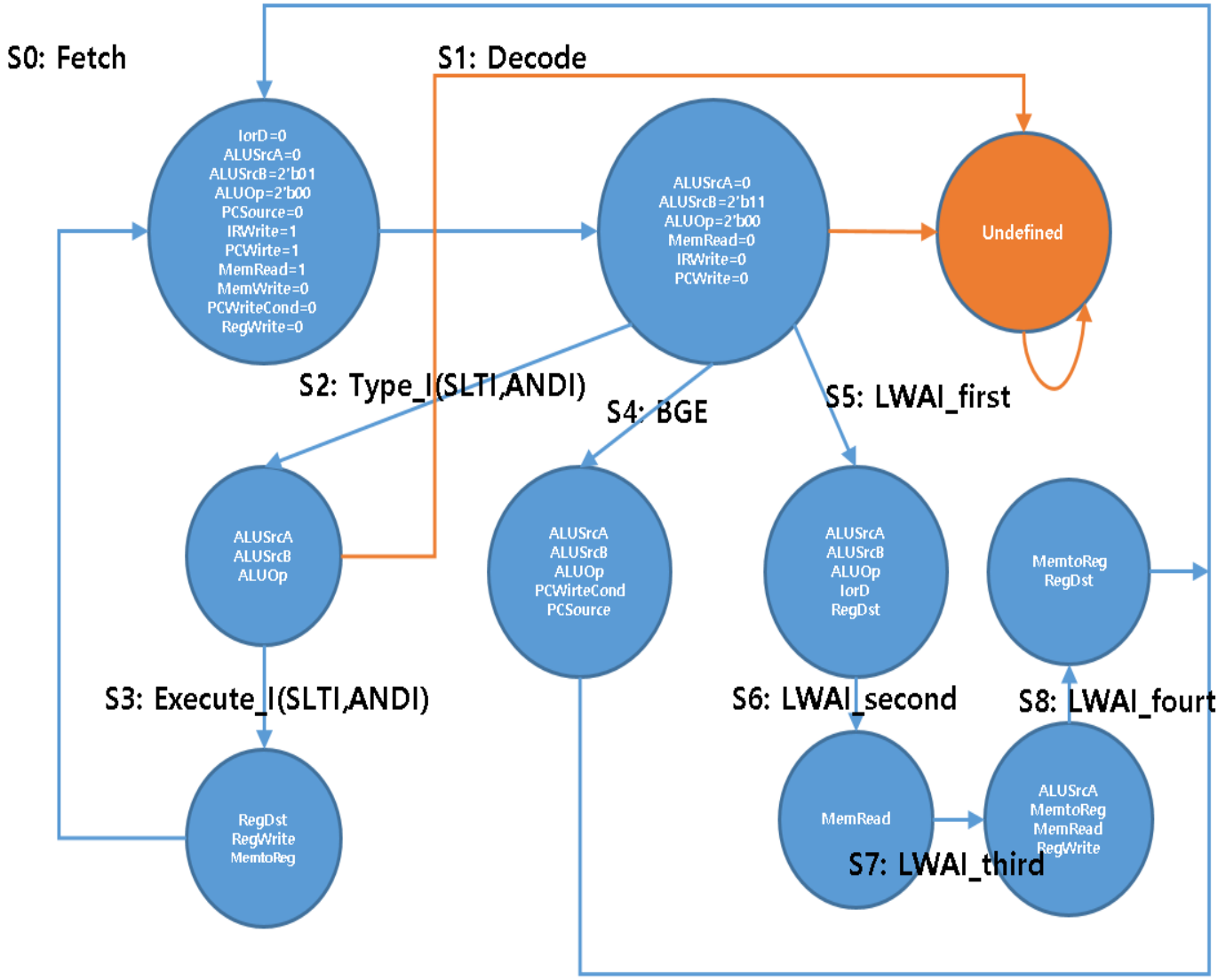
또한 구현된 FSM의 특징으로는 I type을 판단하는 state와 Decode state에서 MainFSM으로 권한을 넘기기 위한 Undefined가 정의되어 있으며, 모든 state들이 정상적으로 완료된다면, 다시 Fetch로 넘어가게 된다. 마지막으로 프로젝트의 TopFSM이 수정되기 전에 타이밍이 맞지 않는 문제가 있었으며, 해당 문제는 모든 Fetch state전에 타이밍을 #8만큼 밀어주면서 동기화를 시켜 문제를 해결한 경험이 있다.

B. 문제해결 방안

문제해결 방안은 1차 과제와 마찬가지로 제공된 MainFSM을 참고하였으며, 각각의 instruction이 동작하는 원리와 data path를 이해하고, FSM이므로 1차 과제와는 다르게 TopFSM수정 전 타이밍을 맞추는 작업을 하게 되었다. Multiple cycle의 전반적인 이해와 동작은 설계시간에 배운 내용이며, 구현하게 되는 명령의 동작은 배우내용을 응용하여 구현하게 되었다.

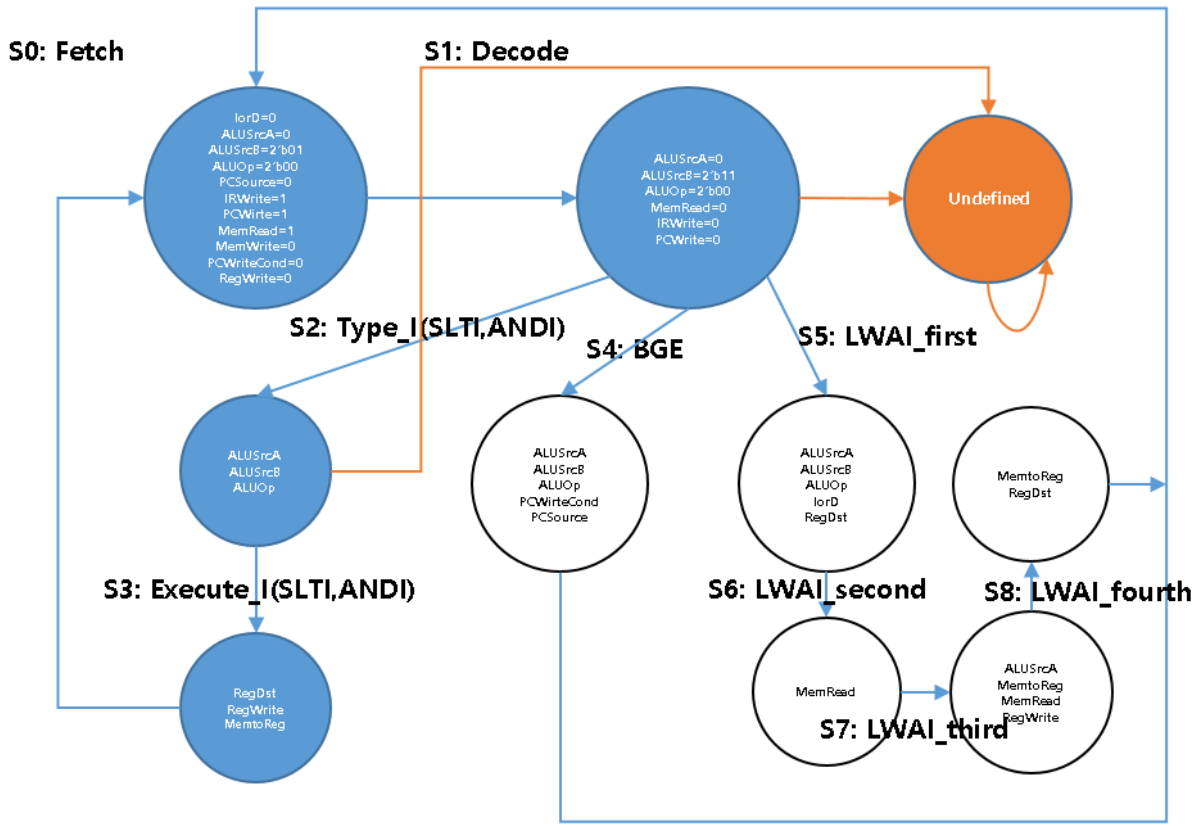
2. 설계 의도와 방법

A. 구현한 Multi Cycle CPU FSM Diagram



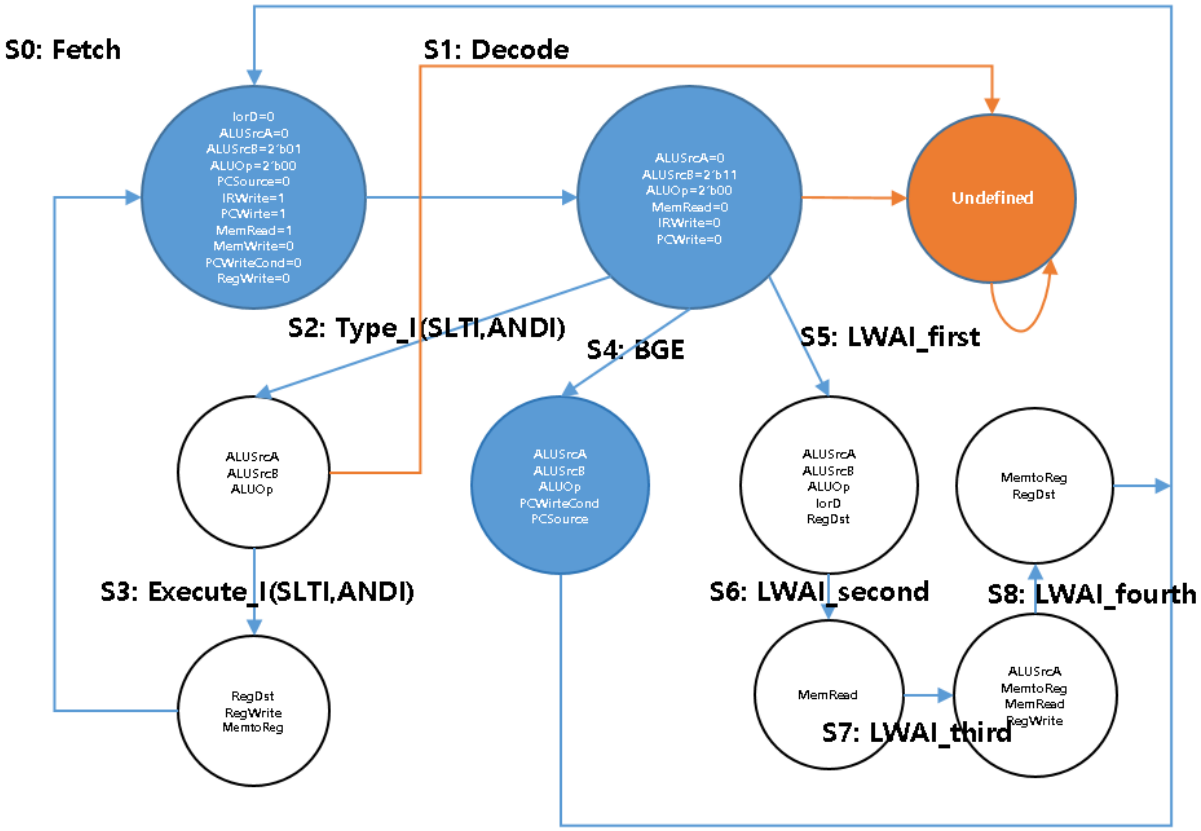
다음은 구현한 FSM의 동작 순서이며, 위에서 미리 언급한 것과 마찬가지로 동작을 하며, 각각의 Signal의 변화는 data들이 알맞게 들어가기 위한 signal이다. 각각 명령의 세부적인 FSM과 동작은 아래 그림과 같다.

1. ANDI and SLTI



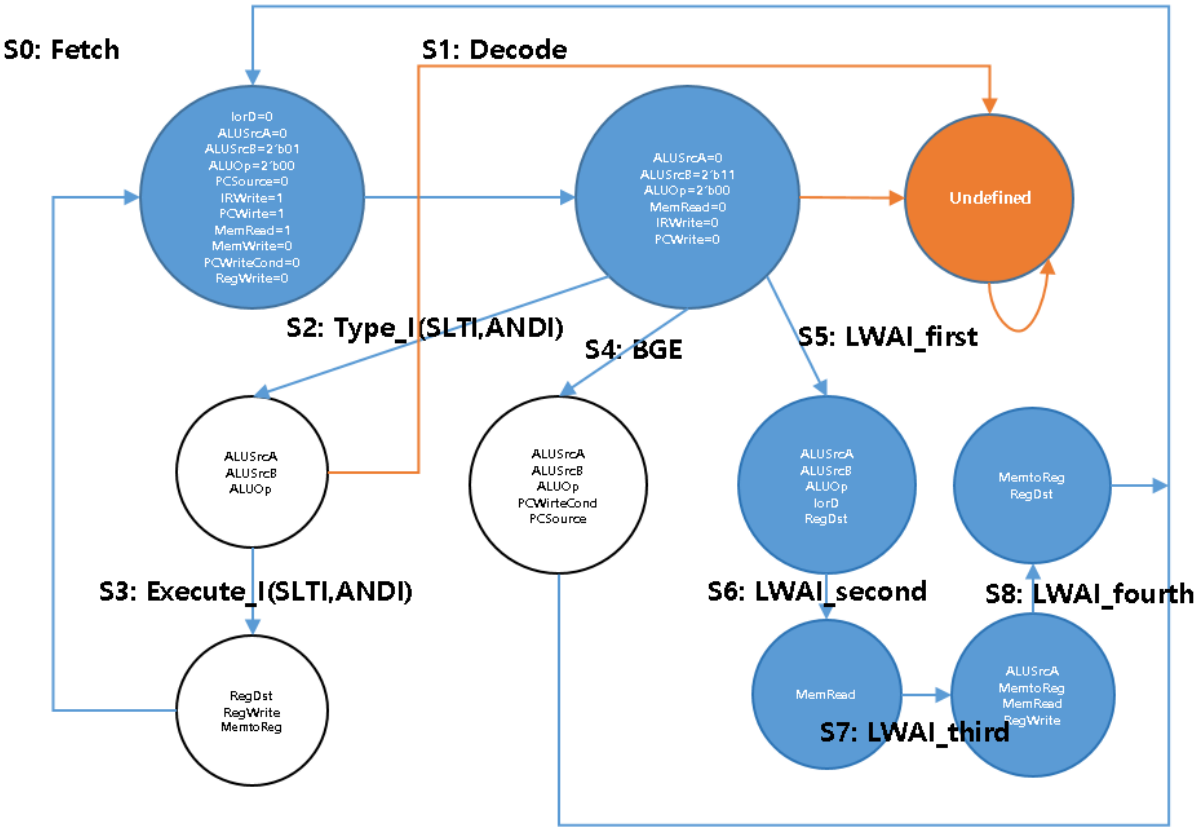
State	Working
Fetch	메모리에서 instruction을 가져오며, PC+4를 미리 계산하여 PC를 write한다.
Decide	명령에 알맞은 state에 들어가기 전 단계로 op code를 통해 명령을 판단하여 다음 state를 결정한다. 추가적으로 Branch 명령을 미리 계산하는 과정도 더해져 있다.
Type_I	직접 구현한 state로 SLTI나 ANDI가 들어오게 되며, ALUOp을 정해주는 것이 가장 중요한 수행 요소다.
Execute	ALU를통해 수행한 값을 알맞은 register에 write하는 state다.

2. BGE



State	Working
Fetch	메모리에서 instruction을 가져오며, PC+4를 미리 계산하여 PC를 write한다.
Decide	명령에 알맞은 state에 들어가기 전 단계로 op code를 통해 명령을 판단하여 다음 state를 결정한다. 추가적으로 Branch 명령을 미리 계산하는 과정도 더해져 있다.
BGE	BGE의 반대 명령은 SLT가 성립하기 때문에 ALUOp으로 SLT를 설정해주며, flag를 판단하여 branch를 할지 하지 않을지를 결정해주는 state다.

3. LWAI



State	Working
Fetch	메모리에서 instruction을 가져오며, PC+4를 미리 계산하여 PC를 write한다.
Decide	명령에 알맞은 state에 들어가기 전 단계로 op code를 통해 명령을 판단하여 다음 state를 결정한다. 추가적으로 Branch 명령을 미리 계산하는 과정도 더해져 있다.
LWAI_first	Rs와 Rt의 값을 더하여 메모리에 접근할 위치를 결정하는 과정
LWAI_second	접근한 memory를 읽어 쓰기 위해 읽어주는 과정
LWAI_third	읽은 값을 register에 쓰고, Rt를 Rt+4로 업데이트 해주기 위한 과정
LWAI_fourth	계산한 Rt+4를 Rt에 write해주는 과정이다.

B. 시뮬레이션 결과와 예상결과 비교분석

시뮬레이션은 다음과 같은 어셈블리 코드를 기반으로 동작하게 된다.

Assembly code

addi $v0 $v0 0 ->0

addi $v1 $v1 10 ->10

BGE $v0 $v1, 2

addi $v5 $v5 4 ->4

slti $v2 $v0 5 ->1

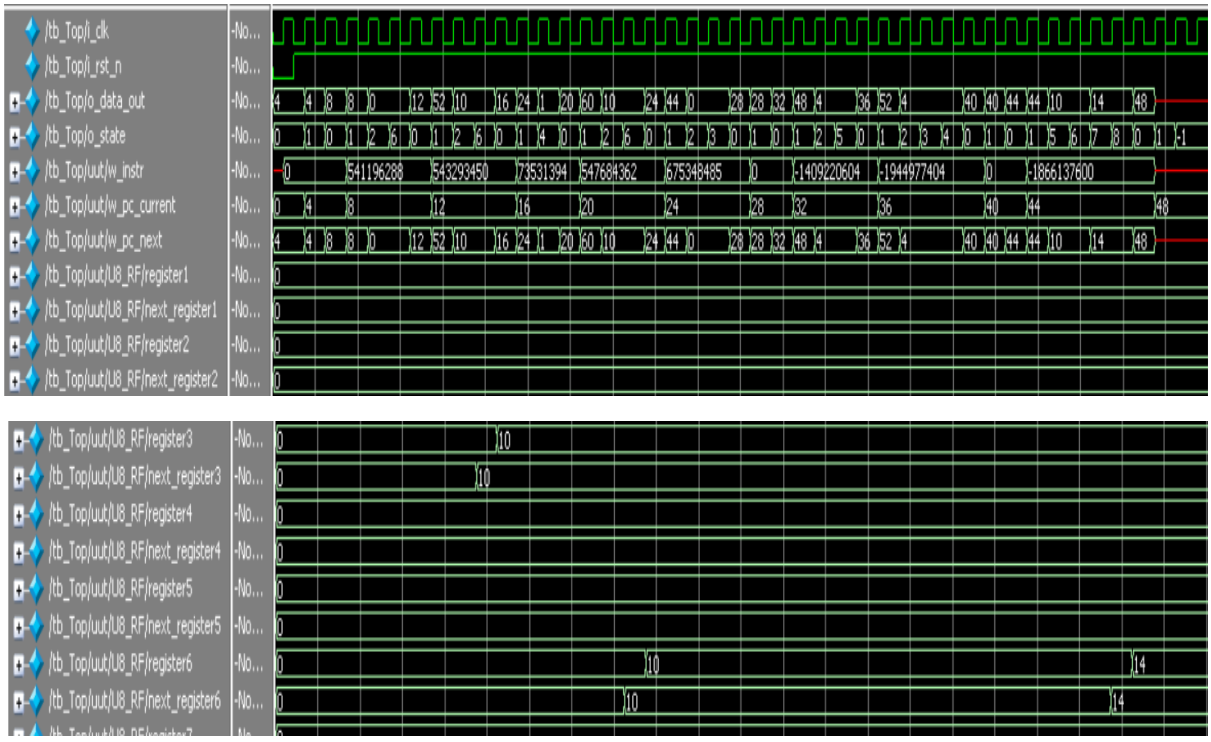
NOP

sw $v3, 4($zero)

lw $v2, 4($zero)

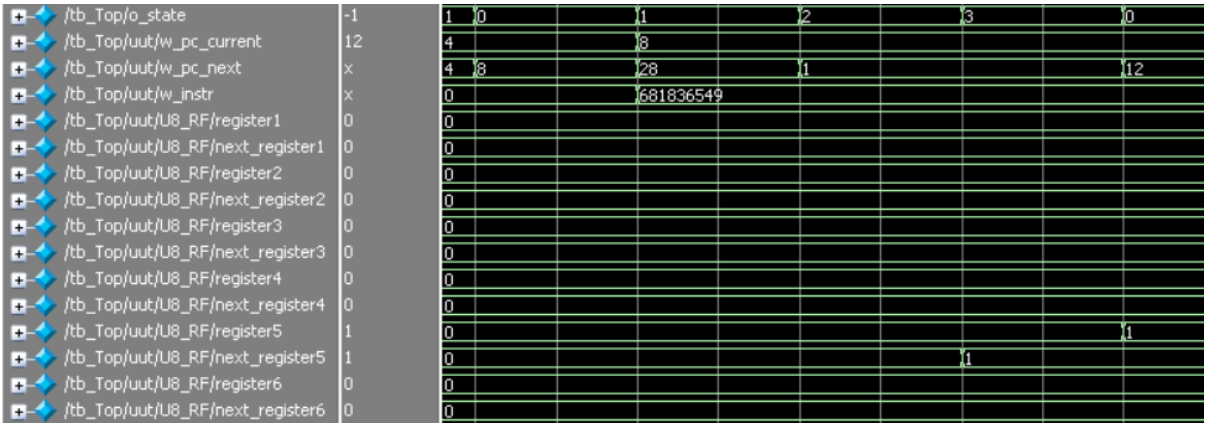
NOP

lwai $v4, $v5, $zero

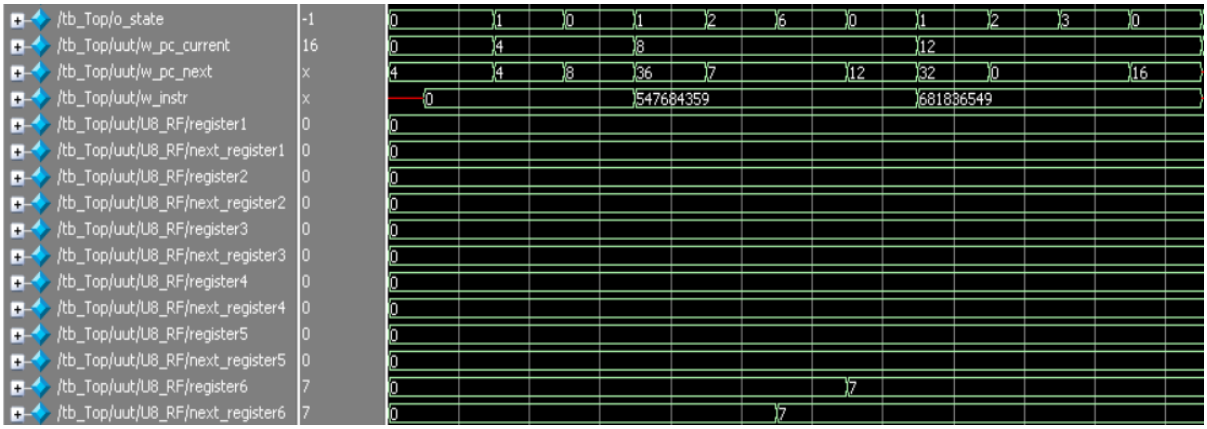


다음은 JAL을 뺀 제공된 명령에서 값이 들어가는 모습을 나타낸 것이다.

SLTI r5 r6 5

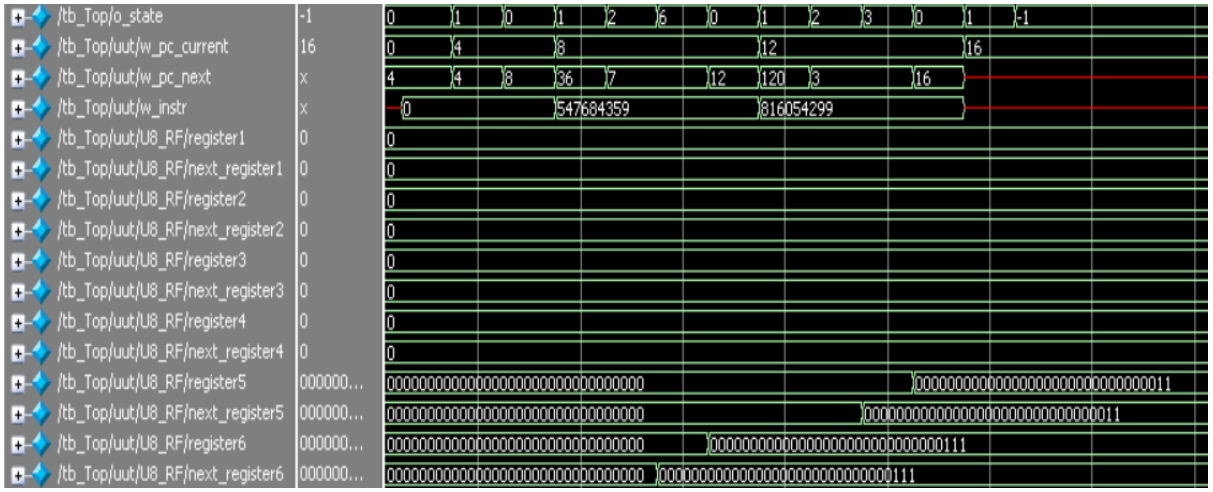


우선 r6에는 0이 저장되어있고 5랑 비교하기 때문에 r6의 값이 상대적으로 적기 때문에 r5에 1이 set되는 것을 확인할 수 있다. 이에 비교하여 r6의 값을 5보다 큰 값으로 설정해놓으면 아래와 같은 결과를 확인할 수 있다.



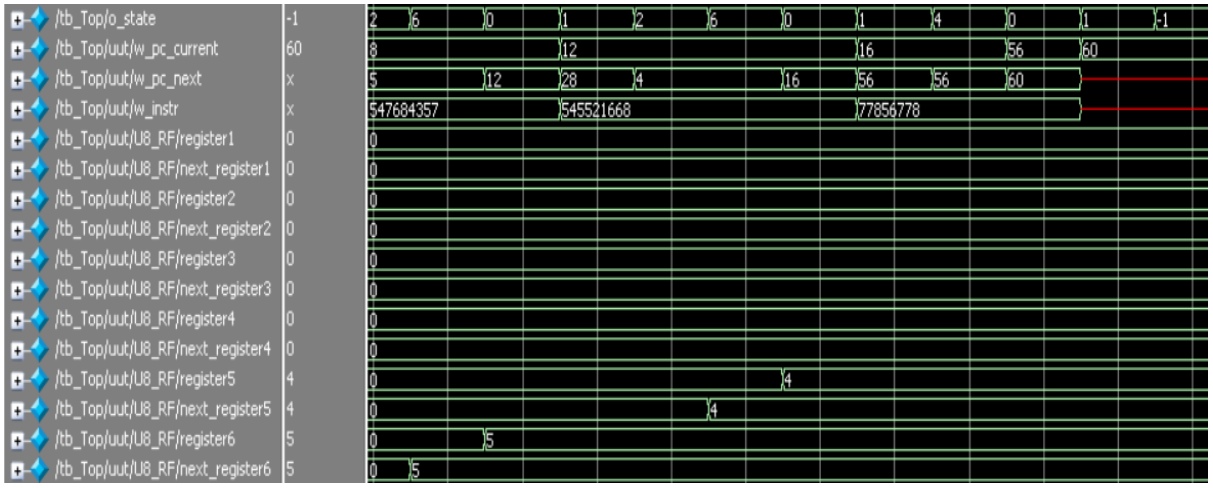
비교 값인 r6는 7을 저장하고 있으므로 5보다 값이 커 r5에 1이 set되지 않는 모습을 확인할 수 있다.

ANDI r5 r6 27



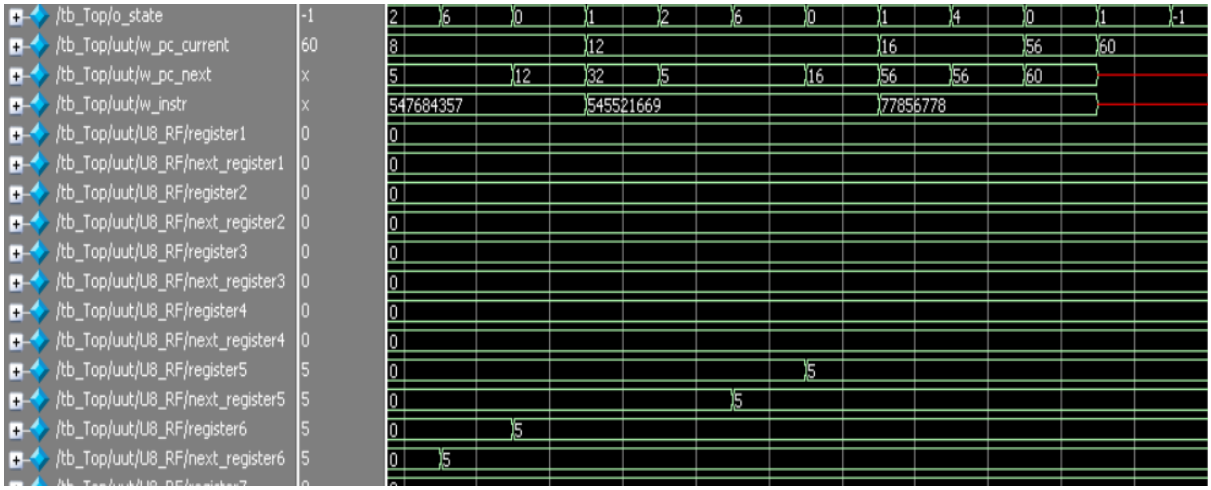
R6에는 마찬가지로 7, 즉 111의 값이 들어있고 imm값으로 27, 11011을 넣어주어 ANDI를 하게 되면 r5에 두 개의 값이 AND된 결과가 저장되는데 결과 화면을 보면 11로 잘 들어가있는 것을 확인할 수 있다.

BGE r6(5) r5(4) 10



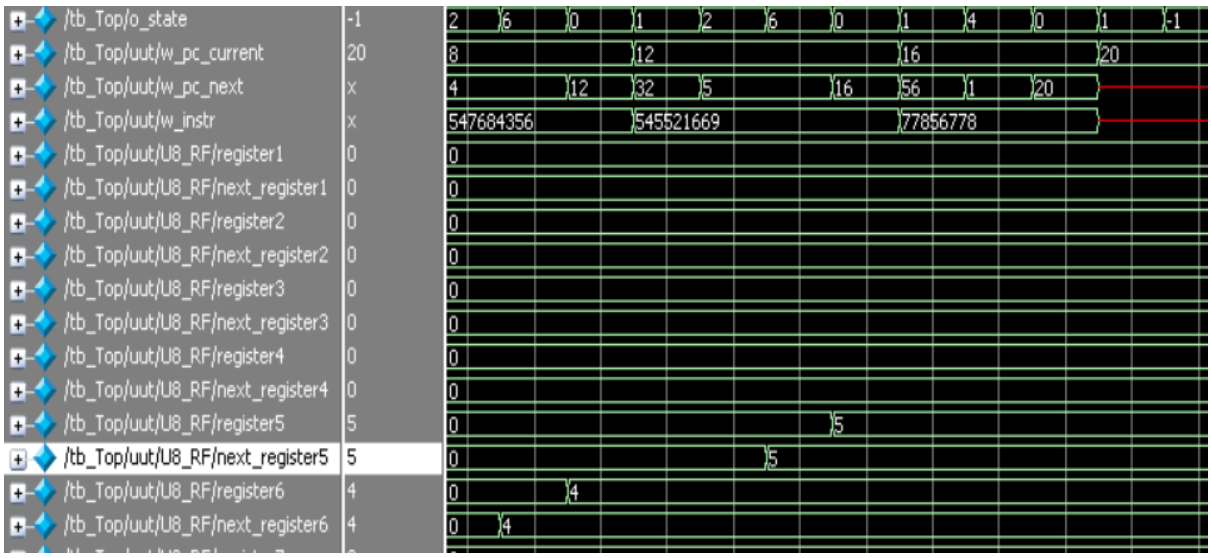
R6의 값이 5이며, r5의 값이 4이기 때문에 BGE가 성립하여 Branch가 성립한 모습이다. 10을 왼쪽으로 두 번 shift하면 40이고 원래 PC에 40을 더한 값인 56이 PC값으로 변하게 된다. 아래는 값이 같을 때와 BGE를 성립하지 않는 경우다.

BGE r6(5) r5(5) 10



같은 경우도 Branch가 동작하는 모습을 볼 수 있다.

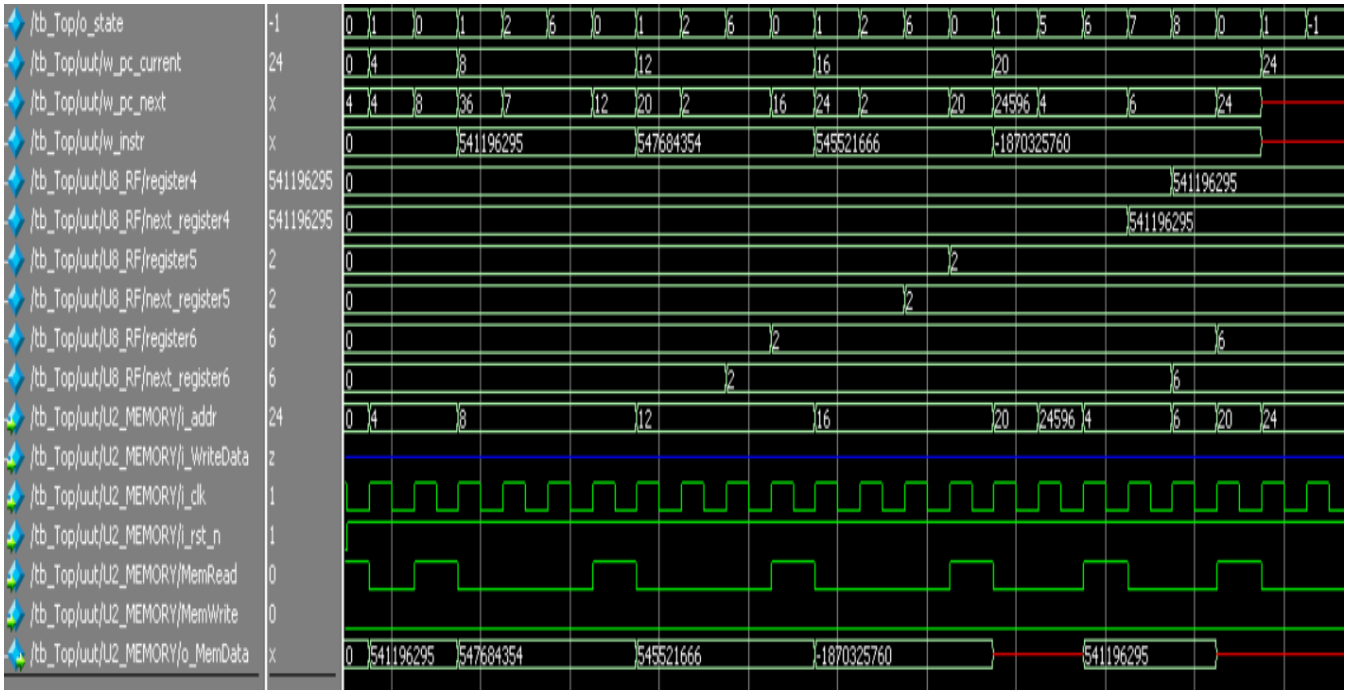
BGE r6(4) r5(5) 10



PC값이 16으로 12에 이어 진행되는 것을 확인할 수 있다.

LWAI r4 r5 r6

해당 명령은 원래 정의되어있지 않은 명령어지만 이번 프로젝트에서 구현해보는 명령이다. 해당 명령의 동작은 rs, rt의 값을 더하여, 즉 r6[r5]위치에 있는 데이터를 불러와 rd값에 넣어주고, rt는 rt+4로 업데이트 해주면 된다. 해당 명령의 실행은 다음과 같이 동작한다.



동작을 보면 rs인 r5와 rt인 r6을 더하여 4를 만들고 memory에서 4에 해당하는 값인 541196295를 불러와 r4에 저장하는 모습을 확인할 수 있다. 또한 rt인 r6의 값이 2에서 6으로 업데이트되는 것을 확인할 수 있으며, 해당 명령어가 정상적으로 동작하는 것을 확인할 수 있다.