Computer Architecture

Project 2

(MIPS Multi-Cycle CPU Implementation)

학　　과: 컴퓨터정보공학부

담당교수: 이성원 교수님

실습분반: 금34

학　　번: 2020202037

성　　명: 염정호

1. 프로젝트 개요

본 프로젝트는 MIPS(Microprocessor without Interlocked Pipeline Stages) 아키텍처의 멀티 사이클 CPU를 구현하는 것을 목표로 한다. 해당 프로젝트에서 모든 명령어에서 공통적으로 들어가는 명령어를 Fetch하는 부분과 Instr Decode & Register Fetch는 구현 되어 있으며 이후에 실행되는 명령어의 동작 과정을 작성하면 된다. LW, SW, ORI, ADD, SUB, J, LUI, BREAK, LLO, LHI 명령어들은 이미 구현 되어 있으며 추가로 XORI, SLT, SUBU, SRA, MULTU, MFLO, SB, LHU, BLEZ, JR 총 10개의 명령어를 구현하고 동작결과를 확인한다.

2. 설계 세부사항

멀티 사이클 CPU module

module name	Description/function
PC	Program counter
	다음에 읽어올 명령어의 위치를 가리키는 레지스터
MEM	Memory (IM+DM)
	명령어들과 프로그램에서 사용하는 데이터가 저장가능한 레지스터
IR	Instruction Register
	명령어를 읽어와서 각 명령어에 맞는 동작을 각 모듈로 전달
MDR	Memory data register
	메모리에서 값을 가져와서 사용할 때 이용하는 레지스터
FSM	Finite State Machine Main Controller
	IR에서 받은 opcode를 이용해 다른 모듈 동작
RF	Register File
	레지스터들의 모음 해당 레지스터에 값을 쓰기/읽기 가능
A	Temporary register for Read data1
	alu에서 사용하기 위해 레지스터에서 읽어온 값
B	Temporary register for Read data2
	alu에서 사용하기 위해 메모리 또는 레지스터 또는 imm value가 저장
SEU	Sign Extend Unit
	imm value이용시 16bit데이터를 32bit로 확장 할 때 사용
ALU	Arithmetic Logical Unit
	주소, 데이터 연산에 사용되는 모듈
MUL	Multiplier Unit
	곱셈을 연산 할 때 사용되는 모듈
ALUO	Temporary register for ALU result
	ALU의 연산 결과가 저장되는 모듈

명령어별 수행 과정

Stage #	R-type instructions	Memory instructions	Branches	Jumps	Remarks
0	IR = Memory[PC] PC = PC + 4				Instr Fetch
1	A = Reg [IR[25:21]] B = Reg [IR[20:16]] ALUOut = PC + (Sign-Extened (IR[15:0]) << 2)				Instr Decode & Register Fetch
2	ALUOut = A op B	ALUOut = A + Sign-Extended (IR[15:0])	If (A==B) then PC = ALUOut	PC = { PC[31:28], (IR[25:0]<	Execution or Branch/Jump Completion
3	Reg [IR[15:11]] = ALUOut	Load: MDR = Memory[ALUOut] Store: Memory [ALUOut] = B	If (A==B) then PC = ALUOut	PC = { PC[31:28], (IR[25:0]<<2) }	Memory Access & R-type Completion
4		Load: Reg [IR[20:16]] = MDR			Memory Read Completion

수행 과정 중 0번(instr fetch)와 1번(decde)단계는 이미 구현

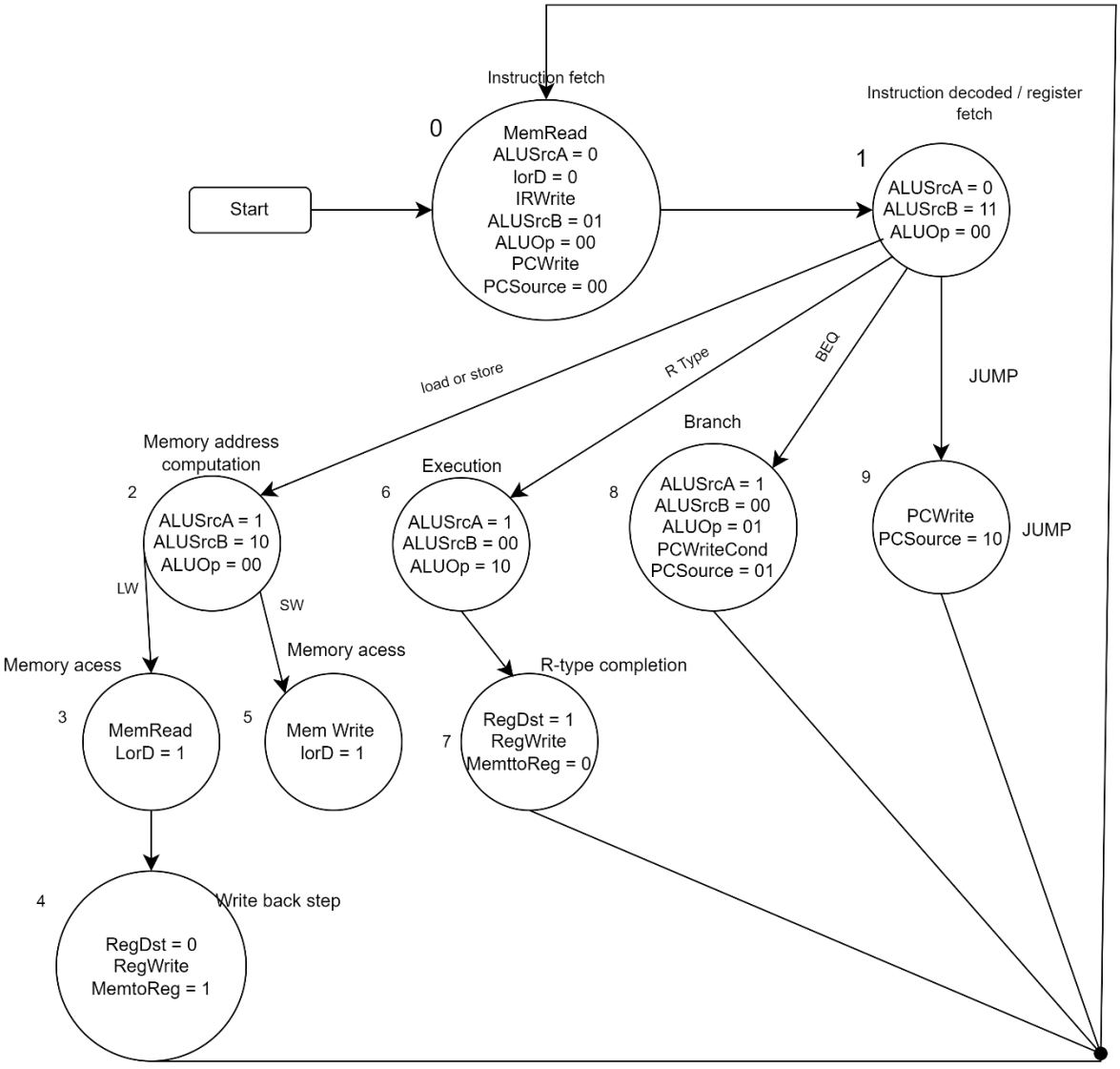
명령어의 종류에 따라 총 5단계로 나뉜다.

control signal

port name	Description
IorD	Memory Access for Instruction or Data
MemRead	Memory Read Access Enable
MemWrite	Memory Read Access Enable
DatWidth	Data Width for Memory Access
IRwrite	Instruction Register Write Enable
RegDst	Selection for Destination Register
RegDatSel	To Select Data Source for Register Write
RegWrite	Write Enable to Register File
EXTmode	Immediate Data Extension Mode
ALUsrcA	ALU Input A Source Selection
ALUsrcB	ALU Input B Source Selection
ALUop	ALU Operation Code
ALUctrl[1:0]	Extra ALU Control Signal
Branch	Branch Options

PCsrc	PC Data Source Selection
PCwrite	PC Register Write Enable
StateSel	Next State Selection Signal

멀티사이클 CPU FSM



기능 및 동작

모든 명령어 공통 – fetch

0_1_0_000_1_xx_xxx_0_x_011_001_00100_00_000_00_1_xxxxxxxx_11

IorD	0	Instruction
MemRead	1	Memory Read Access
MemWrite	0	No Memory Write
DatWidth	000	32-bit Word
IRwrite	1	Instruction Register Write Enable
RegDst	xx	레지스터에 값 저장x
RegDatSel	xxx	레지스터에 값 저장x
RegWrite	0	레지스터 쓰기 x
EXTmode	x	imm value 사용 x
ALUsrcA	011	PC to ALU input A
ALUsrcB	001	0x4
ALUop	00100	a + b
ALUctrl[1:0]	00	Normal ALU input (a,b)/ Shift = Shift Amount
Branch	000	v
PCsrc	00	From ALU output
PCwrite	1	PC Register Write Enable
StateSel	11	Next State = Current State + 1

명령어 공통 - Instr Decode & Register Fetch

x_x_0_xxx_0_xx_xxx_0_1_011_100_00100_00_xxx_xx_0_xxxxxxxx_01

IorD	x	메모리 사용 x
MemRead	x	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	xx	레지스터에 값 저장x
RegDatSel	xxx	레지스터에 값 저장x
RegWrite	0	레지스터 쓰기 x
EXTmode	1	sign Extension

ALUsrcA	011	PC to ALU input A
ALUsrcB	100	SEU output << 2
ALUop	00100	a + b
ALUctrl[1:0]	00	Normal ALU input (a,b)/ Shift = Shift Amount
Branch	000	No branch condition or Jump
PCsrc	xx	pc에 값 load x
PCwrite	0	No PC Register Write
StateSel	11	Next State = Current State + 1

XORI $t = $s ^ ZE(i)

stage3)

x_0_0_xxx_0_xx_xxx_0_1_000_011_00011_0x_xxx_xx_0_xxxxxxxx_11

IorD	x	메모리 사용 x
MemRead	0	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	xx	레지스터에 값 저장x
RegDatSel	xxx	레지스터에 값 저장x
RegWrite	0	레지스터 쓰기 x
EXTmode	0	Zero Extension
ALUsrcA	000	Register A to ALU input A
ALUsrcB	011	SEU output to ALU input B
ALUop	00011	Bitwise XOR
ALUctrl[1:0]	0x	Normal ALU input (a,b)
Branch	xxx	branch 사용 x
PCsrc	xx	pc에 값 load x
PCwrite	0	No PC Register Write
StateSel	11	Next State = Current State + 1

zero extend IR값과 A레지스터 값의 xor연산 결과가 ALUOUT에 저장.

stage4)

x_0_0_xxx_0_00_000_1_x_xxx_xxx_xxxxx_xx_xxx_xx_0_xxxxxxxx_00

IorD	x	메모리 사용 x
MemRead	0	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	1	Instruction Register Write Enable
RegDst	00	Write to $rt
RegDatSel	000	Write ALUOut to Register file
RegWrite	1	Write to Register file
EXTmode	x	EXT사용 x
ALUsrcA	xxx	ALU사용 x
ALUsrcB	xxx	ALU사용 x
ALUop	xxxxx	ALU사용 x
ALUctrl[1:0]	xx	ALU사용 x
Branch	xxx	branch사용 x
PCsrc	xx	pc에 값 load x
PCwrite	0	No PC Register Write
StateSel	00	Next State = 0

ALUOUT에 저장된 값이 선택된 레지스터에 저장.

slt $d = ($s < $t)

stage 3)

x_x_0_xxx_0_xx_xxx_0_1_000_000_10000_0x_xxx_xx_0_xxxxxxxx_11

IorD	x	메모리 사용 x
MemRead	x	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	xx	레지스터에 값 저장x
RegDatSel	xxx	레지스터에 값 저장x
RegWrite	0	레지스터 쓰기 x

EXTmode	1	Sign Extension
ALUsrcA	000	Register A to ALU input A
ALUsrcB	000	SEU output to ALU input B
ALUop	10000	Set Less Than
ALUctrl[1:0]	0x	Normal ALU input (a,b)
Branch	xxx	No branch condition or Jump
PCsrc	xx	pc에 값 load x
PCwrite	0	No PC Register Write
StateSel	11	Next State = Current State + 1

$s<$t연산 수행 결과를 ALUOUT에 저장한다

stage 4)

x_x_0_xxx_0_01_000_1_x_xxx_xxx_xxxxx_xx_xxx_xx_0_xxxxxxxx_00

IorD	x	메모리 사용 x
MemRead	x	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	01	Write to $rd
RegDatSel	000	Write ALUOut to Register file
RegWrite	1	Write to Register file
EXTmode	x	Sign Extension
ALUsrcA	xxx	ALU미사용
ALUsrcB	xxx	ALU미사용
ALUop	xxxxx	ALU미사용
ALUctrl[1:0]	xx	ALU미사용
Branch	xxx	branch 미사용
PCsrc	xx	branch 미사용
PCwrite	0	No branch condition or Jump
StateSel	00	Next State = 0

ALUOUT에 있는 값을 읽어와 $d레지스터에 저장

subu $d = $s - $t

stage 3)

x_x_0_xxx_0_xx_xxx_0_x_000_000_00111_00_xxx_xx_0_xxxxxxxx_11

IorD	x	메모리 사용 x
MemRead	x	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	xx	레지스터에 값 저장x
RegDatSel	xxx	레지스터에 값 저장x
RegWrite	0	레지스터 쓰기 x
EXTmode	x	Imm value 사용 x
ALUsrcA	000	Register A to ALU input A
ALUsrcB	000	SEU output to ALU input B
ALUop	00111	Unsigned a – b
ALUctrl[1:0]	00	Normal ALU input (a,b)/shift사용 안함
Branch	xxx	branch 미사용
PCsrc	xx	branch 미사용
PCwrite	0	branch 미사용
StateSel	11	Next State = Current State + 1

$s - $t연산을 수행한 결과를 ALUout에 저장

stage4)

x_x_0_xxx_0_01_000_1_x_xxx_xxx_xxxxx_xx_xxx_xx_0_xxxxxxxx_00

IorD	x	메모리 사용 x
MemRead	x	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	01	Write to $rd
RegDatSel	000	Write ALUOut to Register file
RegWrite	1	Write to Register file
EXTmode	x	Sign Extension
ALUsrcA	xxx	ALU미사용
ALUsrcB	xxx	ALU미사용
ALUop	xxxxx	ALU미사용
ALUctrl[1:0]	xx	ALU미사용

Branch	xxx	branch 미사용
PCsrc	xx	branch 미사용
PCwrite	0	No branch condition or Jump
StateSel	00	Next State = 0

ALUOUT에 있는 값을 읽어와 $d레지스터에 저장

sra $d = $t >>> a

stage 3)

x_x_0_xxx_0_xx_xxx_0_x_000_000_01111_0x_xxx_xx_0_xxxxxxxx_11

IorD	x	메모리 사용 x
MemRead	x	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	xx	레지스터에 값 저장x
RegDatSel	xxx	레지스터에 값 저장x
RegWrite	0	레지스터 쓰기 x
EXTmode	x	Imm value 사용 x
ALUsrcA	000	Register A to ALU input A
ALUsrcB	000	SEU output to ALU input B
ALUop	01111	b >>> a
ALUctrl[1:0]	0x	Normal ALU input (a,b)/shift사용 안함
Branch	xxx	branch 미사용
PCsrc	xx	branch 미사용
PCwrite	0	branch 미사용
StateSel	11	Next State = Current State + 1

$t레지스터 값을 읽어와 shamt만큼 shift right한 뒤 해당 값을 aluout에 저장

stage 4

x_x_0_xxx_0_01_000_1_x_xxx_xxx_xxxxx_xx_xxx_xx_0_xxxxxxxx_00

IorD	x	메모리 사용 x
MemRead	x	No Memory Read Access
MemWrite	0	No Memory Write

DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	01	Write to $rd
RegDatSel	000	Write ALUOut to Register file
RegWrite	1	Write to Register file
EXTmode	x	Sign Extension
ALUsrcA	xxx	ALU미사용
ALUsrcB	xxx	ALU미사용
ALUop	xxxxx	ALU미사용
ALUctrl[1:0]	xx	ALU미사용
Branch	xxx	branch 미사용
PCsrc	xx	branch 미사용
PCwrite	0	No branch condition or Jump
StateSel	00	Next State = 0

ALUOUT에 있는 값을 읽어와 $d레지스터에 저장

MULTU : multu hi:lo = $s * $t

stage 3

x_x_0_xxx_0_xx_xxx_0_x_000_000_01010_0x_xxx_xx_0_xxxxxxxx_00

IorD	x	메모리 사용 x
MemRead	x	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	xx	레지스터에 값 저장x
RegDatSel	xxx	레지스터에 값 저장x
RegWrite	0	레지스터 쓰기 x
EXTmode	x	Imm value 사용 x
ALUsrcA	000	Register A to ALU input A
ALUsrcB	000	SEU output to ALU input B
ALUop	01010	Unsigned a × b
ALUctrl[1:0]	0x	Normal ALU input (a,b)/shift사용 안함
Branch	xxx	branch 미사용

PCsrc	xx	branch 미사용
PCwrite	0	branch 미사용
StateSel	00	Next State = Current State + 1

$s와 $t를 곱연산 해당 결과는 mul레지스터 내부에 보관됨

레지스터에 쓰기를 하지 않기 때문에 3사이클에 끝남

mflo $d = lo

stage 3

x_0_0_xxx_0_01_010_1_x_xxx_xxx_xxxxx_xx_xxx_xx_x_xxxxxxxx_00

IorD	x	메모리 사용 x
MemRead	0	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	01	Write to $rd
RegDatSel	010	Write LO to Register file
RegWrite	1	Write to Register file
EXTmode	x	Imm value 사용 x
ALUsrcA	xxx	ALU미사용
ALUsrcB	xxx	ALU미사용
ALUop	xxxxx	ALU미사용
ALUctrl[1:0]	xx	ALU미사용
Branch	xxx	branch 미사용
PCsrc	xx	branch 미사용
PCwrite	x	branch 미사용
StateSel	00	Next State = Current State + 1

이전 곱 연산의 결과로 남아있는 결과값 중 하위값을 가져온다

SB

stage 3

x_x_0_xxx_0_xx_xxx_0_1_000_011_00100_0x_xxx_xx_0_xxxxxxxx_11

IorD	x	메모리 사용 x
MemRead	x	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	xx	레지스터 쓰기x
RegDatSel	xxx	레지스터 쓰기x
RegWrite	0	레지스터 쓰기x
EXTmode	1	sign extend
ALUsrcA	000	A
ALUsrcB	011	imm vlaue
ALUop	00100	+연산
ALUctrl[1:0]	0x	a,b값 그대로 사용 /shift미사용
Branch	xxx	branch 미사용
PCsrc	xx	branch 미사용
PCwrite	0	branch 미사용
StateSel	11	Next State = Current State + 1

저장할 메모리의 주소를 계산해 aluout에 저장

stage 4

1_x_1_011_0_xx_xxx_0_x_xxx_xxx_xxxxx_xx_xxx_xx_0_xxxxxxxx_00

IorD	1	메모리 사용 x
MemRead	x	No Memory Read Access
MemWrite	1	No Memory Write
DatWidth	011	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	xx	Write to $rd

RegDatSel	xxx	Write ALUOut to Register file
RegWrite	0	Write to Register file
EXTmode	x	Sign Extension
ALUsrcA	xxx	ALU미사용
ALUsrcB	xxx	ALU미사용
ALUop	xxxxx	ALU미사용
ALUctrl[1:0]	xx	ALU미사용
Branch	xxx	branch 미사용
PCsrc	xx	branch 미사용
PCwrite	0	No branch condition or Jump
StateSel	00	Next State = 0

aluout에 존재하는 주소에 $t데이터 저장.

LHU load half word

stage3

x_x_0_xxx_0_xx_xxx_0_1_000_011_00100_0x_xxx_xx_0_xxxxxxxx_11

IorD	x	메모리 사용 x
MemRead	x	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	xx	레지스터 쓰기x
RegDatSel	xxx	레지스터 쓰기x
RegWrite	0	레지스터 쓰기x
EXTmode	1	sign extend
ALUsrcA	000	A
ALUsrcB	011	imm vlaue
ALUop	00100	+연산
ALUctrl[1:0]	0x	a,b값 그대로 사용
Branch	xxx	branch 미사용
PCsrc	xx	branch 미사용
PCwrite	0	branch 미사용
StateSel	11	Next State = Current State + 1

stage 4

1_1_0_010_0_xx_xxx_0_x_xxx_xxx_xxxxx_xx_xxx_xx_0_xxxxxxxx_11

IorD	1	메모리 사용 x
MemRead	1	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	010	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	xx	레지스터 쓰기x
RegDatSel	xxx	레지스터 쓰기x
RegWrite	0	레지스터 쓰기x
EXTmode	x	sign extend
ALUsrcA	xxx	A
ALUsrcB	xxx	imm vlaue
ALUop	xxxxx	+연산
ALUctrl[1:0]	xx	a,b값 그대로 사용
Branch	xxx	branch 미사용
PCsrc	xx	branch 미사용
PCwrite	0	branch 미사용
StateSel	11	Next State = Current State + 1

stage 50

x_0_0_xxx_0_00_001_1_x_xxx_xxx_xxxxx_xx_xxx_xx_0_xxxxxxxx_00

IorD	x	메모리 사용 x
MemRead	0	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	00	레지스터 쓰기x
RegDatSel	001	레지스터 쓰기x
RegWrite	1	레지스터 쓰기x
EXTmode	x	sign extend
ALUsrcA	xxx	A
ALUsrcB	xxx	imm vlaue
ALUop	xxxx	+연산

ALUctrl[1:0]	xx	a,b값 그대로 사용
Branch	xxx	branch 미사용
PCsrc	xx	branch 미사용
PCwrite	0	branch 미사용
StateSel	00	Next State = Current State + 1

BLEZ if ($s <= 0) pc += i << 2

x_0_0_xxx_0_xx_xxx_0_x_000_010_00100_0x_110_01_1_xxxxxxxx_00

IorD	x	메모리 사용 x
MemRead	0	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write
RegDst	xx	레지스터 쓰기x
RegDatSel	xxx	레지스터 쓰기x
RegWrite	0	레지스터 쓰기x
EXTmode	x	sign extend
ALUsrcA	000	A
ALUsrcB	010	imm vlaue
ALUop	00100	+연산
ALUctrl[1:0]	0x	a,b값 그대로 사용 / shift x
Branch	110	0보다 작거나 같은 경우 즉 양수가 아닌 경우branch
PCsrc	01	decode과정에서 imm + pc값 생성
PCwrite	1	pc값 입력
StateSel	00	명령어 종료

JR

x_0_0_xxx_0_xx_xxx_0_x_000_010_00001_0x_000_00_1_xxxxxxxx_00

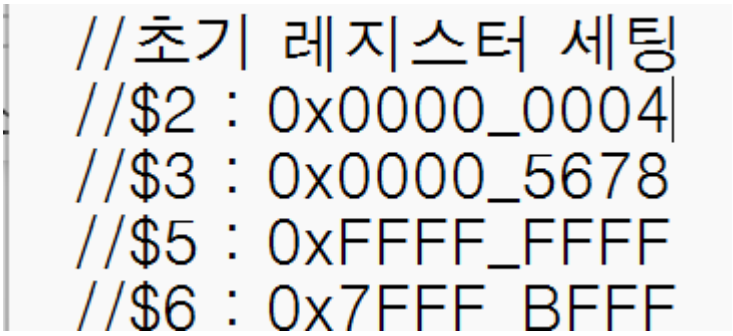
IorD	x	메모리 사용 x
MemRead	0	No Memory Read Access
MemWrite	0	No Memory Write
DatWidth	xxx	메모리 사용 x
IRwrite	0	No Instruction Register Write

RegDst	xx	레지스터 쓰기x
RegDatSel	xxx	레지스터 쓰기x
RegWrite	0	레지스터 쓰기x
EXTmode	x	sign extend
ALUsrcA	000	A = $s
ALUsrcB	010	0
ALUop	00001	+연산
ALUctrl[1:0]	0x	a,b값 그대로 사용 / shift x
Branch	000	조건 없는 분기
PCsrc	00	alu연산 값을 사용
PCwrite	1	pc값 입력
StateSel	00	명령어 종료

3. 결과

괄호 안은 예상 값

초기 레지스터 저장 값 해당 값을 이용해 연산 수행



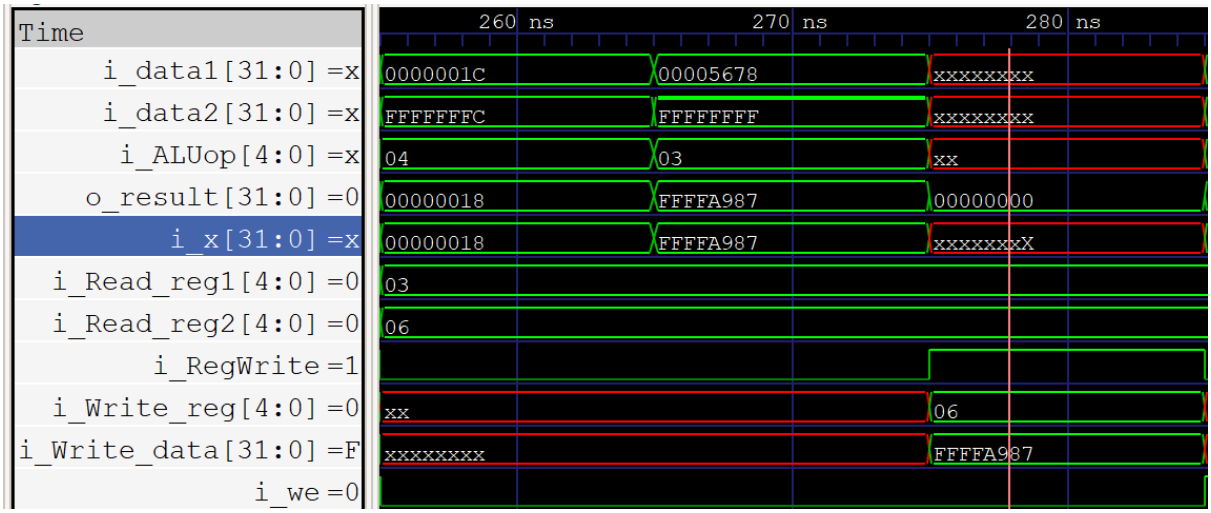
XORI XORI $t = $s ^ ZE(i)

```
00111000_01100110_11111111_11111111 // $3(5678)xori FFFF -> $0(0000A987
```

수행 결과

```
00000006 : 11111111_11111111_10101001_10000111 : ffffa987
```

테스트벤치



1. register A와 imm value를 읽어와 aluout에 저장
2. aluout값을 regiser $6에 저장

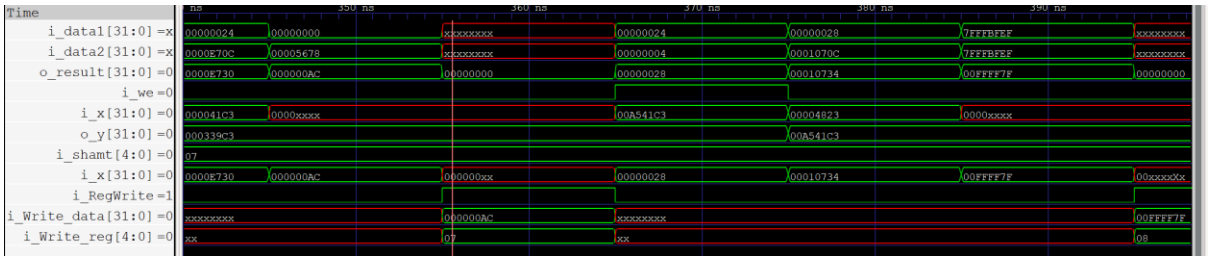
SRA $d = $t >>> a

```
000000_00000_00011_00111_00111_000011 // $7 = $3 >>>7(AC)
000000_00101_00101_01000_00111_000011 // $8 = $5>>>7(00ffff7f)
```

수행결과

```
000000_00000_00011_00111_00111_000011 // $7 = $3 >>>7(AC)
000000_00101_00101_01000_00111_000011 // $8 = $5>>>7(00ffff7f)
```

테스트 벤치



1. $t레지스터 값을 shamt만큼 shift해 aluout에 저장
2. aluout값을 regiser $d에 저장

SUBU $d = $s - $t

```
000000_00101_00011 _01001_00000_100011 // $5-$3 = $9(fff 6977)

000000_00101_00110_01010_00000_100011 // $5-$6 = $10(7fff 1668)
```

수행결과

```
00000009 : 01111111_11111111_01101001_01110111 : 7fff6977
0000000a : 01111111_11111111_00010110_01101000 : 7fff1668
```

테스트벤치

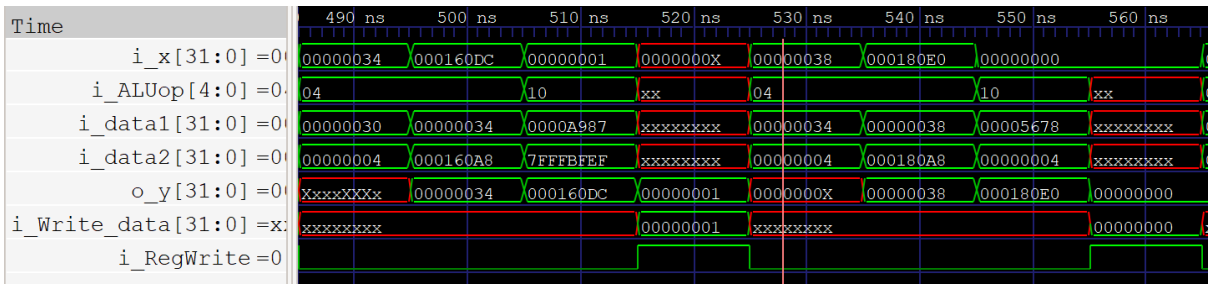
1. $s - $t연산을 수행해 aluout에 값을 저장한다.
2. aluout값을 register $d에 저장한다.

SLT slt $d = ($s < $t)

```
000000_00110_00101_01011_00000_101010 // $6 < $5? => $11(1)
000000 00011 00010 01100 00000 101010 // $3 < $2? => $12(0)
```

수행 결과

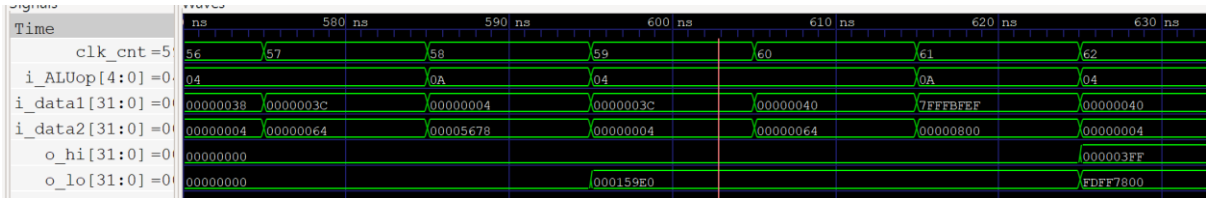
```
0000000b : 00000000_00000000_00000000_00000001 : 00000001
0000000c : 00000000_00000000_00000000_00000000 : 00000000
```



1. $S와 $T값을 비교해 $S가 작은 경우1 $S가 더 큰 경우 0을 ALUOUT에 저장한다.
2. aluout값을 regiser $d에 저장

MULTU multu hi:lo = $s * $t

```
00000000_10100100_00000000_00011001// $5 * $4 (hi: 0000 03FF lo: FDFF7800)
```



두 레지스터의 값을 곱한 결과가 각각 RESULT HI와LO에 저장된다. 해당 연산의 결과는 테스트벤치를 통해 확인 가능함.

MFLO mflo $d = lo

```
00000000_10100100_00000000_00011001// $5 * $4 (hi: 0000 03FF lo: FDFF7800)
```

결과값

```
0000000d : 11111101_11111111_01111000_00000000 : fdff7800
```

이전 테스트벤치에서 나온 결과의 LO값이 저장된다.

sb

```
10100000_10000101_00000000_00000000 // FF -> MEM$(4
10100000_10000101_00000000_00000001// FF -> MEM$(4+1
```

연산 결과

```
00000800 : xxxxxxxx_xxxxxxxx_11101111_11101111 : xxxxefef
```

테스트 벤치

aluout으로 받은 주소에 데이터 저장

lhu

명령어

```
10010100_10001110_00000000_00000000//MEM$4(4) ->$14(efef)
```

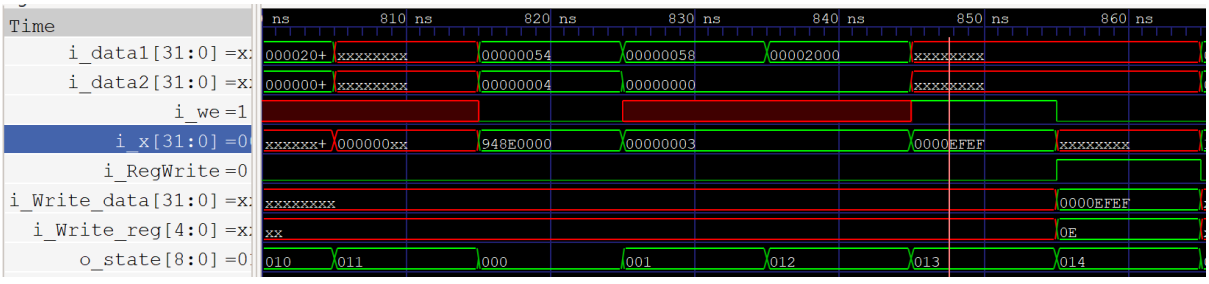
메모리 데이터

```
00000800 : xxxxxxxx_xxxxxxxx_11101111_11101111 : xxxxefef
00000801 : xxxxxxxx_xxxxxxxx_xxxxxxxx_xxxxxxxx : xxxxxxxx
```

결과값

```
0000000e : 00000000_00000000_11101111_11101111 : 0000efef
```

테스트 벤치



1. 데이터를 가져올 주소 연산
2. 데이터를 가져와 mdr에 저장.
3. mdr에서 해당 레지스터로 값 전달

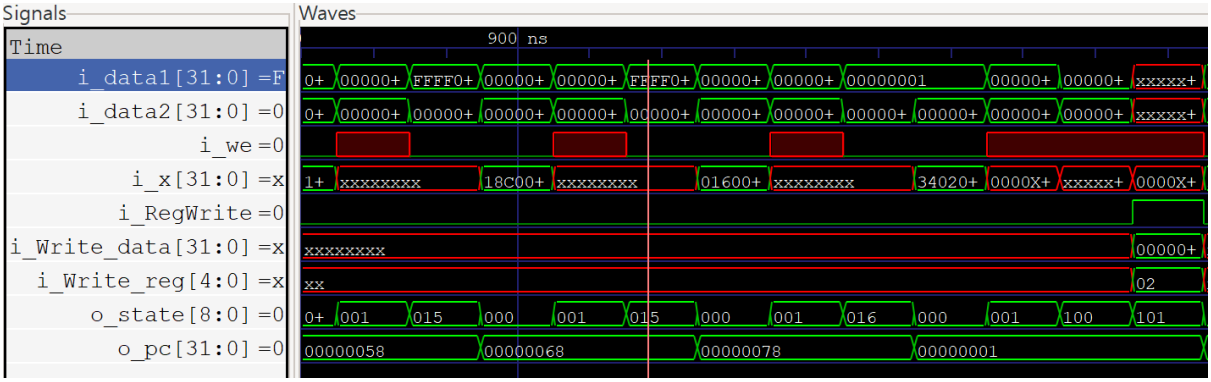
BLEZ/ JR

명령어

```
00011000_11000000_00000000_00000011// if($6<0)pc = pc+4+3<<2
XXXXXXXX_XXXXXXXX_XXXXXXXX_XXXXXXXX
XXXXXXXX_XXXXXXXX_XXXXXXXX_XXXXXXXX
XXXXXXXX_XXXXXXXX_XXXXXXXX_XXXXXXXX

00011000_11000000_00000000_00000011// if($6<0)pc = pc+4+3<<2
XXXXXXXX_XXXXXXXX_XXXXXXXX_XXXXXXXX
XXXXXXXX_XXXXXXXX_XXXXXXXX_XXXXXXXX
XXXXXXXX_XXXXXXXX_XXXXXXXX_XXXXXXXX

00000001_01100000_00000000_00001000// pc = $11
```



alu결과에 따라 분기 수행, $s주소(1)로 분기 실행 후 프로그램 처음부터 다시 실행

4. 고찰

멀티사이클 CPU의 control signal을 작성하고 실행 결과를 확인하는 프로젝트를 수행하였다. 이전 프로젝트에서 했던 signal과 비슷한 코드를 작성하면 될 줄 알았으나 멀티 사이클의 경우 사이클마다 특정하게 수행되는 단계가 존재하고 해당 단계에서 그 이상의 일을 할 경우 정해진 clock을 넘어가는 수행시간이 걸리기 때문에 프로그램이 동작하지 않는다는 것을 알게 되었고 이론 수업 때 배운 멀티사이클 동작 방식을 이해하는 데 많은 도움이 되었다.