**컴퓨터 구조 Term Project**

**이**

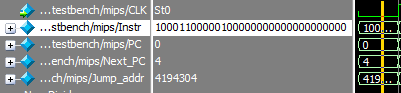
**(1) 코드분석**

**0. 0ns ~ 50ns**

<1> Instruction

mem[0] = 32'b100011\_00000\_10000\_00000\_00000\_000000의 명령어를 가지며 lw s0 0($zero) , s0 = 1을 의미한다.

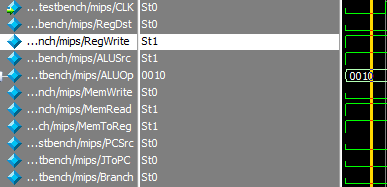
초기 PC = 0이므로 Next PC = 4일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| lw | 0 | 1 | 1 | 0010 | 0 | 1 | 1 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@0($zero)에 저장된 값 Read data1 = 0 일 것이다.

@Write register = 16(s0)이므로 16을 가질 것이다.

@Branch\_or\_offset의 경우, 16’h0000이 32’h00000000=0으로 sign extend 될 것이다.

@Lw\_Sw\_offset= Branch\_or\_offset/4 이므로 0/4 = 0일 것이다.

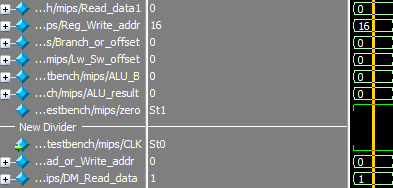
@lw 명령어에서는 ALU\_B가 Lw\_Sw\_offset와 같으므로 0 일 것이다.

@lw 명령어에서는 ALU\_result = Read\_data1 + ALU\_B이므로 0일 것이다.

@ALU\_result = 0이므로 zero는 1이 될 것이다(branch 명령어에서 쓰인다).

@Read\_or\_Write\_addr = ALU\_result =0 일 것이다.

@ DM\_Read\_data = Reg\_Write\_data = 1 일 것이다.(s0 = 1)



* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[0]($zero)의 값은 0일 것이고, 다음 clock의 positive edge에서 mem[16](s0)에   
Reg\_Write\_data = DM\_Read\_data = 1이 저장될 것이다.



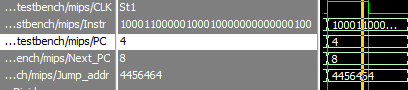
->전부 예상과 동일하다

**1. 50ns ~ 150ns**

<1> Instruction

mem[1] = 32'b100011\_00000\_10001\_00000\_00000\_000100의 명령어를 가지며 lw s1 4($zero) , s1 = 0을 의미한다.

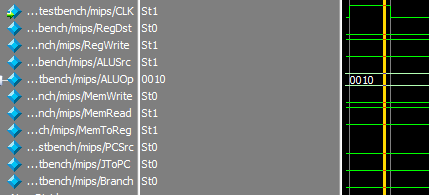
현재 PC = 4이므로 Next PC = 8일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| lw | 0 | 1 | 1 | 0010 | 0 | 1 | 1 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@0($zero)에 저장된 값 Read data1 = 0 일 것이다.

@Write register = 17(s1)이므로 17을 가질 것이다.

@Branch\_or\_offset의 경우, 16’h0004이 32’h00000004=4으로 sign extend 될 것이다.

@Lw\_Sw\_offset= Branch\_or\_offset/4 이므로 4/4 = 1일 것이다.

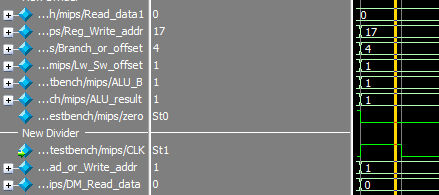
@lw 명령어에서는 ALU\_B가 Lw\_Sw\_offset와 같으므로 1 일 것이다.

@lw 명령어에서는 ALU\_result = Read\_data1 + ALU\_B이므로 1일 것이다.

@ALU\_result =0이 아니므로 zero는 0이 될 것이다. (branch 명령어에서 쓰인다).

@Read\_or\_Write\_addr = ALU\_result =1 일 것이다.(s1 = 1)

@ DM\_Read\_data = Reg\_Write\_data =0 일 것이다.



->전부 예상과 동일하다

<4> Memory 값 확인

Mem[0]($zero)의 값은 0일 것이고, 다음 clock의 positive edge에서 mem[17](s1)에   
Reg\_Write\_data = DM\_Read\_data = 0이 저장될 것이다.



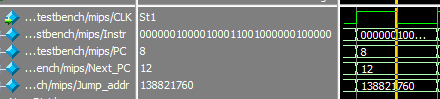
->전부 예상과 동일하다

**2. 150ns ~ 250ns**

<1> Instruction

mem[2] = 32'b000000\_10000\_10001\_10010\_00000\_100000의 명령어를 가지며 add s2 s0 s1 , s2 = 1을 의미한다.

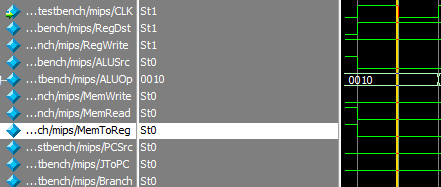
현재 PC = 8이므로 Next PC = 12일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| Add | 1 | 1 | 0 | 0010 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 16(s0)이다.

@s0에 저장된 값 Read data1 = 1 일 것이다.

@Read Register2 = 17(s1)이다.

@s1에 저장된 값 Read data2 = 0 일 것이다.

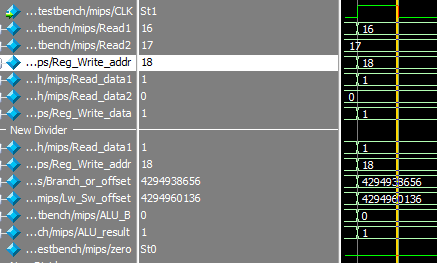
@Write register = 18(s2)이므로 18을 가질 것이다.

@add 명령어에서는 ALU\_B가 Read\_data2와 같으므로 0 일 것이다.

@add 명령어에서는 ALU\_result = Read\_data1 + ALU\_B이므로 1일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =1 일 것이다.



* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[16]($s0)의 값은 1일 것이고, Mem[17]($s1)의 값은 0일 것이며, 다음 clock의 positive edge에서 mem[18](s2)에 Read\_data1 + Read\_data2 = 1이 저장될 것이다.



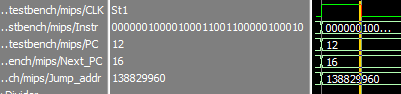
->전부 예상과 동일하다

**3. 250ns ~ 350ns**

<1> Instruction

mem[3] = 32'b000000\_10000\_10001\_10011\_00000\_100010의 명령어를 가지며, sub s3 s0 s1 , s3 = 1을 의미한다.

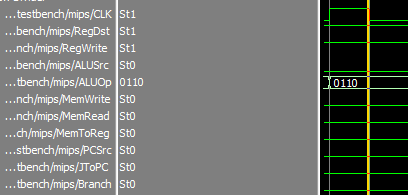
현재 PC = 12이므로 Next PC = 16일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| sub | 1 | 1 | 0 | 0110 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 16(s0)이다.

@s0에 저장된 값 Read data1 = 1 일 것이다.

@Read Register2 = 17(s1)이다.

@s1에 저장된 값 Read data2 = 0 일 것이다.

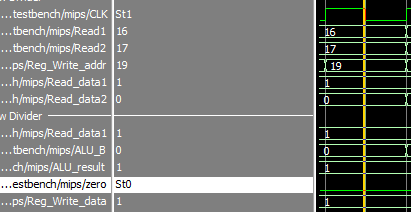
@Write register = 19(s3)이므로 19을 가질 것이다.

@sub 명령어에서는 ALU\_B가 Read\_data2와 같으므로 0 일 것이다.

@sub 명령어에서는 ALU\_result = Read\_data1 - ALU\_B이므로 1일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =1 일 것이다.



* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[16]($s0)의 값은 1일 것이고, Mem[17]($s1)의 값은 0일 것이며, 다음 clock의 positive edge에서 mem[19](s3)에 Read\_data1 - Read\_data2 = 1이 저장될 것이다.



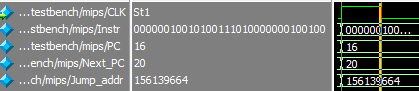
->전부 예상과 동일하다

**4. 350ns ~ 450ns**

<1> Instruction

mem[4] = 32'b000000\_10010\_10011\_10100\_00000\_100100의 명령어를 가지며, and s4 s2 s3 , s4 = 1을 의미한다.

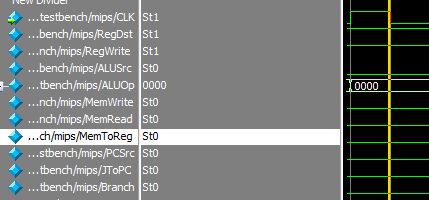
현재 PC = 16이므로 Next PC = 20일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| And | 1 | 1 | 0 | 0000 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 18(s2)이다.

@s2에 저장된 값 Read data1 = 1 일 것이다.

@Read Register2 = 19(s3)이다.

@s3에 저장된 값 Read data2 = 1 일 것이다.

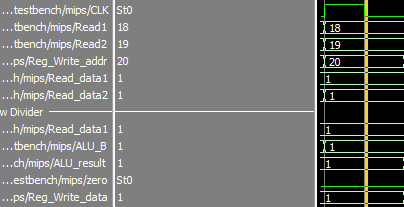
@Write register = 20(s4)이므로 20을 가질 것이다.

@and 명령어에서는 ALU\_B가 Read\_data2와 같으므로 1 일 것이다.

@and 명령어에서는 ALU\_result = Read\_data1 & ALU\_B이므로 1일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =1 일 것이다.



* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[18]($s2)의 값은 1일 것이고, Mem[19]($s3)의 값은 1일 것이며, 다음 clock의 positive edge에서 mem[20](s4)에 Read\_data1 & Read\_data2 = 1이 저장될 것이다.



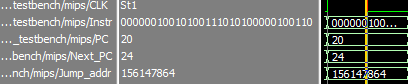
->전부 예상과 동일하다

**5. 450ns ~ 550ns**

<1> Instruction

mem[5] = 32'b000000\_10010\_10011\_10101\_00000\_100110의 명령어를 가지며, xor s5 s2 s3 , s5 = 0을 의미한다.

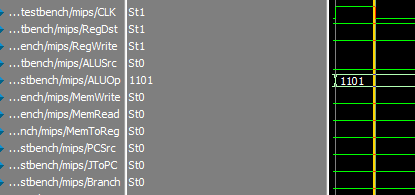
현재 PC = 20이므로 Next PC = 24일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| Xor | 1 | 1 | 0 | 1101 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 18(s2)이다.

@s2에 저장된 값 Read data1 = 1 일 것이다.

@Read Register2 = 19(s3)이다.

@s3에 저장된 값 Read data2 = 1 일 것이다.

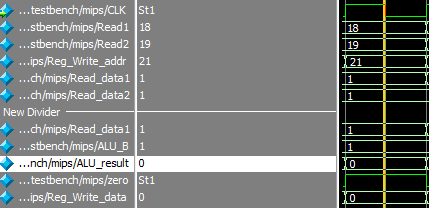
@Write register = 21(s5)이므로 21을 가질 것이다.

@xor 명령어에서는 ALU\_B가 Read\_data2와 같으므로 1 일 것이다.

@xor 명령어에서는 ALU\_result = Read\_data1 ^ ALU\_B이므로 0일 것이다.

@ALU\_result = 0이므로 zero는 1이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =0 일 것이다.



* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[18]($s2)의 값은 1일 것이고, Mem[19]($s3)의 값은 1일 것이며, 다음 clock의 positive edge에서 mem[21](s5)에 Read\_data1 ^ Read\_data2 = 0이 저장될 것이다.



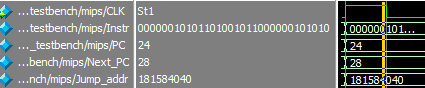
->전부 예상과 동일하다

**6. 550ns ~ 650ns**

<1> Instruction

mem[6] = 32'b000000\_10101\_10100\_10110\_00000\_101010의 명령어를 가지며, slt s6 s5 s4 , s6 = 1을 의미한다.

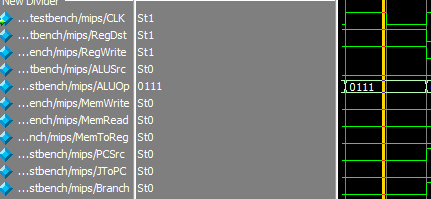
현재 PC = 24이므로 Next PC = 28일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| Slt | 1 | 1 | 0 | 0111 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 21(s5)이다.

@s5에 저장된 값 Read data1 = 0 일 것이다.

@Read Register2 = 20(s4)이다.

@s4에 저장된 값 Read data2 = 1 일 것이다.

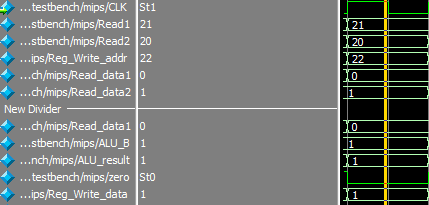
@Write register = 22(s6)이므로 22을 가질 것이다.

@slt 명령어에서는 ALU\_B가 Read\_data2와 같으므로 1 일 것이다.

@slt 명령어에서는 Read\_data1이 ALU\_B보다 작으므로 ALU\_result =1일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =1 일 것이다.



* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[21]($s5)의 값은 0일 것이고, Mem[20]($s4)의 값은 1일 것이며, 다음 clock의 positive edge에서 mem[22](s6)에는 Read\_data1이 Read\_data2보다 작으므로 1이 저장될 것이다.



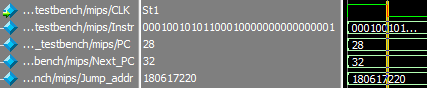
->전부 예상과 동일하다

**7. 650ns ~ 750ns**

<1> Instruction

mem[7] = 32'b000100\_10101\_10001\_00000\_00000\_000001의 명령어를 가지며, beq s5 s1 1 , s5 = s1 = 0을 의미한다.

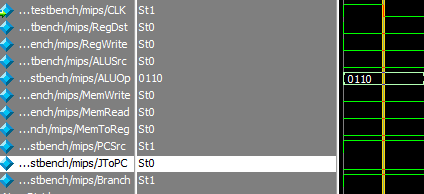
현재 PC = 28이므로 Next PC = 32일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| Beq | X | 0 | 0 | 0110 | 0 | 0 | X | 1 | 0 | 1 |

위와 같은 control signal일 것이다.



Branch 명령어 이므로 PCSrc와 Branch 값이 1로 set 된 것을 확인할 수 있다.( PCSrc가 1인 것으로 보아 ALU의 zero = 1일 것이다.)

<3> 관여하는 값들

@Read Register1 = 21(s5)이다.

@s5에 저장된 값 Read data1 = 0 일 것이다.

@Read Register2 = 17(s1)이다.

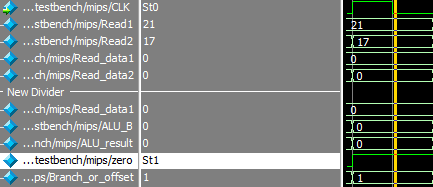
@s1에 저장된 값 Read data2 = 0 일 것이다.

@beq 명령어에서 ALU\_B가 Read\_data2와 같으므로 0 일 것이다.

@beq 명령어에서 ALUOp가 sub명령어와 같으므로, ALU\_result =Read\_data1-ALU\_B = 0일 것이다.

@ALU\_result = 0이므로 zero는 1이 될 것이다. (branch 명령어 이므로 branch 실행 여부를 판별하는 중요한 값이다. Zero가 1이면 PCSrc가 1이 된다.)

@Branch\_or\_offset의 경우, 16’h0001이 32’h00000001=1으로 sign extend 될 것이다.



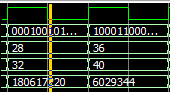
* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[21]($s5)의 값은 0, Mem[17]($s1)의 값은 0이고 다른 register와 memory 값도 변화는 없을 것이다.

Zero =1이므로 Branch 명령어가 실행되어, 다음 clock의 positive edge에서의 PC= 32가아닌 36일 것이다.





->전부 예상과 동일하다

**8. 750ns ~ 850ns**

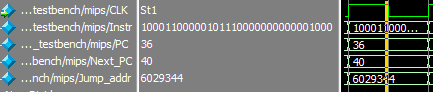
<1> Instruction

원래대로면 mem[8] = 32'b000010\_00000\_00000\_00000\_00000\_001101의 명령어를 가져야 한다.

하지만, 이전 cycle에서의 branch 동작으로 인해 PC = 32가 아닌 36이므로

mem[9] = 32'b100011\_00000\_10111\_00000\_00000\_001000의 명령어를 수행하게 되고, 이것은 lw s7 8($zero)을 의미한다.

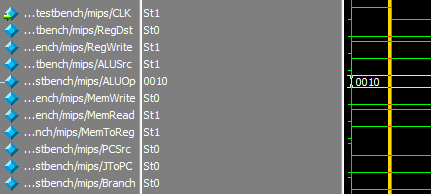
현재 PC = 36이므로 Next\_PC = 40일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| lw | 0 | 1 | 1 | 0010 | 0 | 1 | 1 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@0($zero)에 저장된 값 Read data1 = 0 일 것이다.

@Write register = 23(s7)이므로 23을 가질 것이다.

@Branch\_or\_offset의 경우, 16’h0008이 32’h00000008=8로 sign extend 될 것이다.

@Lw\_Sw\_offset= Branch\_or\_offset/4 이므로 8/4 = 2일 것이다.

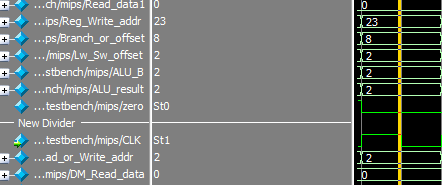
@lw 명령어에서는 ALU\_B가 Lw\_Sw\_offset와 같으므로 2 일 것이다.

@lw 명령어에서는 ALU\_result = Read\_data1 + ALU\_B이므로 2일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다(branch 명령어에서 쓰인다).

@Read\_or\_Write\_addr = ALU\_result =2 일 것이다.

@DM\_Read\_data = Reg\_Write\_data = 0 일 것이다.(s7=0)



* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[0]($zero)의 값은 0일 것이고, 다음 clock의 positive edge에서 mem[23](s7)에   
Reg\_Write\_data = DM\_Read\_data = 0이 저장될 것이다.



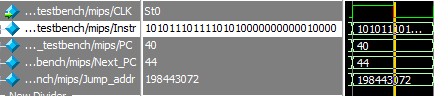
->전부 예상과 동일하다

**9. 850ns ~ 950ns**

<1> Instruction

mem[10]= 32'b101011\_10111\_10101\_00000\_00000\_010000의 명령어를 수행하게 되고, 이것은 sw s5 16(s7) 을 의미한다.

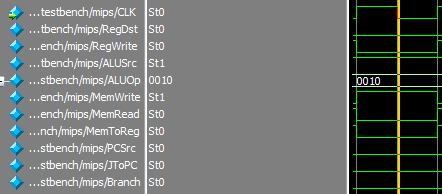
현재 PC = 40이므로 Next\_PC = 44일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| sw | X | 0 | 1 | 0010 | 1 | 0 | X | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@21($s5)에 저장된 값 Read data2 = 0 일 것이다.

@Write register = 21(s5)이므로 21을 가질 것이다.

@Branch\_or\_offset의 경우, 16’h0010이 32’h00000010=16으로 sign extend 될 것이다.

@Lw\_Sw\_offset= Branch\_or\_offset/4 이므로 16/4 = 4일 것이다.

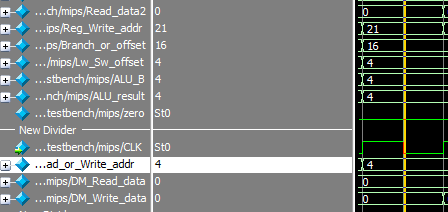
@sw 명령어에서는 ALU\_B가 Lw\_Sw\_offset와 같으므로 4 일 것이다.

@sw 명령어에서는 ALU\_result = Read\_data1 + ALU\_B이므로 4일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다(branch 명령어에서 쓰인다).

@Read\_or\_Write\_addr = ALU\_result =4일 것이다.

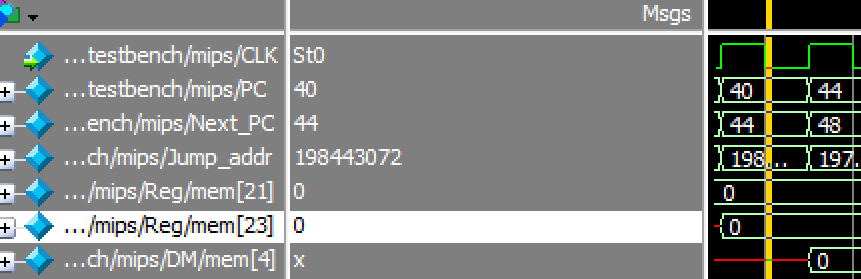
@DM\_Read\_data = DM\_Write\_data = Read\_data2 = 0 일 것이다.(mem[4]=0)



* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[21](s5), Mem[23](s7)의 값은 0일 것이고, 다음 clock의 positive edge에서 Mem[4] 에   
Reg\_Write\_data = DM\_Read\_data = 0이 저장될 것이다.



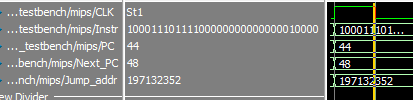
->전부 예상과 동일하다

**10. 950ns ~ 1050ns**

<1> Instruction

mem[11]= 32'b100011\_10111\_10000\_00000\_00000\_010000의 명령어를 수행하게 되고, lw s0 16(s7)을 의미한다.

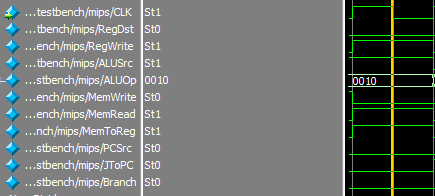
현재 PC = 44이므로 Next\_PC = 48일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| lw | 0 | 1 | 1 | 0010 | 0 | 1 | 1 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@23($s7)에 저장된 값 Read data1 = 0 일 것이다.

@Write register = 16(s0)이므로 16을 가질 것이다.

@Branch\_or\_offset의 경우, 16’h0010이 32’h00000010=16으로 sign extend 될 것이다.

@Lw\_Sw\_offset= Branch\_or\_offset/4 이므로 16/4 = 4일 것이다.

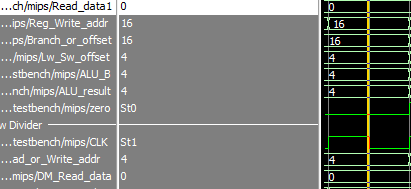
@lw 명령어에서는 ALU\_B가 Lw\_Sw\_offset와 같으므로 4 일 것이다.

@lw 명령어에서는 ALU\_result = Read\_data1 + ALU\_B이므로 4일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다(branch 명령어에서 쓰인다).

@Read\_or\_Write\_addr = ALU\_result =4 일 것이다.

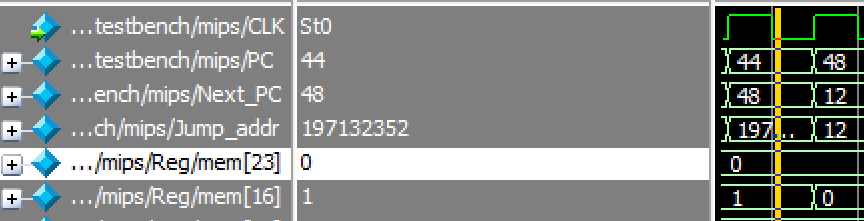
@DM\_Read\_data = Reg\_Write\_data = 0 일 것이다.(s0=mem[4]=0)



* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[23]($s7)의 값은 0일 것이고, 다음 clock의 positive edge에서 mem[16](s0)에   
Reg\_Write\_data = DM\_Read\_data = 0이 저장될 것이다.



->전부 예상과 동일하다

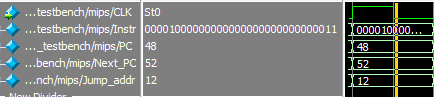
**11. 1050ns ~ 1150ns**

<1> Instruction

mem[12]= 32'b000010\_00000\_00000\_00000\_00000\_000011의 명령어를 수행하게 되고, j 3을 의미한다.

현재 PC = 48이므로 Next\_PC = 52일 것이다.

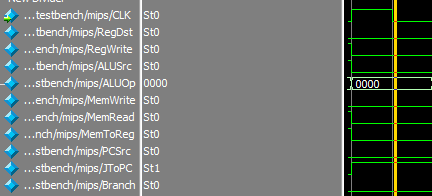
하지만. JToPC가 1이므로 실제 점프 위치는 Jump\_addr = 12 일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| J | X | 0 | X | X | 0 | 0 | X | X | 1 | 0 |

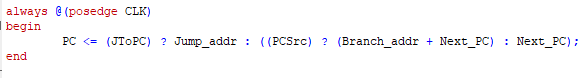
위와 같은 control signal일 것이다.



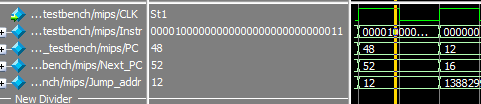
<3> 관여하는 값들

MIPS 코드에서



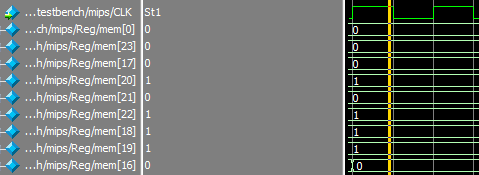


로 인해 Jump\_addr = 12가 되고(j 3이므로), 이것이 다음 PC값이 된다.



<4> Memory 값 확인

J 명령어를 수행하면 Register와 memory의 저장된 값에는 변화가 없을 것이다,.



->전부 예상과 동일하다

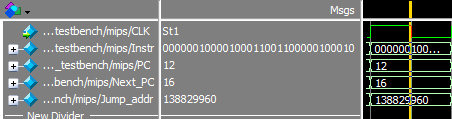
**12. 1150ns ~ 1250ns**

<1> Instruction

이전 j명령어로 인해 현재 PC = 12이므로

mem[3] = 32'b000000\_10000\_10001\_10011\_00000\_100010의 명령어를 가지며, sub s3 s0 s1 을 의미한다.

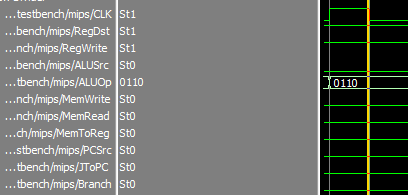
Next PC = 16일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| sub | 1 | 1 | 0 | 0110 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 16(s0)이다.

@s0에 저장된 값 Read data1 = 0일 것이다.

@Read Register2 = 17(s1)이다.

@s1에 저장된 값 Read data2 = 0 일 것이다.

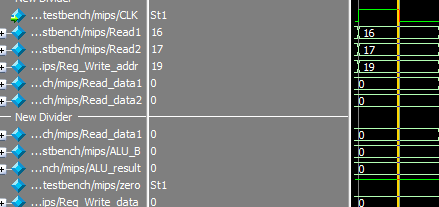
@Write register = 19(s3)이므로 19을 가질 것이다.

@sub 명령어에서는 ALU\_B가 Read\_data2와 같으므로 0 일 것이다.

@sub 명령어에서는 ALU\_result = Read\_data1 - ALU\_B이므로 0 것이다.

@ALU\_result = 0이므로 zero는 1 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =0일 것이다.



* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[16]($s0)의 값은 0 것이고, Mem[17]($s1)의 값은 0일 것이며, 다음 clock의 positive edge에서 mem[19](s3)에 Read\_data1 - Read\_data2 = 0 저장될 것이다.



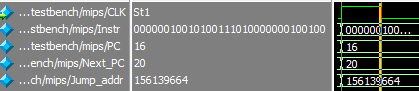
->전부 예상과 동일하다

**13.1250ns ~ 1350ns**

<1> Instruction

mem[4] = 32'b000000\_10010\_10011\_10100\_00000\_100100의 명령어를 가지며, and s4 s2 s3 을 의미한다.

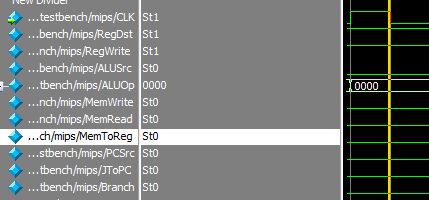
현재 PC = 16이므로 Next PC = 20일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| And | 1 | 1 | 0 | 0000 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 18(s2)이다.

@s2에 저장된 값 Read data1 = 1 일 것이다.

@Read Register2 = 19(s3)이다.

@s3에 저장된 값 Read data2 = 0 일 것이다.

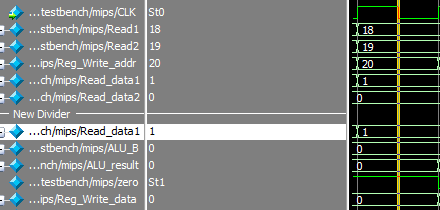
@Write register = 20(s4)이므로 20을 가질 것이다.

@and 명령어에서는 ALU\_B가 Read\_data2와 같으므로 0 일 것이다.

@and 명령어에서는 ALU\_result = Read\_data1 & ALU\_B이므로 0일 것이다.

@ALU\_result = 0이므로 zero는 1이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =0 일 것이다.



* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[18]($s2)의 값은 1일 것이고, Mem[19]($s3)의 값은 0일 것이며, 다음 clock의 positive edge에서 mem[20](s4)에 Read\_data1 & Read\_data2 = 0이 저장될 것이다.



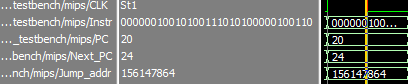
->전부 예상과 동일하다

**14. 1350ns ~ 1450ns**

<1> Instruction

mem[5] = 32'b000000\_10010\_10011\_10101\_00000\_100110의 명령어를 가지며, xor s5 s2 s3을 의미한다.

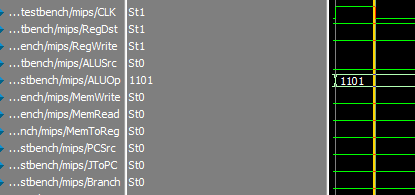
현재 PC = 20이므로 Next PC = 24일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| Xor | 1 | 1 | 0 | 1101 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 18(s2)이다.

@s2에 저장된 값 Read data1 = 1 일 것이다.

@Read Register2 = 19(s3)이다.

@s3에 저장된 값 Read data2 = 0 일 것이다.

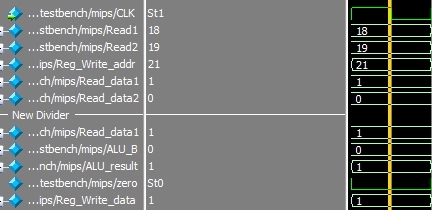
@Write register = 21(s5)이므로 21을 가질 것이다.

@xor 명령어에서는 ALU\_B가 Read\_data2와 같으므로 0 일 것이다.

@xor 명령어에서는 ALU\_result = Read\_data1 ^ ALU\_B이므로 1일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =1일 것이다.



* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[18]($s2)의 값은 1일 것이고, Mem[19]($s3)의 값은 0일 것이며, 다음 clock의 positive edge에서 mem[21](s5)에 Read\_data1 ^ Read\_data2 = 1이 저장될 것이다.



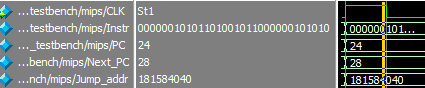
->전부 예상과 동일하다

**15. 1450ns ~ 1550ns**

<1> Instruction

mem[6] = 32'b000000\_10101\_10100\_10110\_00000\_101010의 명령어를 가지며, slt s6 s5 s4 을 의미한다.

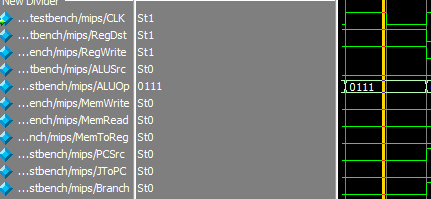
현재 PC = 24이므로 Next PC = 28일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| Slt | 1 | 1 | 0 | 0111 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 21(s5)이다.

@s5에 저장된 값 Read data1 = 1 일 것이다.

@Read Register2 = 20(s4)이다.

@s4에 저장된 값 Read data2 = 0 일 것이다.

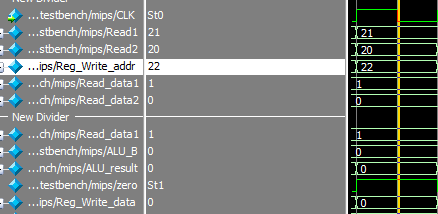
@Write register = 22(s6)이므로 22을 가질 것이다.

@slt 명령어에서는 ALU\_B가 Read\_data2와 같으므로 0 일 것이다.

@slt 명령어에서는 Read\_data1이 ALU\_B보다 작지 않으므로 ALU\_result =0일 것이다.

@ALU\_result = 0이므로 zero는 1이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =0 일 것이다.



* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[21]($s5)의 값은 1일 것이고, Mem[20]($s4)의 값은 0일 것이며, 다음 clock의 positive edge에서 mem[22](s6)에는 Read\_data1이 Read\_data2보다 작지 않으므로 0이 저장될 것이다.



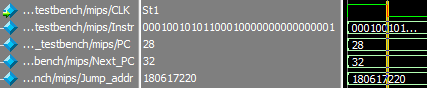
->전부 예상과 동일하다

**16. 650ns ~ 750ns**

<1> Instruction

mem[7] = 32'b000100\_10101\_10001\_00000\_00000\_000001의 명령어를 가지며, beq s5 s1 1을 의미한다.

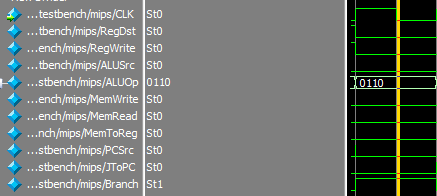
현재 PC = 28이므로 Next PC = 32일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| Beq | X | 0 | 0 | 0110 | 0 | 0 | X | 1 | 0 | 1 |

위와 같은 control signal일 것이다.



Branch 명령어 이므로 Branch 값이 1로 set 되었지만 PCSrc =0인 것을 확인할 수 있다.( PCSrc가 0인 것으로 보아 ALU의 zero = 0일 것이다.)

따라서 위의 표가 아닌

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| Beq | X | 0 | 0 | 0110 | 0 | 0 | X | 0 | 0 | 1 |

가 된다.

<3> 관여하는 값들

@Read Register1 = 21(s5)이다.

@s5에 저장된 값 Read data1 = 1 일 것이다.

@Read Register2 = 17(s1)이다.

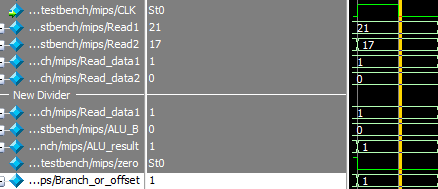
@s1에 저장된 값 Read data2 = 0 일 것이다.

@beq 명령어에서 ALU\_B가 Read\_data2와 같으므로 0 일 것이다.

@beq 명령어에서 ALUOp가 sub명령어와 같으므로, ALU\_result =Read\_data1-ALU\_B = 1일 것이다.

@ALU\_result = 1이므로 zero는 0이 될 것이다. (branch 명령어 이므로 branch 실행 여부를 판별하는 중요한 값이다. Zero가 1이면 PCSrc가 1이 된다.)

@Branch\_or\_offset의 경우, 16’h0001이 32’h00000001=1으로 sign extend 될 것이다.



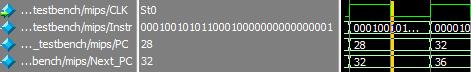
* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[21]($s5)의 값은 1, Mem[17]($s1)의 값은 0이고 다른 register와 memory 값도 변화는 없을 것이다.

Zero =0이므로 다음 clock의 positive edge에서의 PC= 32가 될 것이다.





->전부 예상과 동일하다

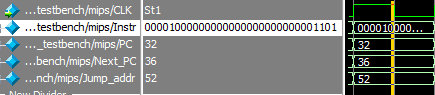
**17. 1650ns ~ 1750ns**

<1> Instruction

mem[8] = 32'b000010\_00000\_00000\_00000\_00000\_001101의 명령어를 수행하게 되고, j 13을 의미한다.

현재 PC = 32이므로 Next\_PC = 36일 것이다.

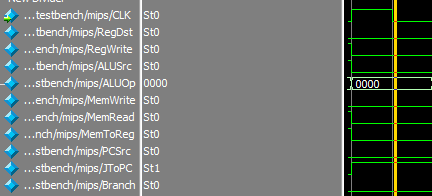
하지만. JToPC가 1이므로 실제 점프 위치는 Jump\_addr = 52 일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| J | X | 0 | X | X | 0 | 0 | X | X | 1 | 0 |

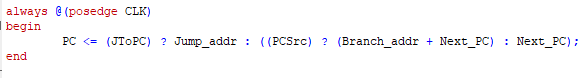
위와 같은 control signal일 것이다.



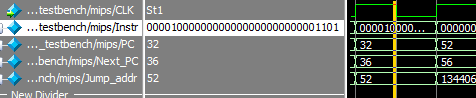
<3> 관여하는 값들

MIPS 코드에서



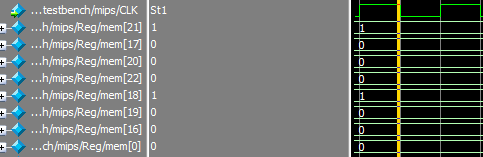


로 인해 Jump\_addr = 52가 되고(j 13이므로), 이것이 다음 PC값이 된다.



<4> Memory 값 확인

J 명령어를 수행하면 Register와 memory의 저장된 값에는 변화가 없을 것이다,.



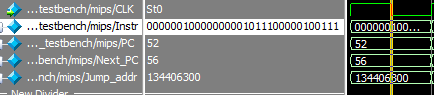
->전부 예상과 동일하다

**18. 1750ns ~ 1850ns**

<1> Instruction

mem[13]= 32'b000000\_10000\_00000\_10111\_00000\_100111의 명령어를 가지며, nor s7 s0 $zero을 의미한다.

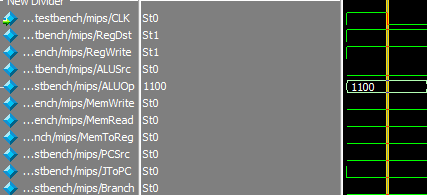
현재 PC = 52이므로 Next PC = 56일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| nor | 1 | 1 | 0 | 1100 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 16(s0)이다.

@s2에 저장된 값 Read data1 = 0 일 것이다.

@Read Register2 = 0($zero)이다.

@$zero에 저장된 값 Read data2 = 0 일 것이다.

@Write register = 23(s7)이므로 23을 가질 것이다.

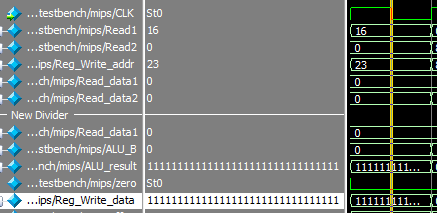
@nor 명령어에서는 ALU\_B가 Read\_data2와 같으므로 0 일 것이다.

@nor 명령어에서는 ALU\_result = ~(Read\_data1 | ALU\_B)이므로 모든 비트가 1로 바뀌어

32'b1111\_1111\_1111\_1111\_1111\_1111\_1111\_1111일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =32'b1111\_1111\_1111\_1111\_1111\_1111\_1111\_1111일 것이다.

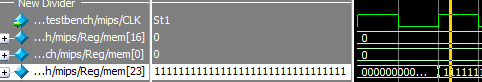




* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[16]($s0)의 값은 0일 것이고, Mem[0]($zero)의 값은 0일 것이며, 다음 clock의 positive edge에서 mem[23](s7)에 32'b1111\_1111\_1111\_1111\_1111\_1111\_1111\_1111이 저장될 것이다.



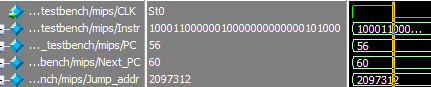
->전부 예상과 동일하다

**19. 1850ns ~ 1950ns**

<1> Instruction

mem[14]= 32'b100011\_00000\_01000\_00000\_00000\_101000의 명령어를 수행하게 되고, lw t0 40($zero)을 의미한다.

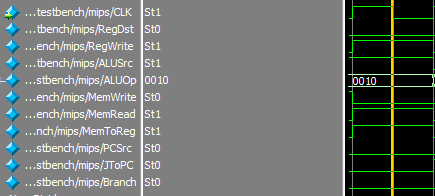
현재 PC = 56이므로 Next\_PC = 60일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| lw | 0 | 1 | 1 | 0010 | 0 | 1 | 1 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@0($zero)에 저장된 값 Read data1 = 0 일 것이다.

@Write register = 8(t0)이므로 8을 가질 것이다.

@Branch\_or\_offset의 경우, 16’h0028이 32’h00000028=40으로 sign extend 될 것이다.

@Lw\_Sw\_offset= Branch\_or\_offset/4 이므로 40/4 = 10일 것이다.

@lw 명령어에서는 ALU\_B가 Lw\_Sw\_offset와 같으므로 10 일 것이다.

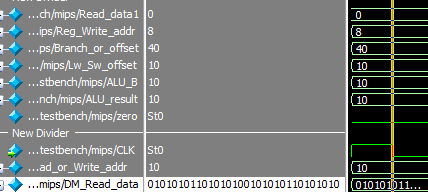
@lw 명령어에서는 ALU\_result = Read\_data1 + ALU\_B이므로 10일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다(branch 명령어에서 쓰인다).

@Read\_or\_Write\_addr = ALU\_result =10 일 것이다.

@DM\_Read\_data = Reg\_Write\_data = 32'b0101\_0101\_1010\_1010\_0101\_0101\_1010\_1010일 것이다.

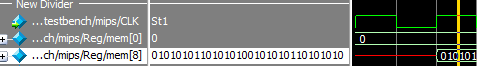
( t0 =32'b0101\_0101\_1010\_1010\_0101\_0101\_1010\_1010)

)

* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[0]($zero)의 값은 0일 것이고, 다음 clock의 positive edge에서 mem[8](t0)에   
Reg\_Write\_data = DM\_Read\_data = 32'b0101\_0101\_1010\_1010\_0101\_0101\_1010\_1010이 저장될 것이다.



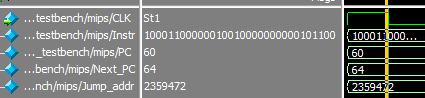
->전부 예상과 동일하다

**20. 1950ns ~ 2050ns**

<1> Instruction

mem[15]= 32'b100011\_00000\_01001\_00000\_00000\_101100의 명령어를 수행하게 되고lw t1 44($zero)을 의미한다.

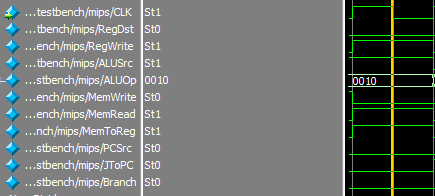
현재 PC = 60이므로 Next\_PC = 64일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| lw | 0 | 1 | 1 | 0010 | 0 | 1 | 1 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@0($zero)에 저장된 값 Read data1 = 0 일 것이다.

@Write register = 9(t1)이므로 9를 가질 것이다.

@Branch\_or\_offset의 경우, 16’h002C이 32’h0000002C=44으로 sign extend 될 것이다.

@Lw\_Sw\_offset= Branch\_or\_offset/4 이므로 44/4 = 11일 것이다.

@lw 명령어에서는 ALU\_B가 Lw\_Sw\_offset와 같으므로 11 일 것이다.

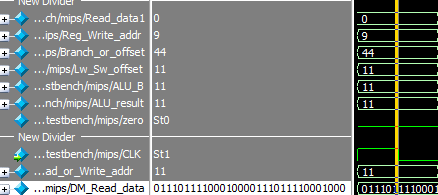
@lw 명령어에서는 ALU\_result = Read\_data1 + ALU\_B이므로 11일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다(branch 명령어에서 쓰인다).

@Read\_or\_Write\_addr = ALU\_result =11 일 것이다.

@DM\_Read\_data = Reg\_Write\_data = 32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000일 것이다.

( t1 =32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000)



)

* 전부 예상과 동일하다.

<4> Memory 값 확인

Mem[0]($zero)의 값은 0일 것이고, 다음 clock의 positive edge에서 mem[9](t1)에   
Reg\_Write\_data = DM\_Read\_data = 32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000이 저장될 것이다.



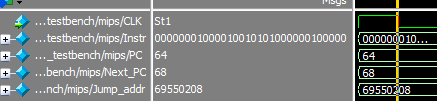
->전부 예상과 동일하다

**21. 2050ns ~ 2150ns**

<1> Instruction

mem[16]= 32'b000000\_01000\_01001\_01010\_00000\_100000의 명령어를 가지며 add t2 t0 t1을 의미한다.

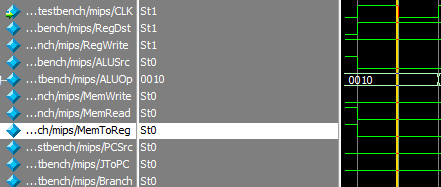
현재 PC = 64이므로 Next PC = 68일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| Add | 1 | 1 | 0 | 0010 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 8(t0)이다.

@t0에 저장된 값 Read data1 = 32'b0101\_0101\_1010\_1010\_0101\_0101\_1010\_1010 일 것이다.

@Read Register2 = 9(t1)이다.

@t1에 저장된 값 Read data2 = 32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000 일 것이다.

@Write register = 10(t2)이므로 10을 가질 것이다.

@add 명령어에서는 ALU\_B가 Read\_data2와 같으므로

32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000 일 것이다.

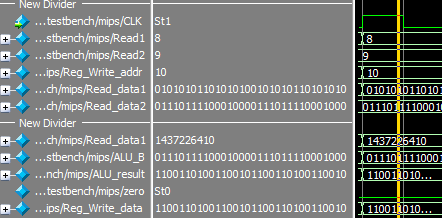
@add 명령어에서는 ALU\_result = Read\_data1 + ALU\_B이므로

32'b1100\_1101\_0011\_0010\_1100\_1101\_0011\_0010일 것이다.

(overflow가 발생하지만 양의 값이다.)

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =32'b1100\_1101\_0011\_0010\_1100\_1101\_0011\_0010 일 것이다.



* 전부 예상과 동일하다.

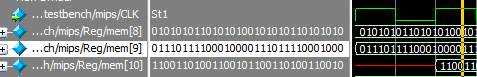
<4> Memory 값 확인

Mem[8]($t0)의 값은

32'b0101\_0101\_1010\_1010\_0101\_0101\_1010\_1010 일 것이고,

Mem[9]($t1)의 값은

32'b1100\_1101\_0011\_0010\_1100\_1101\_0011\_0010일 것이며, 다음 clock의 positive edge에서 mem[10](t2)에 Read\_data1 + Read\_data2 = 32'b1100\_1101\_0011\_0010\_1100\_1101\_0011\_0010 이 저장될 것이다.



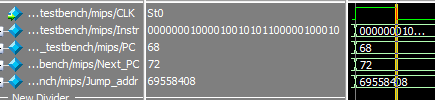
->전부 예상과 동일하다

**22. 2150ns ~ 2250ns**

<1> Instruction

mem[17]= 32'b000000\_01000\_01001\_01011\_00000\_100010의 명령어를 가지며, sub t3 t0 t1을 의미한다.

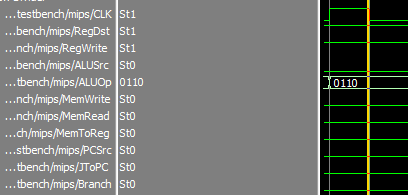
현재 PC = 68이므로 Next PC = 72일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| sub | 1 | 1 | 0 | 0110 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 8(t0)이다.

@t0에 저장된 값 Read data1 = 32'b0101\_0101\_1010\_1010\_0101\_0101\_1010\_1010 일 것이다.

@Read Register2 = 9(t1)이다.

@t1에 저장된 값 Read data2 =32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000 일 것이다.

@Write register = 11(t3)이므로 11을 가질 것이다.

@sub 명령어에서는 ALU\_B가 Read\_data2와 같으므로

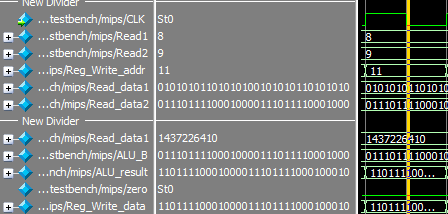
32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000 일 것이다.

@sub 명령어에서는 ALU\_result = Read\_data1 - ALU\_B이므로

32'b1100\_1101\_0011\_0010\_1100\_1101\_0011\_0010일 것이다. (음의 값이다.)

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =32'b1101\_1110\_0010\_0001\_1101\_1110\_0010\_0010일 것이다.



* 전부 예상과 동일하다.

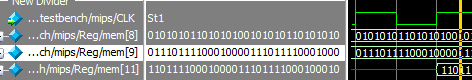
<4> Memory 값 확인

Mem[8]($t0)의 값은

32'b0101\_0101\_1010\_1010\_0101\_0101\_1010\_1010 일 것이고,

Mem[9]($t1)의 값은

32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000 일 것이며, 다음 clock의 positive edge에서 mem[11](t3)에 Read\_data1 - Read\_data2 = 32'b1101\_1110\_0010\_0001\_1101\_1110\_0010\_0010이 저장될 것이다.



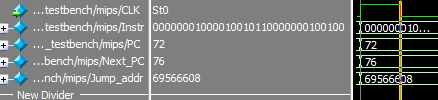
* 전부 예상과 동일하다.

**23. 2250ns ~ 2350ns**

<1> Instruction

mem[18]= 32'b000000\_01000\_01001\_01100\_00000\_100100의 명령어를 가지며, and t4 t0 t1을 의미한다.

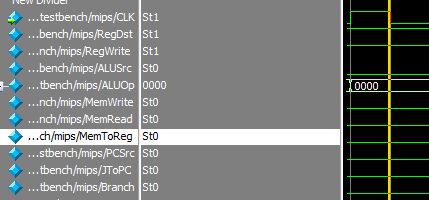
현재 PC = 72이므로 Next PC = 76일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| And | 1 | 1 | 0 | 0000 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 8(t0)이다.

@t0에 저장된 값 Read data1 = 32'b0101\_0101\_1010\_1010\_0101\_0101\_1010\_1010 일 것이다.

@Read Register2 = 9(t1)이다.

@t1에 저장된 값 Read data2 = 32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000 일 것이다.

@Write register = 12(t4)이므로 12를 가질 것이다.

@and 명령어에서는 ALU\_B가 Read\_data2와 같으므로

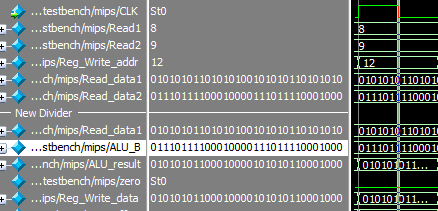
32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000 일 것이다.

@and 명령어에서는 ALU\_result = Read\_data1 & ALU\_B이므로

32'b0101\_0101\_1000\_1000\_0101\_0101\_1000\_1000일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =32'b0101\_0101\_1000\_1000\_0101\_0101\_1000\_1000 일 것이다.



* 전부 예상과 동일하다.

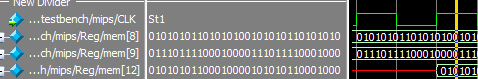
<4> Memory 값 확인

Mem[8]($t0)의 값은

32'b0101\_0101\_1010\_1010\_0101\_0101\_1010\_1010 일 것이고,

Mem[9]($t1)의 값은

32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000일 것이며, 다음 clock의 positive edge에서 mem[12](t4)에 Read\_data1 & Read\_data2 = 32'b0101\_0101\_1000\_1000\_0101\_0101\_1000\_1000 이 저장될 것이다.



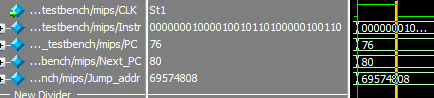
->전부 예상과 동일하다

**24. 2350ns ~ 2450ns**

<1> Instruction

mem[19]= 32'b000000\_01000\_01001\_01101\_00000\_100110의 명령어를 가지며, xor t5 t0 t1을 의미한다.

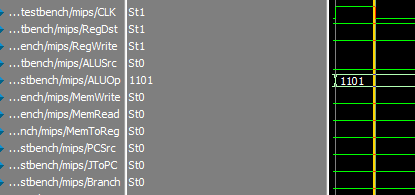
현재 PC = 76이므로 Next PC = 80일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| Xor | 1 | 1 | 0 | 1101 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 8(t0)이다.

@t0에 저장된 값 Read data1 = 32'b0101\_0101\_1010\_1010\_0101\_0101\_1010\_1010일 것이다.

@Read Register2 = 9(t1)이다.

@t1에 저장된 값 Read data2 = 32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000일 것이다.

@Write register = 13(t5)이므로 13을 가질 것이다.

@xor 명령어에서는 ALU\_B가 Read\_data2와 같으므로

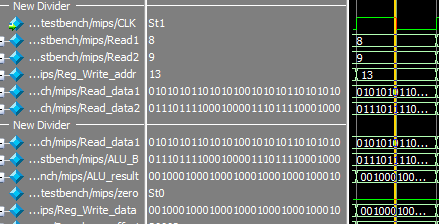
32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000 일 것이다.

@xor 명령어에서는 ALU\_result = Read\_data1 ^ ALU\_B이므로

32'b0010\_0010\_0010\_0010\_0010\_0010\_0010\_0010일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =32'b0010\_0010\_0010\_0010\_0010\_0010\_0010\_0010 일 것이다.



* 전부 예상과 동일하다.

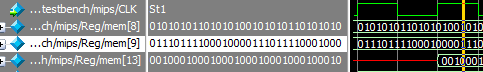
<4> Memory 값 확인

Mem[8]($t0)의 값은

32'b0101\_0101\_1010\_1010\_0101\_0101\_1010\_1010 일 것이고,

Mem[9]($t1)의 값은

32'b0111\_0111\_1000\_1000\_0111\_0111\_1000\_1000일 것이며, 다음 clock의 positive edge에서 mem[13](t5)에 Read\_data1 ^ Read\_data2 = 32'b0010\_0010\_0010\_0010\_0010\_0010\_0010\_0010이 저장될 것이다.



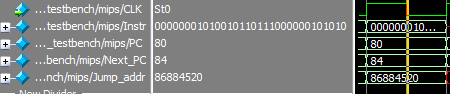
->전부 예상과 동일하다

**25. 550ns ~ 650ns**

<1> Instruction

mem[20]= 32'b000000\_01010\_01011\_01110\_00000\_101010의 명령어를 가지며, slt t6 t2 t3을 의미한다.

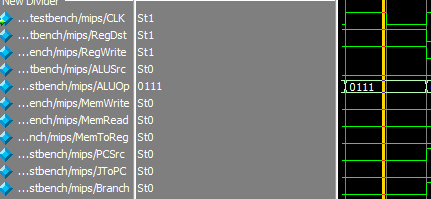
현재 PC = 80이므로 Next PC = 84일 것이다.



<2> Control Signal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | ALUOp | MemWrite | MemRead | MemToReg | PCSrc | JToPC | Branch |
| Slt | 1 | 1 | 0 | 0111 | 0 | 0 | 0 | X | 0 | 0 |

위와 같은 control signal일 것이다.



<3> 관여하는 값들

@Read Register1 = 13(t5)이다.

@t5에 저장된 값 Read data1 = 32'b0010\_0010\_0010\_0010\_0010\_0010\_0010\_0010 일 것이다.

@Read Register2 = 12(t4)이다.

@t4에 저장된 값 Read data2 = 32'b0101\_0101\_1000\_1000\_0101\_0101\_1000\_1000일 것이다.

@Write register = 14(t6)이므로 14을 가질 것이다.

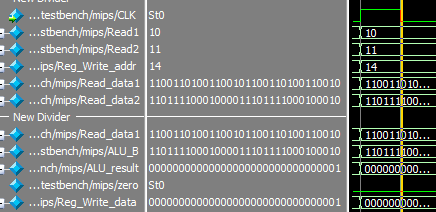
@slt 명령어에서는 ALU\_B가 Read\_data2와 같으므로

32'b0101\_0101\_1000\_1000\_0101\_0101\_1000\_1000 일 것이다.

@slt 명령어에서는 Read\_data1이 ALU\_B보다 작으므로 ALU\_result =1일 것이다.

@ALU\_result = 0이 아니므로 zero는 0이 될 것이다. (branch 명령어에서 쓰인다).

@Reg\_Write\_data = ALU\_result =1 일 것이다.



* 전부 예상과 동일하다.

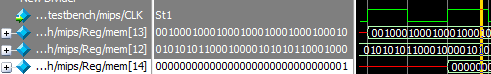
<4> Memory 값 확인

Mem[13]($t5)의 값은

32'b0010\_0010\_0010\_0010\_0010\_0010\_0010\_0010 일 것이고,

Mem[12]($t4)의 값은

32'b0101\_0101\_1000\_1000\_0101\_0101\_1000\_1000 일 것이며, 다음 clock의 positive edge에서 mem[14](t6)에는 Read\_data1이 Read\_data2보다 작으므로 1이 저장될 것이다.



->전부 예상과 동일하다

**25. 2550ns~**

2550ns를 끝으로 동작이 종료되지만,

13250ns부터 위의 과정들이 다시 반복되고, 모든 과정이 끝나면 동작이 종료된 후 10700ns후에

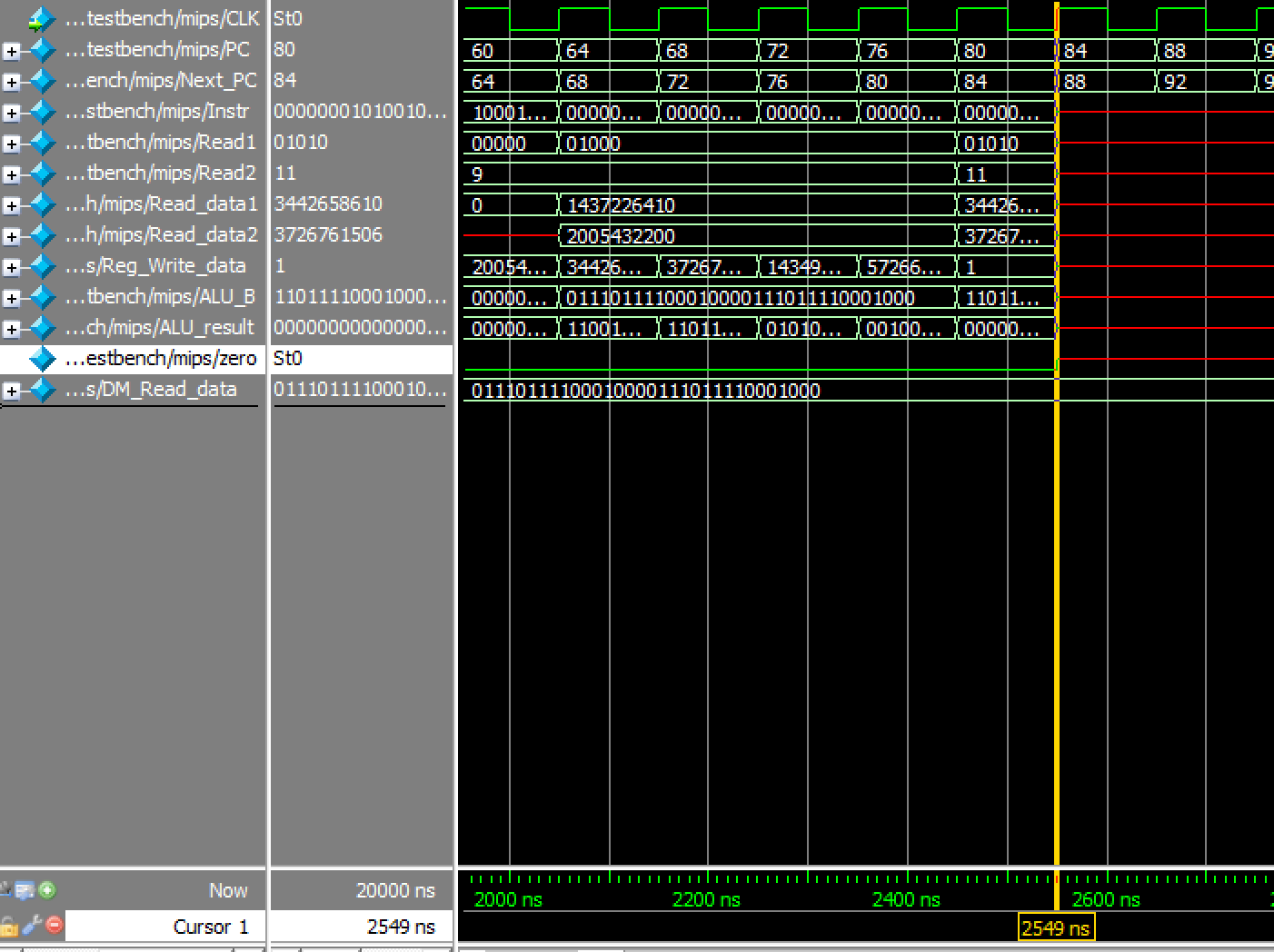
다시 반복한다. 이것은 위의 과정에서 첫 cycle에 50ns를 소모하고, 이후 한 cycle당 100ns씩

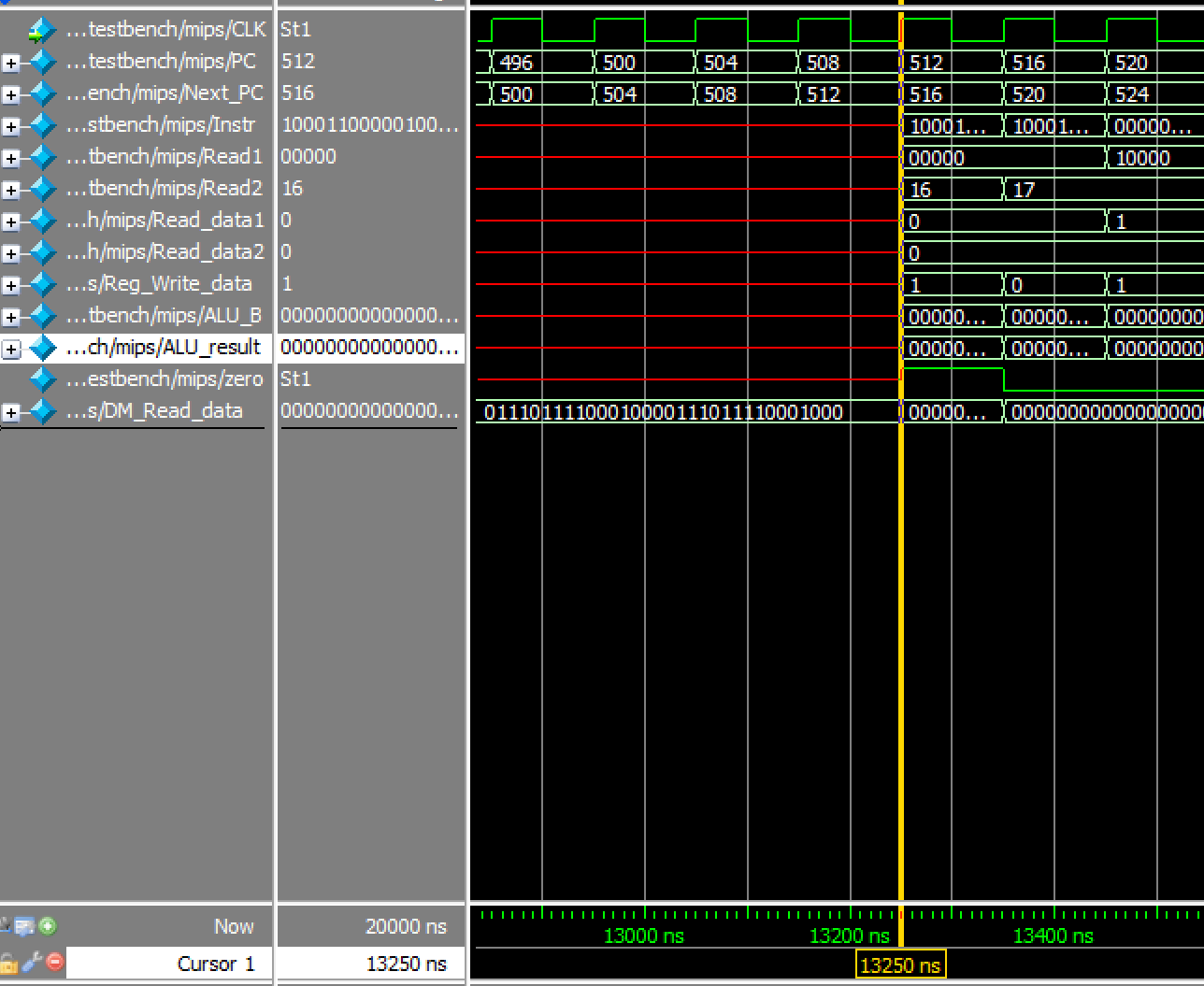
소모하였고, 명령어에 의해 추가적으로 5번의 명령어를 더 실행시켰으며 코드에서 128 entries의

메모리를 선언하였기 때문에 1\*50+(127+5)\*100 = 13250ns로 위의 과정이 다시 반복되는 시간이

도출된다.

즉, 모든 memory에 대해 시뮬레이션을 진행하면 다시 처음으로 돌아가는 구조임을 알 수 있었다.



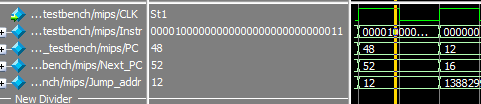


**(2) 오류 찾기**

**1. Next PC가 실제 다음 cycle의 PC가 아닌 문제**

기존 코드에서는 Next\_PC와 Jump\_addr가 달라서 다음 cycle에 PC가 이전 cycle의 Next\_PC가 아닌, Jump\_addr 값이 되어버리는 경우가 있었다.

Ex)



하지만, Mips에서 nPC 사용할 때, nPC를 다음 cycle에서 PC값이 되는 값으로 정의하여 사용하고 있다. 그렇기 때문에 이 코드에서도 Next\_PC 값을 실제로 다음 cycle에 PC가 갖게 되는 값으로 설정되도록 수정할 필요가 있을 것 이다.

**2. Read data에 write back 된 후의 값이 저장된다.**

정상적인 동작이라면 ,write back에 의해 register의 write data에 저장된 값은 다음 cycle의 positive edge에서 Read\_data2에 저장되어야 한다.

하지만, 기존 코드를 실행시키면 한 cycle 내에서 write data = Read\_data2인 현상이 발생 한다.

Write data의 값이 다음 cycle clock edge에서 Read\_data2에 입력되도록 코드를 수정해야 할 것 이다.

**3. overflow?**

2050ns에서 Mem[16] 명령어를 실행시 overflow가 발생하였다.

하지만 overflow가 발생했는지 안 했는지 따로 계산을 해보지 않으면 알 수 없는 문제가 있다.

Overflow가 발생했는지 모른다면 signed bit의 오인으로 아예 다른 수로 여겨질 수도 있다.

그래서 overflow의 발생 여부를 알려주는 wire를 선언해서 시뮬레이션에서 overflow가 발생했음

을 확인하고 발생한 시점을 알 수 있도록 코드를 수정해야 할 것 이다.

**4. branch\_offset을 4로 나누는 문제**

기존의 코드에서는 branch\_offset을 4로 나누어 lw\_sw\_offset 값을 설정하고 있다.

하지만, 이론적으로 branch\_offset이 4로 나누어져야 하는 시점은 ALU를 빠져나와 data memory에 입력될 때 이다.

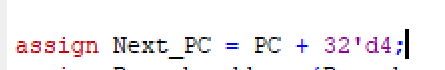
따라서 기존의 코드를 수정하여 branch\_offset이 ALU를 빠져나왔을 때 4로 나누어지지 않도록 만들어야 할 것 이다.

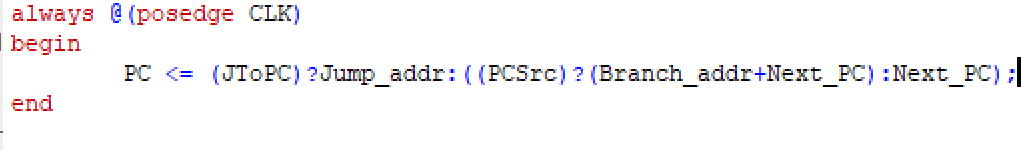
**(3) 오류 해결**

**1. 1번의 해결**

1번 오류를 해결하기 위해 Mips.v 파일에 있는 코드를 수정하였다.

기존의

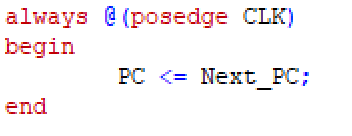




코드를

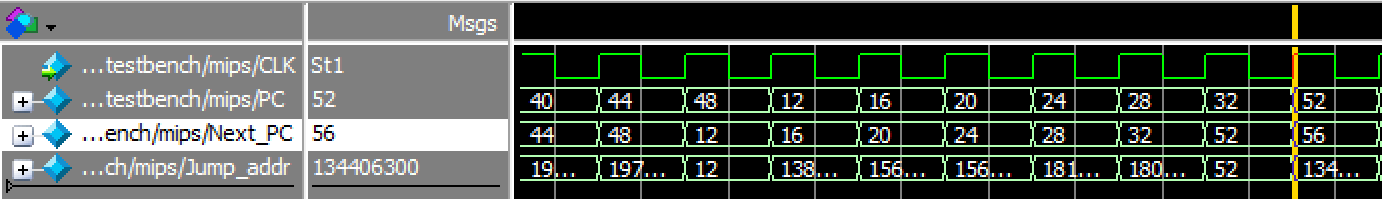
아래와 같이





로 수정하였다.

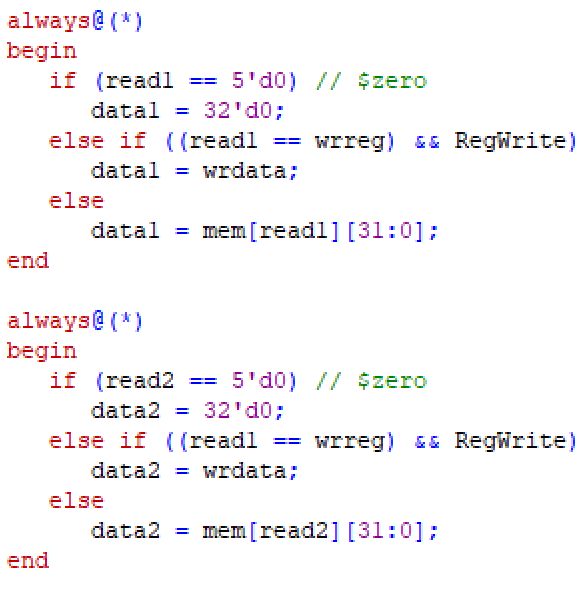
시뮬레이션 시킨 결과



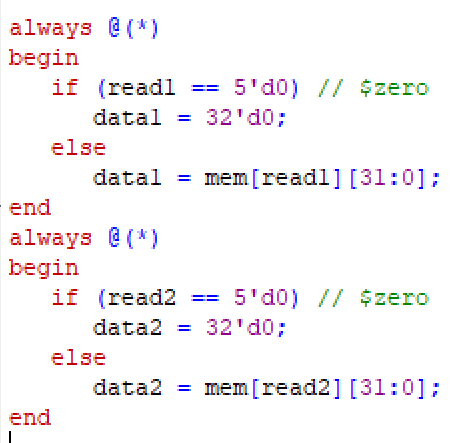
기존에는 Next\_PC와 Jump\_addr가 서로 다를 때 다음 cycle에서 Jump\_addr값을 PC로 갖던 것과 달리,

다음 cycle의 PC값이 Jump\_addr와 같은 경우에 Next\_PC또한 Jump\_addr과 같은 값을 갖는 것을 확인할 수 있다.

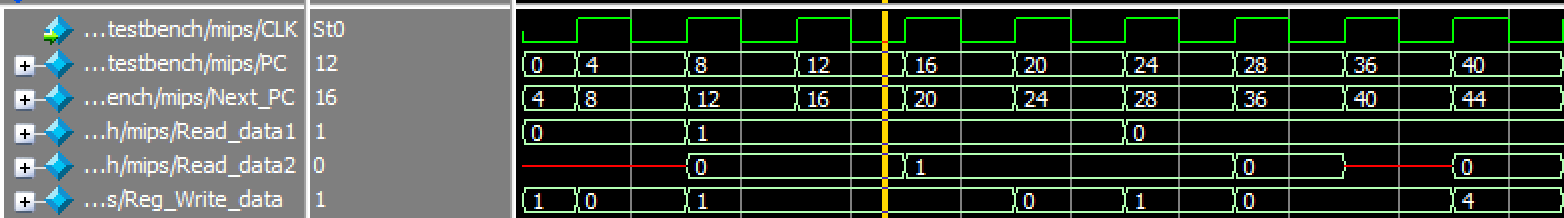
**2. 2번의 해결**

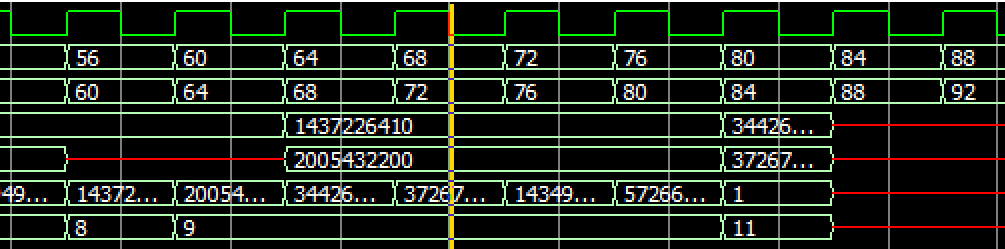


위의 기존 코드에서 else if 문을 제거해서



위의 코드로 수정을 하고 시뮬레이션 해본 결과



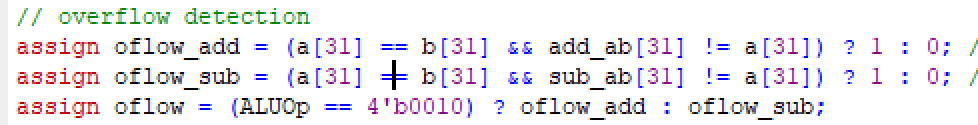


위와 같이 Read\_data2에 unknown 구간이 생기면서

Read\_data2가 write\_data와 동일한 값을 가지던 문제를 해결한 것을 확인하였다.

**3. 3번의 해결**

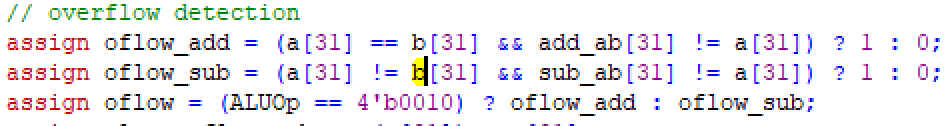
Overflow 여부를 판단하기 위해 ALU.v에 있던 다음 코드를 이용하기로 했다.



그런데, 덧셈과 뺄셈에서 overflow가 일어날 수 있는 조건을 생각했을 때

뺄셈에서 a[31]과 b[31]가 같다면, overflow가 발생할 수 없다.

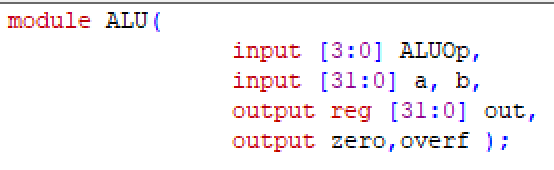
따라서 코드를

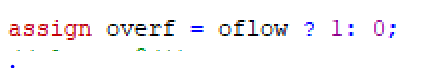
"

위과 같이 수정하였고

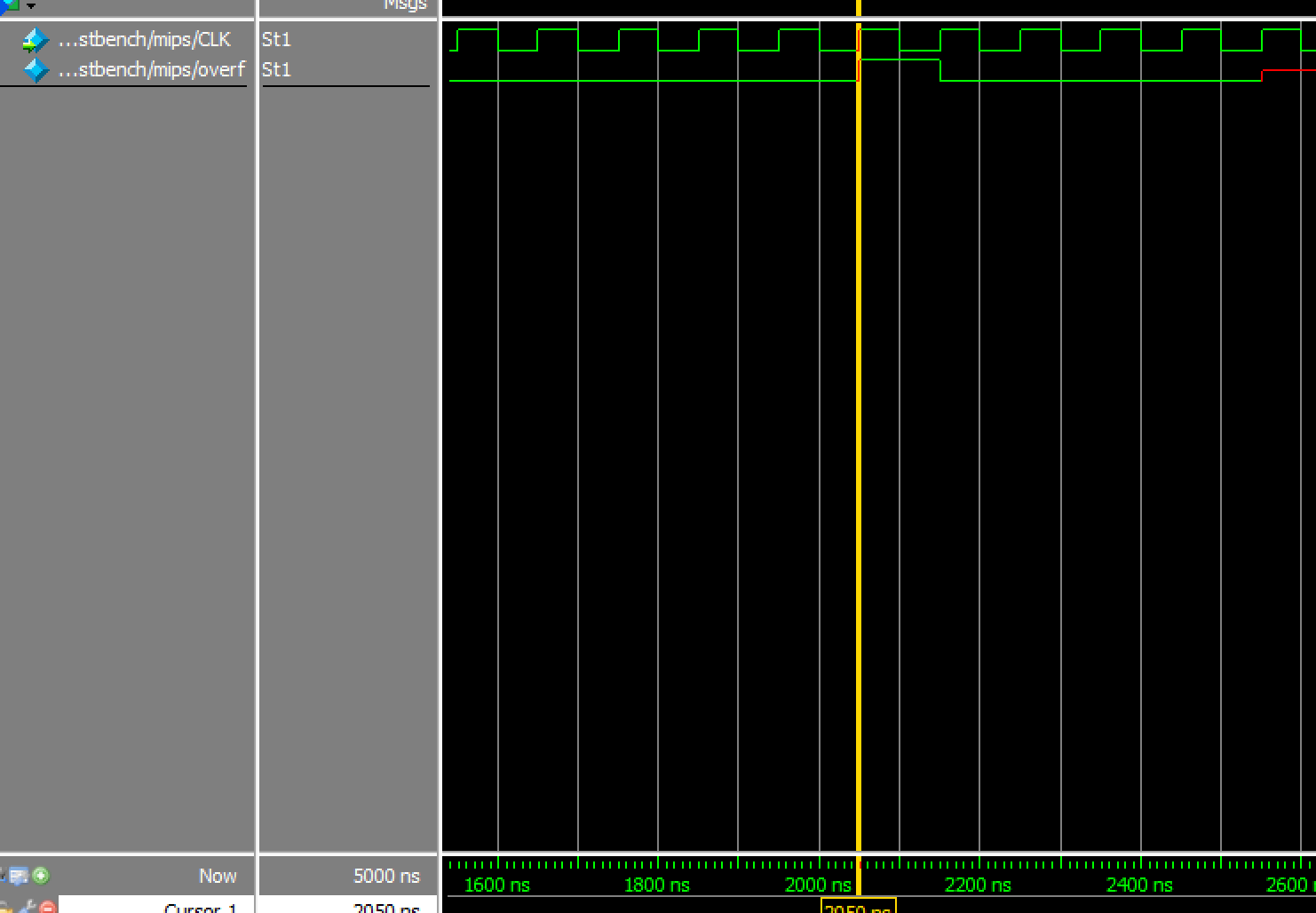
Mips.v에서 overf 라는 wire를 생성시키고 ALU에 입력 시킨후 ALU.v에서 oflow값을 받아오도록 코드를 작성하였다.

(overf가 1이면 overflow 발생, 0이면 발생 X)





시뮬레이션 시켜본 결과

기존에 overflow가 발생하던 시점에서 정확히 overf가 1로 상승하였고 해당 cycle이 끝난 후 다시 0의 값을 갖는 것을 확인하였다.

**4. 4번의 해결**

기존의



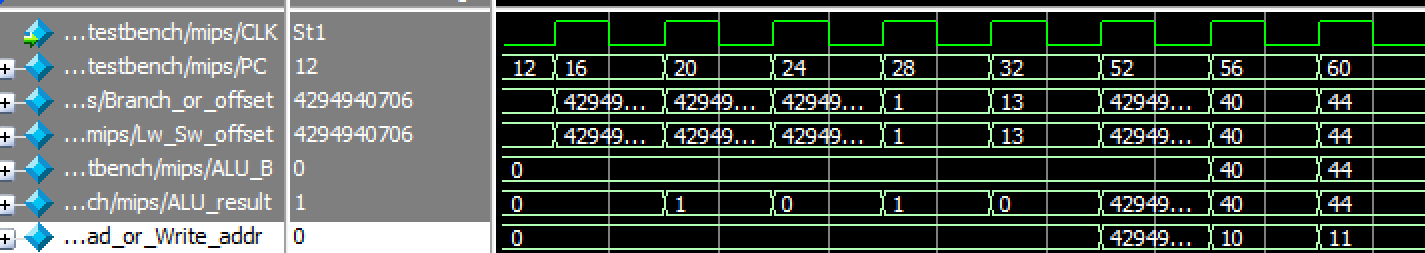


코드를 아래와 같이 수정하였다.





그 결과



위와 같이 Branch\_offset과 lw\_sw\_offset이 서로 일치하고 있으며, ALU\_B, ALU\_result와 Read\_Write\_addr 값 또한 정상적인 모습을 보이고 있다.