컴퓨터구조

이성원교수님

Project #1

학과 : 컴퓨터 정보 공학부

학번 : 2019202103

이름 : 이은비

제출 날짜 : 2023.04.16

[각 명령어에 대해 기능과 동작에 대한 설명]

ADDU (R-type instruction),

Fetch instruction: Instruction ← MEM[PC]

Fetch operands: data1 ← Reg(Rs), data2 ← Reg(Rt)

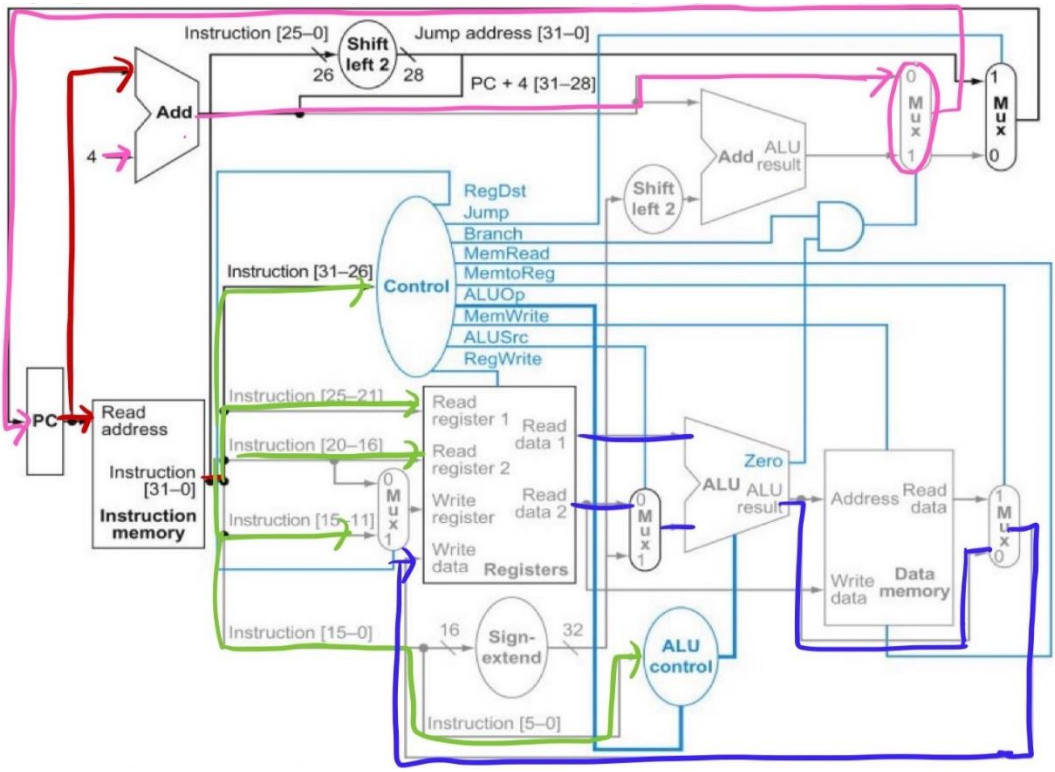
Execute operation: ALU_result ← func(data1, data2) ($d = $s + $t)

Write ALU result: Reg(Rd) ← ALU_result

Next PC address: PC ← PC + 4

Operation: $d = $s + $t

[data path]



OR (R-type instruction),

Fetch instruction: Instruction ← MEM[PC]

Fetch operands: data1 ← Reg(Rs),

data2 ← Reg(Rt)

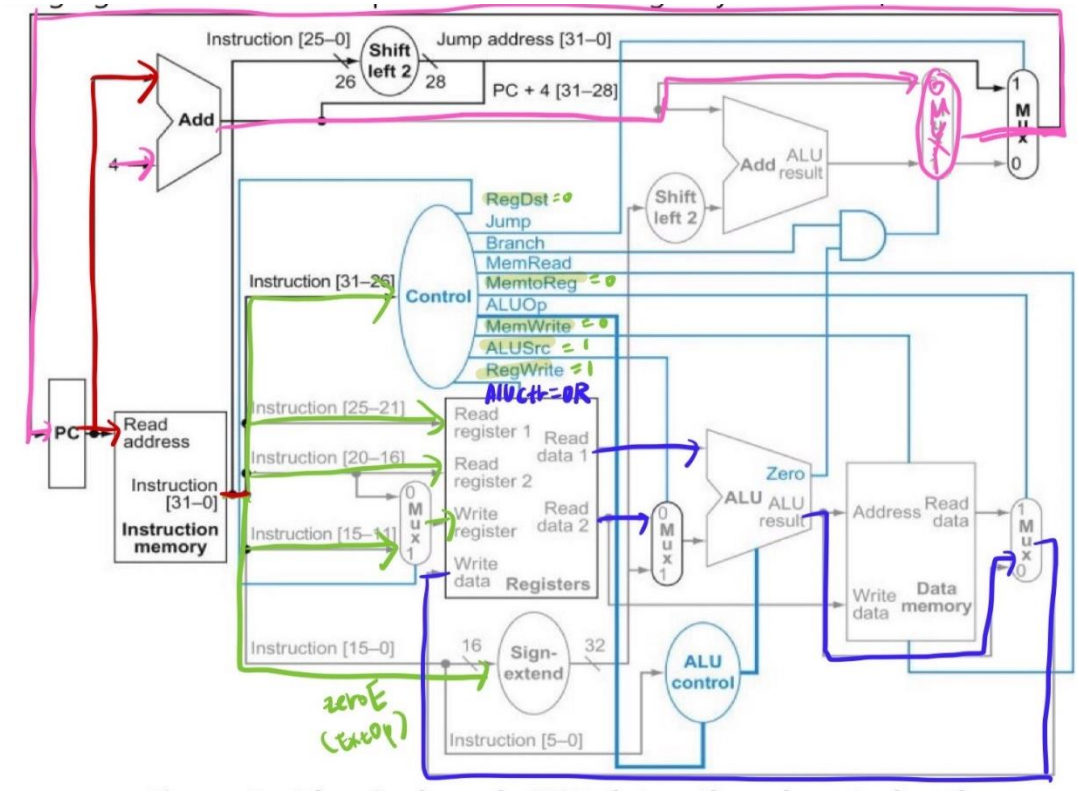
Execute operation: ALU_result ← func(data1, data2) ($d = $s | $t)

Write ALU result: Reg(Rd) ← ALU_result

Next PC address: PC ← PC + 4

Operation: $d = $s | $t

[data path]



ADDIU (RegIm),

Fetch instruction: Instruction ← MEM[PC] Fetch operands:

data1 ← Reg(Rs),

data2 ← Extend(imm16) (shift-extended를 거친)

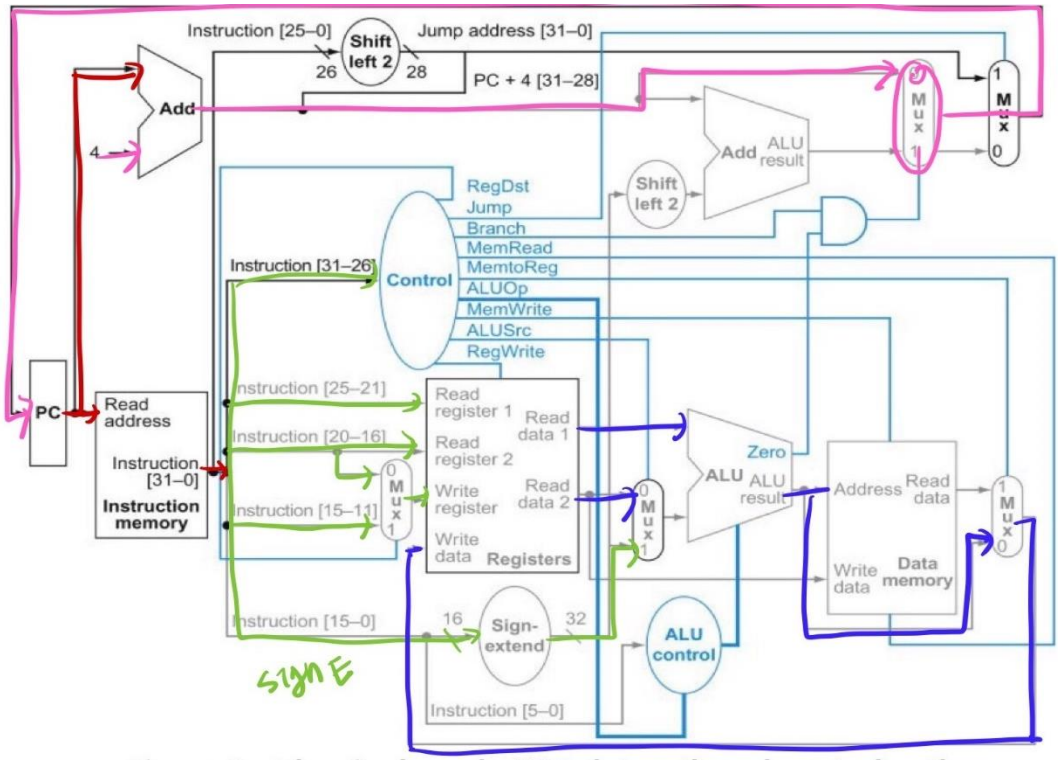
Execute operation: ALU_result ← op(data1, data2) ($t = $s + SE(i))

Write ALU result: Reg(Rt) ← ALU_result

Next PC address: PC ← PC + 4

Operation: $t = $s + SE(i)

[data path]



XORI

Fetch instruction: Instruction ← MEM[PC]

Fetch operands: data1 ← Reg(Rs), data2 ← Extend(imm16) (zero-extended를 거친)

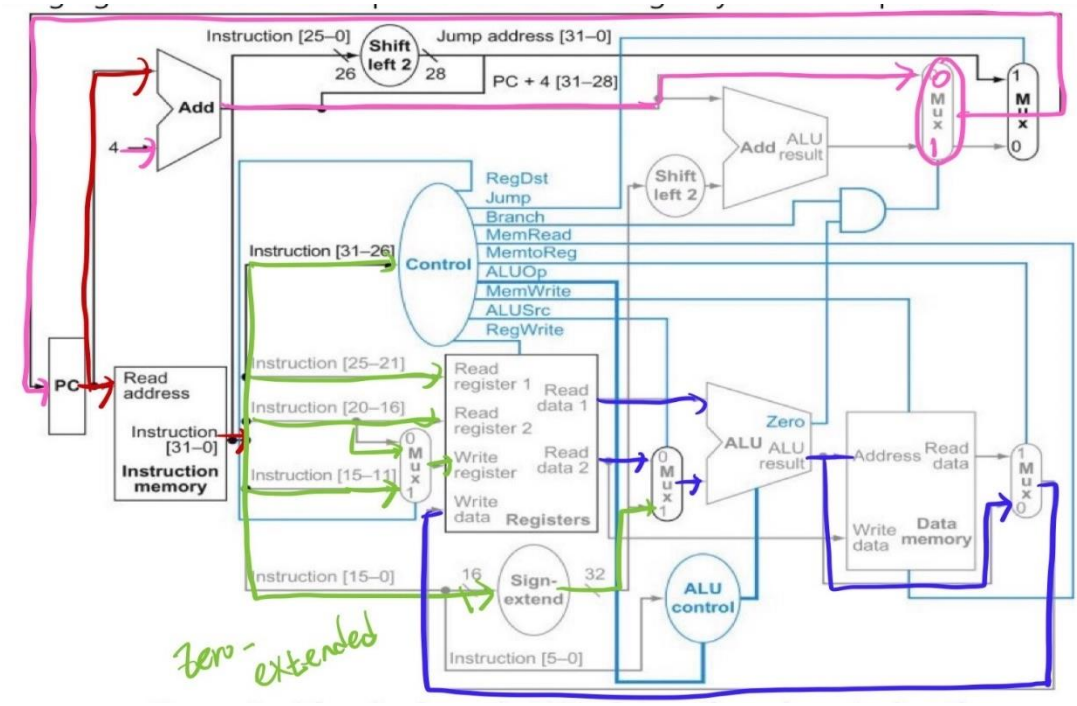
Execute operation: ALU_result ← op(data1, data2) ($t = $s ^ ZE(i))

Write ALU result: Reg(Rt) ← ALU_result

Next PC address: PC ← PC + 4

Operation: $t = $s ^ ZE(i)

[data path]



SLL(R-type instruction)

Fetch instruction: Instruction ← MEM[PC]

Fetch operands: data1 ← Reg(Rs), data2 ← Reg(Rt)

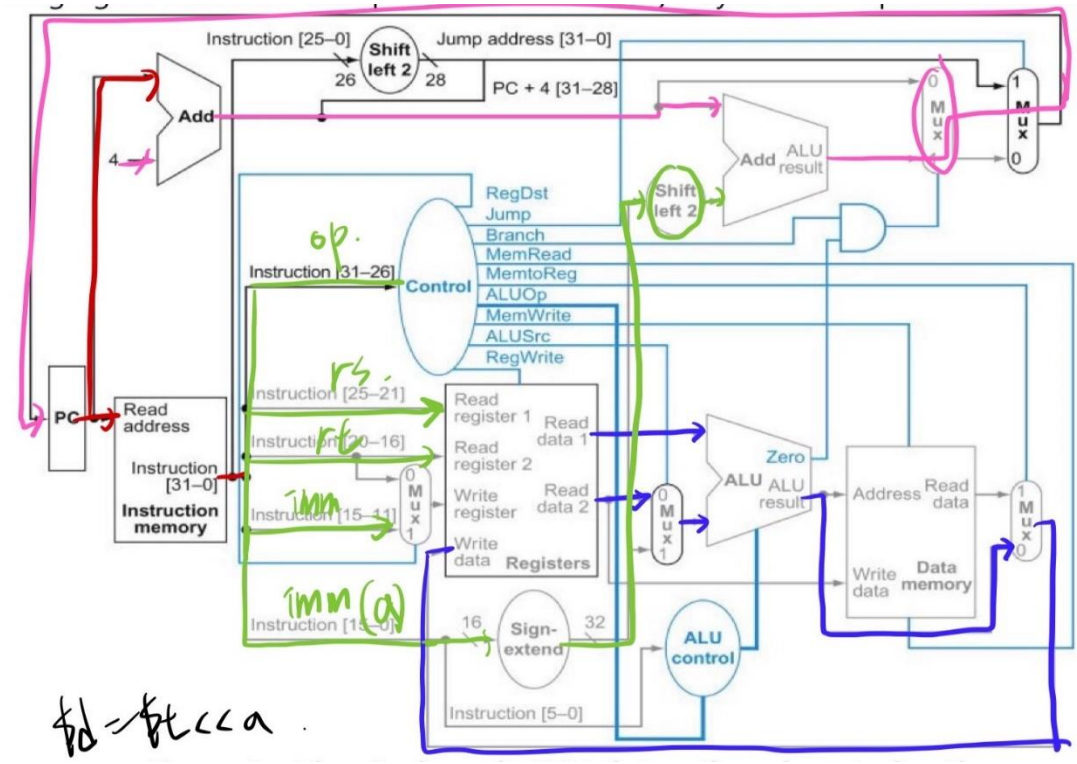
Execute operation: ALU_result ← func(data1, data2) ($d = $t << a)

Write ALU result: Reg(Rd) ← ALU_result

Next PC address: PC ← PC + 4

Operation: $d = $t << a

[data path]



SRAV(R-type instruction),

Fetch instruction: Instruction ← MEM[PC]

Fetch operands: data1 ← Reg(Rs),

data2 ← Reg(Rt)

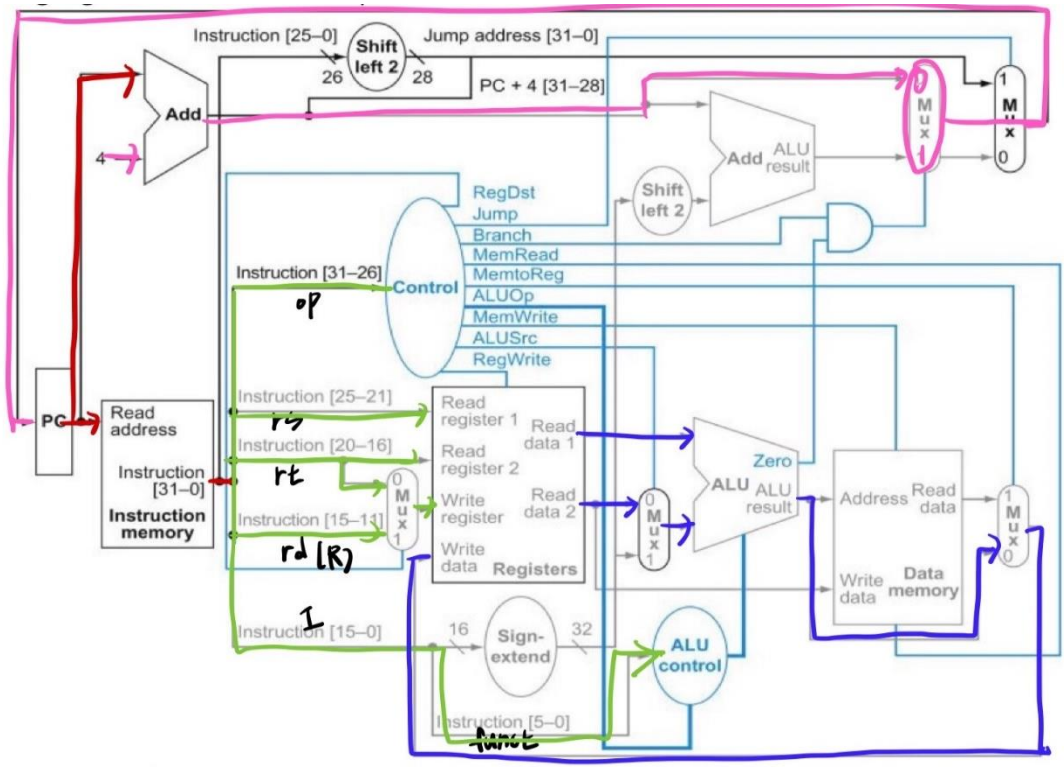
Execute operation: ALU_result ← func(data1, data2) ($d = $t >>> $s)

Write ALU result: Reg(Rd) ← ALU_result

Next PC address: PC ← PC + 4

Operation: $d = $t >>> $s

[data path]



SH(store)

Store half word,

Fetch instruction: Instruction ← MEM[PC]

Fetch registers: base ← Reg(Rs), data ← Reg(Rt)

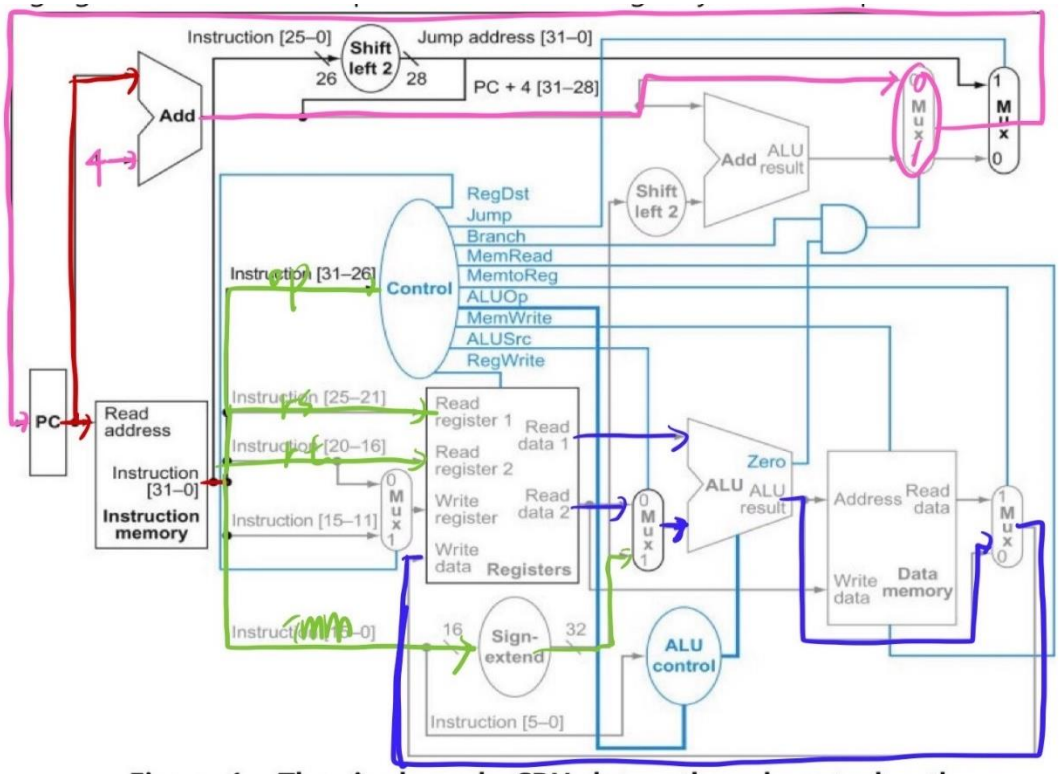
Calculate address: address ← base + sign extend(imm16)

Write memory: MEM[address] ← data (MEM [$s + i]:2 = LH ($t))

Next PC address: PC ← PC + 4

Operation: MEM [$s + i]:2 = LH ($t)

[data path]



LH(load)

Fetch instruction: Instruction ← MEM[PC]

Fetch base register: base ← Reg(Rs)

Calculate address: address ← base + sign extend(imm16) (SE (MEM [$s + i]:2)

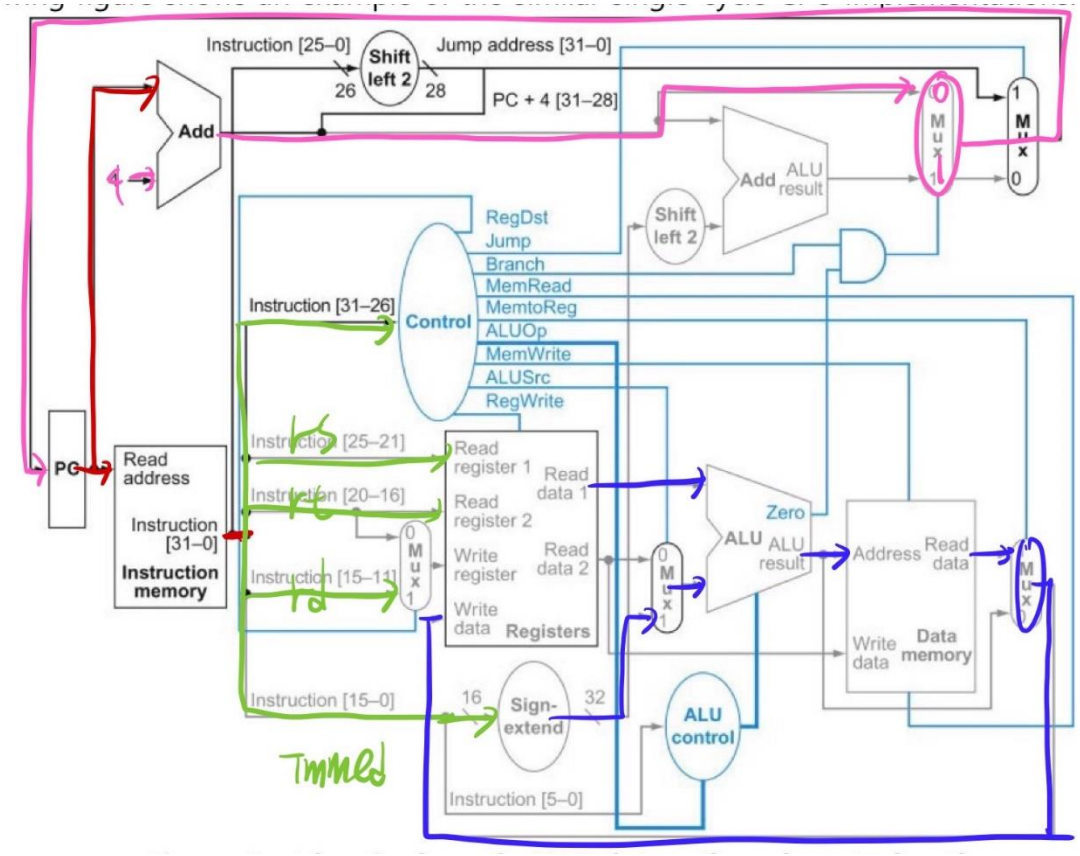
Read memory: data ← MEM[address]

Write register Rt: Reg(Rt) ← data ($t = SE (MEM [$s + i]:2))

Next PC address: PC ← PC + 4

Operation: $t = SE (MEM [$s + i]:2)

[data path]



BLTZ(RegIm),

Fetch instruction: Instruction ← MEM[PC]

Fetch operands: data1 ← Reg(Rs), data2 ← Reg(0)

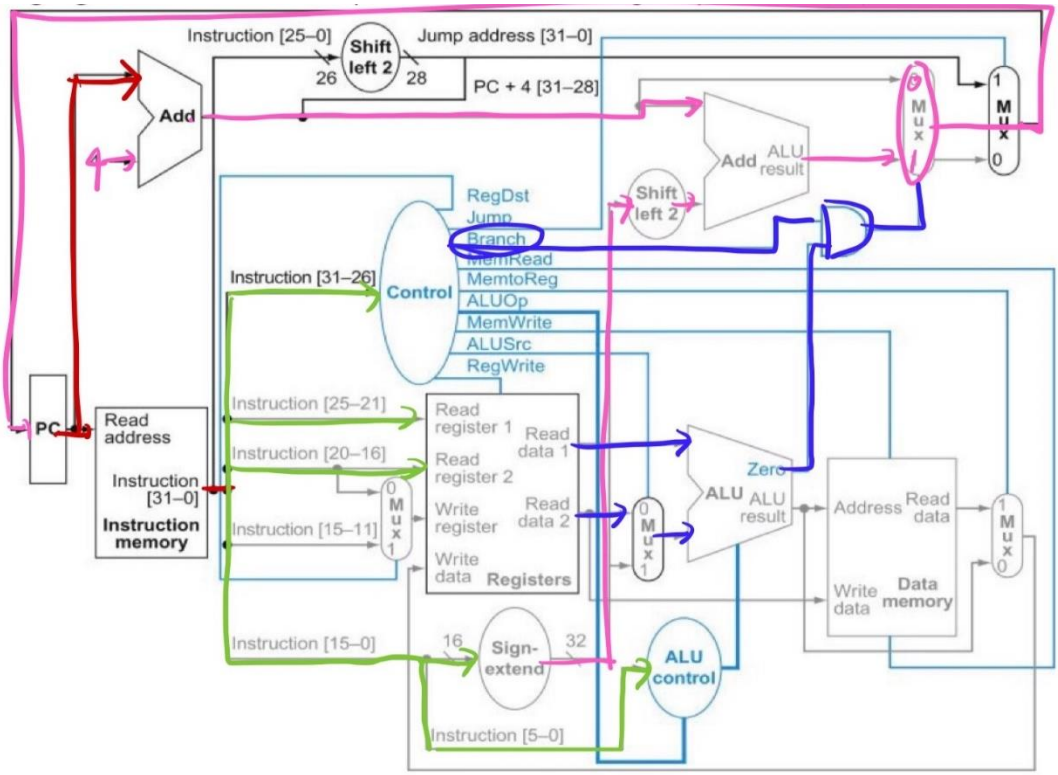
Conditional check: if (rs < 0),

Branch: if (zero) PC ← PC + 4 + 4×sign_ext(imm16)

else PC ← PC + 4

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

[data path]



JAL(jal)

Fetch instruction: Instruction ← MEM[PC]

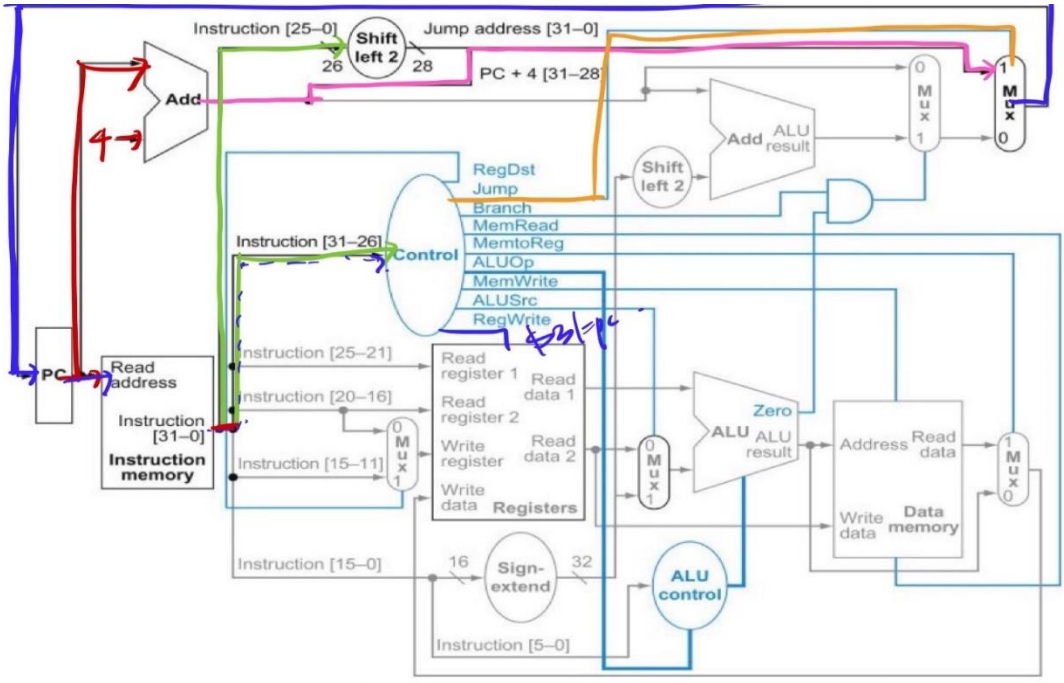
Target PC address: target ← PC[31:28] , Imm26 <<2

Jump: PC ← target

Write result: Reg($31) ← PC

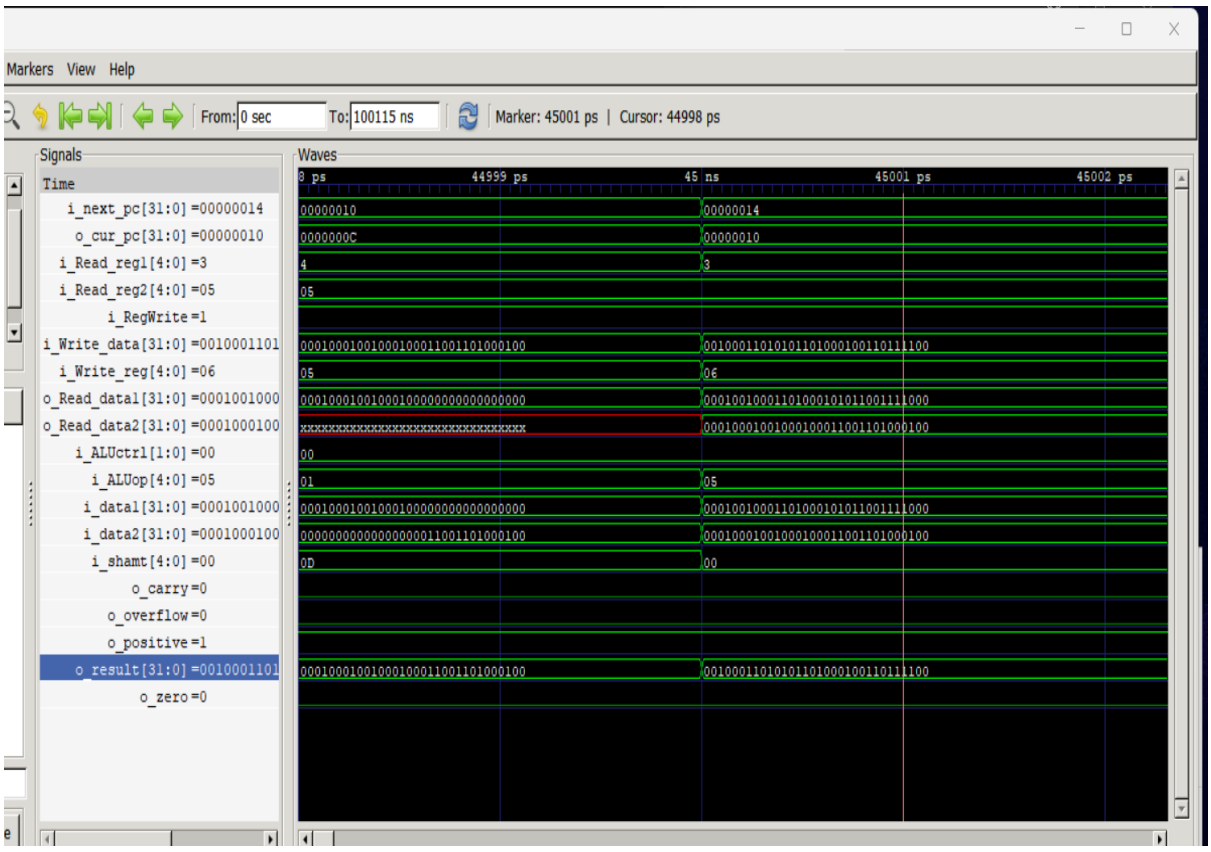
Operation: $31 = pc; pc = pc4 | i26 << 2

[data path]



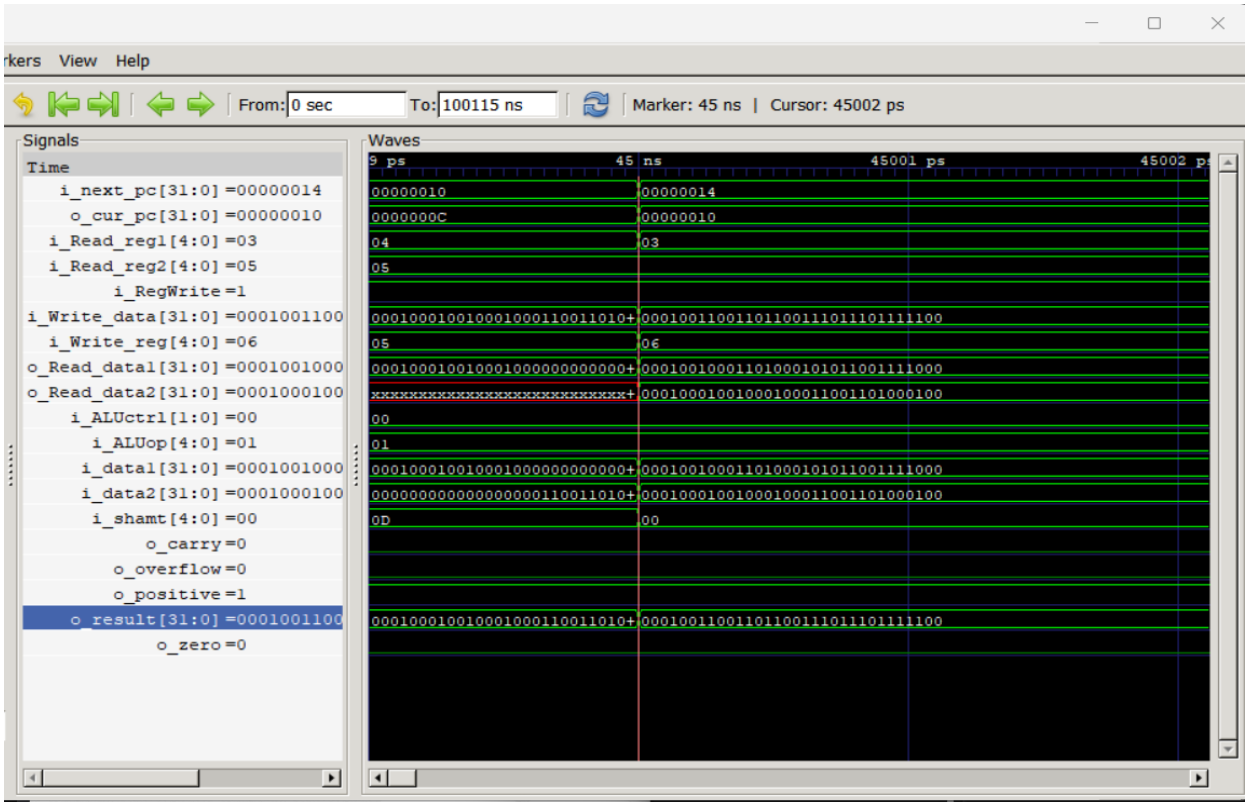
[Testbench]

ADDU



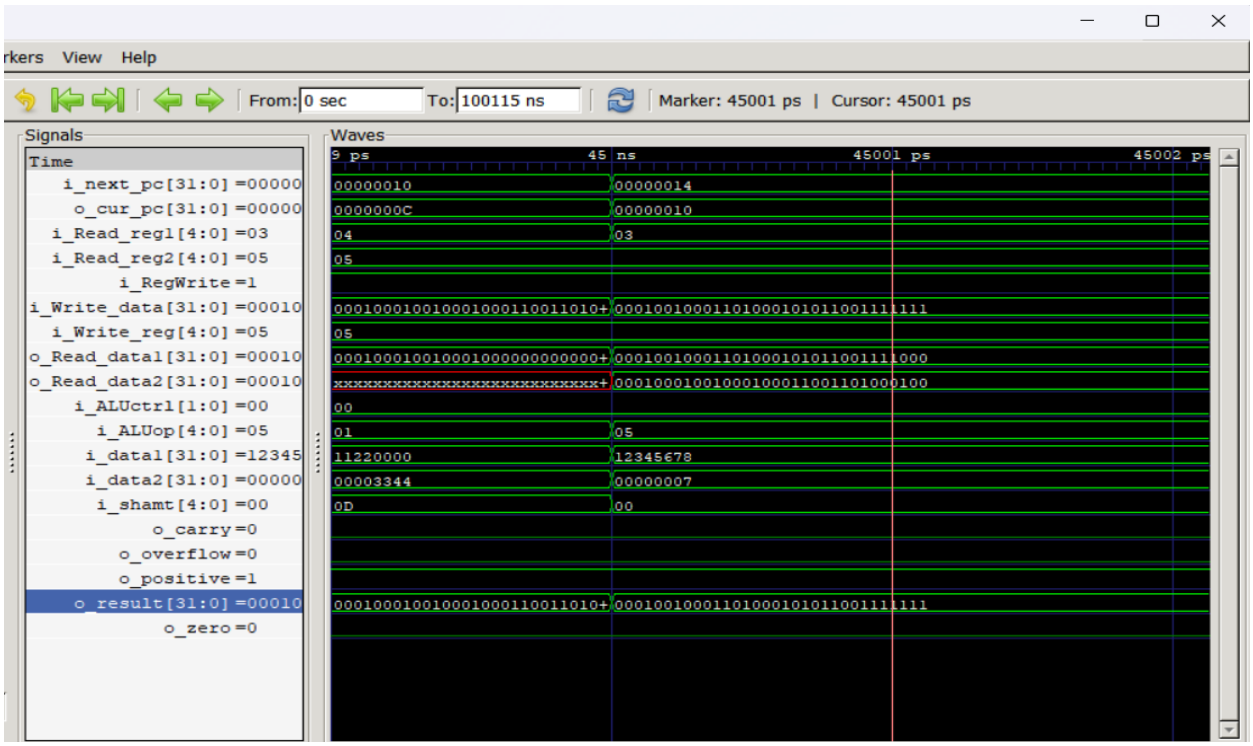
Reg1에는 3,reg2에는 5일때의 값을 확인합니다. $d=$s+$t 연산을 o_Read_data1[31:0] o_Read_data2[31:0] 의 ADDU연산을 진행한 결과 값을 o_result[31:0] 로 확인 할 수 있었습니다.

OR



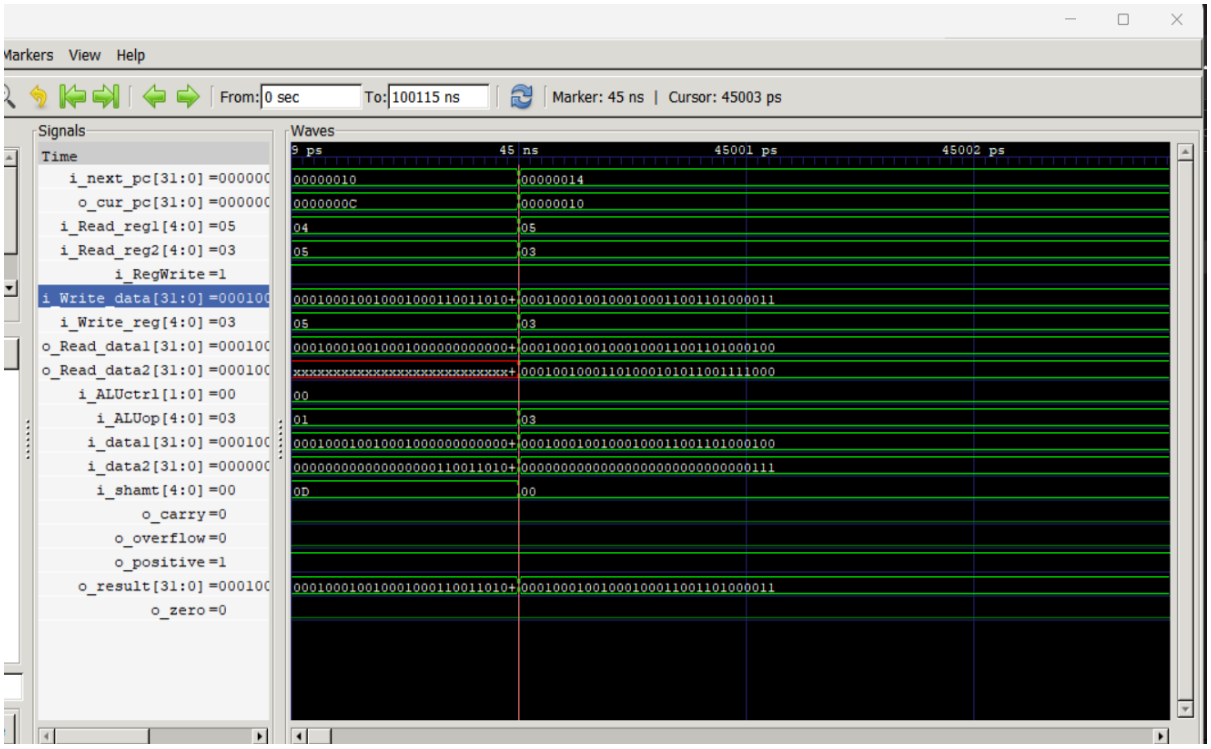
$d = $s | $t reg1, reg2가 각각 3 과 5 일 때, 이에 맞는 연산 결과입니다. o_Read_data1과o_Read_data2의 or연산 결과 값이 o_result[31:0]으로 확인 된 결과 입니다.

ADDIU



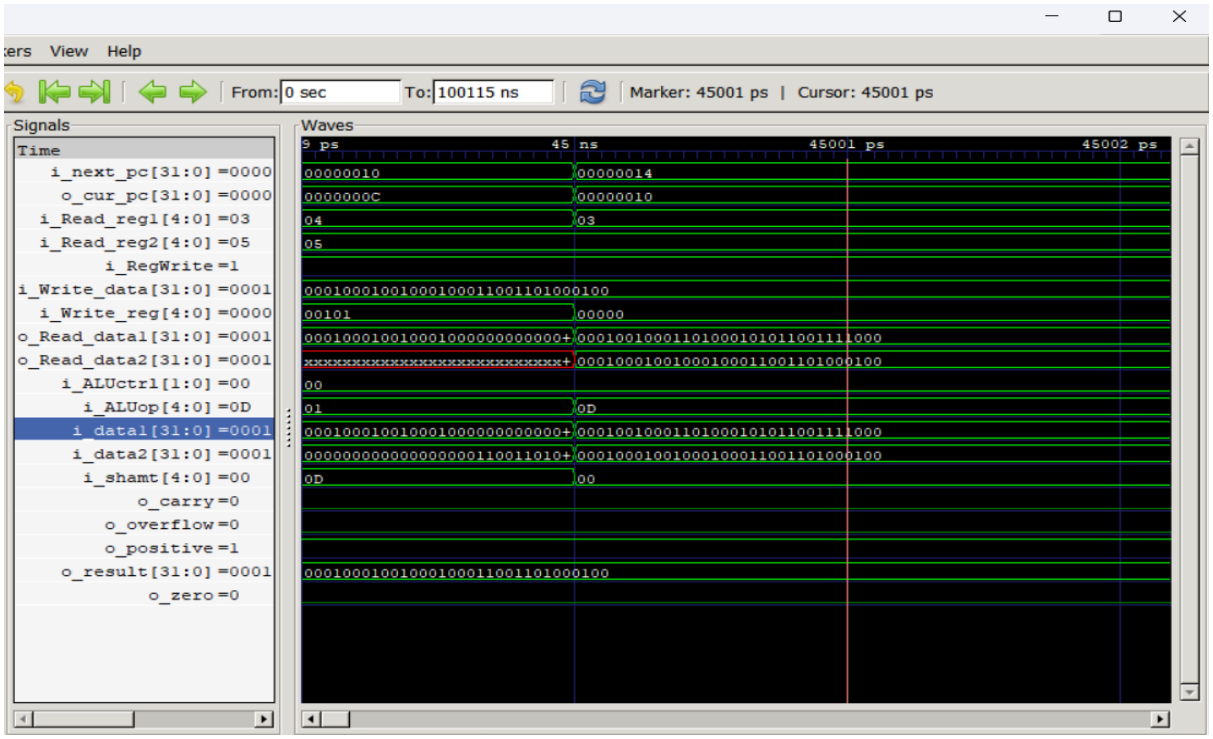
ADDIU $t = $s + SE(I) 의 연산을 I-type의 형식에 맞춰서 진행 한 결과로 imm값을 7이고 s와 t는 각각 3과 5입니다. 그리고 그 때 해당하는 o_Read_data1 과 o_Read_data2의 ADDIU연산을 진행한 결과로 o_result값이 나온 것을 확인 할 수 있습니다.

XORI



XORI $t = $s ^ZE(I) 의 연산을 I-type의 형식에 맞춰서 진행 한 결과로 imm값을 7이고 s와 t는 각각 3과 5입니다. imm값의 zero extend연산 이후 XOR연산을 진행하는 과정을 수행합니다. 그리고 그 때 해당하는 o_Read_data1 과 o_Read_data2의 XORI연산을 진행한 결과로 o_result값이 나온 것을 확인할 수 있습니다.

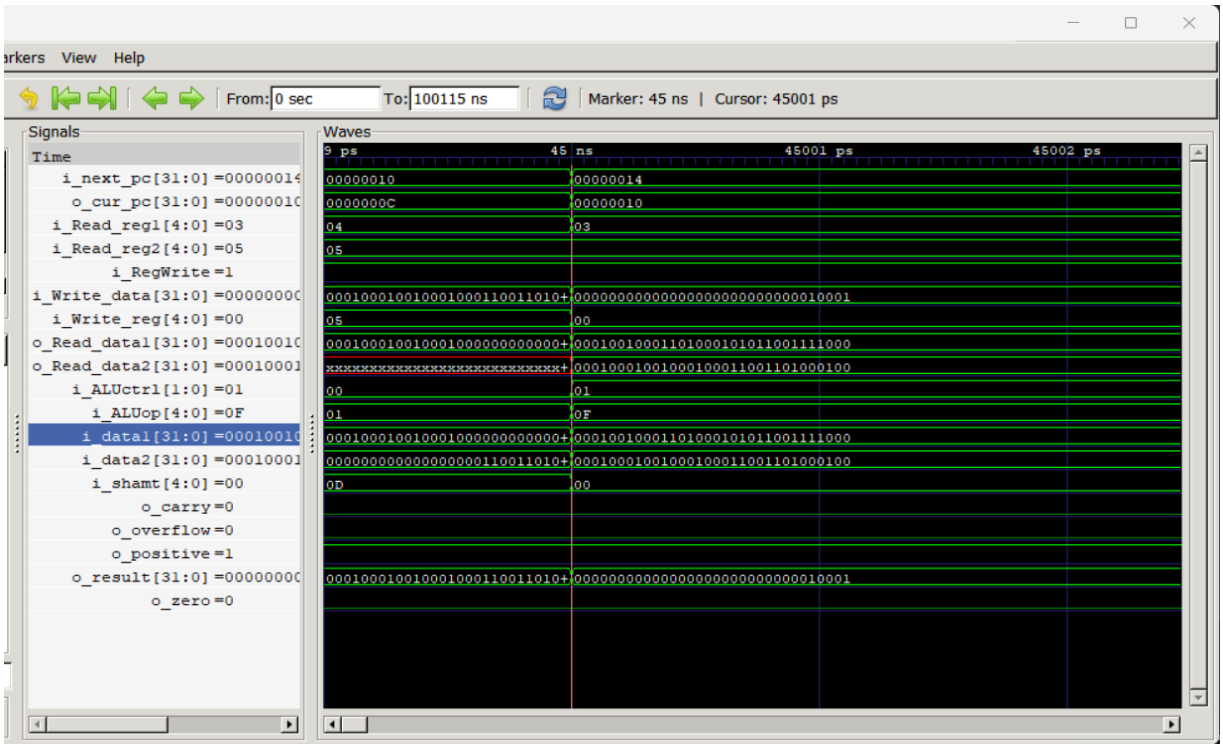
SLL



$d = $t << a

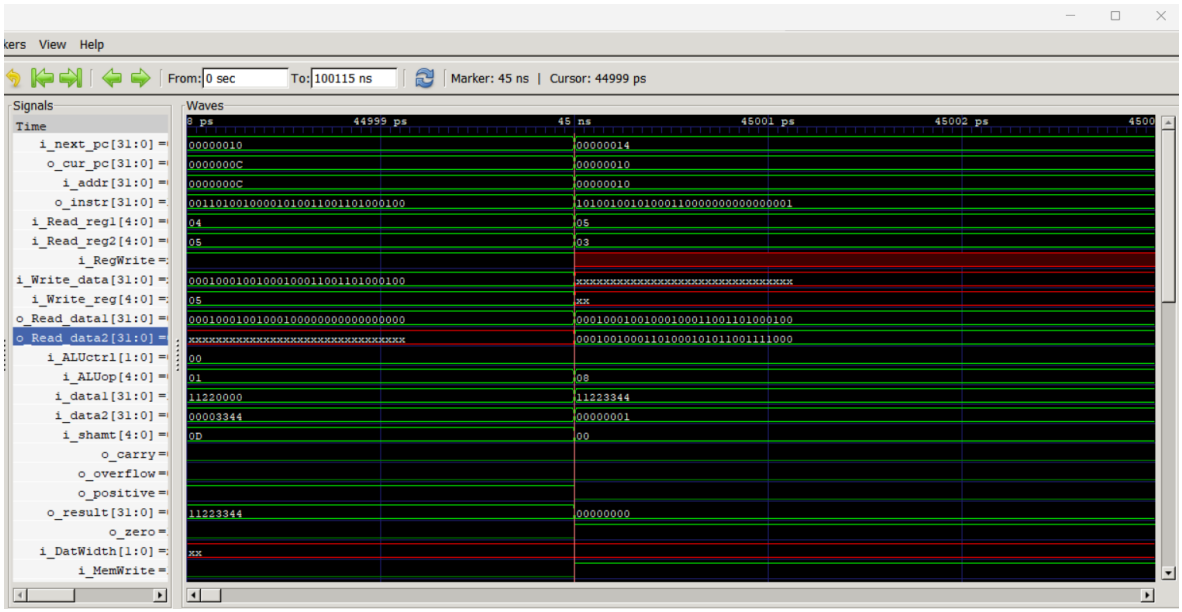
SLL 즉, $d = $t << a 의 연산을 R-type의 형식에 맞춰서 진행 한 결과로 s와 t는 각각 3과 5입니다. 그리고 그 때 해당하는 o_Read_data1 과 o_Read_data2의 SLL연산을 진행한 결과로 o_result값이 나온 것을 확인 할 수 있습니다.

SRAV



SRAV $d = $t >>>$a 의 연산을 R-type의 형식에 맞춰서 진행 한 결과로 imm 값을 7이고 s와 t는 각각 3과 5입니다. 그리고 그 때 해당하는 o_Read_data1 과 o_Read_data2의 SRAV연산을 진행한 결과로 o_result 값이 나온 것을 확인할 수 있습니다.

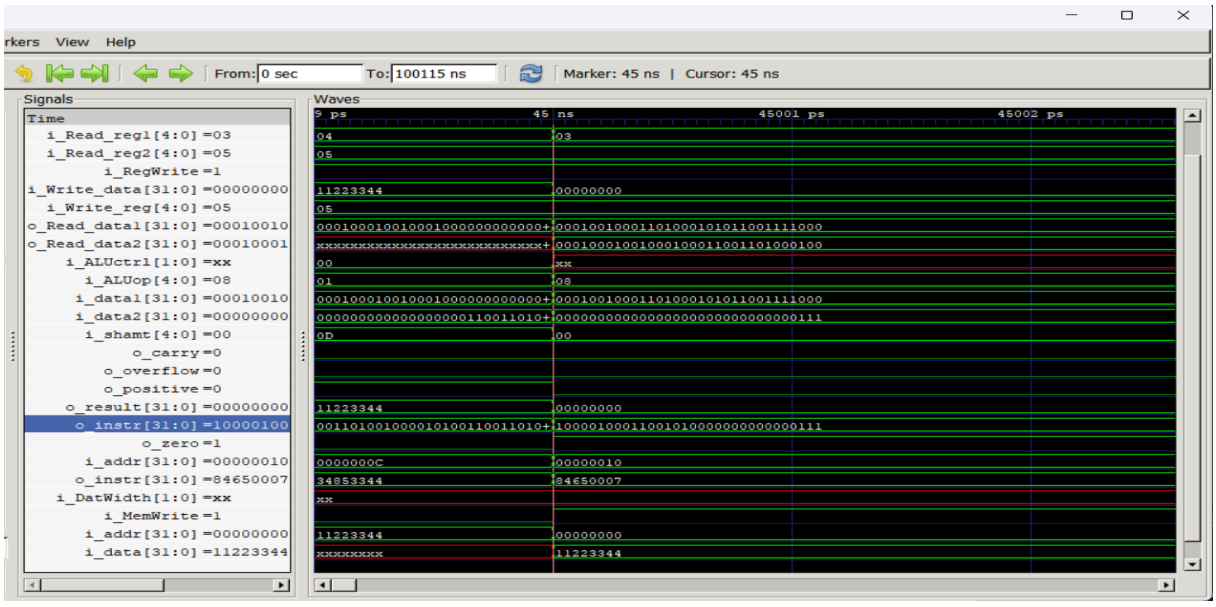
SH



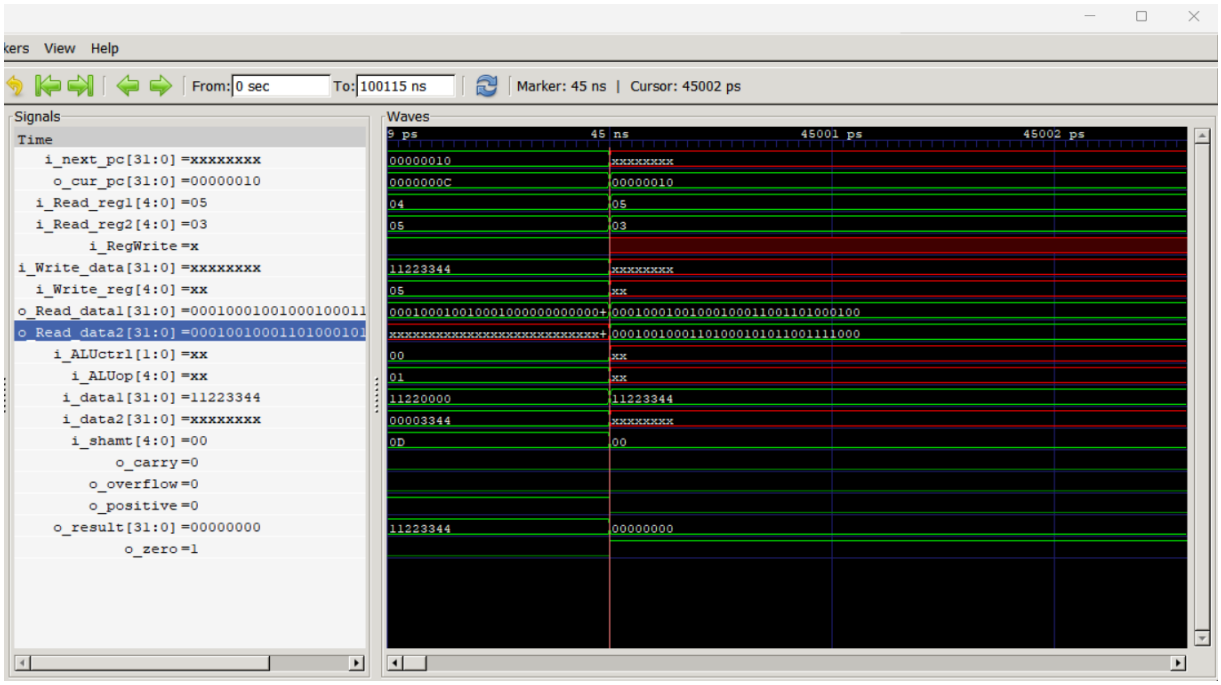
SH MEM[$s + i]:2 = LH($t)의 연산을 I-type의 형식에 맞춰서 진행 한 결과로 memory에 halfword만큼 로드 된 rt를 대입한 결과 입니다.

LH

LH $t = SE(MEM[$s+i]:2)의 연산을 I-type의 형식에 맞춰서 진행 한 결과로 memory의 값을 sign extension한 값을 rt에 halfword만큼 대입한 것 입니다.

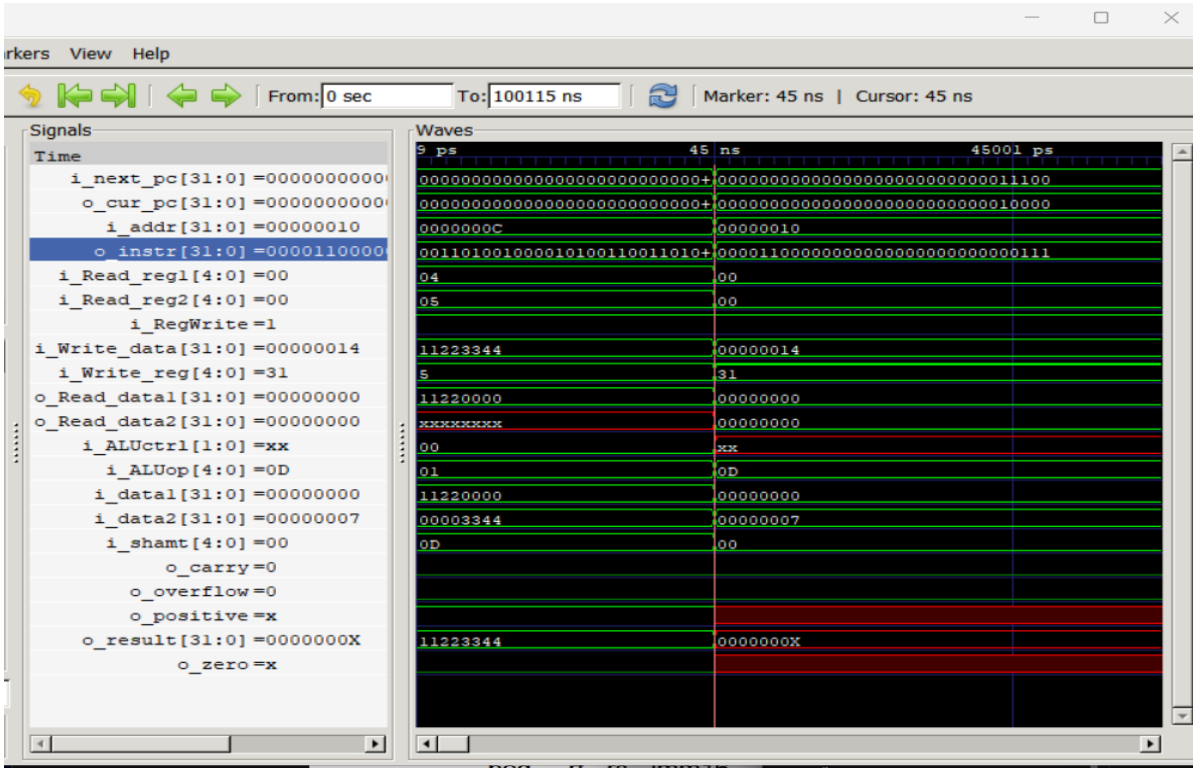


BLTZ



BLTZ 즉, If ($s < 0) pc += i <<2 의 연산을 진행 한 결과로 s에 5값 즉 양수 값이 들어가서 pc += i<<2와 같은 연산은 진행 되지 않은 것을 확인 할 수 있습니다.

JAL



JAL 즉, $31 = pc; pc =pc4 | i26 << 2일 때 imm값은 7이고 write되는 reg 31번에 해당하는 i_Write_data

를 통해 해당 연산이 진행된 것을 확인 할 수 있습니다.

[Single Cycle CPU 블록도]

pc에서 주소 받아오고 명령어 해석하는 과정에서 Rtype, Itype, Jtype을 구분해서 IM에서 rs, rt, rd, imm 각각의 주소를 맞게 register로 받아옵니다. 그리고 이 register에 해당하는 data값을 alu로 넘겨주면 Data Memory 혹은 register로 연산 결과를 write하는 과정이 필요합니다.