CSE 331

HW3

Burak Kocausta

1901042605

Content

Important Note

- After restoring the project using QAR file, if test benches are wanted to be run, after opening the modelsim, please add the initialization files from initialization_file directory that I provided to "/simulation/modelsim" directory. If this is not done, modelsim cannot find the initialization files.

Testing with Mips Assembly Language:

- If instructions are given as mips assembly language, inside assembler.py input and output file names are commented out, entering the file names

there will be enough. Please put the output file to the "/simulation/modelsim" directory after modelsim is run.

Inside assembler.py

```
# test for spec1
input_file = open("spec1.asm", "r")
output_file = open("instruction_init_spec1.mem", "w")
```

Inside test bench

```
// initialize instruction memory from mem file
$readmemb("instruction_init_spec1.mem", processor.inst_mem.memory);
```

More detail for the assembler is in the assembler part.

Testing with Machine Codes:

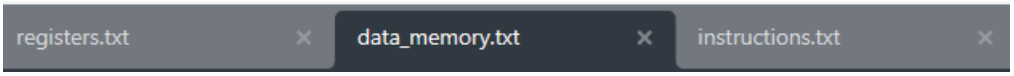
- For testing other than the test benches that I provided, if instructions are given as machine codes, after opening QAR, please provide .mem file in the simulation/modelsim directory. Example .mem file formats are inside initialization_file directory.

Sample instruction memory initialization usage:

```
// initialize instruction memory from mem file
$readmemb("instruction_init_tb1.mem", processor.inst_mem.memory);
```

Memory Outputs:

- After running test benches; data_memory, instruction_memory, registers are outputted as data_memory.txt, instruction_memory.txt and registers.txt in the /simulation/modelsim directory.

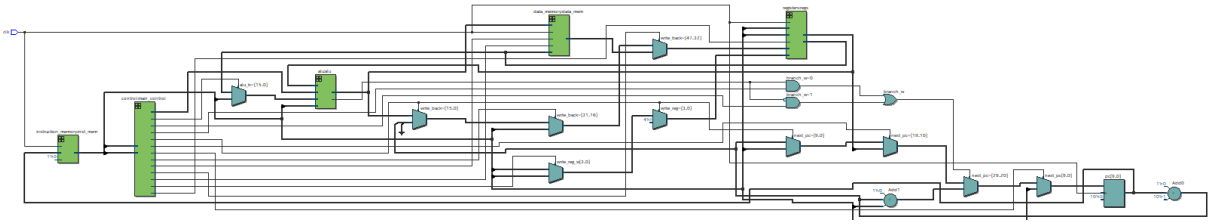


With this way, last situation of the memories can be observed.

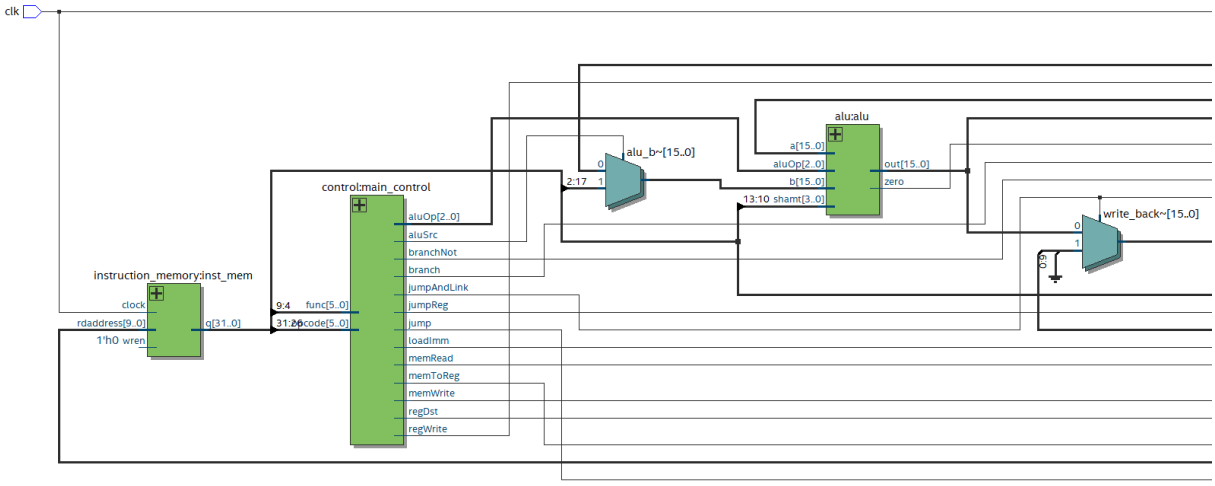
Design Explanation and Views

1- General View of the Processor

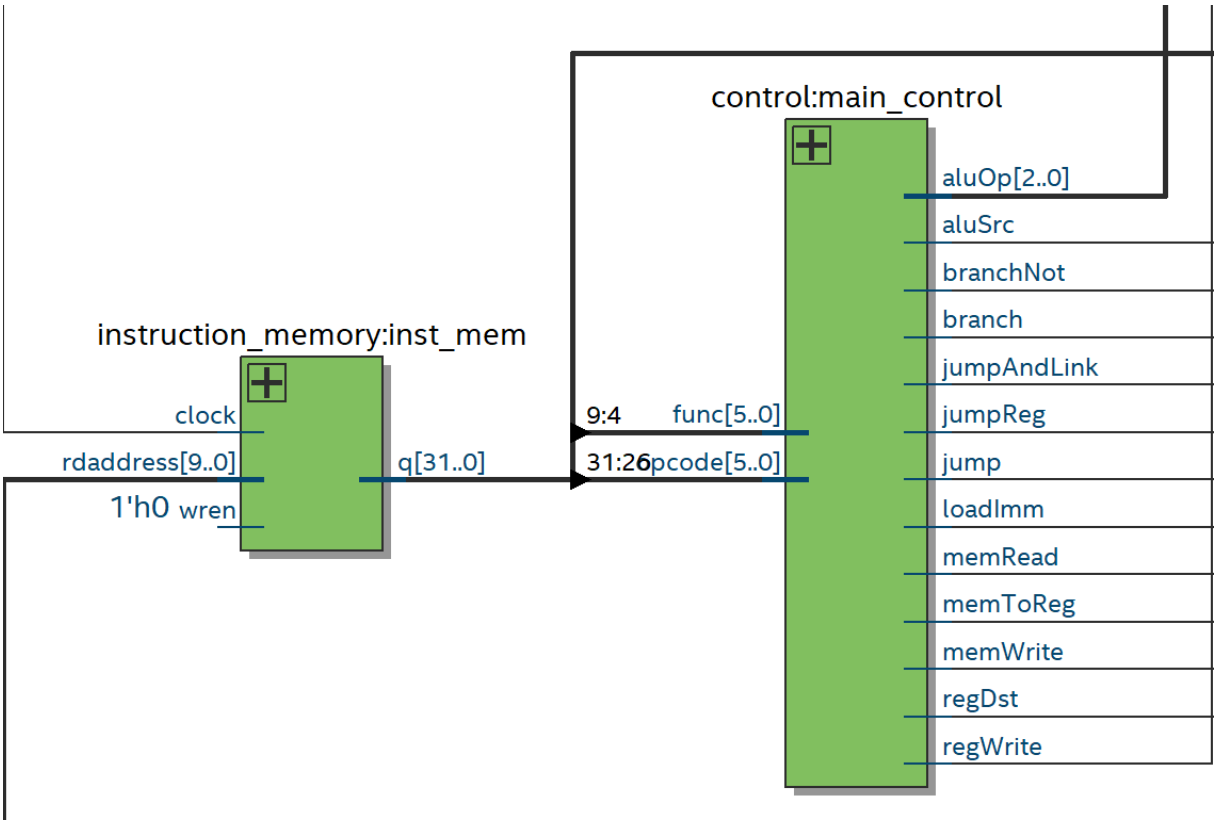
(all)



Looking closer

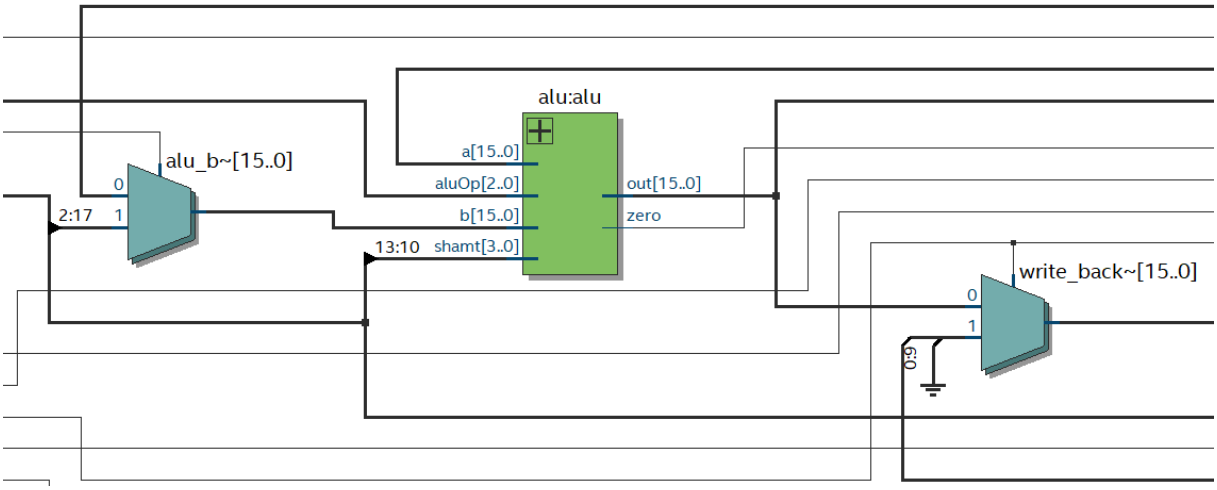


- Instruction Memory is initialized by given .mem file. All instructions are read from the instruction memory. It is not read only memory but, in the processor, I used it as read only.
- Control unit is the only controller in my design. In the lectures, there is an alu controller, but I removed it with giving function and opcode fields of the instruction to the main control. I explained it in detail in the control part.

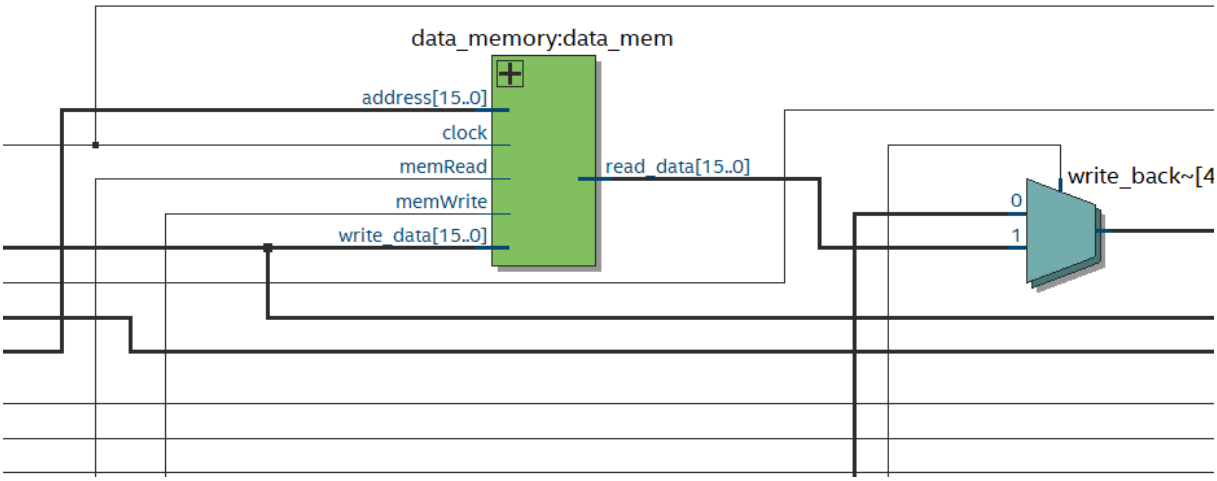


Controller generates these signals, on the right. Each signals meaning are:

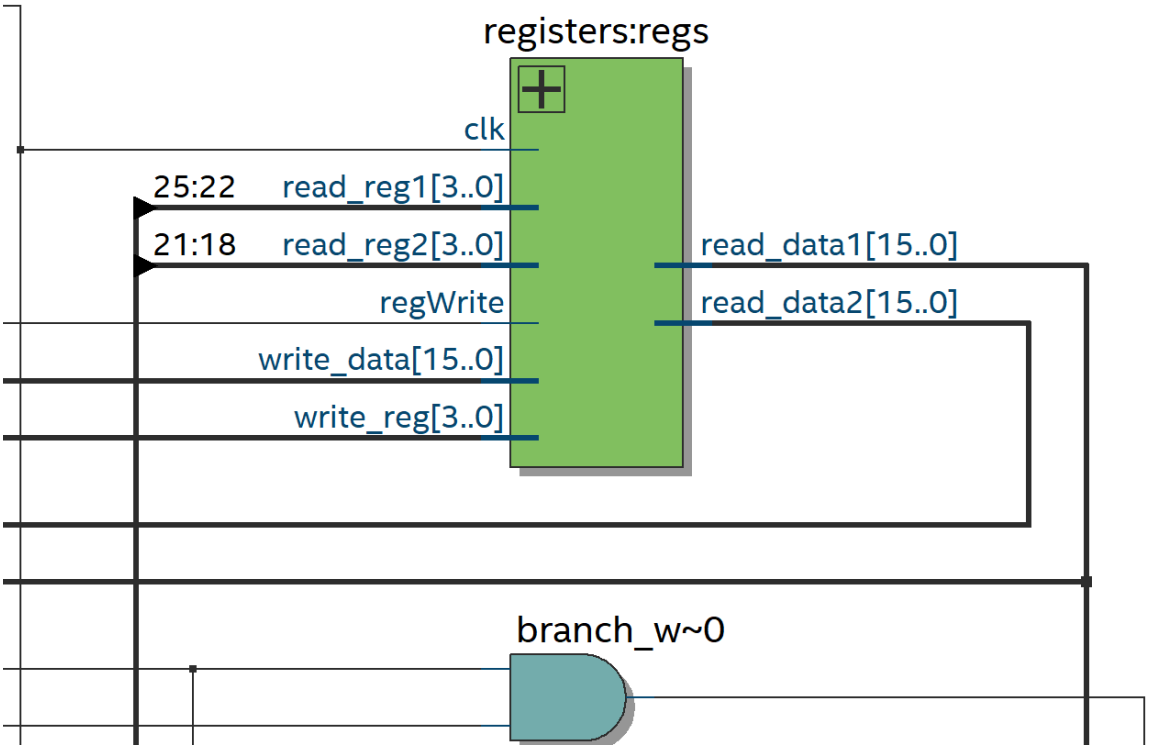
- aluOp: Alu operation to control alu.
- aluSrc: alu source, determines if alu input is immediate or rt value.
- branchNot, branch: As the name suggests, these are generated for beq, and bne instructions.
- jumpAndLink: generated for jal instruction.
- jumpReg, jump: generated for jr, j instructions.
- loadImm: generated for li instruction.
- memRead: It controls the memory for reading.
- memToReg: It controls the write back signal.
- memWrite: Controls the memory for writing to it.
- regDst: Controls if instruction is r type or not.
- regWrite: Controls register writing.



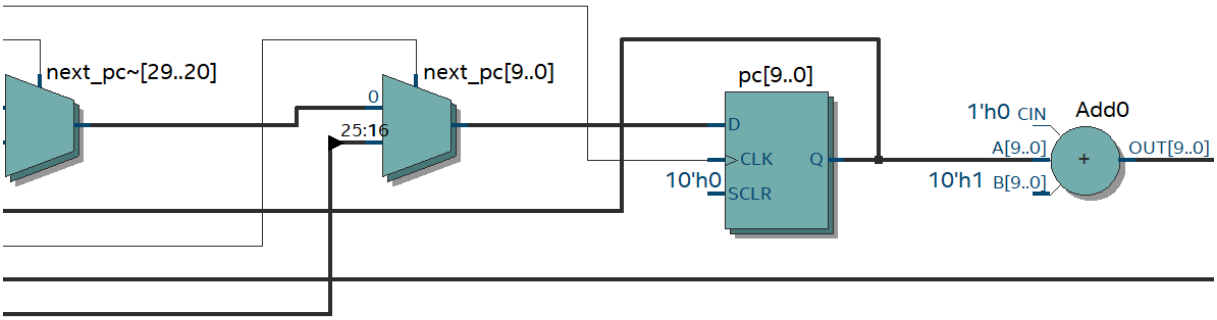
- Alu takes a, b 16 bit inputs. Also, it gets aluOp to determine which operation is done. Then 16 bit output, and zero output. Zero output is used for branch operations to compute next program counter value. Alu has shift amount input also for shift operations.



- Processor has data memory. It takes memRead signal, this signal is true when data is read from memory. memWrite is true when data is written to the memory. 16 bit address must be provided, when one of these operation is performed to the memory.



- Registers, have read_reg1, read_reg2 4 bit inputs, to determine which register is read. It has 16 register inside. Each of them is 16 bit. When data is wanted to written to register, regWrite must be true, and 16 bit data, also write register 4 bit must be provided.

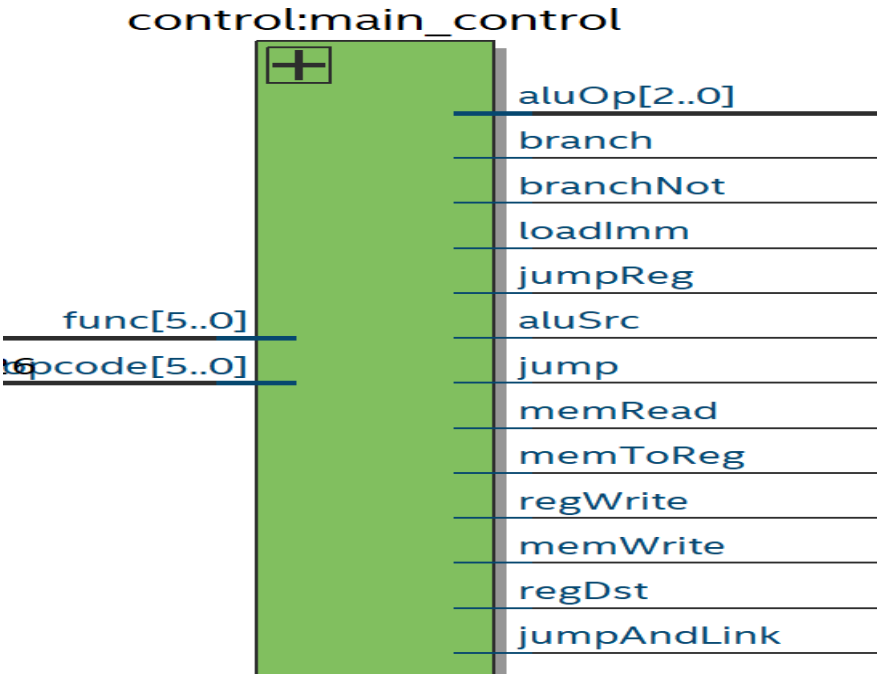


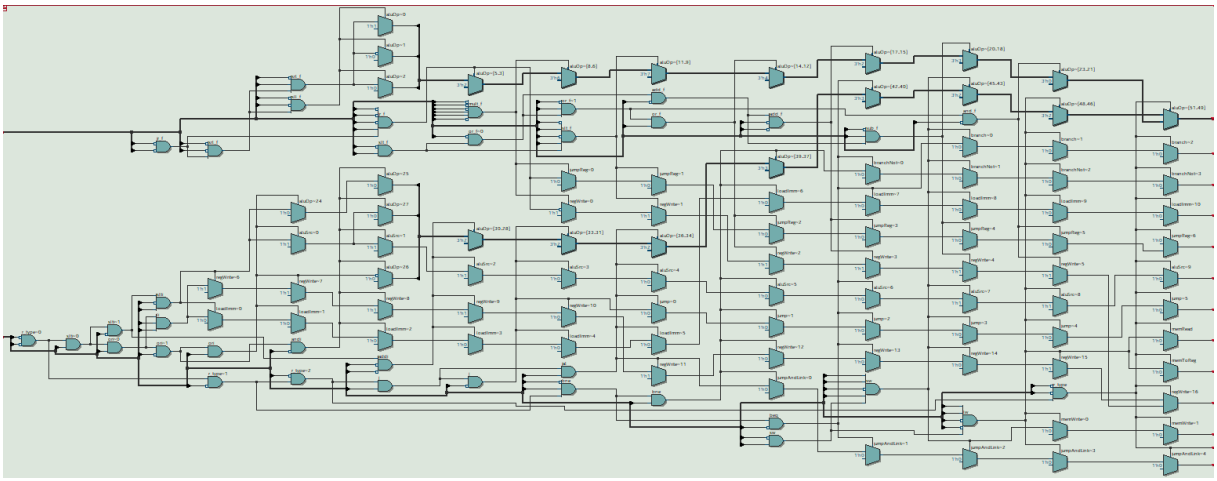
- 10 bit program counter must be in the processor to determine next instructions address. It has several conditions which is affected by branch, and jump instructions. To get the next address pc+1 is used. Incrementing by 1 is enough to get next instruction. Jump instructions immediate field is directly assigned to the program counter. Branch instructions are pc+1+(imm).

There are some important wires, and multiplexers in the processor.

- write_back: It is important because it is used for register write. To determine write_back value multiplexer is needed. It will be read_data memory if memToReg is true because memory is read, and data will be stored to the register. It can be immediate field, when load immediate instruction is the current instruction. If jump and link instruction is the current instruction, then the next address must be stored to the register 15, so if jal signal is true write back will be that data. If none of them is true, then alu result is the write back. Of course register signal must be true, to store it in the register also.
- next_pc: It is another important signal because determines the next instruction. It is affected from branch and jump operations. If jump or jump and link signals are true, it directly sets the next instruction to the given jump address. If it is jump register, then directly sets to the given registers data. If none of them are true, then it is pc+1.
- write_reg: It holds the 4 bit data for which registers is used for writing. Initially it is determined according to the instructions type(r type or not). Then jump and link signal is checked, if jump and link is true 15th register will be used.

2- Controller





- Controller rtl looks complicated, but it simply generates all the signals according to the 6 bit opcode, and function field. Initially it determines, if given instruction is r type or not. If it is r type then, checks for the function field, and determines all the result signals according to that. If instruction is not r type then it is easier, just determine the signals with looking opcode.
- All of the output signals can be generated just by looking opcode, and function field, these signals determines how the processor performs completely.
- Here are the tables for each instructions generated signals.

r-types: results determined according to the func(6 bits)

	add	sub	and	or	slt	sll	srl	jr	mult
branch	0	0	0	0	0	0	0	0	0
branchNot	0	0	0	0	0	0	0	0	0
loadImm	0	0	0	0	0	0	0	0	0
jumpReg	0	0	0	0	0	0	0	1	0
aluSrc	0	0	0	0	0	0	0	0	0
Jump	0	0	0	0	0	0	0	0	0
memRead	0	0	0	0	0	0	0	0	0
memToReg	0	0	0	0	0	0	0	0	0
regWrite	1	1	1	1	1	1	1	0	1
memWrite	0	0	0	0	0	0	0	0	0
regDst	1	1	1	1	1	1	1	1	1
jumpAndLink	0	0	0	0	0	0	0	0	0
aluOp	010	110	000	001	111	100	101	x	011

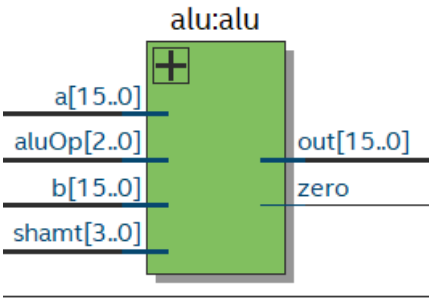
Some similar Immediate Types:

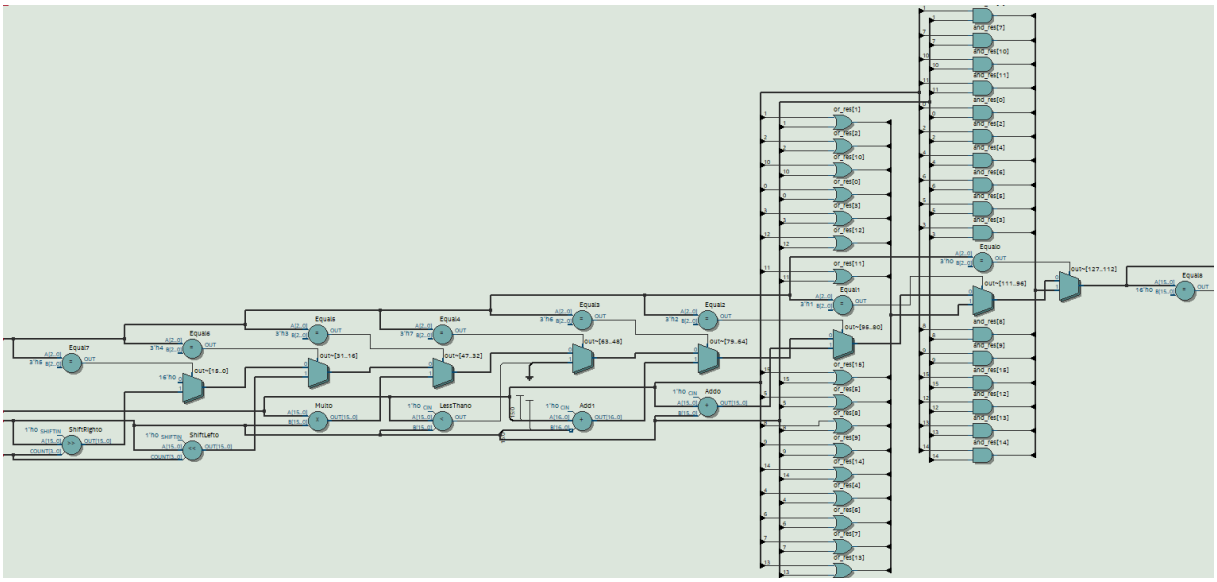
	addi	li	andi	ori	slti
branch	0	0	0	0	0
branchNot	0	0	0	0	0
loadImm	0	1	0	0	0
jumpReg	0	0	0	0	0
aluSrc	1	0	1	1	1
Jump	0	0	0	0	0
memRead	0	0	0	0	0
memToReg	0	0	0	0	0
regWrite	1	1	1	1	1
memWrite	0	0	0	0	0
regDst	0	0	0	0	0
jumpAndLink	0	0	0	0	0
aluOp	010	x	000	001	111

Other instructions

	j	jal	beq	bne	lw	sw
branch	0	0	1	1	0	0
branchNot	0	0	0	0	0	0
loadImm	0	0	0	0	0	0
jumpReg	0	0	0	0	0	0
aluSrc	0	0	0	0	1	1
Jump	1	0	0	0	0	0
memRead	0	0	0	0	1	0
memToReg	0	0	0	0	1	0
regWrite	0	1	0	0	1	0
memWrite	0	0	0	0	0	1
regDst	0	0	0	0	0	0
jumpAndLink	0	1	0	0	0	0
aluOp	x	x	110	110	010	010

3- ALU



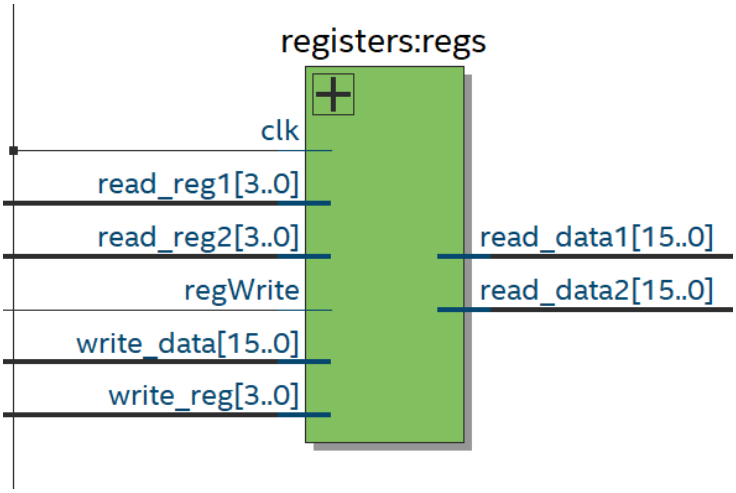


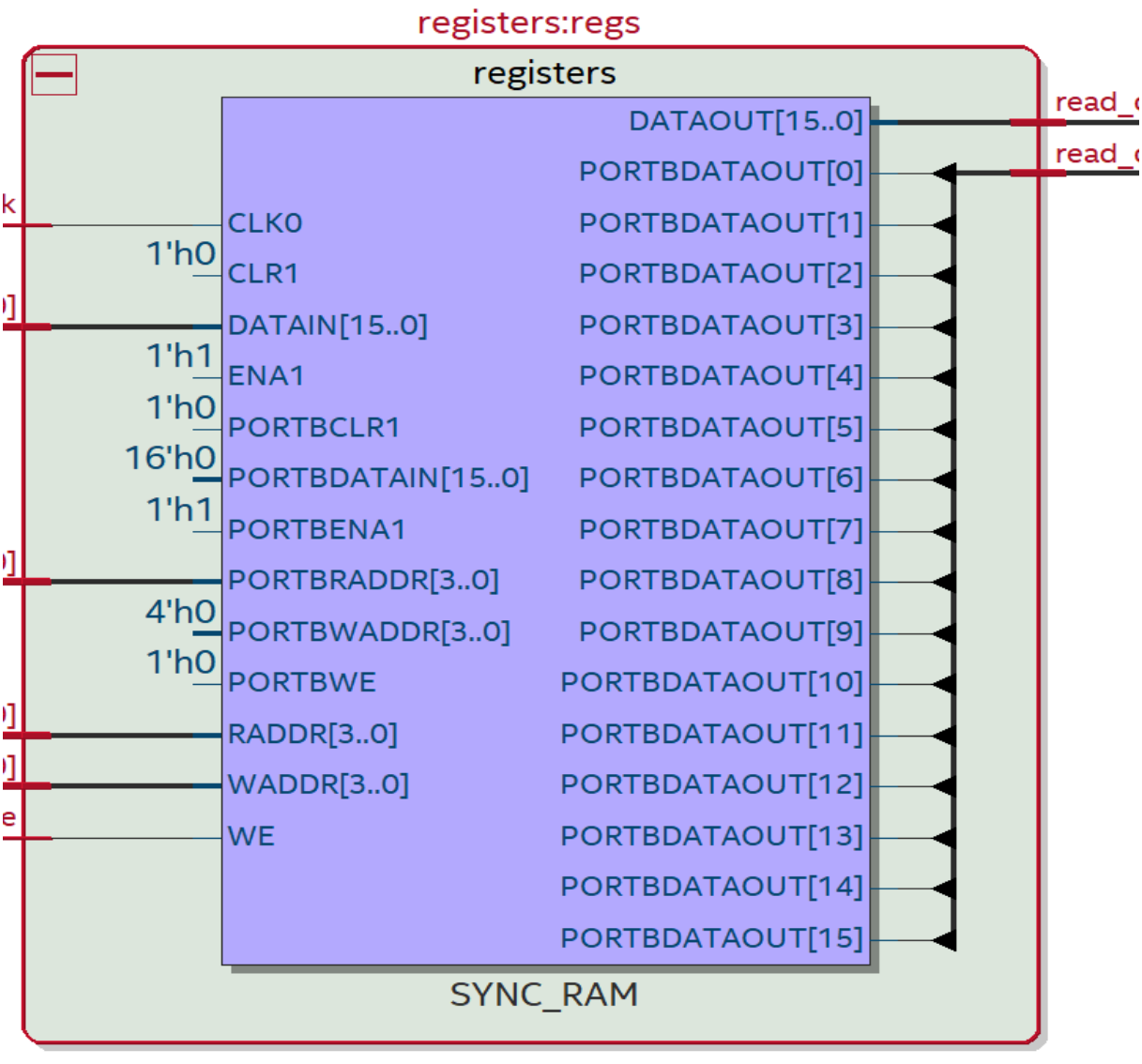
- ALU does all the operations, according to the aluop signals, and generates zero, and output signal. Multiplexers are needed to determine the output result. All results are calculated in parallel. Shift amount input is needed because I decide to do shifting in alu. Shift operations have their own aluOp, therefore shamt input is enough.

Here is the 3 bit aluOp input value for each r type instruction:

and	000
or	001
add	010
sub	110
slt	111
mult	011
sll	100
srl	101

4- Registers

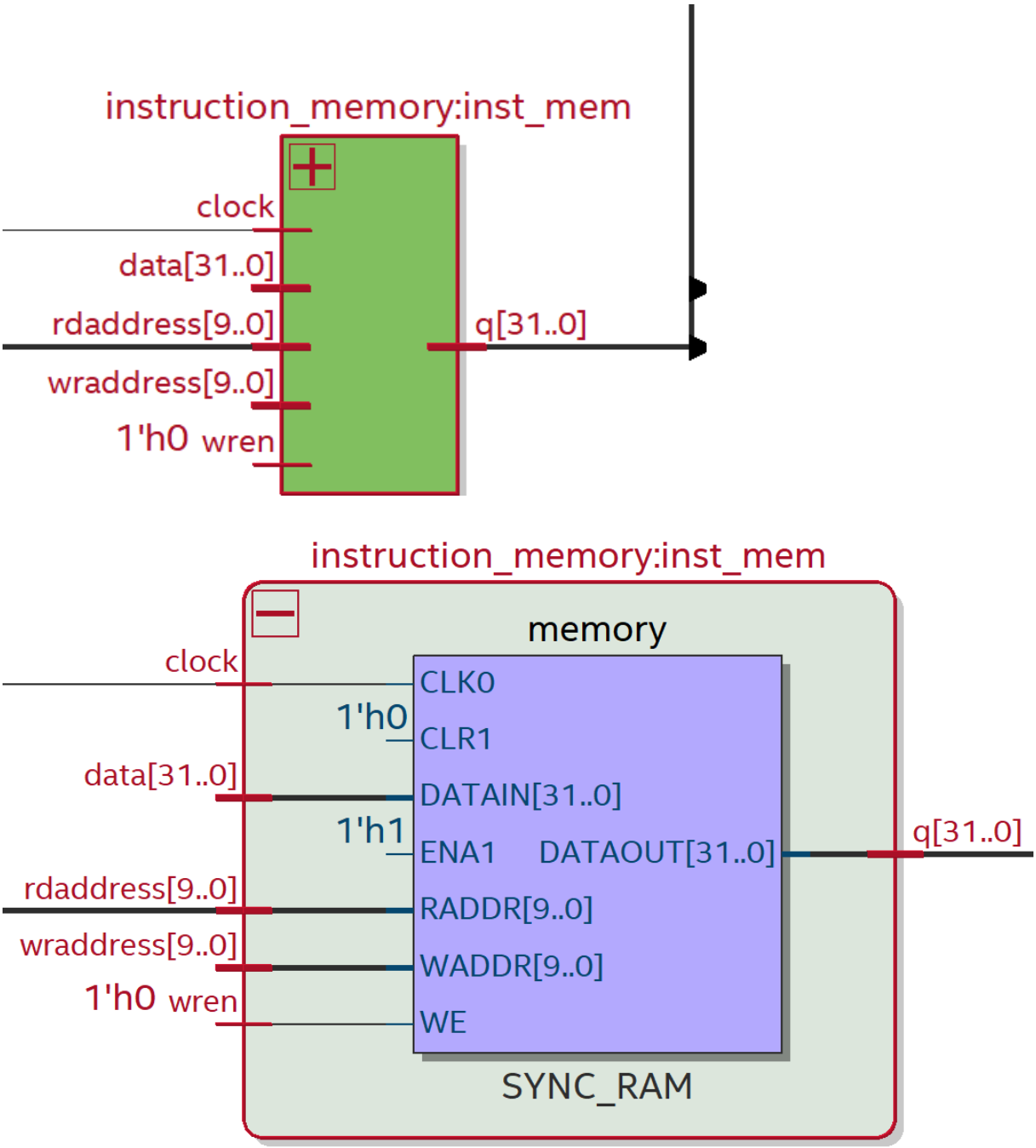

registers:regs
clk
read_reg1[3..0]
read_reg2[3..0]
regWrite
write_data[15..0]
write_reg[3..0]
read_data1[15..0]
read_data2[15..0]


registers:regs
registers
CLK0
1'h0
CLR1
DATAIN[15..0]
1'h1
ENA1
1'h0
PORTBCLR1
16'h0
PORTBDATAIN[15..0]
1'h1
PORTBENA1
PORTBRADDR[3..0]
4'h0
PORTBWADDR[3..0]
1'h0
PORTBWE
RADDR[3..0]
WADDR[3..0]
WE
DATAOUT[15..0]
PORTBDATAOUT[0]
PORTBDATAOUT[1]
PORTBDATAOUT[2]
PORTBDATAOUT[3]
PORTBDATAOUT[4]
PORTBDATAOUT[5]
PORTBDATAOUT[6]
PORTBDATAOUT[7]
PORTBDATAOUT[8]
PORTBDATAOUT[9]
PORTBDATAOUT[10]
PORTBDATAOUT[11]
PORTBDATAOUT[12]
PORTBDATAOUT[13]
PORTBDATAOUT[14]
PORTBDATAOUT[15]
SYNC_RAM

- I used register file to implement the registers, it holds 16 16 bit register inside, and it is used for read, write operations as explained in the general overview part. 0 register is initialized to 0.

5- Memory(s)

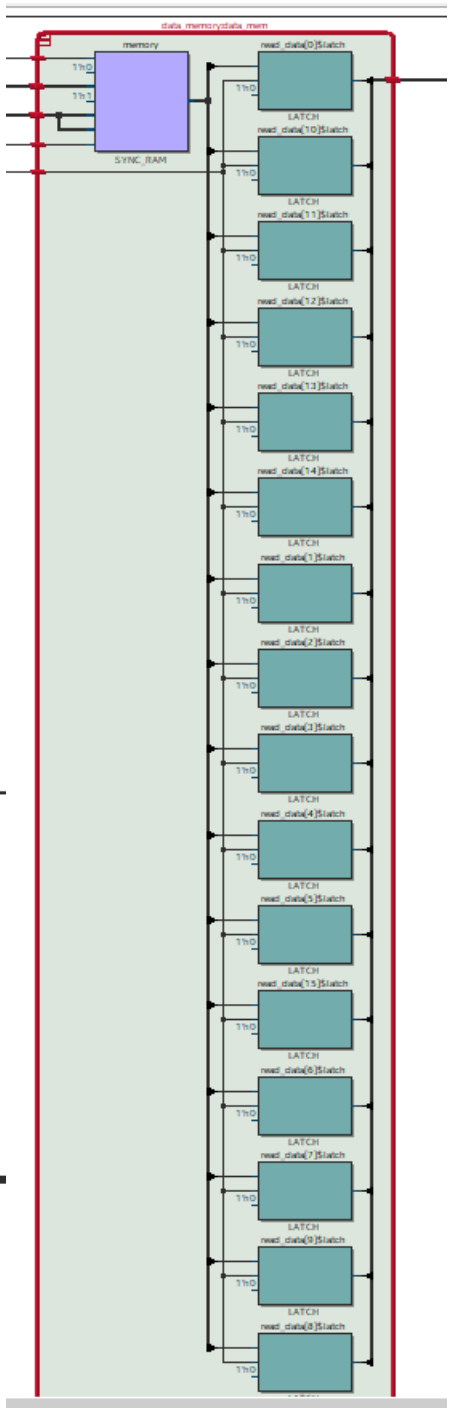
a- Instruction Memory

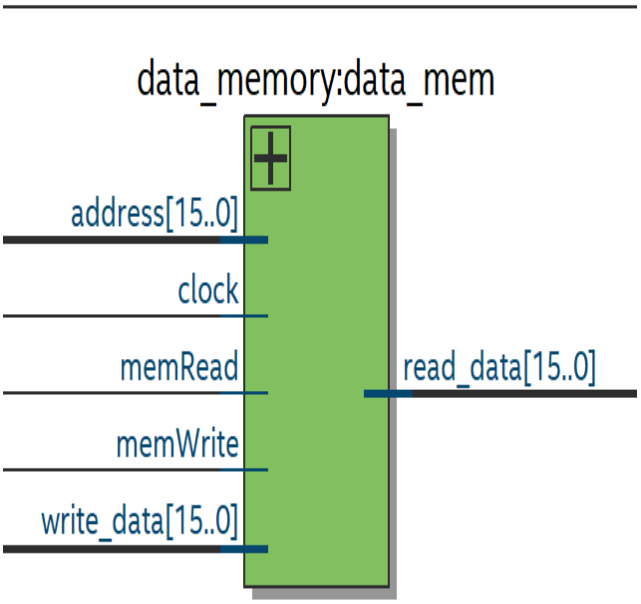


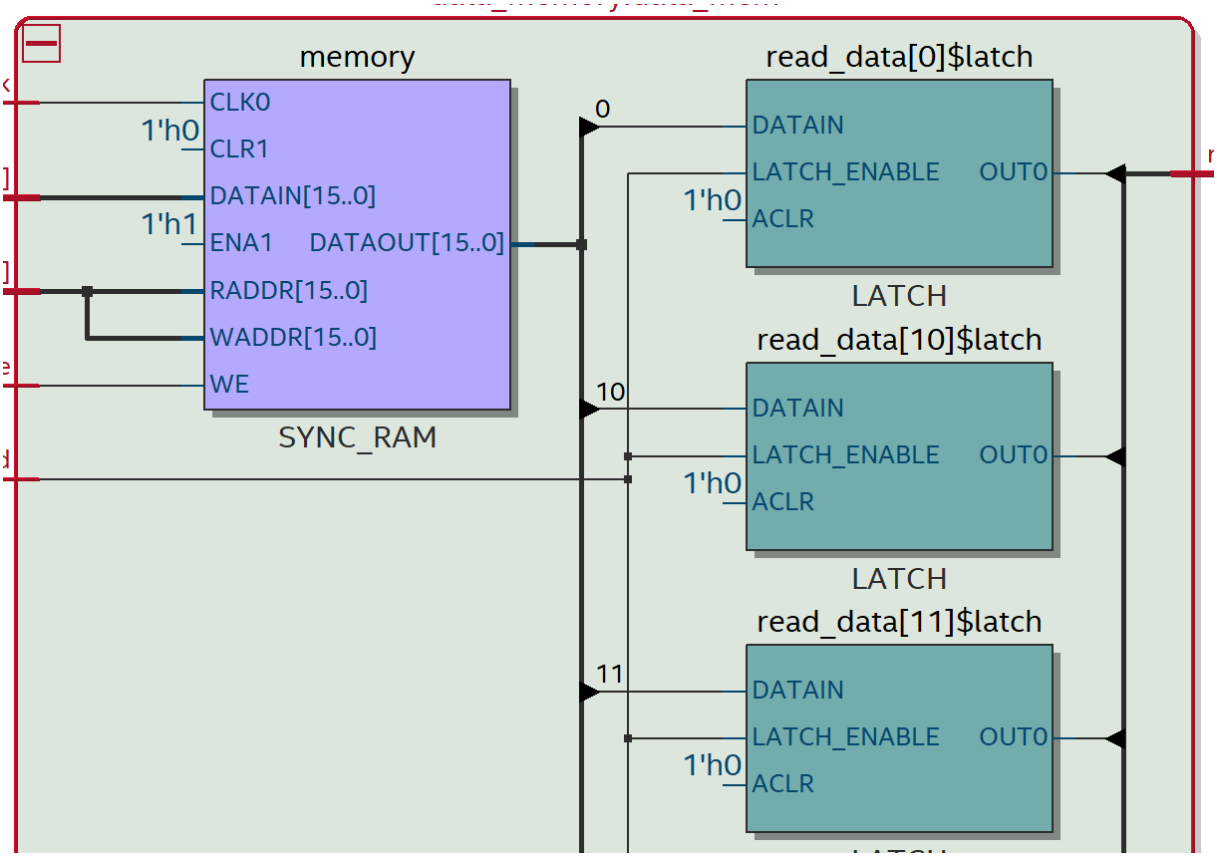
- Instruction memory holds 1024 32 bit instructions, it can be used to write instruction but I used as it is read only memory. I initalized the

instruction memory using .mem files while testing. These files include 32 bit machine codes which are instructions.

b- Data Memory





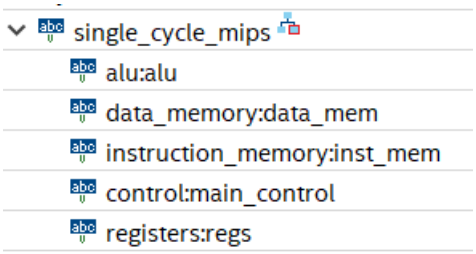


- Data memory can hold 2^16 16 bit data inside. It can be both reading and writing operation done to the data memory.

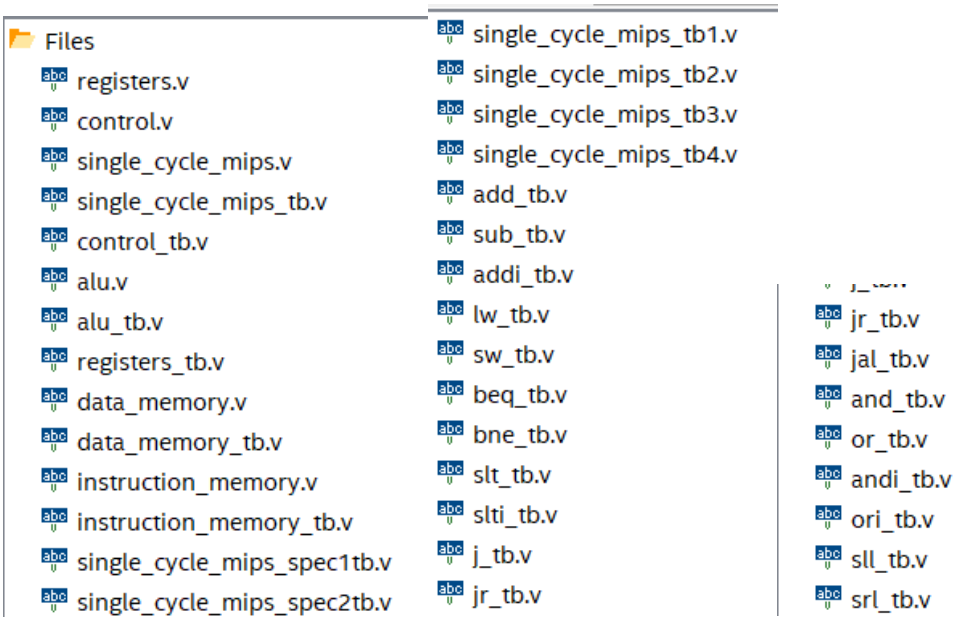
Verilog Modules and Assembler

1- Verilog Modules

Hierarchy



Files



Each module is on different Verilog file from each other, all of the test benches are tested and screenshots and explanations are on the test bench section.

2- Assembler

- I write a simple assembler in Python to convert assembly codes to machine code. It generates .mem file which can be used as an initialization file for instruction memory.
- This assembler tokenizes a simple assembly code line by line.

sample .asm file (spec1.asm)

```
1   li $3, 15
2   li $2, 256
3   sw $3, 100($2)
4   lw $1, 100($2)
5   addi $1, $1, 4
6   sw $1, 100($2)
```

run the assembler

```
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse331/homework/hw3/assembler_src$ python assembler.py
['li', '3', '15']
['li', '2', '256']
['sw', '3', '100(2)']
['lw', '1', '100(2)']
['addi', '1', '1', '4']
['sw', '1', '100(2)']
```

output file (instruction_init_spec1.mem)

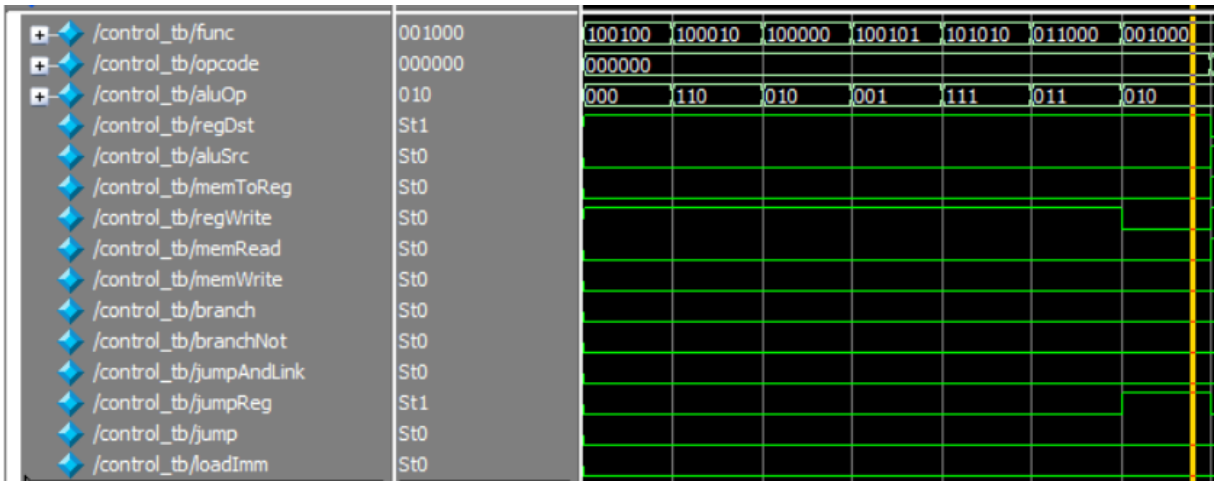
```
1  00111100000011000000000000111100
2  00111100000010000000010000000000
3  10101100100011000000000110010000
4  10001100100001000000000110010000
5  00100000010001000000000000010000
6  10101100100001000000000110010000
7
```

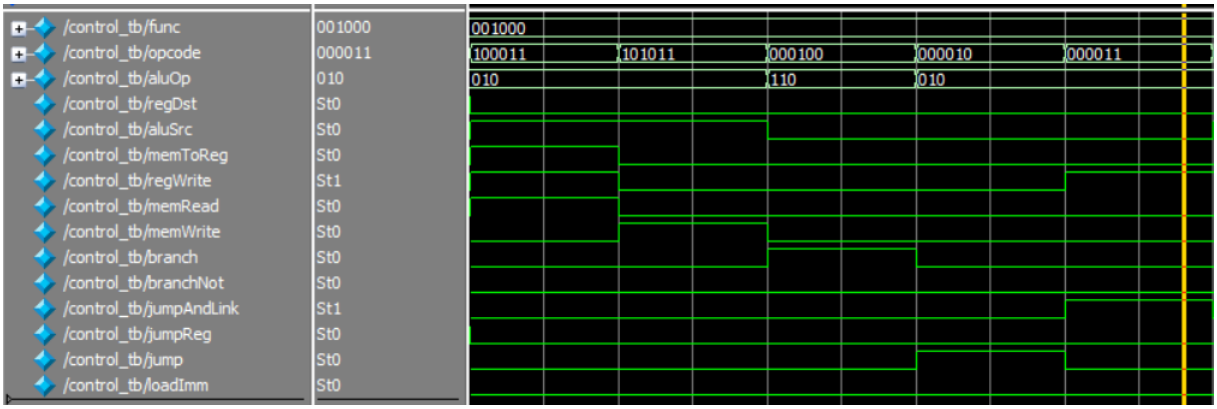
Test Bench

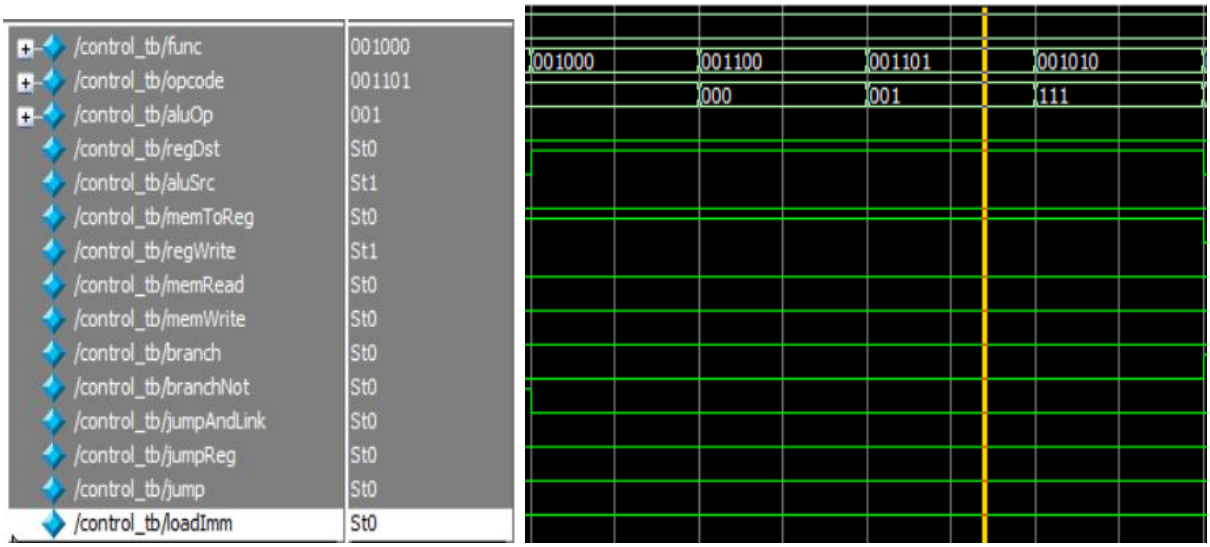
1- Sub Module Test

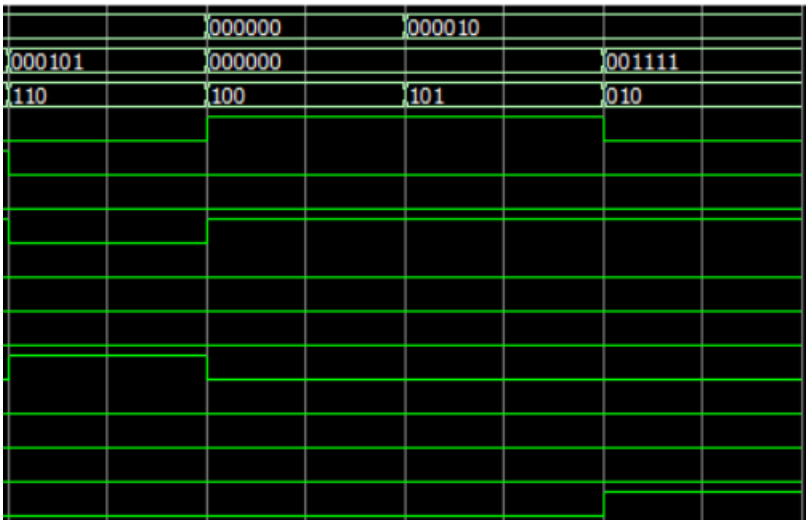
- control_tb

- Test bench for the controller. Inputs are function and opcode field, all other signals generated according to these two.

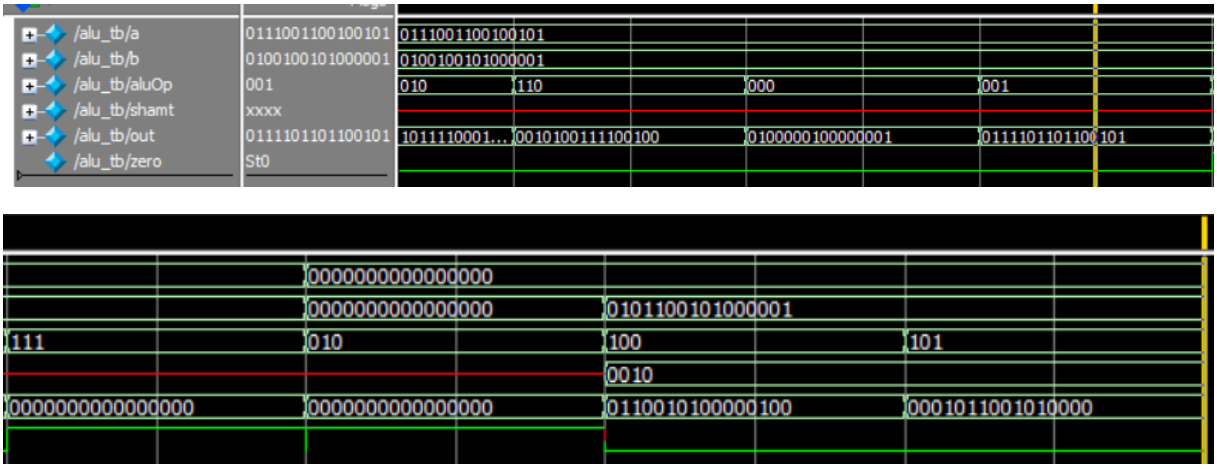




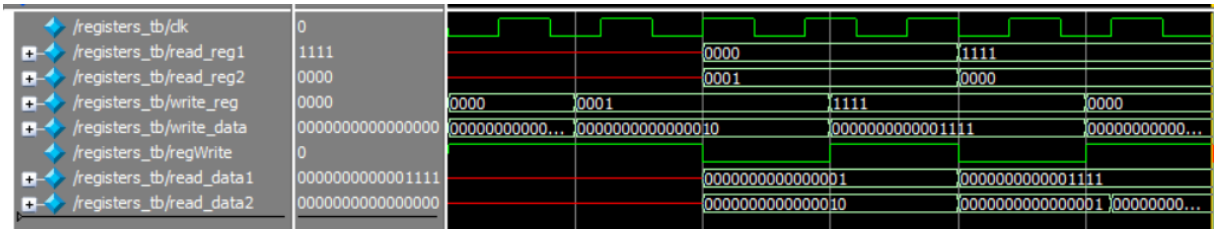




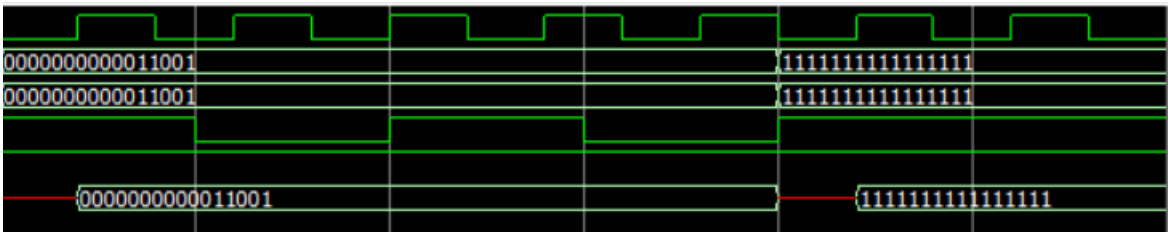
- alu_tb
- This test bench tests, all alu operations. Inputs are a, b, aluOp, shamt.



- registers_tb
- Testing register write, and read



- data_memory_tb
- Testing memory write, and read



- instruction_memory_tb
- Testing instruction initalization, all instructions in the mem file are loaded after run.

```
initial begin

    $readmemb("instruction_init.mem", inst.memory);

    // test initialization
    #50

    // write to output file
    $writememb("instructions.txt", inst.memory);
    $stop;
end

endmodule
```

Output of memory:

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/instruction_memory_tb/inst/memory
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00111100000001000000000000110000
00111100000001000000000000111100
00000000010010001110000100000000
00000000100001010000001000100000
00000000010010010100001010100000
00111100000110000000000000000000
00100000010001000000000000000100
00010001010110000000000000000100
00001000000000110000000000000000
10101100110001000000000000000000
10001100110110000000000000000000
00000001100100011100001000000000
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

2- Simple Tests For All Instructions Separately

!!!For every instruction assembly code is written, then they are generated to the machine code using assembler, after that that machine code is used as instruction initialization file in the test bench. Results waveform, display, registers, data memory(if needed).

- add_tb
  - li, add instructions are used, register 1, 2 has the data, then add them to register 3.

assembly code

```
1   li $1, 10
2   li $2, 20
3   add $3, $1, $2
```

generated machine code

```
C: > Users > burak kocausta > Desktop > cse331 > hom
1   00111100000001000000000000101000
2   00111100000001000000000000101000
3   00000000010010001110000100000000
4
```

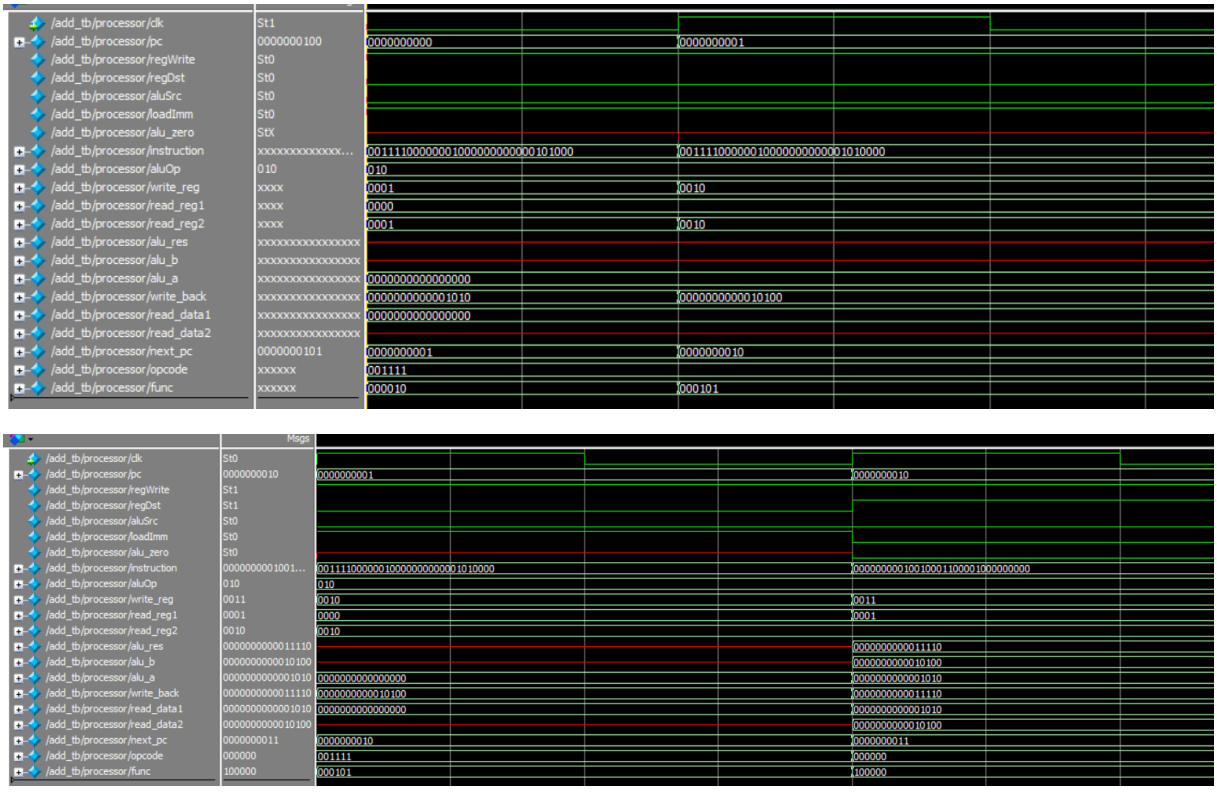
Results

Registers

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/add_tb/processor/regs/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
0000000000000000
0000000000001010
0000000000010100
0000000000011110
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
```

- register 3 has the addition result, 1 and 2 have the initalized values.

Waveform



- During li loadImm signal is 1, also regWrite is 1 to write the result to the register. For the add signal regDst must be 1 because it is r-type. write_back signal is the alu_res.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000000101000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001010
# -----

# clock = 1
# PC= 0000000001
# instruction = 00111100000001000000000000101010000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000010100
# -----

# clock = 1
# PC= 0000000010
# instruction = 00000000010010001100001000000000
# opcode = 000000
# RegDst = 1, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000001010
# alu_b = 0000000000010100
# alu_res = 0000000000011110
# alu_zero = 0
# alu_op = 010
# read_reg1 = 0001, read_reg2 = 0010, write_reg = 0011
# read_data1 = 0000000000001010
# read_data2 = 0000000000010100
# write_back = 0000000000011110
# -----
```

- sub_tb
- li is used for filling registers, sub result is saved to the register 3.

assembly machine code

```
1   li $1, 30
2   li $2, 10
3   sub $3, $1, $2
```

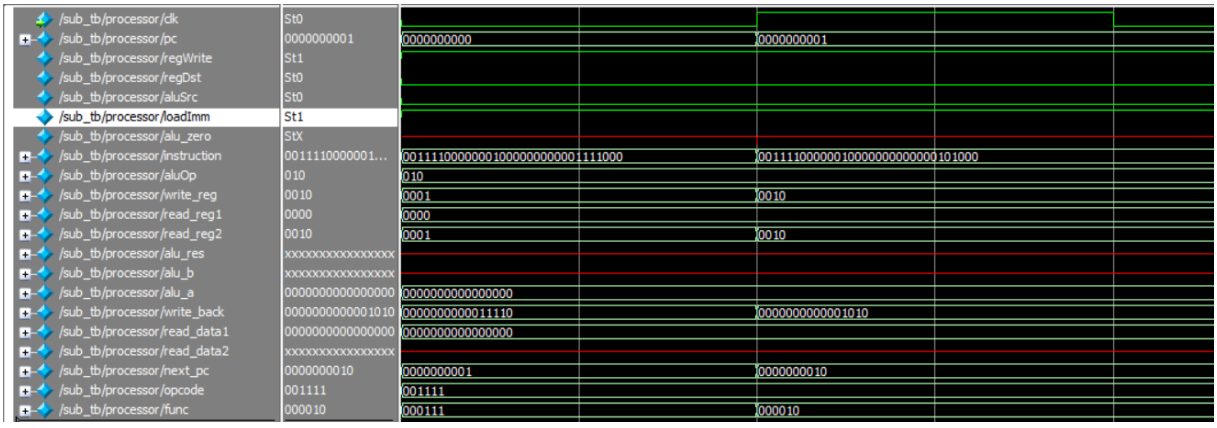
```
1   00111100000001000000000001111000
2   00111100000010000000000000101000
3   00000000010010001100001000100000
4
```

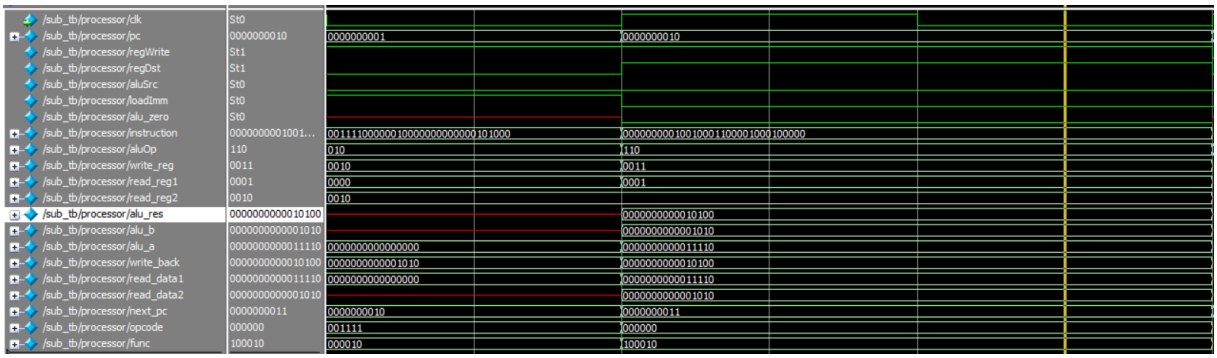
Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/sub_tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000011110
6   0000000000001010
7   0000000000010100
8   xxxxxxxxxxxxxxxx
9   xxxxxxxxxxxxxxxx
10  xxxxxxxxxxxxxxxx
11  xxxxxxxxxxxxxxxx
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
```

Waveform





- li during li operation loadImm is 1. And write_back is alu_res because sub is an r-type. ALU inputs are taken from register reads.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000001111000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000011110
# -----

# clock = 1
# PC= 0000000001
# instruction = 00111100000001000000000000101000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001010
# -----
```

```
# clock = 1
# PC= 0000000010
# instruction = 00000000010010001100001000100000
# opcode = 000000
# RegDst = 1, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000011110
# alu_b = 0000000000001010
# alu_res = 0000000000010100
# alu_zero = 0
# alu_op = 110
# read_reg1 = 0001, read_reg2 = 0010, write_reg = 0011
# read_data1 = 0000000000011110
# read_data2 = 0000000000001010
# write_back = 0000000000010100
# -----
```

- addi_tb
  - li is used for filling register 1, then value on the register 1 is used for addition with 30.

```
1   li $1, 10
2   addi $2, $1, 30
```

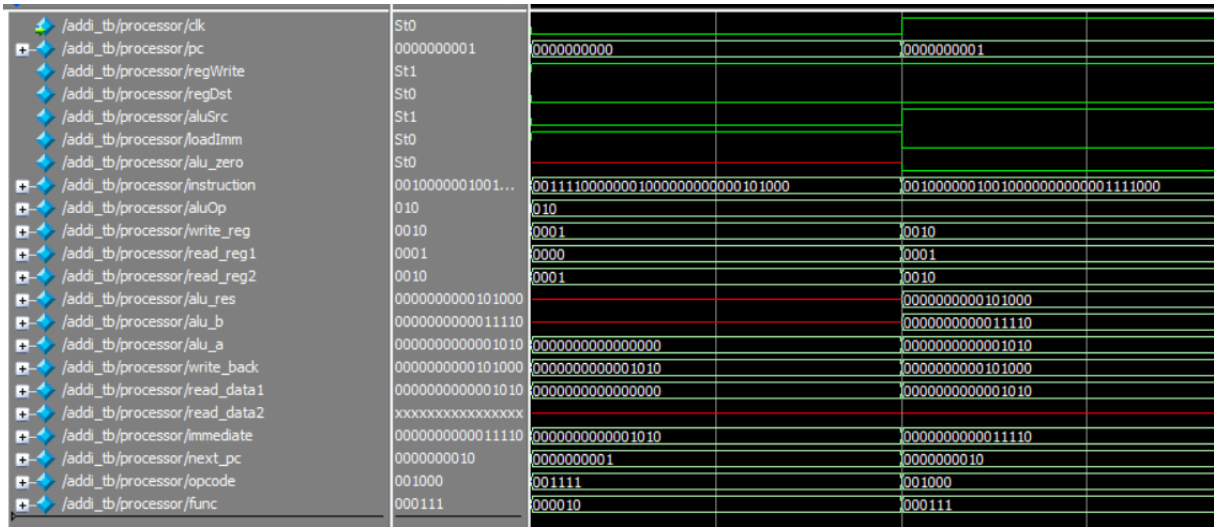
```
1   00111100000001000000000000101000
2   00100000010010000000000001111000
3
```

Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/addi_tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000001010
6   0000000000101000
7   xxxxxxxxxxxxxxxx
8   xxxxxxxxxxxxxxxx
9   xxxxxxxxxxxxxxxx
10  xxxxxxxxxxxxxxxx
11  xxxxxxxxxxxxxxxx
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
```

Waveform



- for addi, aluop is same with add operation, but regDst is 0 because addi is not an r-type instruction.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000000101000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001010
# -----
```

```
# clock = 1
# PC= 0000000001
# instruction = 00100000010010000000000001111000
# opcode = 001000
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 1, loadImm = 0
# alu_a = 0000000000001010
# alu_b = 0000000000011110
# alu_res = 0000000000101000
# alu_zero = 0
# alu_op = 010
# read_reg1 = 0001, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000001010
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000101000
# -----
```

- lw_tb

- To test load word, sw is used because data must be saved to memory to read. It is a simple test for start, it reads the same location which is written by sw.

```
li $1, 12
sw $1, 4($0)
lw $2, 4($0)
```

```
00111100000001000000000000110000
10101100000001000000000000010000
10001100000010000000000000010000
```

Results

Memory

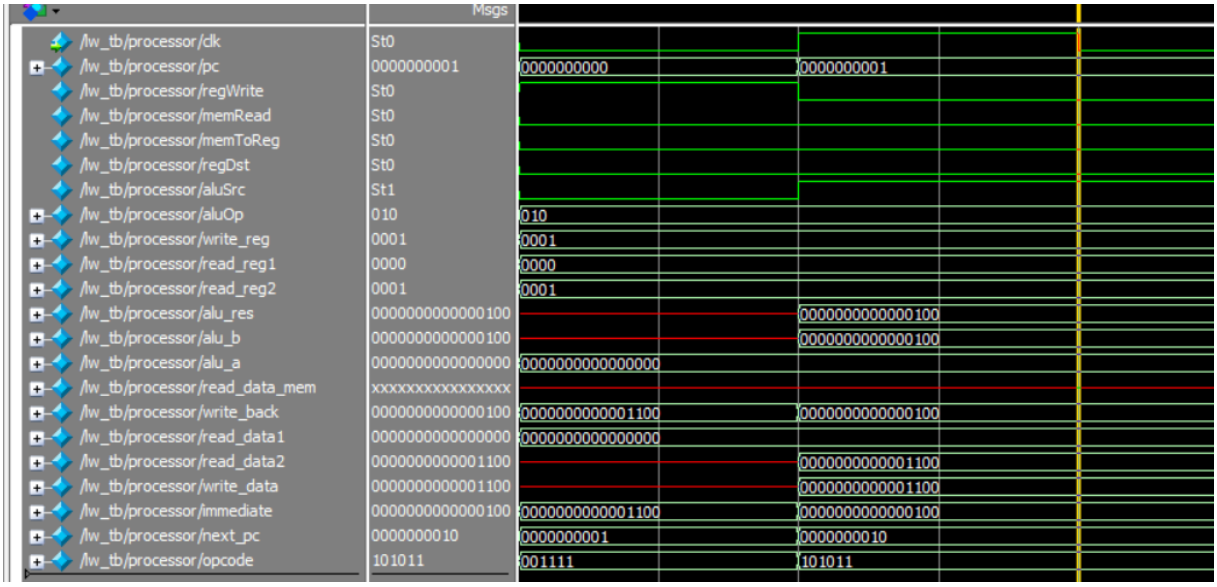
memory after sw

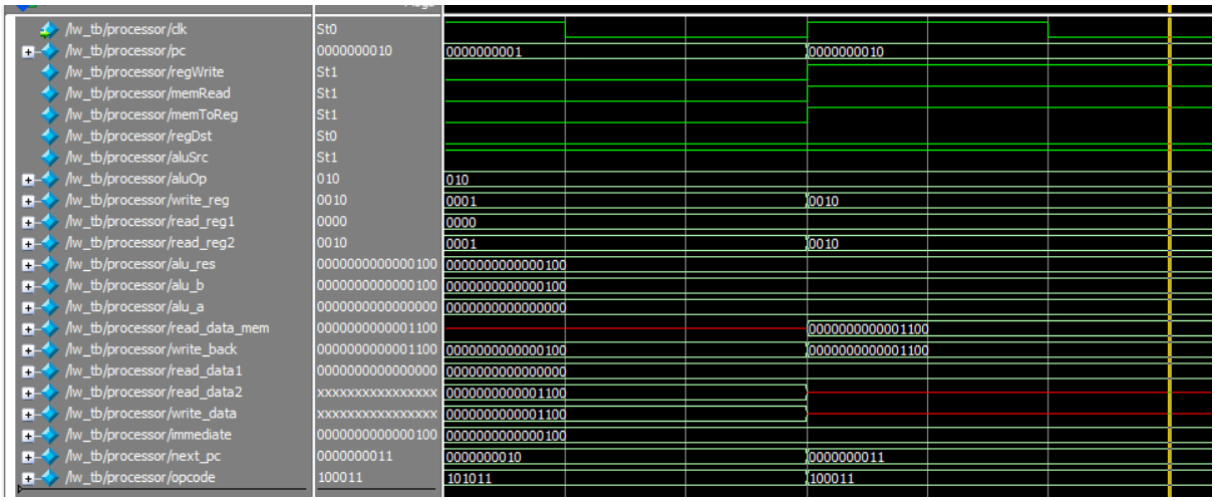
```
// memory data file (do not edit the following line - required for mem load use)
// instance=/lw_tb/processor/data_mem/memory
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
0000000000001100
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
```

Registers

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/lw_tb/processor/regs/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
0000000000000000
0000000000001100
0000000000001100
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
```

Waveform





- During lw memRead, regWrite, memToReg signals are 1. During sw memWrite is 1 as expected. I forgot to add memWrite to the waveform but it can be seen on the display(2nd screenshot). I added the memWrite signal on the sw instruction test below for the waveform also.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000000110000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxx
# write_back = 0000000000001100
# read_data_mem = xxxxxxxxxxxxxxx
# write_data_mem = xxxxxxxxxxxxxxx
# address = xxxxxxxxxxxxxxx
# memRead = 0, memWrite = 0
# -----

#
# clock = 1
# PC= 0000000001
# instruction = 10101100000001000000000000010000
# opcode = 101011
# RegDst = 0, RegWrite = 0, MemWrite = 1, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 1, loadImm = 0
# alu_a = 0000000000000000
# alu_b = 0000000000000100
# alu_res = 0000000000000100
# alu_zero = 0
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = 0000000000001100
# write_back = 0000000000000100
# read_data_mem = xxxxxxxxxxxxxxx
# write_data_mem = 0000000000001100
# address = 0000000000000100
# memRead = 0, memWrite = 1
# -----
```

```
# clock = 1
# PC= 0000000010
# instruction = 10001100000010000000000000010000
# opcode = 100011
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 1, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 1, loadImm = 0
# alu_a = 0000000000000000
# alu_b = 0000000000000100
# alu_res = 0000000000000100
# alu_zero = 0
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001100
# read_data_mem = 0000000000001100
# write_data_mem = xxxxxxxxxxxxxxxx
# address = 0000000000000100
# memRead = 1, memWrite = 0
# -----
```

- sw_tb

- sw test, it uses 10 as an offset.

```
1   li $1, 10
2   li $2, 15
3   sw $1, 10($2)
```

```
1   00111100000001000000000000101000
2   00111100000001000000000000111100
3   10101100100001000000000000101000
```

Results

Memory

(10+15 = 25) (it is on the 29th line because first memory cell is on the 4th line in the memory output txt file.

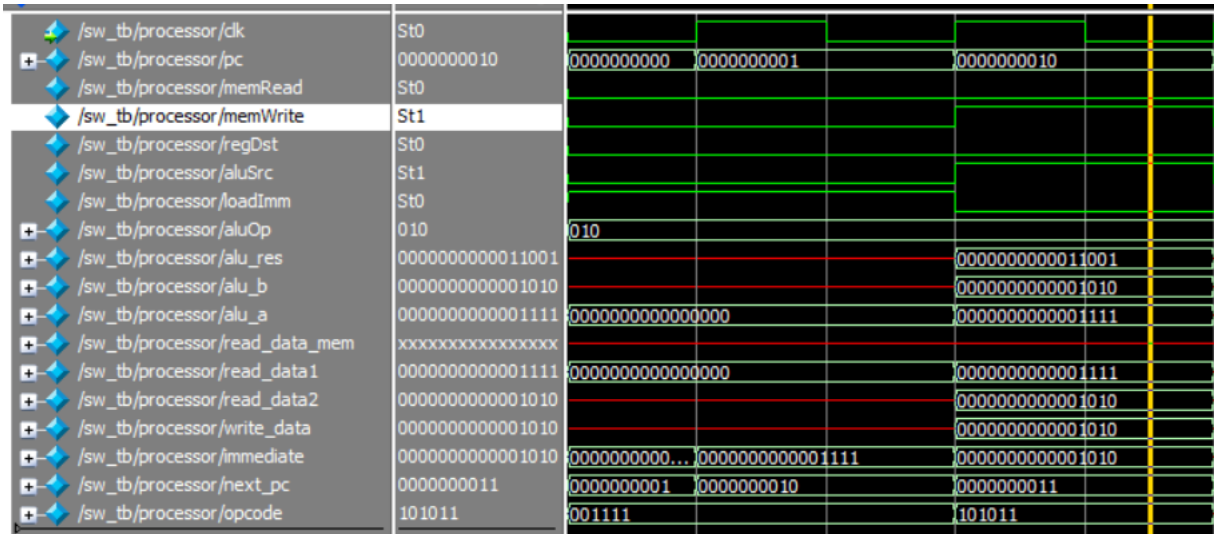
```
20   xxxxxxxxxxxxxxxx
21   xxxxxxxxxxxxxxxx
22   xxxxxxxxxxxxxxxx
23   xxxxxxxxxxxxxxxx
24   xxxxxxxxxxxxxxxx
25   xxxxxxxxxxxxxxxx
26   xxxxxxxxxxxxxxxx
27   xxxxxxxxxxxxxxxx
28   xxxxxxxxxxxxxxxx
29   0000000000001010
30   xxxxxxxxxxxxxxxx
31   xxxxxxxxxxxxxxxx
32   xxxxxxxxxxxxxxxx
33   xxxxxxxxxxxxxxxx
34   xxxxxxxxxxxxxxxx
35   xxxxxxxxxxxxxxxx
36   xxxxxxxxxxxxxxxx
37   xxxxxxxxxxxxxxxx
38   xxxxxxxxxxxxxxxx
39   xxxxxxxxxxxxxxxx
```

(29 – 4 = 25) 25th cell

Registers

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/sw_tb/processor/regs/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
0000000000000000
0000000000001010
0000000000001111
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
```

Waveform



- memWrite is 1 for the store word instruction.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000000101000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001010
# read_data_mem = xxxxxxxxxxxxxxxx
# write_data_mem = xxxxxxxxxxxxxxxx
# address = xxxxxxxxxxxxxxxx
# memRead = 0, memWrite = 0
# -----
```

```
# clock = 1
# PC= 0000000001
# instruction = 00111100000010000000000000111100
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001111
# read_data_mem = xxxxxxxxxxxxxxxx
# write_data_mem = xxxxxxxxxxxxxxxx
# address = xxxxxxxxxxxxxxxx
# memRead = 0, memWrite = 0
# -----
# clock = 1
# PC= 0000000010
# instruction = 10101100100001000000000000101000
# opcode = 101011
# RegDst = 0, RegWrite = 0, MemWrite = 1, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 1, loadImm = 0
# alu_a = 0000000000001111
# alu_b = 0000000000001010
# alu_res = 0000000000011001
# alu_zero = 0
# alu_op = 010
# read_reg1 = 0010, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000001111
# read_data2 = 0000000000001010
# write_back = 0000000000011001
# read_data_mem = xxxxxxxxxxxxxxxx
# write_data_mem = 0000000000001010
# address = 0000000000011001
# memRead = 0, memWrite = 1
# -----
```

- beq_tb
  - Values are equal, so only 6th line must be executed. $5 must be filled.

```
1   li $1, 10
2   li $2, 10
3   beq $1, $2, 2
4   add $3, $1, $2
5   sub $4, $1, $2
6   addi $5, $1, 10
```

C: > Users > burak kocausta > Desktop > cse331 > home

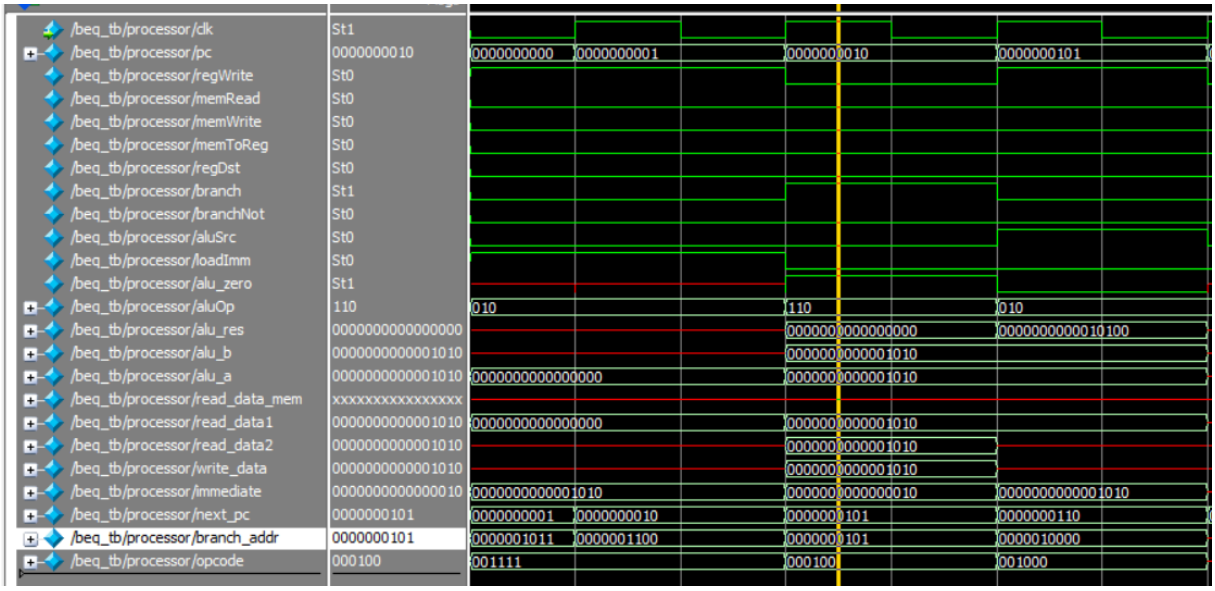
```
1   00111100000001000000000000101000
2   00111100000010000000000000101000
3   00010000010010000000000000001000
4   00000000010010001100001000000000
5   00000000010010010000001000100000
6   00100000010101000000000000101000
7
```

Results

Registers

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/beq_tb/processor/regs/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
0000000000000000
0000000000001010
0000000000001010
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
0000000000010100
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
```

Waveform



- Observing the change on program counter means branching is successful, also branch address can be observed. Also branch signal is 1 when beq instruction comes. It goes to and gate with alu_zero if result is 1 branch will be done.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000000101000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxx
# write_back = 0000000000001010
# branch address = 0000001011
# -----

# clock = 1
# PC= 0000000001
# instruction = 00111100000001000000000000101000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxx
# write_back = 0000000000001010
# branch address = 0000001100
# -----

# clock = 1
# PC= 0000000010
# instruction = 00010000010010000000000000001000
# opcode = 000100
# RegDst = 0, RegWrite = 0, MemWrite = 0, memToReg = 0, branch = 1
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000001010
# alu_b = 0000000000001010
# alu_res = 0000000000000000
# alu_zero = 1
# alu_op = 110
# read_reg1 = 0001, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000001010
# read_data2 = 0000000000001010
# write_back = 0000000000000000
# branch address = 0000000101
# -----
```

```
#
# clock = 1
# PC= 0000000101
# instruction = 00100000010101000000000000101000
# opcode = 001000
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 1, loadImm = 0
# alu_a = 0000000000001010
# alu_b = 0000000000001010
# alu_res = 0000000000010100
# alu_zero = 0
# alu_op = 010
# read_reg1 = 0001, read_reg2 = 0101, write_reg = 0101
# read_data1 = 0000000000001010
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000010100
# branch address = 0000010000
# -----
```

- bne_tb

- It is very similar test like beq_tb.

```
1   li $1, 10
2   li $2, 12
3   bne $1, $2, 2
4   add $3, $1, $2
5   sub $4, $1, $2
6   addi $5, $1, 10
```

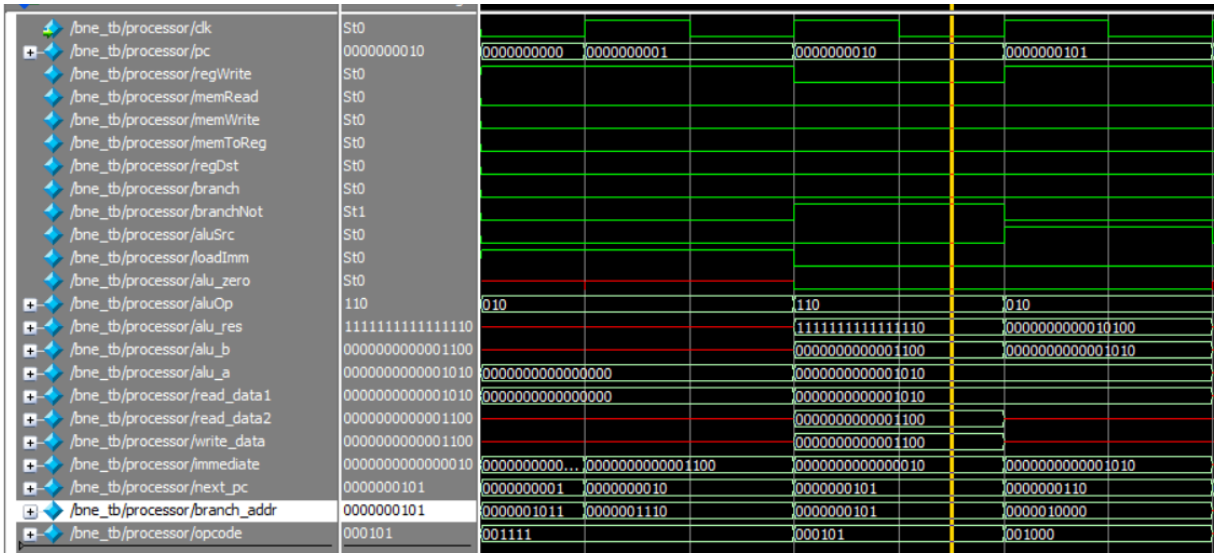
```
1   00111100000001000000000000101000
2   00111100000010000000000000110000
3   00010100010010000000000000001000
4   00000000010010001100001000000000
5   00000000010010010000001000100000
6   00100000010101000000000000101000
```

Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/bne_tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000001010
6   0000000000001100
7   xxxxxxxxxxxxxxxx
8   xxxxxxxxxxxxxxxx
9   0000000000010100
10  xxxxxxxxxxxxxxxx
11  xxxxxxxxxxxxxxxx
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
```

Waveform



- In this case branchNot signal is 1. Program counter is setted to the branch addr calculation.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000000101000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001010
# branch address = 0000001011
# -----
```

```
# clock = 1
# PC= 0000000001
# instruction = 00111100000010000000000000110000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001100
# branch address = 0000001110
# -----
```

```
# clock = 1
# PC= 0000000010
# instruction = 00010100010010000000000000001000
# opcode = 000101
# RegDst = 0, RegWrite = 0, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 1, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000001010
# alu_b = 0000000000001100
# alu_res = 1111111111111110
# alu_zero = 0
# alu_op = 110
# read_reg1 = 0001, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000001010
# read_data2 = 0000000000001100
# write_back = 1111111111111110
# branch address = 0000000101
# -----
```

```
# clock = 1
# PC= 0000000101
# instruction = 00100000010101000000000000101000
# opcode = 001000
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 1, loadImm = 0
# alu_a = 0000000000001010
# alu_b = 0000000000001010
# alu_res = 0000000000010100
# alu_zero = 0
# alu_op = 010
# read_reg1 = 0001, read_reg2 = 0101, write_reg = 0101
# read_data1 = 0000000000001010
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000010100
# branch address = 0000010000
# -----
```

- slt_tb
- set on less than instruction to compare 10 < 20 and 20 < 10.

```
1   li $1, 10
2   li $2, 20
3   slt $3, $1, $2
4   slt $4, $2, $1
```

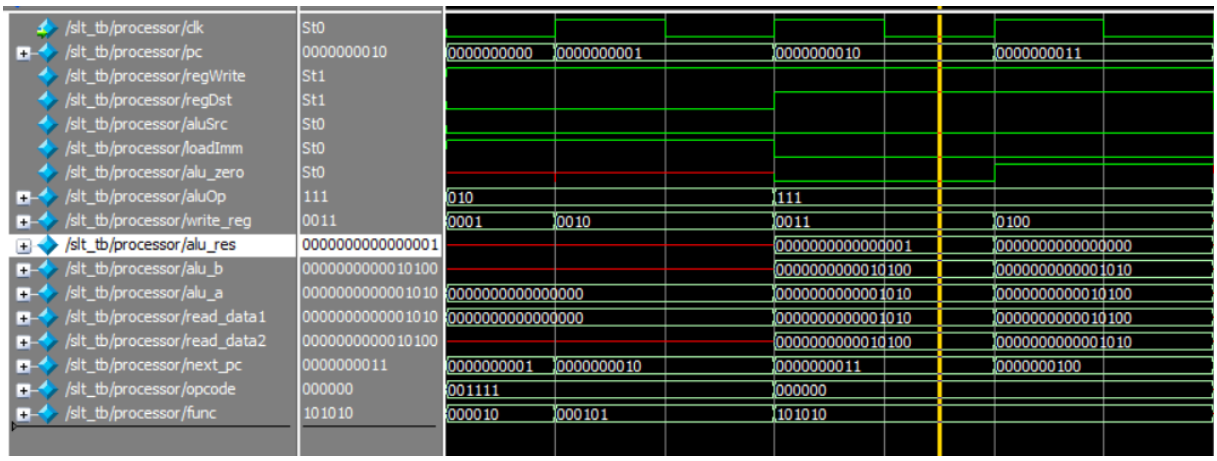
```
1   00111100000001000000000000101000
2   00111100000010000000000001010000
3   00000000010010001100001010100000
4   00000000010000101000000101010000
5
```

Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/slt_tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000001010
6   0000000000010100
7   0000000000000001
8   0000000000000000
9   xxxxxxxxxxxxxxxx
10  xxxxxxxxxxxxxxxx
11  xxxxxxxxxxxxxxxx
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
```

Waveform



- slt is an r-type instruction so regDst must be 1. ALU inputs are read from registers, and result is determined with write_back signal.

Display

```
#
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000000101000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001010
# -----
#
# clock = 1
# PC= 0000000001
# instruction = 00111100000001000000000000101000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000010100
# -----
#
# clock = 1
# PC= 0000000010
# instruction = 00000000010010001100001010100000
# opcode = 000000
# RegDst = 1, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000001010
# alu_b = 0000000000010100
# alu_res = 0000000000000001
# alu_zero = 0
# alu_op = 111
# read_reg1 = 0001, read_reg2 = 0010, write_reg = 0011
# read_data1 = 0000000000001010
# read_data2 = 0000000000010100
# write_back = 0000000000000001
# -----
```

```
#
# clock = 1
# PC= 0000000011
# instruction = 00000000100001010000001010100000
# opcode = 000000
# RegDst = 1, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000010100
# alu_b = 0000000000001010
# alu_res = 0000000000000000
# alu_zero = 1
# alu_op = 111
# read_reg1 = 0010, read_reg2 = 0001, write_reg = 0100
# read_data1 = 0000000000010100
# read_data2 = 0000000000001010
# write_back = 0000000000000000
# -----
```

- slti_tb
- It is an immediate version of set less than.

```
1   li $1, 10
2   slti $3, $1, 20
3   slti $4, $1, 5
```

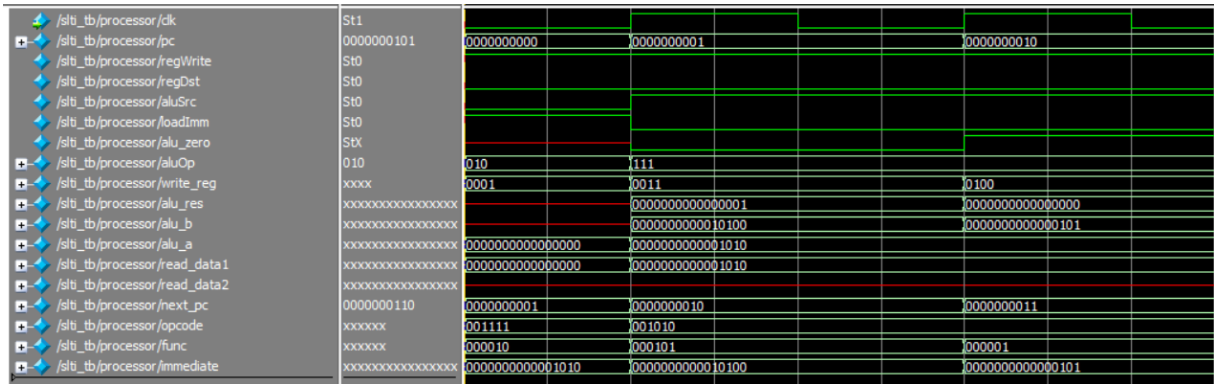
```
1   00111100000001000000000000101000
2   00101000010011000000000001010000
3   00101000010100000000000000010100
```

Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/slti_tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000001010
6   xxxxxxxxxxxxxxxx
7   0000000000000001
8   0000000000000000
9   xxxxxxxxxxxxxxxx
10  xxxxxxxxxxxxxxxx
11  xxxxxxxxxxxxxxxx
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
20
```

Waveform



- Because of not being r-type regDst is 0 immediate field is send to the alu for comparison.

Display

```
clock = 0
PC= 0000000000
instruction = 00111100000001000000000000101000
opcode = 001111
RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
alu_a = 0000000000000000
alu_b = xxxxxxxxxxxxxxxx
alu_res = xxxxxxxxxxxxxxxx
alu_zero = x
alu_op = 010
read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
read_data1 = 0000000000000000
read_data2 = xxxxxxxxxxxxxxxx
write_back = 0000000000001010
-----
```

```
# clock = 1
# PC= 0000000001
# instruction = 00101000010011000000000001010000
# opcode = 001010
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 1, loadImm = 0
# alu_a = 0000000000001010
# alu_b = 0000000000010100
# alu_res = 0000000000000001
# alu_zero = 0
# alu_op = 111
# read_reg1 = 0001, read_reg2 = 0011, write_reg = 0011
# read_data1 = 0000000000001010
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000000001
# -----
```

```
clock = 1
PC= 0000000010
instruction = 00101000010100000000000000010100
opcode = 001010
RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 1, loadImm = 0
alu_a = 0000000000001010
alu_b = 0000000000000101
alu_res = 0000000000000000
alu_zero = 1
alu_op = 111
read_reg1 = 0001, read_reg2 = 0100, write_reg = 0100
read_data1 = 0000000000001010
read_data2 = xxxxxxxxxxxxxxxx
write_back = 0000000000000000
-----
```

- j_tb
  - Simple jump test, it must be jump to the 6th line in this code.

```
1   li $1, 15
2   li $2, 10
3   j 5
4   addi $1, $1, 2
5   addi $2, $1, 3
6   add $3, $1, $2
```

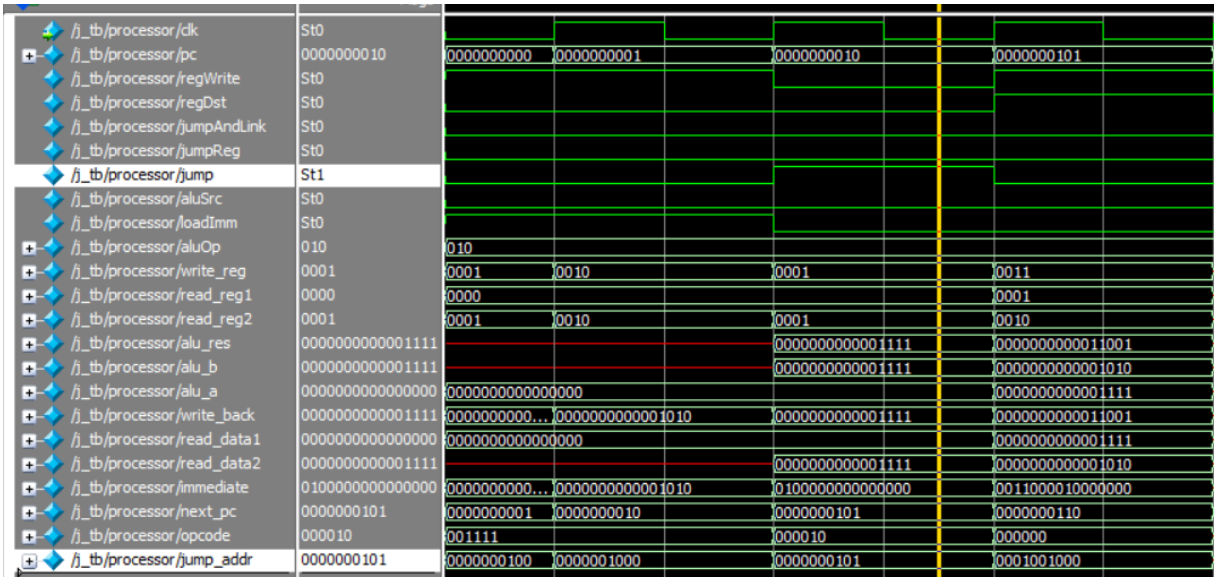
```
1   00111100000001000000000000111100
2   00111100000010000000000000101000
3   00001000000001010000000000000000
4   00100000100010000000000000001000
5   00100000100100000000000000001100
6   00000000010010001100001000000000
7
```

Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/j_tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000001111
6   0000000000001010
7   0000000000011001
8   xxxxxxxxxxxxxxxx
9   xxxxxxxxxxxxxxxx
10  xxxxxxxxxxxxxxxx
11  xxxxxxxxxxxxxxxx
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
20
```

Waveform



- It can be seen that jump signal is 1 when jump instruction comes, also jump address becomes the next pc.

Display

```
#
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000000111100
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001111
# jump address = 0000000100
# -----
```

```
# clock = 1
# PC= 0000000001
# instruction = 00111100000010000000000000101000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001010
# jump address = 0000001000
# -----

# clock = 1
# PC= 0000000010
# instruction = 00001000000001010000000000000000
# opcode = 000010
# RegDst = 0, RegWrite = 0, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 1, aluSrc = 0, loadImm = 0
# alu_a = 0000000000000000
# alu_b = 0000000000001111
# alu_res = 0000000000001111
# alu_zero = 0
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = 0000000000001111
# write_back = 0000000000001111
# jump address = 0000000101
# -----

# clock = 1
# PC= 0000000101
# instruction = 00000000010010001100001000000000
# opcode = 000000
# RegDst = 1, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000001111
# alu_b = 0000000000001010
# alu_res = 0000000000011001
# alu_zero = 0
# alu_op = 010
# read_reg1 = 0001, read_reg2 = 0010, write_reg = 0011
# read_data1 = 0000000000001111
# read_data2 = 0000000000001010
# write_back = 0000000000011001
# jump address = 0001001000
# -----
```

- jr_tb
- It must be directly jump to the line 5.

```
1   li $1, 15
2   li $2, 4
3   jr $2
4   add $3, $2, $1
5   add $4, $2, $1
```

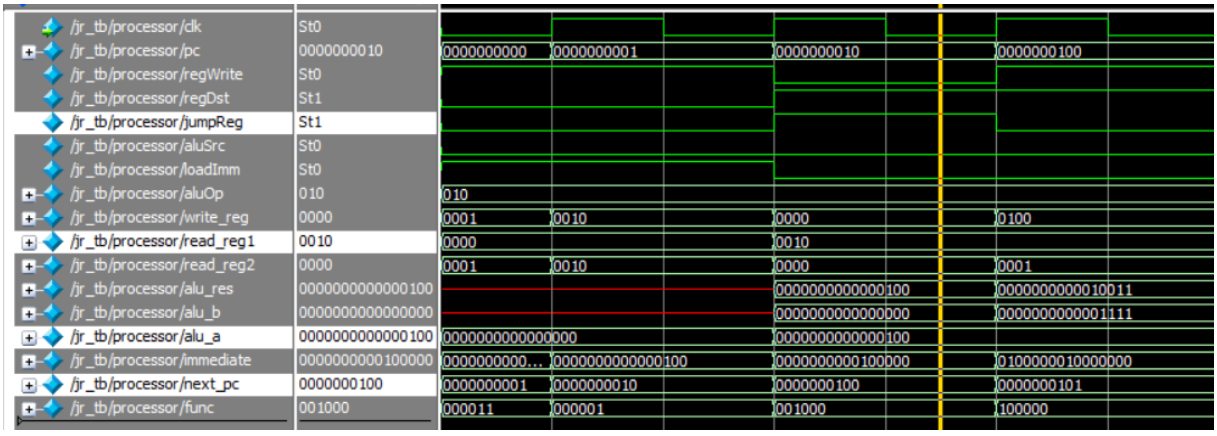
```
1   00111100000001000000000000111100
2   00111100000010000000000000010000
3   00000000100000000000000010000000
4   00000000100001001100001000000000
5   00000000100001010000001000000000
```

Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/jr_tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000001111
6   0000000000000100
7   xxxxxxxxxxxxxxxx
8   0000000000010011
9   xxxxxxxxxxxxxxxx
10  xxxxxxxxxxxxxxxx
11  xxxxxxxxxxxxxxxx
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
20
```

Waveform



- When jr instruction comes, jumpReg signal becomes 1, then next pc is set to the value of the read reg1 result.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000000111100
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001111
# next PC(jump address) = 0000000001
# -----
# clock = 1
# PC= 0000000001
# instruction = 00111100000001000000000000010000
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000000100
# next PC(jump address) = 0000000010
# -----
# clock = 1
# PC= 0000000010
# instruction = 00000000100000000000000010000000
# opcode = 000000
# RegDst = 1, RegWrite = 0, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 1, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000000100
# alu_b = 0000000000000000
# alu_res = 0000000000000100
# alu_zero = 0
# alu_op = 010
# read_reg1 = 0010, read_reg2 = 0000, write_reg = 0000
# read_data1 = 0000000000000100
# read_data2 = 0000000000000000
# write_back = 0000000000000100
# next PC(jump address) = 0000000100
# -----
```

```
clock = 1
PC= 0000000100
instruction = 00000000100001010000001000000000
opcode = 000000
RegDst = 1, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
alu_a = 0000000000000100
alu_b = 0000000000001111
alu_res = 0000000000010011
alu_zero = 0
alu_op = 010
read_reg1 = 0010, read_reg2 = 0001, write_reg = 0100
read_data1 = 0000000000000100
read_data2 = 0000000000001111
write_back = 0000000000010011
next PC(jump address) = 0000000101
-----
```

- jal_tb

- jal is tested with using jr, and beq instructions. It directly jumps to the line 6 than goes up and down again. Line 4, 5 must not be executed.

```
1   li $1, 15
2   jal 5
3   beq $4, $1, 4
4   add $2, $1, $1
5   add $3, $2, $1
6   addi $4, $1, 0
7   jr $15
8   addi $5, $1, 0
```

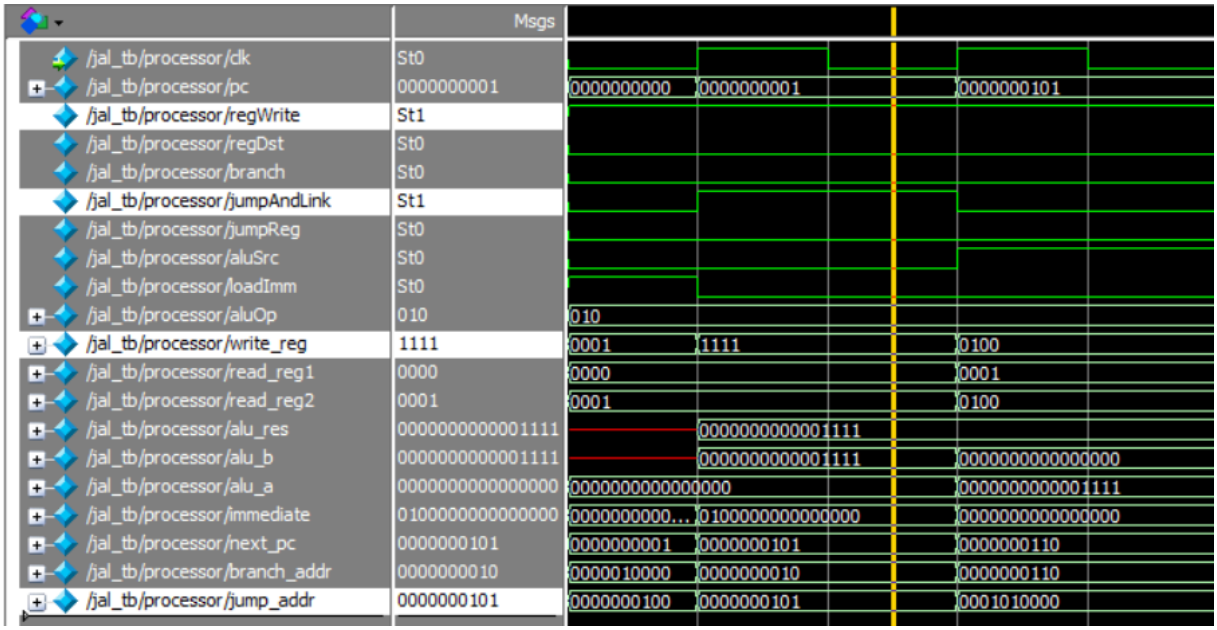
```
1   00111100000001000000000000111100
2   00001100000000101000000000000000
3   00010001000001000000000000010000
4   00000000010001001000001000000000
5   00000000100001001100001000000000
6   00100000101000000000000000000000
7   00000011110000000000000010000000
8   00100000101010000000000000000000
```

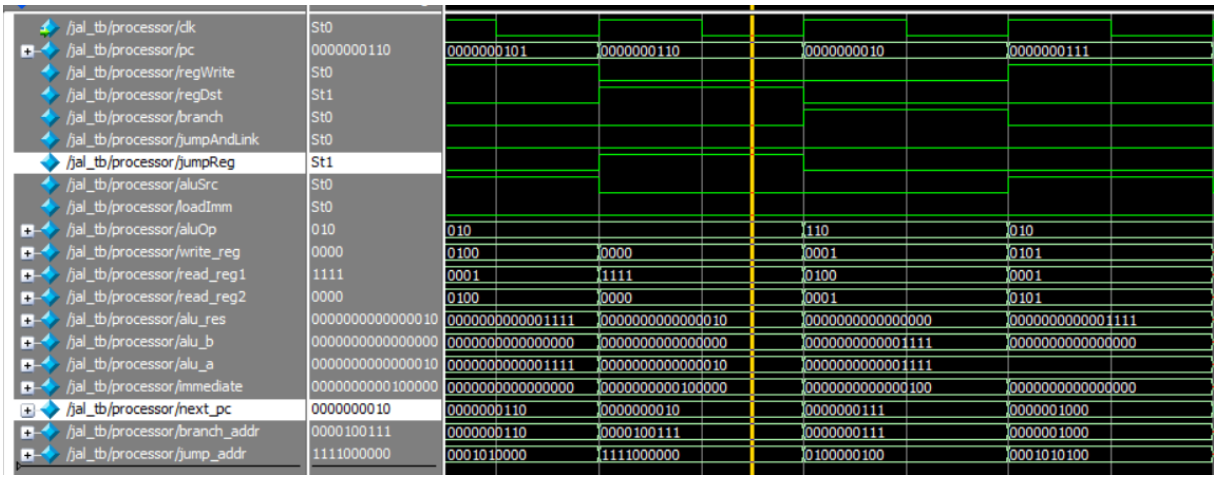
Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/jal_tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000001111
6   xxxxxxxxxxxxxxxx
7   xxxxxxxxxxxxxxxx
8   0000000000001111
9   0000000000001111
0   xxxxxxxxxxxxxxxx
1   xxxxxxxxxxxxxxxx
2   xxxxxxxxxxxxxxxx
3   xxxxxxxxxxxxxxxx
4   xxxxxxxxxxxxxxxx
5   xxxxxxxxxxxxxxxx
6   xxxxxxxxxxxxxxxx
7   xxxxxxxxxxxxxxxx
8   xxxxxxxxxxxxxxxx
9   0000000000000010
```

Waveform





- It can be seen that during jal instruction, jumpAndLink is 1, then pc+1 is saved to the register 15. Because of the multiplexer that checks jumpAndLink signal to set 1111 to write_reg.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000000111100
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001111
# jump address = 0000000100
# -----
```

```
# clock = 1
# PC= 0000000001
# instruction = 00001100000001010000000000000000
# opcode = 000011
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 1, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000000000
# alu_b = 0000000000001111
# alu_res = 0000000000001111
# alu_zero = 0
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 1111
# read_data1 = 0000000000000000
# read_data2 = 0000000000001111
# write_back = 0000000000000010
# jump address = 0000000101
# -----
```

```
# clock = 1
# PC= 0000000101
# instruction = 00100000010100000000000000000000
# opcode = 001000
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 1, loadImm = 0
# alu_a = 0000000000001111
# alu_b = 0000000000000000
# alu_res = 0000000000001111
# alu_zero = 0
# alu_op = 010
# read_reg1 = 0001, read_reg2 = 0100, write_reg = 0100
# read_data1 = 0000000000001111
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001111
# jump address = 0001010000
# -----
```

```
# clock = 1
# PC= 0000000110
# instruction = 00000011110000000000000010000000
# opcode = 000000
# RegDst = 1, RegWrite = 0, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 1, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000000010
# alu_b = 0000000000000000
# alu_res = 0000000000000010
# alu_zero = 0
# alu_op = 010
# read_reg1 = 1111, read_reg2 = 0000, write_reg = 0000
# read_data1 = 0000000000000010
# read_data2 = 0000000000000000
# write_back = 0000000000000010
# jump address = 1111000000
# -----
```

```
#
# clock = 1
# PC= 0000000010
# instruction = 00010001000001000000000000010000
# opcode = 000100
# RegDst = 0, RegWrite = 0, MemWrite = 0, memToReg = 0, branch = 1
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000001111
# alu_b = 0000000000001111
# alu_res = 0000000000000000
# alu_zero = 1
# alu_op = 110
# read_reg1 = 0100, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000001111
# read_data2 = 0000000000001111
# write_back = 0000000000000000
# jump address = 0100000100
# -----
```

```
# clock = 1
# PC= 0000000111
# instruction = 00100000010101000000000000000000
# opcode = 001000
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 1, loadImm = 0
# alu_a = 0000000000001111
# alu_b = 0000000000000000
# alu_res = 0000000000001111
# alu_zero = 0
# alu_op = 010
# read_reg1 = 0001, read_reg2 = 0101, write_reg = 0101
# read_data1 = 0000000000001111
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000001111
# jump address = 0001010100
# -----
```

- and_tb
  - simple r-type and operation sets the result to the register 3.

```
1   li $1, 21
2   li $2, 17
3   and $3, $1, $2
```

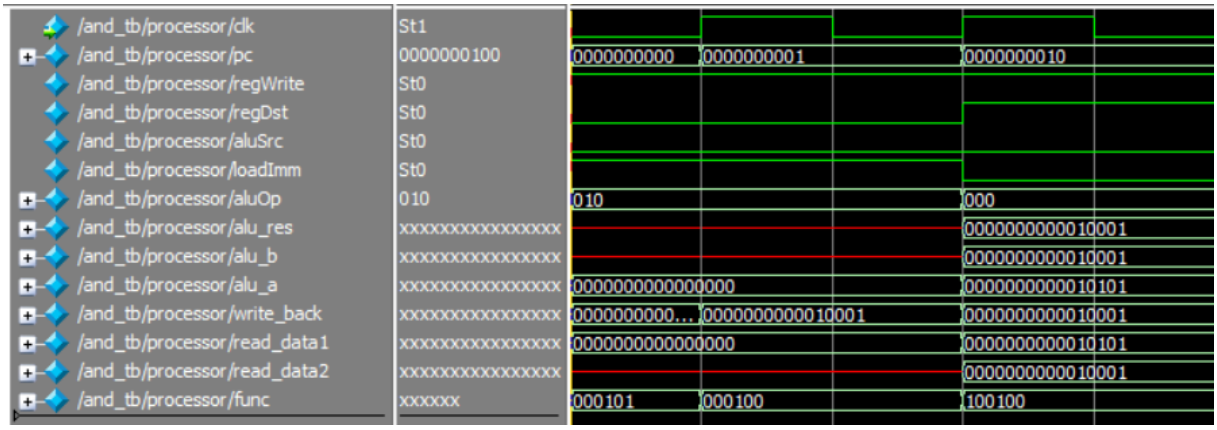
```
1   00111100000001000000000001010100
2   00111100000010000000000001000100
3   00000000010010001100001001000000
4
```

Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/and_tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000010101
6   0000000000010001
7   0000000000010001
8   xxxxxxxxxxxxxxxx
9   xxxxxxxxxxxxxxxx
10  xxxxxxxxxxxxxxxx
11  xxxxxxxxxxxxxxxx
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
```

Waveform



- regDst is 1.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000001010100
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000010101
# -----
```

```
# clock = 1
# PC= 0000000001
# instruction = 00111100000010000000000001000100
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000010001
# -----
```

```
# clock = 1
# PC= 0000000010
# instruction = 00000000010010001100001001000000
# opcode = 000000
# RegDst = 1, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000010101
# alu_b = 0000000000010001
# alu_res = 0000000000010001
# alu_zero = 0
# alu_op = 000
# read_reg1 = 0001, read_reg2 = 0010, write_reg = 0011
# read_data1 = 0000000000010101
# read_data2 = 0000000000010001
# write_back = 0000000000010001
# -----
```

- or_tb
  - It is very similar with and operation.

```
1   li $1, 21
2   li $2, 17
3   or $3, $1, $2
```

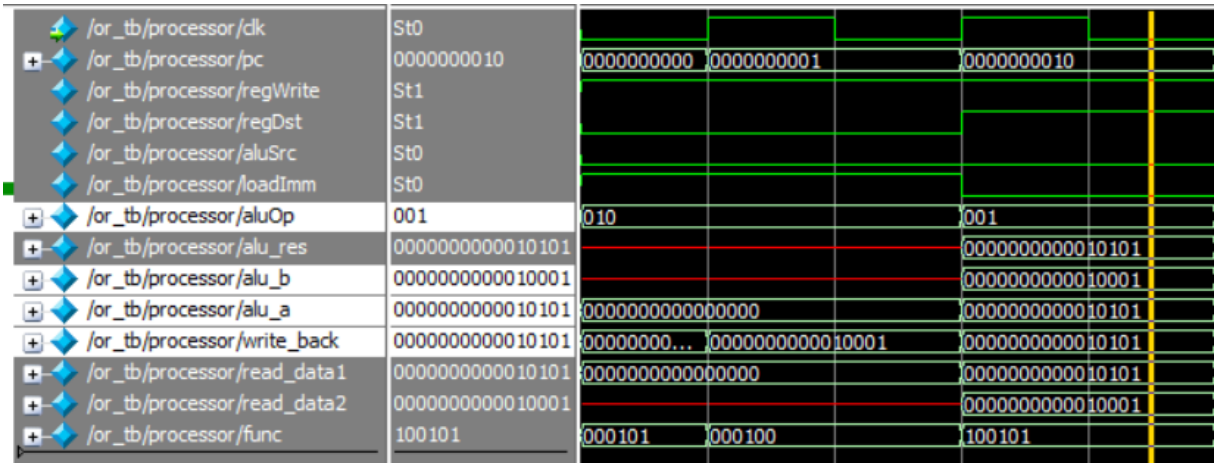
```
1   00111100000001000000000001010100
2   00111100000010000000000001000100
3   00000000010010001100001001010000
```

Results

Registers

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/or_tb/processor/regs/registers
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  0000000000000000
5  0000000000010101
6  0000000000010001
7  0000000000010101
8  xxxxxxxxxxxxxxxx
9  xxxxxxxxxxxxxxxx
10 xxxxxxxxxxxxxxxx
11 xxxxxxxxxxxxxxxx
12 xxxxxxxxxxxxxxxx
13 xxxxxxxxxxxxxxxx
14 xxxxxxxxxxxxxxxx
15 xxxxxxxxxxxxxxxx
16 xxxxxxxxxxxxxxxx
17 xxxxxxxxxxxxxxxx
18 xxxxxxxxxxxxxxxx
19 xxxxxxxxxxxxxxxx
20
```

Waveform



- RegDst is 1 again and the aluOp is 001.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000001010100
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000010101
# -----
```

```
# clock = 1
# PC= 0000000001
# instruction = 00111100000010000000000001000100
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000010001
# -----
```

```
# clock = 1
# PC= 0000000100
# instruction = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
# opcode = xxxxxx
# RegDst = 0, RegWrite = 0, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = xxxxxxxxxxxxxxxx
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = xxxx, read_reg2 = xxxx, write_reg = xxxx
# read_data1 = xxxxxxxxxxxxxxxx
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = xxxxxxxxxxxxxxxx
# -----
```

- andi_tb
  - immediate and instruction.

```
1   li $1, 21
2   andi $2, $1, 17
```

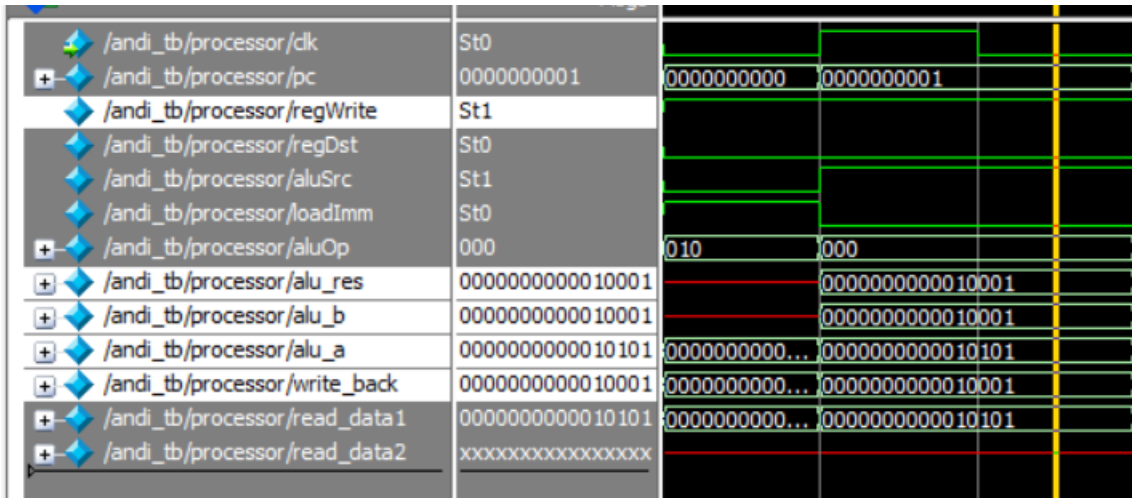
```
1   00111100000001000000000001010100
2   00110000010010000000000001000100
3
```

Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/andi_tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000010101
6   0000000000010001
7   xxxxxxxxxxxxxxxx
8   xxxxxxxxxxxxxxxx
9   xxxxxxxxxxxxxxxx
10  xxxxxxxxxxxxxxxx
11  xxxxxxxxxxxxxxxx
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
```

Waveform



- regDst is 0, but aluOp is 000. Immediate field is send to the alu.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000001010100
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000010101
# -----
```

```
# clock = 1
# PC= 0000000001
# instruction = 00110000010010000000000001000100
# opcode = 001100
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 1, loadImm = 0
# alu_a = 0000000000010101
# alu_b = 0000000000010001
# alu_res = 0000000000010001
# alu_zero = 0
# alu_op = 000
# read_reg1 = 0001, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000010101
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000010001
# -----
```

- ori_tb

- immediate or instruction.

```
1   li $1, 21
2   ori $2, $1, 17
```

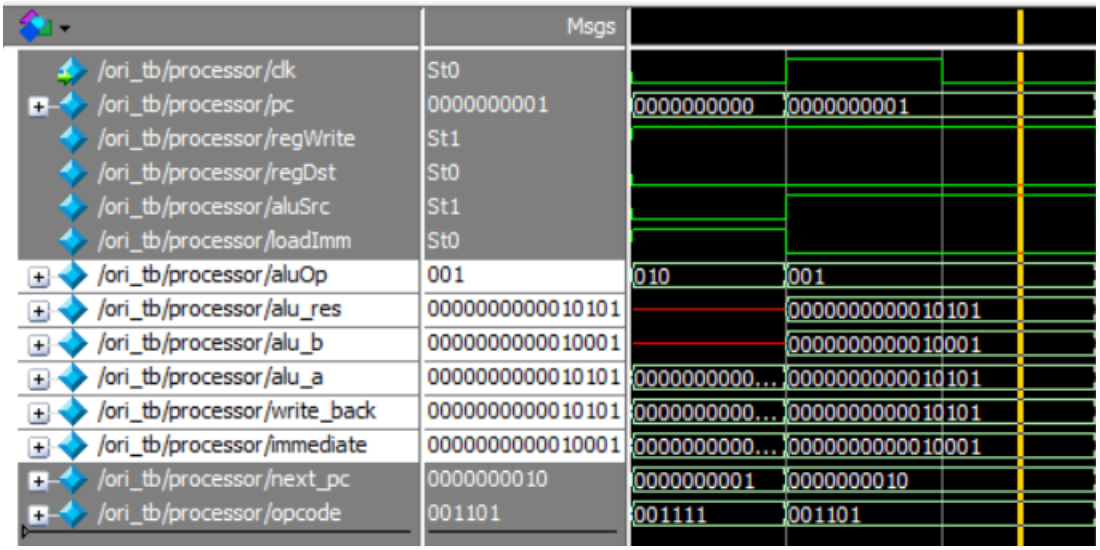
```
1   00111100000001000000000001010100
2   00110100010010000000000001000100
```

Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/ori_tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000010101
6   0000000000010101
7   xxxxxxxxxxxxxxxx
8   xxxxxxxxxxxxxxxx
9   xxxxxxxxxxxxxxxx
10  xxxxxxxxxxxxxxxx
11  xxxxxxxxxxxxxxxx
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
```

Waveform



- Immediate field is send to the alu because regDst is 0.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000001010100
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000010101
# -----

# clock = 1
# PC= 0000000001
# instruction = 00110100010010000000000001000100
# opcode = 001101
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 1, loadImm = 0
# alu_a = 0000000000010101
# alu_b = 0000000000010001
# alu_res = 0000000000010101
# alu_zero = 0
# alu_op = 001
# read_reg1 = 0001, read_reg2 = 0010, write_reg = 0010
# read_data1 = 0000000000010101
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000010101
# -----
```

- sll_tb
- shift left logical instruction.

```
> Users > burak kocausta >
1   li $1, 31
2   sll $2, $1, 3
3   sll $3, $1, 5
```

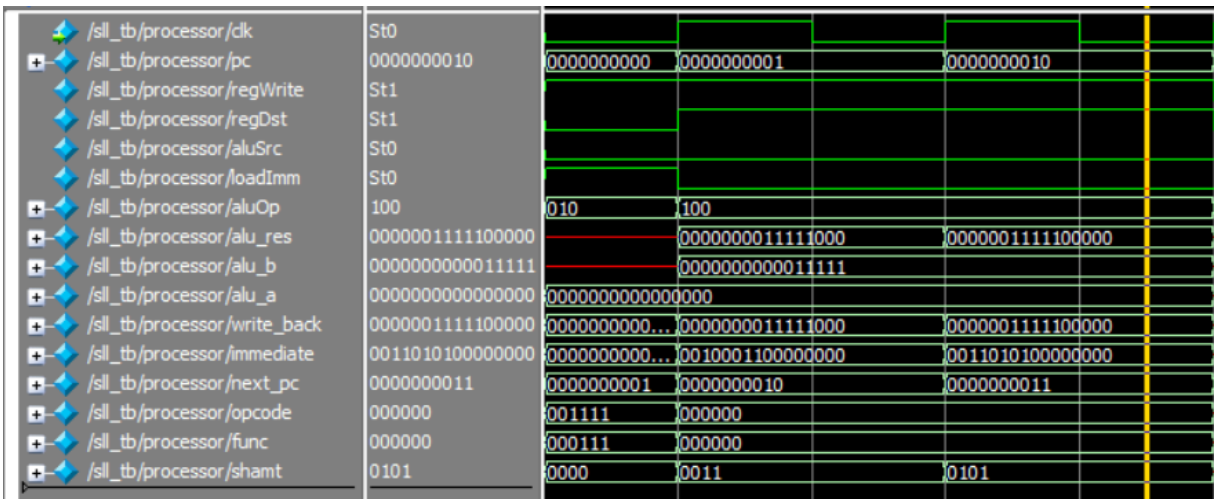
```
1   00111100000001000000000001111100
2   00000000000001001000110000000000
3   00000000000001001101010000000000
```

Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/sll_tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000011111
6   0000000011111000
7   0000001111100000
8   xxxxxxxxxxxxxxxx
9   xxxxxxxxxxxxxxxx
10  xxxxxxxxxxxxxxxx
11  xxxxxxxxxxxxxxxx
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
```

Waveform



- shamt field is 3, and 5. Also regDst is 1. regWrite is 1 because result must be written to the register.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000001111100
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000011111
# shamt = 0000
# -----
# clock = 1
# PC= 0000000001
# instruction = 00000000000001001000110000000000
# opcode = 000000
# RegDst = 1, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000000000
# alu_b = 0000000000011111
# alu_res = 0000000011111000
# alu_zero = 0
# alu_op = 100
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0010
# read_data1 = 0000000000000000
# read_data2 = 0000000000011111
# write_back = 0000000011111000
# shamt = 0011
# -----
# clock = 1
# PC= 0000000010
# instruction = 00000000000001001101010000000000
# opcode = 000000
# RegDst = 1, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000000000
# alu_b = 0000000000011111
# alu_res = 0000001111100000
# alu_zero = 0
# alu_op = 100
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0011
# read_data1 = 0000000000000000
# read_data2 = 0000000000011111
# write_back = 0000001111100000
# shamt = 0101
# -----
```

- srl_tb
- shift right logical very similar to the shift left logical.

```
1   li $1, 31
2   srl $2, $1, 3
3   srl $3, $1, 2
```

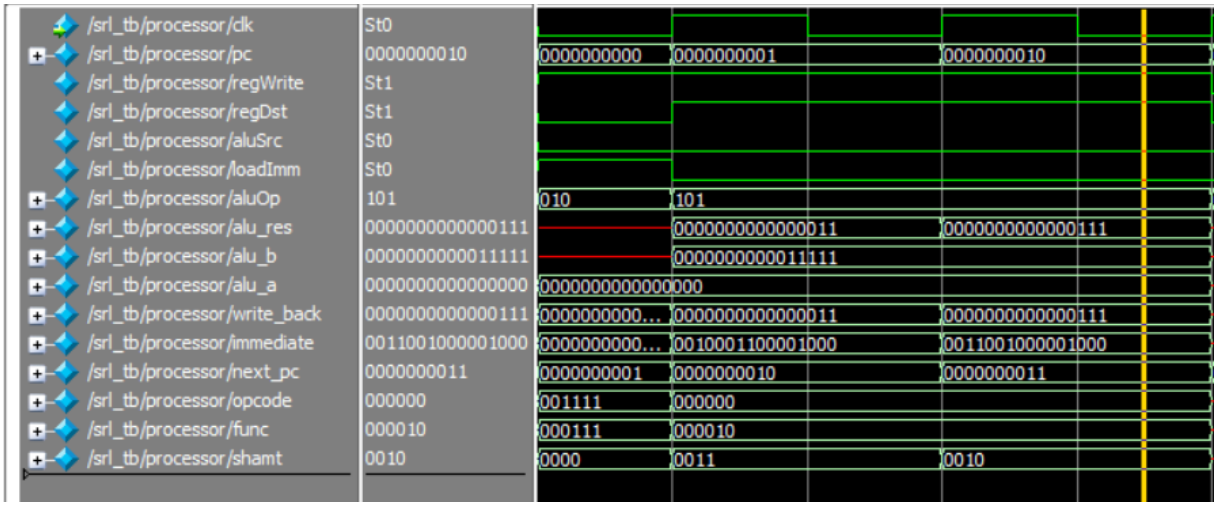
```
1   00111100000001000000000001111100
2   00000000000001001000110000100000
3   00000000000001001100100000100000
```

Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/srl_tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000011111
6   0000000000000011
7   0000000000000111
8   xxxxxxxxxxxxxxxx
9   xxxxxxxxxxxxxxxx
10  xxxxxxxxxxxxxxxx
11  xxxxxxxxxxxxxxxx
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
```

Waveform



- shamt is 3, and 2. regDst, regWrite is 1. aluOp is 101 which is srl aluOp.

Display

```
# clock = 0
# PC= 0000000000
# instruction = 00111100000001000000000001111100
# opcode = 001111
# RegDst = 0, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 1
# alu_a = 0000000000000000
# alu_b = xxxxxxxxxxxxxxxx
# alu_res = xxxxxxxxxxxxxxxx
# alu_zero = x
# alu_op = 010
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0001
# read_data1 = 0000000000000000
# read_data2 = xxxxxxxxxxxxxxxx
# write_back = 0000000000011111
# shamt = 0000
# -----
# clock = 1
# PC= 0000000001
# instruction = 00000000000001001000110000100000
# opcode = 000000
# RegDst = 1, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000000000
# alu_b = 0000000000011111
# alu_res = 0000000000000011
# alu_zero = 0
# alu_op = 101
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0010
# read_data1 = 0000000000000000
# read_data2 = 0000000000011111
# write_back = 0000000000000011
# shamt = 0011
# -----
# clock = 1
# PC= 0000000010
# instruction = 00000000000001001100100000100000
# opcode = 000000
# RegDst = 1, RegWrite = 1, MemWrite = 0, memToReg = 0, branch = 0
# branchNot = 0, jumpAndLink = 0, jumpReg = 0, jump = 0, aluSrc = 0, loadImm = 0
# alu_a = 0000000000000000
# alu_b = 0000000000011111
# alu_res = 0000000000000111
# alu_zero = 0
# alu_op = 101
# read_reg1 = 0000, read_reg2 = 0001, write_reg = 0011
# read_data1 = 0000000000000000
# read_data2 = 0000000000011111
# write_back = 0000000000000111
# shamt = 0010
# -----
```

3- General Assembly Program Tests

- single_cycle_mips_spec1tb

- This test bench is provided by presentation instructions, simply tests lw, sw operations. Memory and register results are shown.

assembly program

```
1   li $3, 15
2   li $2, 256
3   sw $3, 100($2)
4   lw $1, 100($2)
5   addi $1, $1, 4
6   sw $1, 100($2)
```

generated machine code

```
1   00111100000011000000000000111100
2   00111100000010000000010000000000
3   10101100100011000000000110010000
4   10001100100001000000000110010000
5   00100000100010000000000000010000
6   10101100100001000000000110010000
```

Results

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/single_cycle_mips_spec1tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000010011
6   0000000100000000
7   0000000000001111
8   xxxxxxxxxxxxxxxx
9   xxxxxxxxxxxxxxxx
10  xxxxxxxxxxxxxxxx
11  xxxxxxxxxxxxxxxx
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
20
```

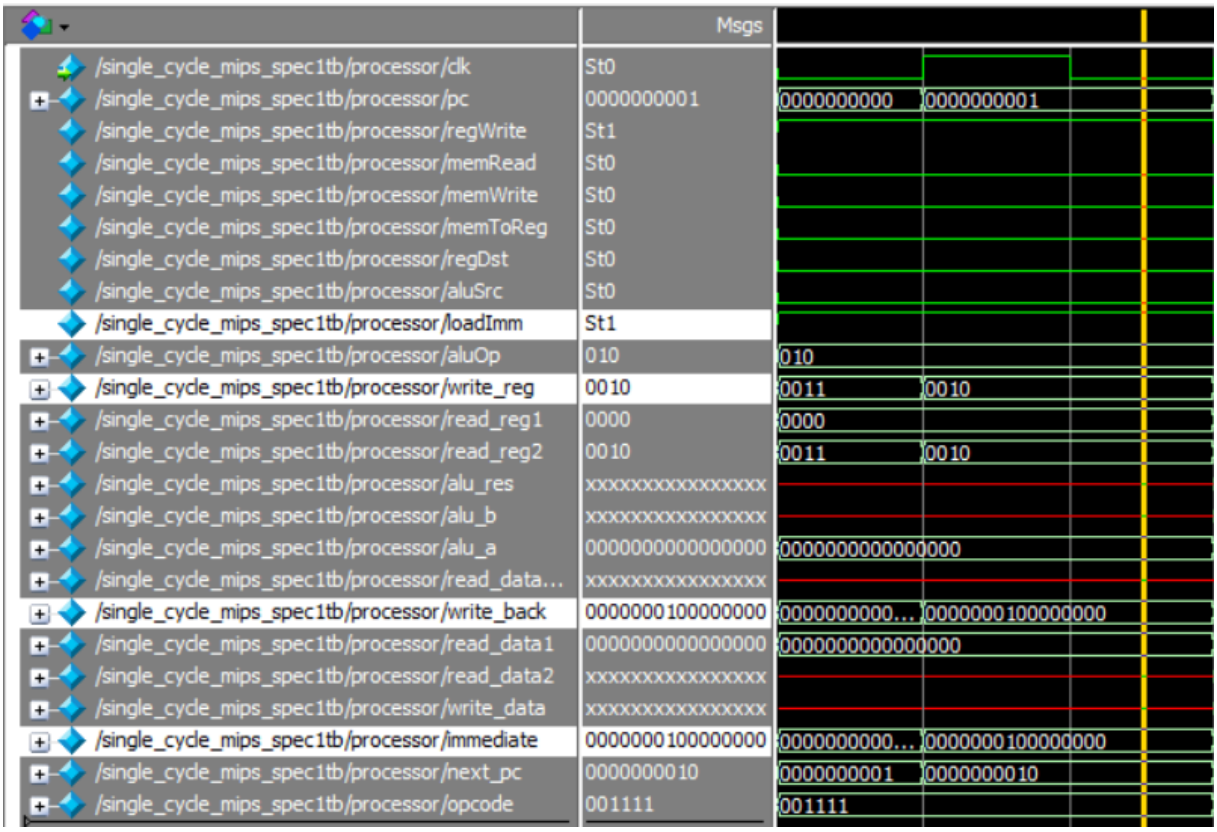
Memory

```
353   xxxxxxxxxxxxxxxx
354   xxxxxxxxxxxxxxxx
355   xxxxxxxxxxxxxxxx
356   xxxxxxxxxxxxxxxx
357   xxxxxxxxxxxxxxxx
358   xxxxxxxxxxxxxxxx
359   xxxxxxxxxxxxxxxx
360   0000000000010011
361   xxxxxxxxxxxxxxxx
362   xxxxxxxxxxxxxxxx
363   xxxxxxxxxxxxxxxx
364   xxxxxxxxxxxxxxxx
365   xxxxxxxxxxxxxxxx
366   xxxxxxxxxxxxxxxx
```

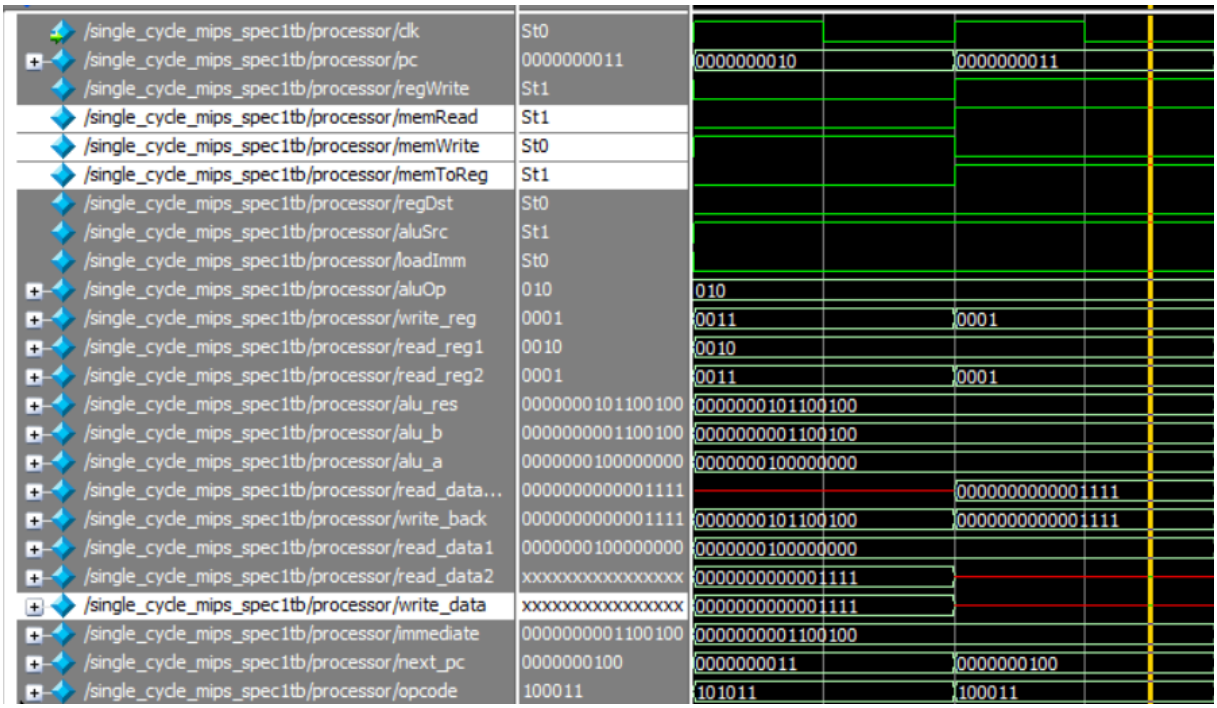
(360-4 = 356th cell) first memory is 4 in the file

Waveform

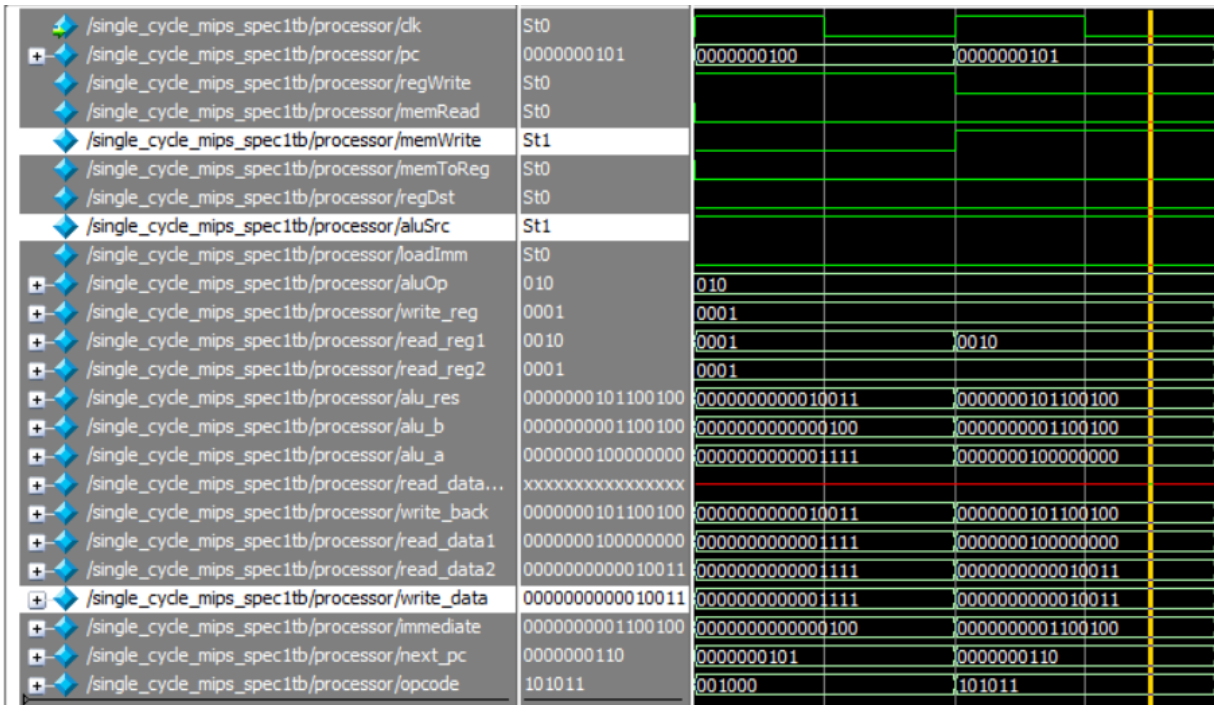
first 2 instructions, load immediate



Store word and load word instructions



Add immediate, and store word again



- Signals are exactly the same with the simple tests of each instruction, writing memory, and reading is done successfully.

- single_cycle_mips_spec2tb
- Test bench provided by presentation file, it tests lw, sw instructions.

assembly program generated machine code

```
1   li $6, 18
2   li $7, 31
3   li $3, 512
4   sw $6, 100($3)
5   sw $7, 80($3)
6   lw $1, 100($3)
7   lw $2, 80($3)
8   and $4, $1, $2
9   slti $5, $4, 1
```

```
1   00111100000110000000000001001000
2   00111100000111000000000001111100
3   00111100000011000000100000000000
4   10101100110110000000000110010000
5   10101100110111000000000101000000
6   10001100110001000000000110010000
7   10001100110010000000000101000000
8   00000000010010010000001001000000
9   00101001000101000000000000000100
```

Result

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/single_cycle_mips_spec2tb/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000010010
6   0000000000011111
7   0000001000000000
8   0000000000010010
9   0000000000000000
10  0000000000010010
11  0000000000011111
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
```

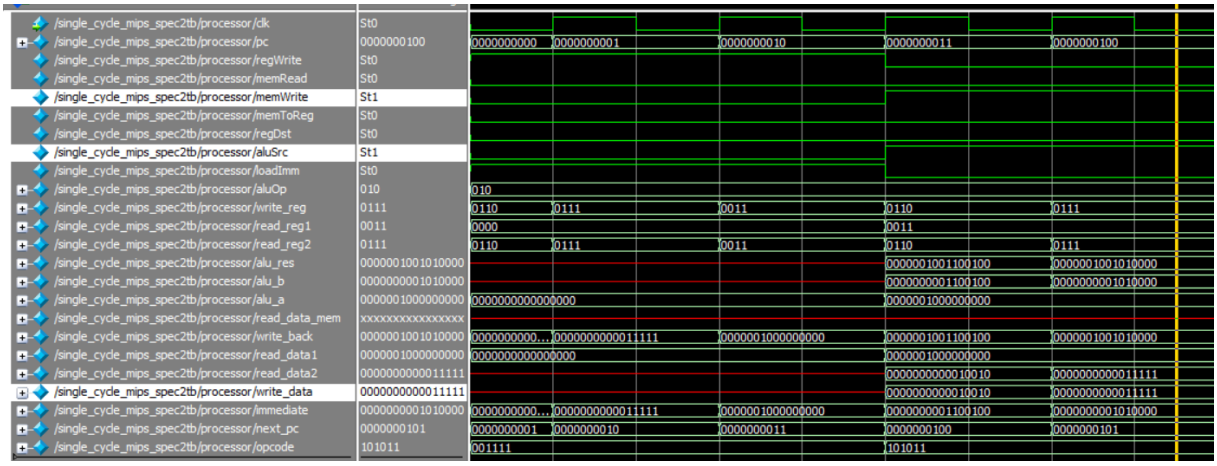
Memory

```
596   0000000000011111
597   xxxxxxxxxxxxxxxx
598   xxxxxxxxxxxxxxxx
599   xxxxxxxxxxxxxxxx
600   xxxxxxxxxxxxxxxx
601   xxxxxxxxxxxxxxxx
602   xxxxxxxxxxxxxxxx
603   xxxxxxxxxxxxxxxx
604   xxxxxxxxxxxxxxxx
605   xxxxxxxxxxxxxxxx
606   xxxxxxxxxxxxxxxx
607   xxxxxxxxxxxxxxxx
608   xxxxxxxxxxxxxxxx
609   xxxxxxxxxxxxxxxx
610   xxxxxxxxxxxxxxxx
611   xxxxxxxxxxxxxxxx
612   xxxxxxxxxxxxxxxx
613   xxxxxxxxxxxxxxxx
614   xxxxxxxxxxxxxxxx
615   xxxxxxxxxxxxxxxx
616   0000000000010010
```

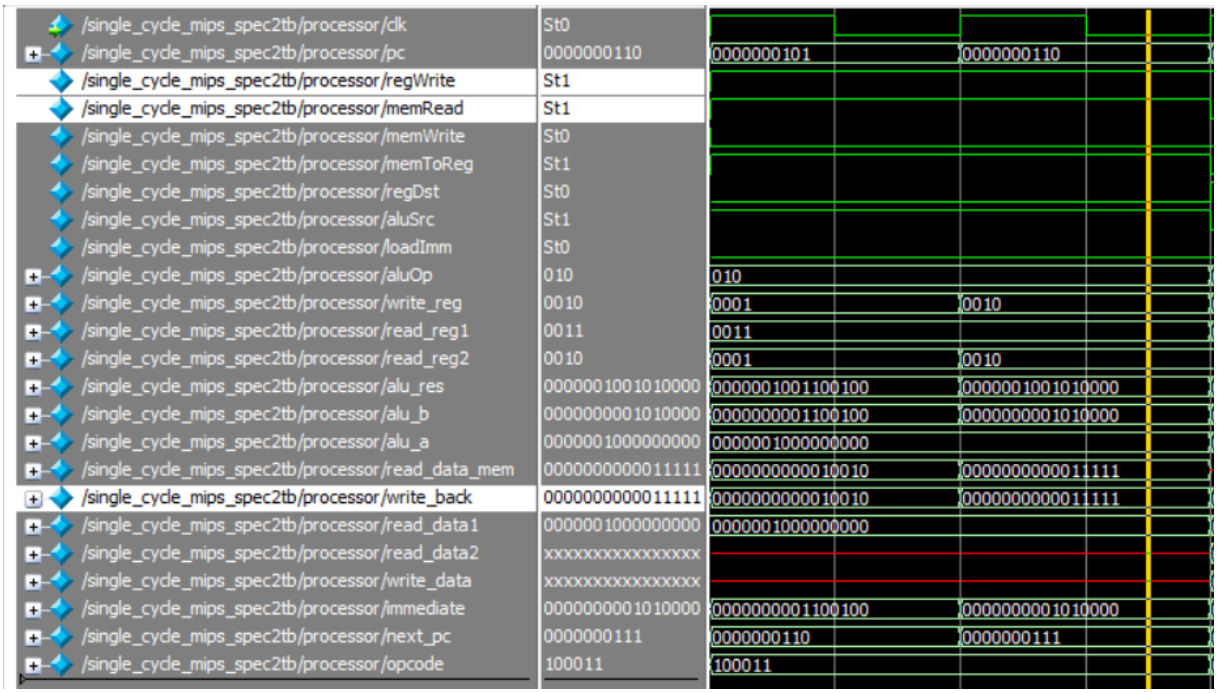
(592, 612) (first memory cell is 4 in the file)

Waveform

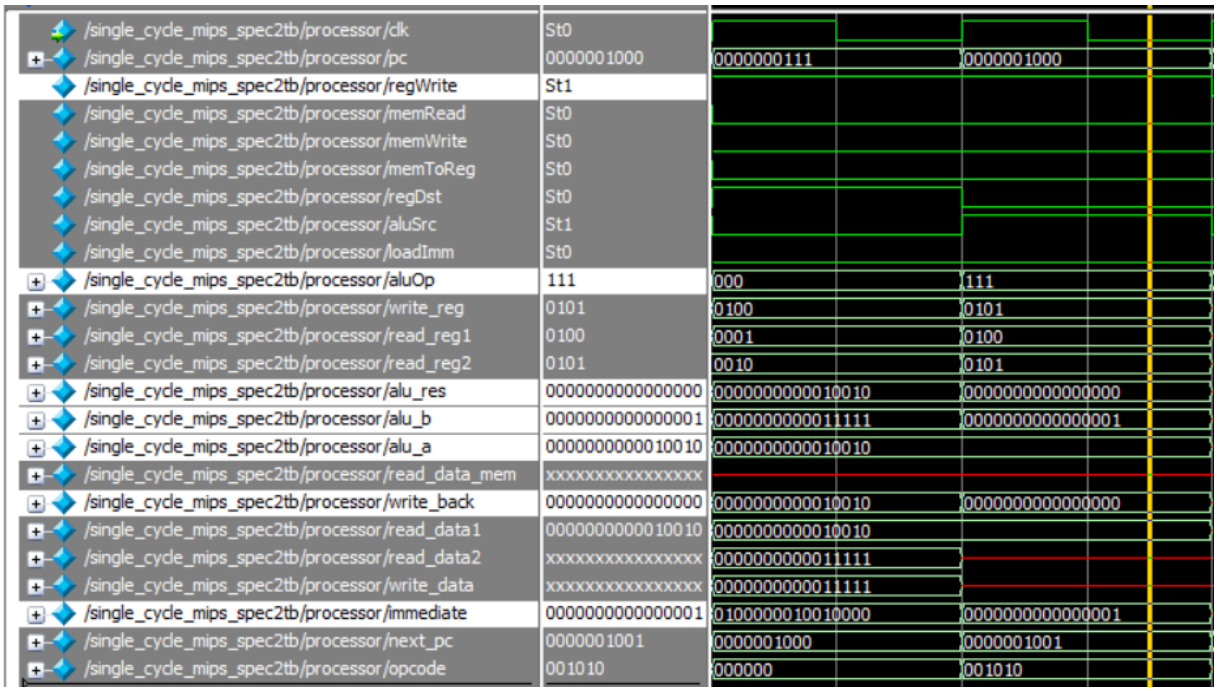
Load immediate and store word instructions



Load word instructions



and, and set less than instructions



- single_cycle_mips_tb1
- This test bench simply tests all basic instructions. All of them uses alu operations. Both r type and I type instructions of these basic operations are tested.

assembly program

```
1   li $1, 17
2   addi $2, $1, 19
3   andi $3, $2, 255
4   ori $4, $3, 19
5   slti $5, $4, 15
6   add $6, $5, $2
7   sub $7, $6, $2
8   li $13, 511
9   and $8, $6, $13
10  or $9, $8, $7
11  slt $10, $9, $8
12  sll $11, $9, 2
13  srl $12, $11, 3
```

generated machine code

```
1   00111100000001000000000000010001
2   00100000010010000000000000011001
3   00110000100011000000001111111100
4   00110100110100000000000000011001
5   00101001000101000000000000111100
6   00000001010010011000001000000000
7   00000001100010011110000100010000
8   00111100001101000000011111111100
9   00000001101101100000001001000000
10  00000010000111100100001001010000
11  00000010011000101000001010100000
12  00000000001001101100100000000000
13  00000000001011110000110000100000
```

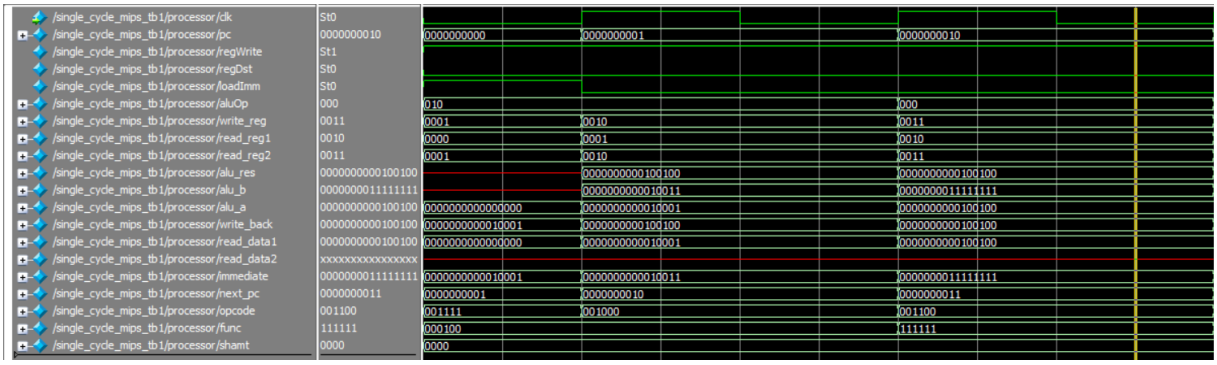
Result

Registers

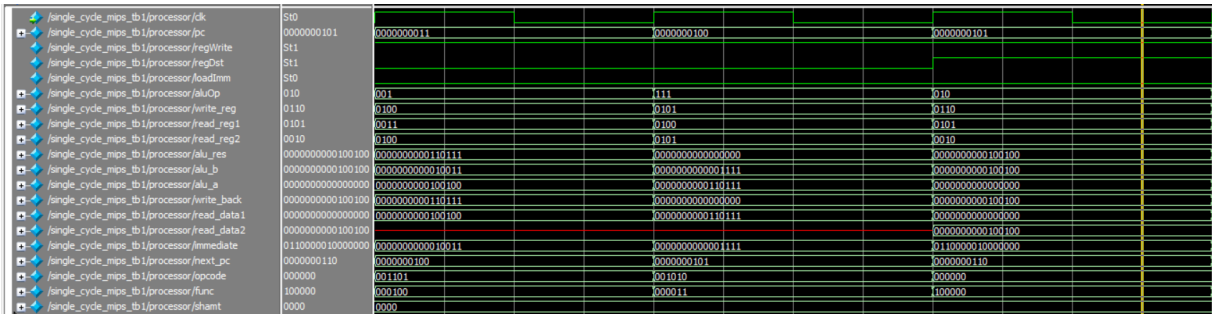
```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/single_cycle_mips_tb1/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000010001
6   0000000000100100
7   0000000000100100
8   0000000000110111
9   0000000000000000
10  0000000000100100
11  0000000000000000
12  0000000000100100
13  0000000000100100
14  0000000000000000
15  0000000010010000
16  0000000000010010
17  0000000111111111
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
```

Waveform

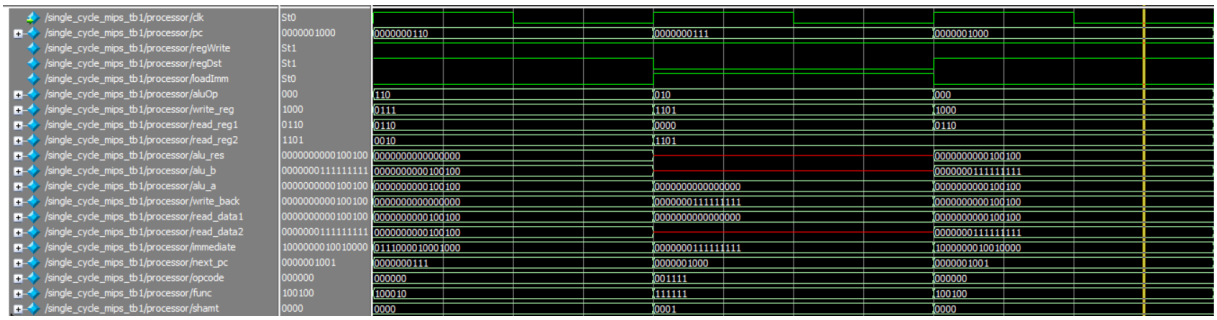
li, addi, andi



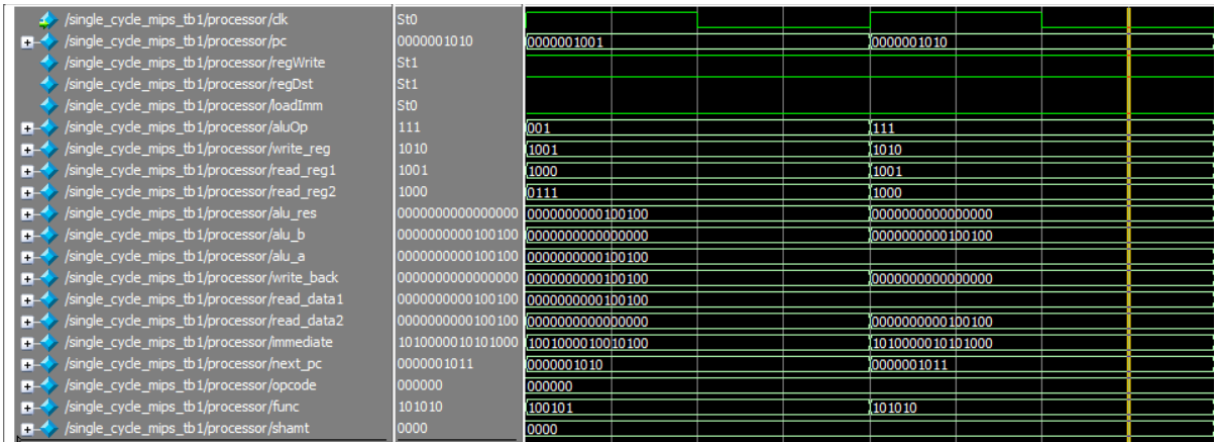
ori, slti, add



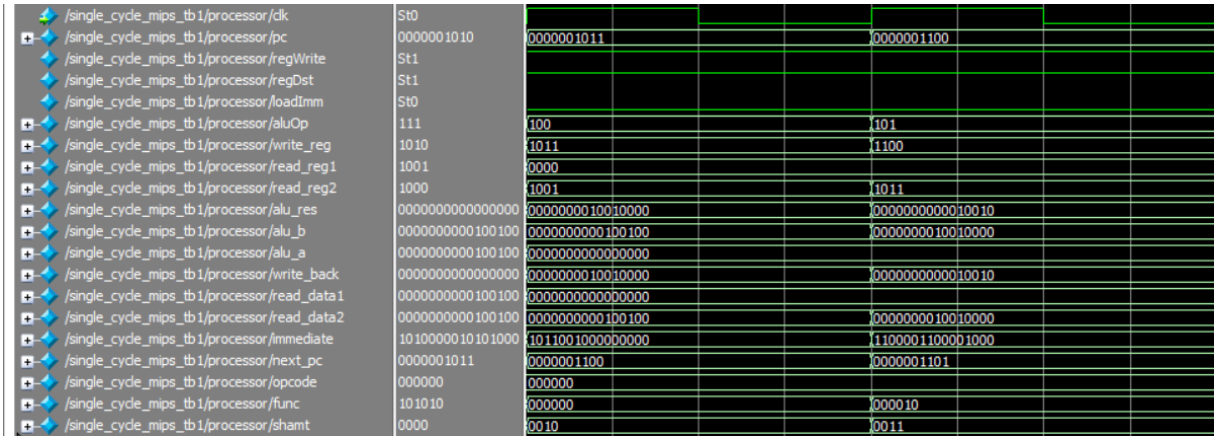
sub, li, and



or, slt



sll, srl



- single_cycle_mips_tb2

- This test bench tests branch and jump instructions. A simple loop is created. 6th register has no value because of bne instruction. Iteration can be followed by tracking the program counter in the waveform.

assembly program

```
1   li $1, 15
2   li $2, 20
3   li $3, 0
4   beq $1, $2, 3
5   addi $3, $3, 1
6   addi $1, $1, 1
7   j 3
8   slti $4, $3, 6
9   bne $4, $0, 1
10  add $6, $1, $2
11  sub $7, $1, $2
```

generated machine code

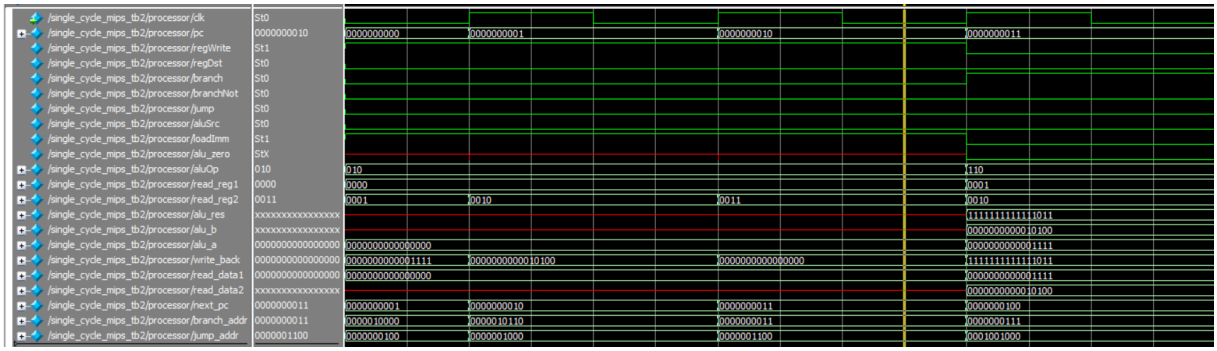
```
1   00111100000001000000000000111100
2   00111100000010000000000001010000
3   00111100000011000000000000000000
4   00010000010010000000000000001100
5   00100000110011000000000000000100
6   00100000010001000000000000000100
7   00001000000000110000000000000000
8   00101000110100000000000000011000
9   00010101000000000000000000000100
10  00000000010010011000001000000000
11  00000000010010011100001000100000
12
```

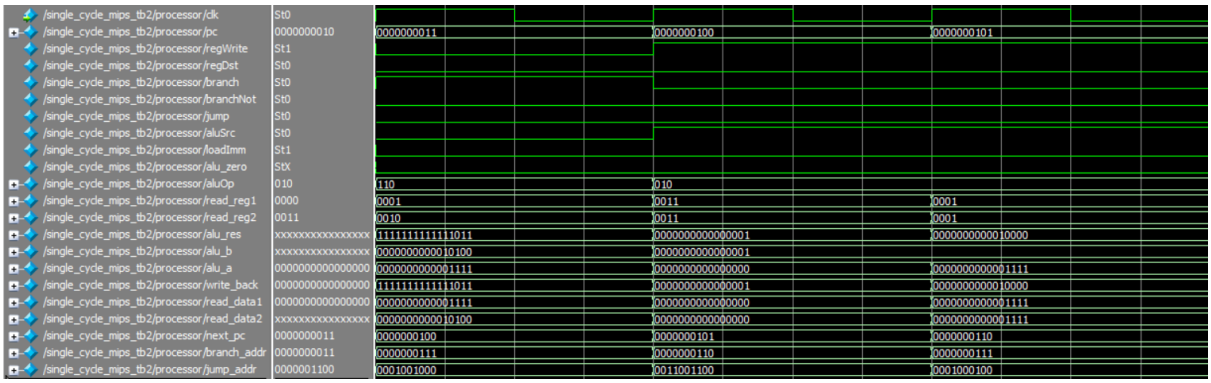
Result

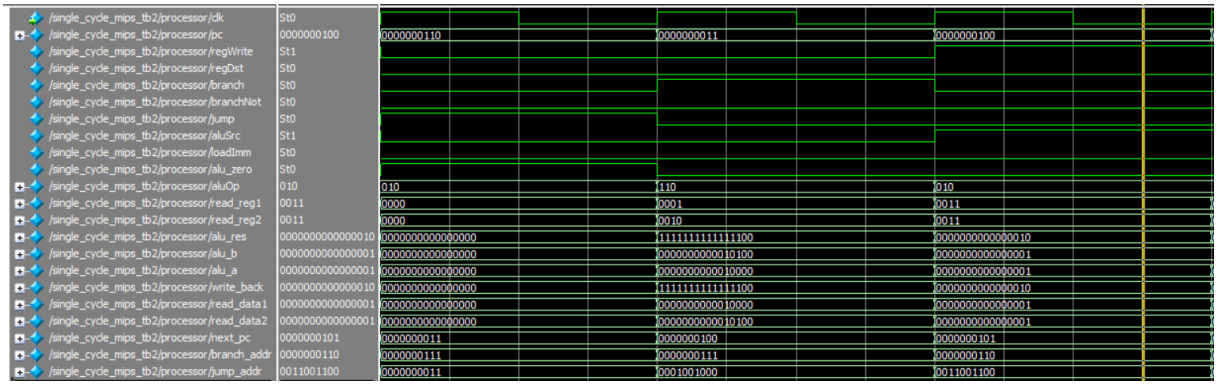
Registers

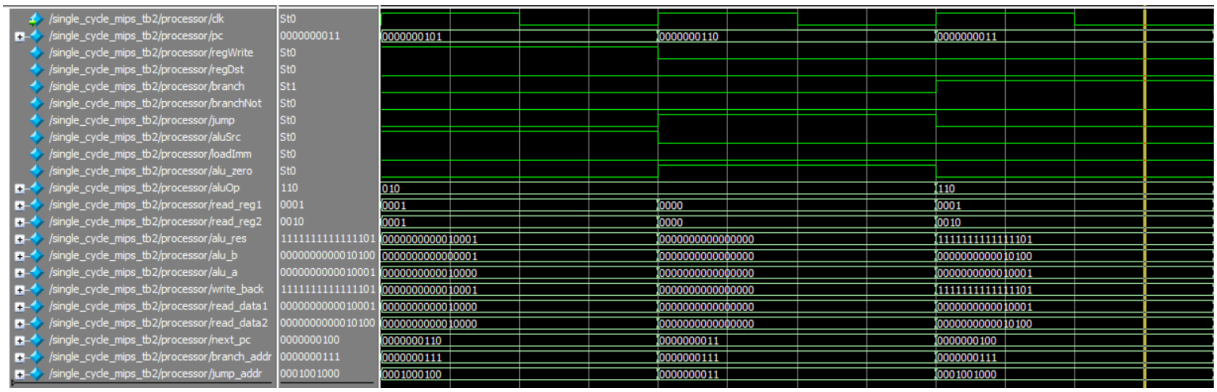
```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/single_cycle_mips_tb2/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000000010100
6   0000000000010100
7   0000000000000101
8   0000000000000001
9   xxxxxxxxxxxxxxxx
10  xxxxxxxxxxxxxxxx
11  0000000000000000
12  xxxxxxxxxxxxxxxx
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
```

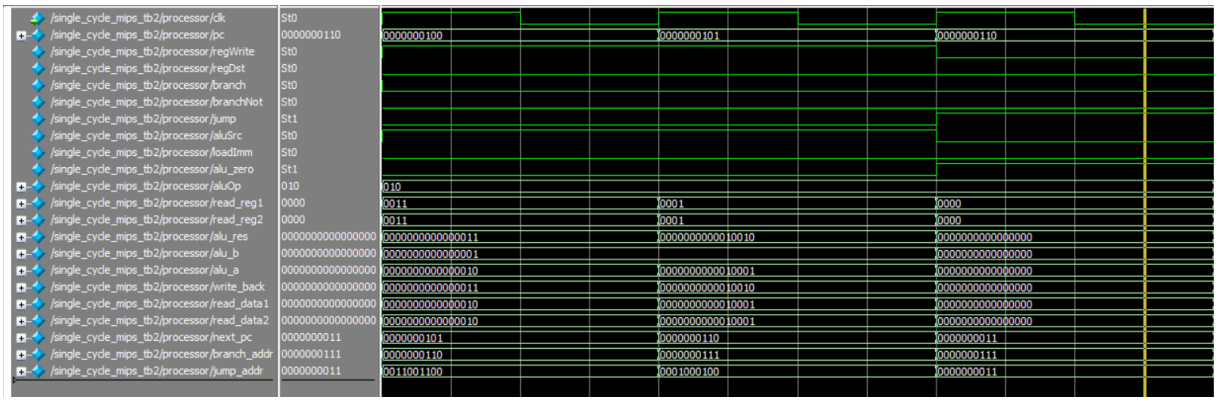
Waveform

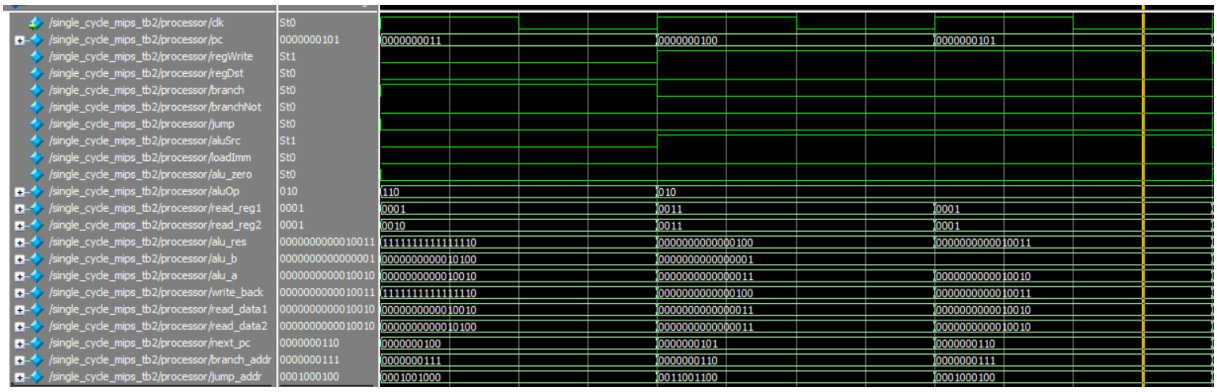

/single_cycle_mips_tb2/processor/clk St0
/single_cycle_mips_tb2/processor/pc 0000000010 0000000000 0000000001 0000000010 0000000011
/single_cycle_mips_tb2/processor/regWrite St1
/single_cycle_mips_tb2/processor/regDst St0
/single_cycle_mips_tb2/processor/branch St0
/single_cycle_mips_tb2/processor/branchNot St0
/single_cycle_mips_tb2/processor/jump St0
/single_cycle_mips_tb2/processor/aluSrc St0
/single_cycle_mips_tb2/processor/loadImm St1
/single_cycle_mips_tb2/processor/alu_zero StX
/single_cycle_mips_tb2/processor/aluOp 010 010 110
/single_cycle_mips_tb2/processor/read_reg1 0000 0000 0001
/single_cycle_mips_tb2/processor/read_reg2 0011 0001 0010 0011 0010
/single_cycle_mips_tb2/processor/alu_res xxxxxxxxxxxxxxxx 1111111111111011
/single_cycle_mips_tb2/processor/alu_b xxxxxxxxxxxxxxxx 0000000000010100
/single_cycle_mips_tb2/processor/alu_a 0000000000000000 0000000000001111
/single_cycle_mips_tb2/processor/write_back 0000000000000000 0000000000011111 0000000000010100 0000000000000000 1111111111111011
/single_cycle_mips_tb2/processor/read_data1 0000000000000000 0000000000000000 0000000000001111
/single_cycle_mips_tb2/processor/read_data2 xxxxxxxxxxxxxxxx 0000000000010100
/single_cycle_mips_tb2/processor/next_pc 0000000011 0000000001 0000000010 0000000011 0000000100
/single_cycle_mips_tb2/processor/branch_addr 0000000011 0000010000 0000010110 0000000011 0000000111
/single_cycle_mips_tb2/processor/jump_addr 0000001100 0000000100 0000001000 0000001100 0001001000

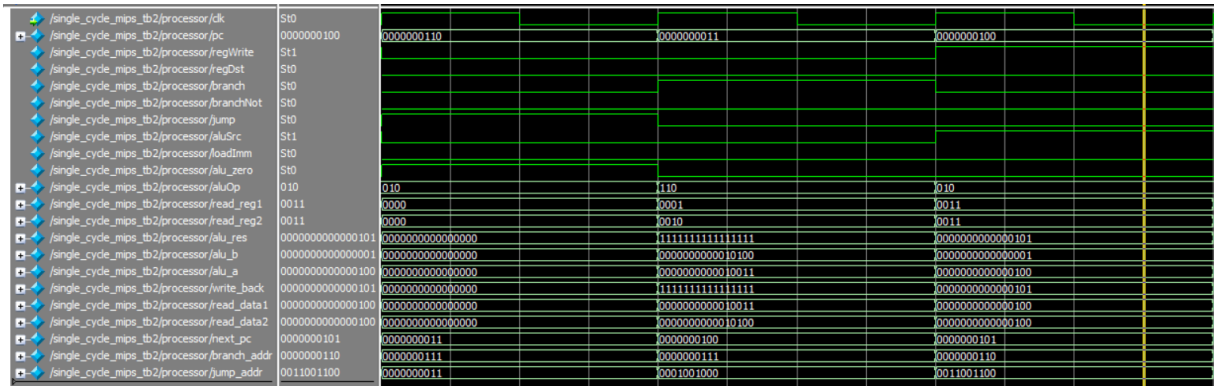

/single_cycle_mips_tb2/processor/clk St0
/single_cycle_mips_tb2/processor/pc 0000000010 0000000011 0000000100 0000000101
/single_cycle_mips_tb2/processor/regWrite St1
/single_cycle_mips_tb2/processor/regDst St0
/single_cycle_mips_tb2/processor/branch St0
/single_cycle_mips_tb2/processor/branchNot St0
/single_cycle_mips_tb2/processor/jump St0
/single_cycle_mips_tb2/processor/aluSrc St0
/single_cycle_mips_tb2/processor/loadImm St1
/single_cycle_mips_tb2/processor/alu_zero StX
/single_cycle_mips_tb2/processor/aluOp 010 110 010
/single_cycle_mips_tb2/processor/read_reg1 0000 0001 0011 0001
/single_cycle_mips_tb2/processor/read_reg2 0011 0010 0011 0001
/single_cycle_mips_tb2/processor/alu_res xxxxxxxxxxxxxxxx 1111111111111011 0000000000000001 0000000000010000
/single_cycle_mips_tb2/processor/alu_b xxxxxxxxxxxxxxxx 0000000000010100 0000000000000001
/single_cycle_mips_tb2/processor/alu_a 0000000000000000 0000000000001111 0000000000000000 0000000000001111
/single_cycle_mips_tb2/processor/write_back 0000000000000000 1111111111111011 0000000000000001 0000000000010000
/single_cycle_mips_tb2/processor/read_data1 0000000000000000 0000000000001111 0000000000000000 0000000000001111
/single_cycle_mips_tb2/processor/read_data2 xxxxxxxxxxxxxxxx 0000000000010100 0000000000000000 0000000000001111
/single_cycle_mips_tb2/processor/next_pc 0000000011 0000000100 0000000101 0000000110
/single_cycle_mips_tb2/processor/branch_addr 0000000011 0000000111 0000000110 0000000111
/single_cycle_mips_tb2/processor/jump_addr 0000001100 0001001000 0011001100 0001000100

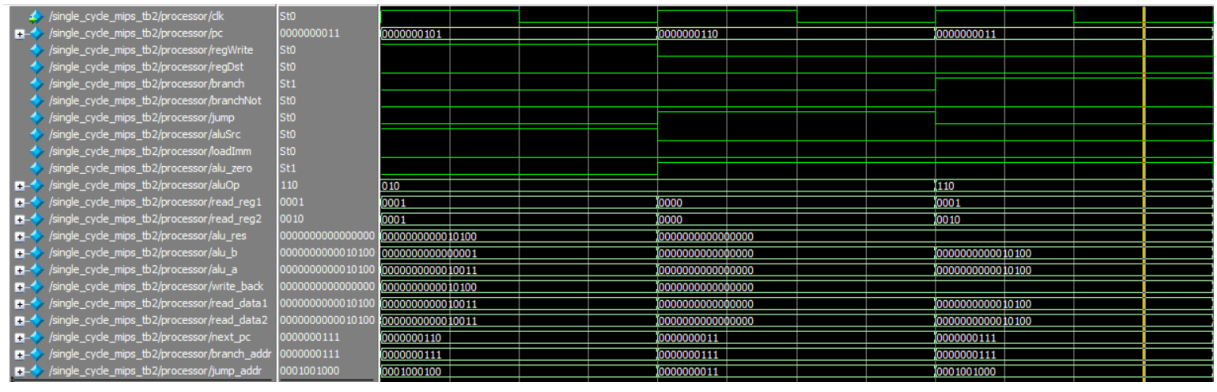

/single_cycle_mips_tb2/processor/clk St0
/single_cycle_mips_tb2/processor/pc 0000000100 0000000110 0000000011 0000000100
/single_cycle_mips_tb2/processor/regWrite St1
/single_cycle_mips_tb2/processor/regDst St0
/single_cycle_mips_tb2/processor/branch St0
/single_cycle_mips_tb2/processor/branchNot St0
/single_cycle_mips_tb2/processor/jump St0
/single_cycle_mips_tb2/processor/aluSrc St1
/single_cycle_mips_tb2/processor/loadImm St0
/single_cycle_mips_tb2/processor/alu_zero St0
/single_cycle_mips_tb2/processor/aluOp 010 110 010
/single_cycle_mips_tb2/processor/read_reg1 0011 0000 0001 0011
/single_cycle_mips_tb2/processor/read_reg2 0011 0000 0010 0011
/single_cycle_mips_tb2/processor/alu_res 0000000000000010 0000000000000000 1111111111111100 0000000000000010
/single_cycle_mips_tb2/processor/alu_b 0000000000000001 0000000000000000 0000000000010100 0000000000000001
/single_cycle_mips_tb2/processor/alu_a 0000000000000001 0000000000000000 0000000000010000 0000000000000001
/single_cycle_mips_tb2/processor/write_back 0000000000000010 0000000000000000 1111111111111100 0000000000000010
/single_cycle_mips_tb2/processor/read_data1 0000000000000001 0000000000000000 0000000000010000 0000000000000001
/single_cycle_mips_tb2/processor/read_data2 0000000000000001 0000000000000000 0000000000010100 0000000000000001
/single_cycle_mips_tb2/processor/next_pc 0000000101 0000000011 0000000100 0000000101
/single_cycle_mips_tb2/processor/branch_addr 0000000110 0000000111 0000000111 0000000110
/single_cycle_mips_tb2/processor/jump_addr 0011001100 0000000011 0001001000 0011001100

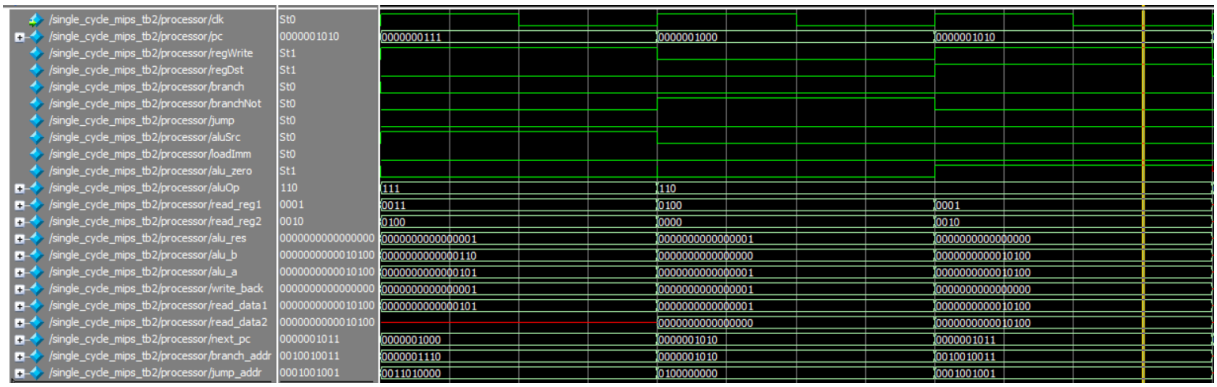

/single_cycle_mips_tb2/processor/clk St0
/single_cycle_mips_tb2/processor/pc 0000000011 0000000101 0000000110 0000000011
/single_cycle_mips_tb2/processor/regWrite St0
/single_cycle_mips_tb2/processor/regDst St0
/single_cycle_mips_tb2/processor/branch St1
/single_cycle_mips_tb2/processor/branchNot St0
/single_cycle_mips_tb2/processor/jump St0
/single_cycle_mips_tb2/processor/aluSrc St0
/single_cycle_mips_tb2/processor/loadImm St0
/single_cycle_mips_tb2/processor/alu_zero St0
/single_cycle_mips_tb2/processor/aluOp 110 010 110
/single_cycle_mips_tb2/processor/read_reg1 0001 0001 0000 0001
/single_cycle_mips_tb2/processor/read_reg2 0010 0001 0000 0010
/single_cycle_mips_tb2/processor/alu_res 1111111111111101 0000000000010001 0000000000000000 1111111111111101
/single_cycle_mips_tb2/processor/alu_b 0000000000010100 0000000000000001 0000000000000000 0000000000010100
/single_cycle_mips_tb2/processor/alu_a 0000000000010001 0000000000010000 0000000000000000 0000000000010001
/single_cycle_mips_tb2/processor/write_back 1111111111111101 0000000000010001 0000000000000000 1111111111111101
/single_cycle_mips_tb2/processor/read_data1 0000000000010001 0000000000010000 0000000000000000 0000000000010001
/single_cycle_mips_tb2/processor/read_data2 0000000000010100 0000000000010000 0000000000000000 0000000000010100
/single_cycle_mips_tb2/processor/next_pc 0000000100 0000000110 0000000011 0000000100
/single_cycle_mips_tb2/processor/branch_addr 0000000111 0000000111 0000000111 0000000111
/single_cycle_mips_tb2/processor/jump_addr 0001001000 0001000100 0000000011 0001001000


/single_cycle_mips_tb2/processor/clk St0
/single_cycle_mips_tb2/processor/pc 0000000110 0000000100 0000000101 0000000110
/single_cycle_mips_tb2/processor/regWrite St0
/single_cycle_mips_tb2/processor/regDst St0
/single_cycle_mips_tb2/processor/branch St0
/single_cycle_mips_tb2/processor/branchNot St0
/single_cycle_mips_tb2/processor/jump St1
/single_cycle_mips_tb2/processor/aluSrc St0
/single_cycle_mips_tb2/processor/loadImm St0
/single_cycle_mips_tb2/processor/alu_zero St1
/single_cycle_mips_tb2/processor/aluOp 010 010
/single_cycle_mips_tb2/processor/read_reg1 0000 0011 0001 0000
/single_cycle_mips_tb2/processor/read_reg2 0000 0011 0001 0000
/single_cycle_mips_tb2/processor/alu_res 0000000000000000 0000000000000011 0000000000010010 0000000000000000
/single_cycle_mips_tb2/processor/alu_b 0000000000000000 0000000000000001 0000000000000000
/single_cycle_mips_tb2/processor/alu_a 0000000000000000 0000000000000010 0000000000010001 0000000000000000
/single_cycle_mips_tb2/processor/write_back 0000000000000000 0000000000000011 0000000000010010 0000000000000000
/single_cycle_mips_tb2/processor/read_data1 0000000000000000 0000000000000010 0000000000010001 0000000000000000
/single_cycle_mips_tb2/processor/read_data2 0000000000000000 0000000000000010 0000000000010001 0000000000000000
/single_cycle_mips_tb2/processor/next_pc 0000000011 0000000101 0000000110 0000000011
/single_cycle_mips_tb2/processor/branch_addr 0000000111 0000000110 0000000111 0000000111
/single_cycle_mips_tb2/processor/jump_addr 0000000011 0011001100 0001000100 0000000011


/single_cycle_mips_tb2/processor/clk St0
/single_cycle_mips_tb2/processor/pc 0000000101 0000000011 0000000100 0000000101
/single_cycle_mips_tb2/processor/regWrite St1
/single_cycle_mips_tb2/processor/regDst St0
/single_cycle_mips_tb2/processor/branch St0
/single_cycle_mips_tb2/processor/branchNot St0
/single_cycle_mips_tb2/processor/jump St0
/single_cycle_mips_tb2/processor/aluSrc St1
/single_cycle_mips_tb2/processor/loadImm St0
/single_cycle_mips_tb2/processor/alu_zero St0
/single_cycle_mips_tb2/processor/aluOp 010 110 010
/single_cycle_mips_tb2/processor/read_reg1 0001 0001 0011 0001
/single_cycle_mips_tb2/processor/read_reg2 0001 0010 0011 0001
/single_cycle_mips_tb2/processor/alu_res 0000000000010011 1111111111111110 0000000000000100 0000000000010011
/single_cycle_mips_tb2/processor/alu_b 0000000000000001 0000000000010100 0000000000000001
/single_cycle_mips_tb2/processor/alu_a 0000000000010010 0000000000010010 0000000000000011 0000000000010010
/single_cycle_mips_tb2/processor/write_back 0000000000010011 1111111111111110 0000000000000100 0000000000010011
/single_cycle_mips_tb2/processor/read_data1 0000000000010010 0000000000010010 0000000000000011 0000000000010010
/single_cycle_mips_tb2/processor/read_data2 0000000000010010 0000000000010100 0000000000000011 0000000000010010
/single_cycle_mips_tb2/processor/next_pc 0000000110 0000000100 0000000101 0000000110
/single_cycle_mips_tb2/processor/branch_addr 0000000111 0000000111 0000000110 0000000111
/single_cycle_mips_tb2/processor/jump_addr 0001000100 0001001000 0011001100 0001000100


/single_cycle_mips_tb2/processor/clk St0
/single_cycle_mips_tb2/processor/pc 0000000100 0000000110 0000000011 0000000100
/single_cycle_mips_tb2/processor/regWrite St1
/single_cycle_mips_tb2/processor/regDst St0
/single_cycle_mips_tb2/processor/branch St0
/single_cycle_mips_tb2/processor/branchNot St0
/single_cycle_mips_tb2/processor/jump St0
/single_cycle_mips_tb2/processor/aluSrc St1
/single_cycle_mips_tb2/processor/loadImm St0
/single_cycle_mips_tb2/processor/alu_zero St0
/single_cycle_mips_tb2/processor/aluOp 010 010 110 010
/single_cycle_mips_tb2/processor/read_reg1 0011 0000 0001 0011
/single_cycle_mips_tb2/processor/read_reg2 0011 0000 0010 0011
/single_cycle_mips_tb2/processor/alu_res 0000000000000101 0000000000000000 1111111111111111 0000000000000101
/single_cycle_mips_tb2/processor/alu_b 0000000000000001 0000000000000000 0000000000010100 0000000000000001
/single_cycle_mips_tb2/processor/alu_a 0000000000000100 0000000000000000 0000000000010011 0000000000000100
/single_cycle_mips_tb2/processor/write_back 0000000000000101 0000000000000000 1111111111111111 0000000000000101
/single_cycle_mips_tb2/processor/read_data1 0000000000000100 0000000000000000 0000000000010011 0000000000000100
/single_cycle_mips_tb2/processor/read_data2 0000000000000100 0000000000000000 0000000000010100 0000000000000100
/single_cycle_mips_tb2/processor/next_pc 0000000101 0000000100 0000000101
/single_cycle_mips_tb2/processor/branch_addr 0000000110 0000000111 0000000110
/single_cycle_mips_tb2/processor/jump_addr 0011001100 0000000011 0001001000 0011001100


/single_cycle_mips_tb2/processor/clk St0
/single_cycle_mips_tb2/processor/pc 0000000011 0000000101 0000000110 0000000011
/single_cycle_mips_tb2/processor/regWrite St0
/single_cycle_mips_tb2/processor/regDst St0
/single_cycle_mips_tb2/processor/branch St1
/single_cycle_mips_tb2/processor/branchNot St0
/single_cycle_mips_tb2/processor/jump St0
/single_cycle_mips_tb2/processor/aluSrc St0
/single_cycle_mips_tb2/processor/loadImm St0
/single_cycle_mips_tb2/processor/alu_zero St1
/single_cycle_mips_tb2/processor/aluOp 110 010 110
/single_cycle_mips_tb2/processor/read_reg1 0001 0001 0000 0001
/single_cycle_mips_tb2/processor/read_reg2 0010 0001 0000 0010
/single_cycle_mips_tb2/processor/alu_res 0000000000000000 0000000000010100 0000000000000000
/single_cycle_mips_tb2/processor/alu_b 0000000000010100 0000000000000001 0000000000000000 0000000000010100
/single_cycle_mips_tb2/processor/alu_a 0000000000010100 0000000000010011 0000000000000000 0000000000010100
/single_cycle_mips_tb2/processor/write_back 0000000000000000 0000000000010100 0000000000000000
/single_cycle_mips_tb2/processor/read_data1 0000000000010100 0000000000010011 0000000000000000 0000000000010100
/single_cycle_mips_tb2/processor/read_data2 0000000000010100 0000000000010011 0000000000000000 0000000000010100
/single_cycle_mips_tb2/processor/next_pc 0000000111 0000000110 0000000011 0000000111
/single_cycle_mips_tb2/processor/branch_addr 0000000111 0000000111 0000000111 0000000111
/single_cycle_mips_tb2/processor/jump_addr 0001001000 0001000100 0000000011 0001001000



- The reason for the waveform being that long is, there is an iteration, it can be observed from pc value that it goes up and down for a while till branching breaks the loop. In each iteration there is an increment.

- single_cycle_mips_tb3

- This test bench tests, jal, jr and jump. Jal instruction must save the next address to the 15th register.

assembly program

```
1   li $1, 117
2   addi $2, $1, 38
3   add $3, $1, $0
4   jal 8
5   add $5, $4, $0
6   addi $6, $3, 1
7   sub $7, $5, $6
8   j 11
9   addi $3, $3, 15
10  or $4, $3, $2
11  jr $15
12  add $8, $3, $1
```

generated machine code

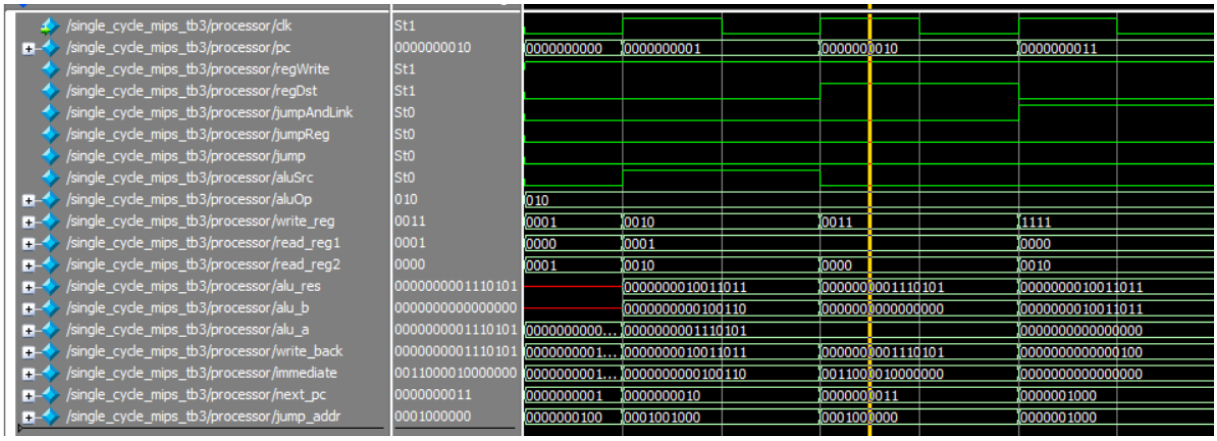
```
1   00111100000001000000000111010100
2   00100000010010000000000010011000
3   00000000010000001100001000000000
4   00001100000010000000000000000000
5   00000001000000010100001000000000
6   00100000110110000000000000000100
7   00000001010110011100001000100000
8   00001000000001011000000000000000
9   00100000110011000000000000111100
10  00000000110010010000001001010000
11  00000011110000000000000010000000
12  00000000110001100000001000000000
```

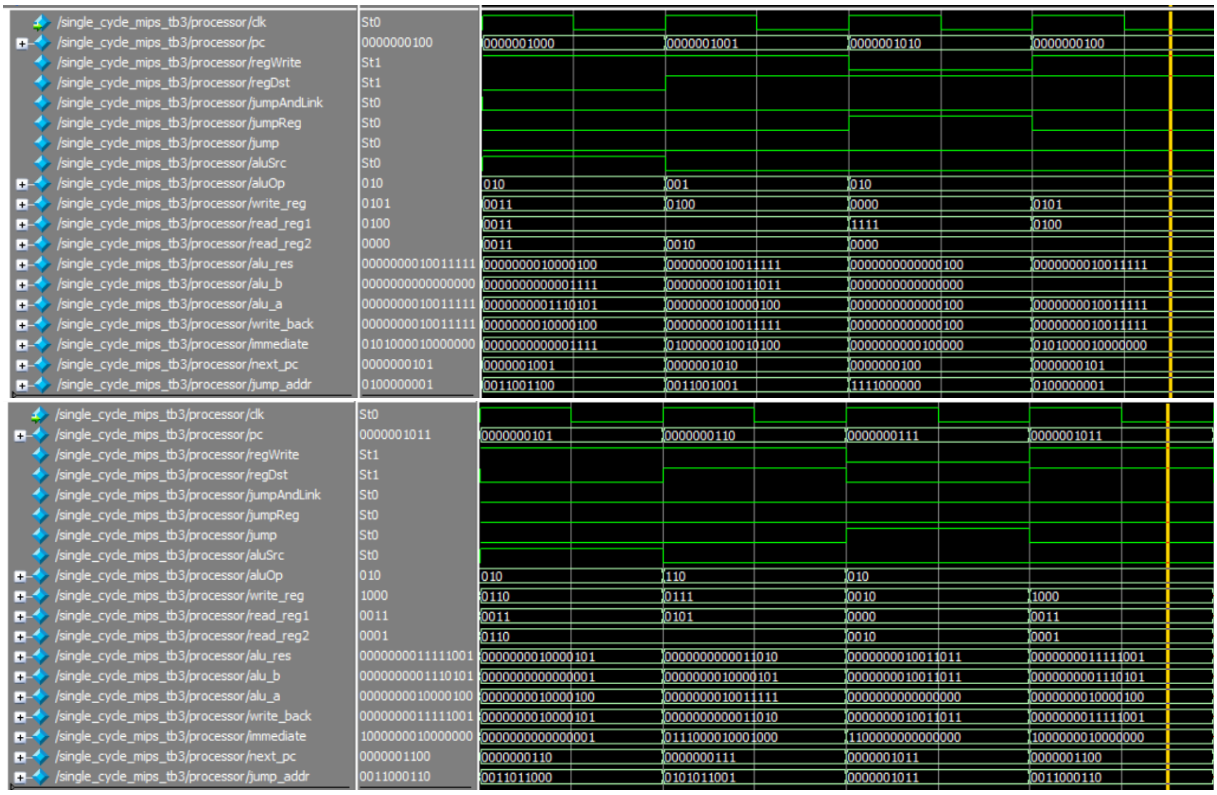
Result

Registers

```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/single_cycle_mips_tb3/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000001110101
6   0000000010011011
7   0000000010000100
8   0000000010011111
9   0000000010011111
10  0000000010000101
11  0000000000011010
12  0000000011111001
13  xxxxxxxxxxxxxxxx
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  0000000000000100
20
```

Waveform





- jal saves the next address to the register 15. Using jr 15 goes to the next address from the jal instruction. With another jump instruction infinite loop is prevented.

- single_cycle_mips_tb4

- This test bench tests, memory operations with iterations. Almost all type of instructions are tested. It stores to memory in a loop, then reads them in a loop.

assembly code

```
1   li $1, 511
2   sw $1, 0($0)
3   lw $2, 0($0)
4   addi $3, $2, $1
5   sw $3, 10($0)
6   li $4, 0
7   li $5, 10
8   addi $4, $4, 1
9   beq $4, $5, 2
10  sw $4, 4($4)
11  j 7
12  li $6, 1
13  lw $7, 4($6)
14  addi $7, $7, 1
15  slt $8, $7, $5
16  bne $8, $0, 1
17  j 12
18  sub $9, $6, $4
```

generated machine code

```
1   00111100000001000000011111111100
2   10101100000001000000000000000000
3   10001100000001000000000000000000
4   00100000100011000000000000000100
5   10101100000011000000000000101000
6   00111100000100000000000000000000
7   00111100000101000000000000101000
8   00100001000100000000000000000100
9   00010001000101000000000000001000
10  10101101000100000000000000010000
11  00001000000001110000000000000000
12  00111100000110000000000000000100
13  10001101100111000000000000010000
14  00100001110111000000000000000100
15  00000001110101100000001010100000
16  00010110000000000000000000000100
17  00001000000011000000000000000000
18  00000001100100100100001000100000
```

Result

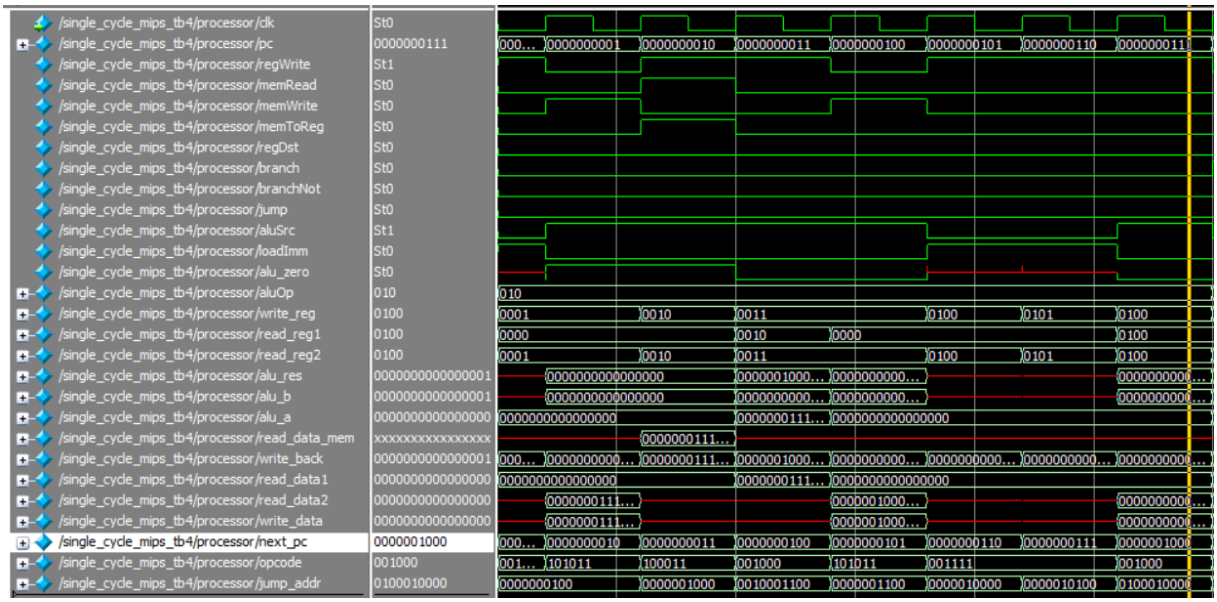
Registers

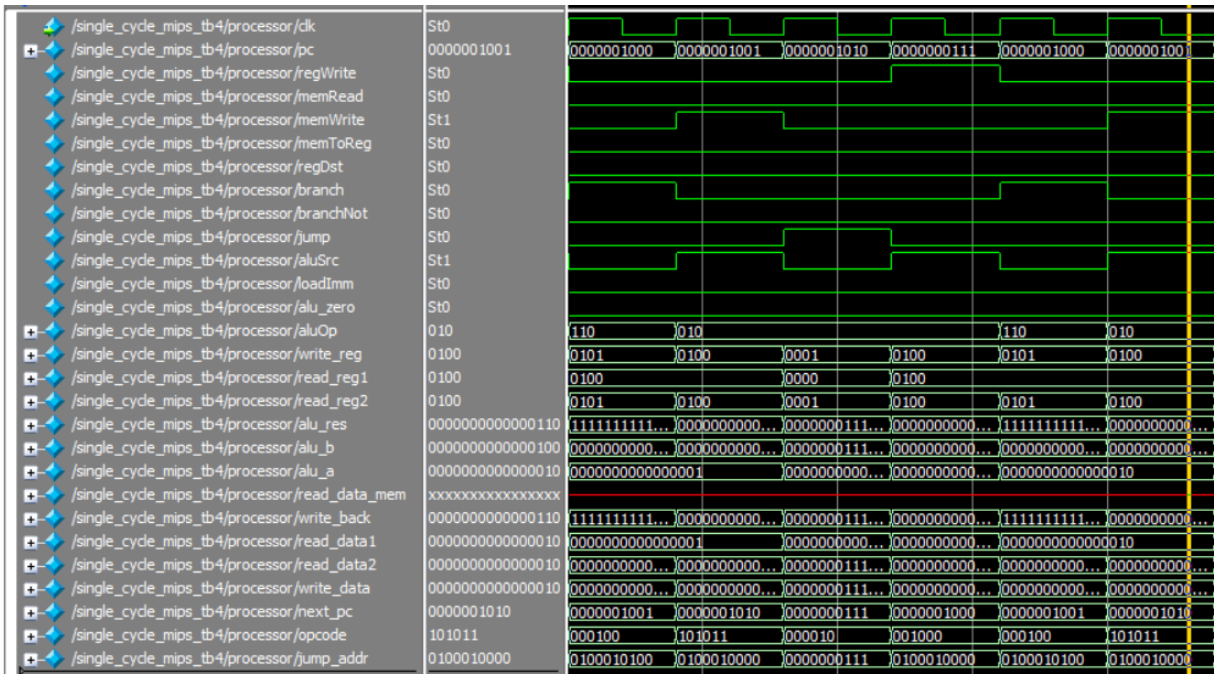
```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/single_cycle_mips_tb4/processor/regs/registers
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000000000000
5   0000000111111111
6   0000000111111111
7   0000001000000000
8   0000000000001010
9   0000000000001010
10  0000000000000001
11  0000000000000010
12  0000000000000001
13  1111111111110111
14  xxxxxxxxxxxxxxxx
15  xxxxxxxxxxxxxxxx
16  xxxxxxxxxxxxxxxx
17  xxxxxxxxxxxxxxxx
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
```

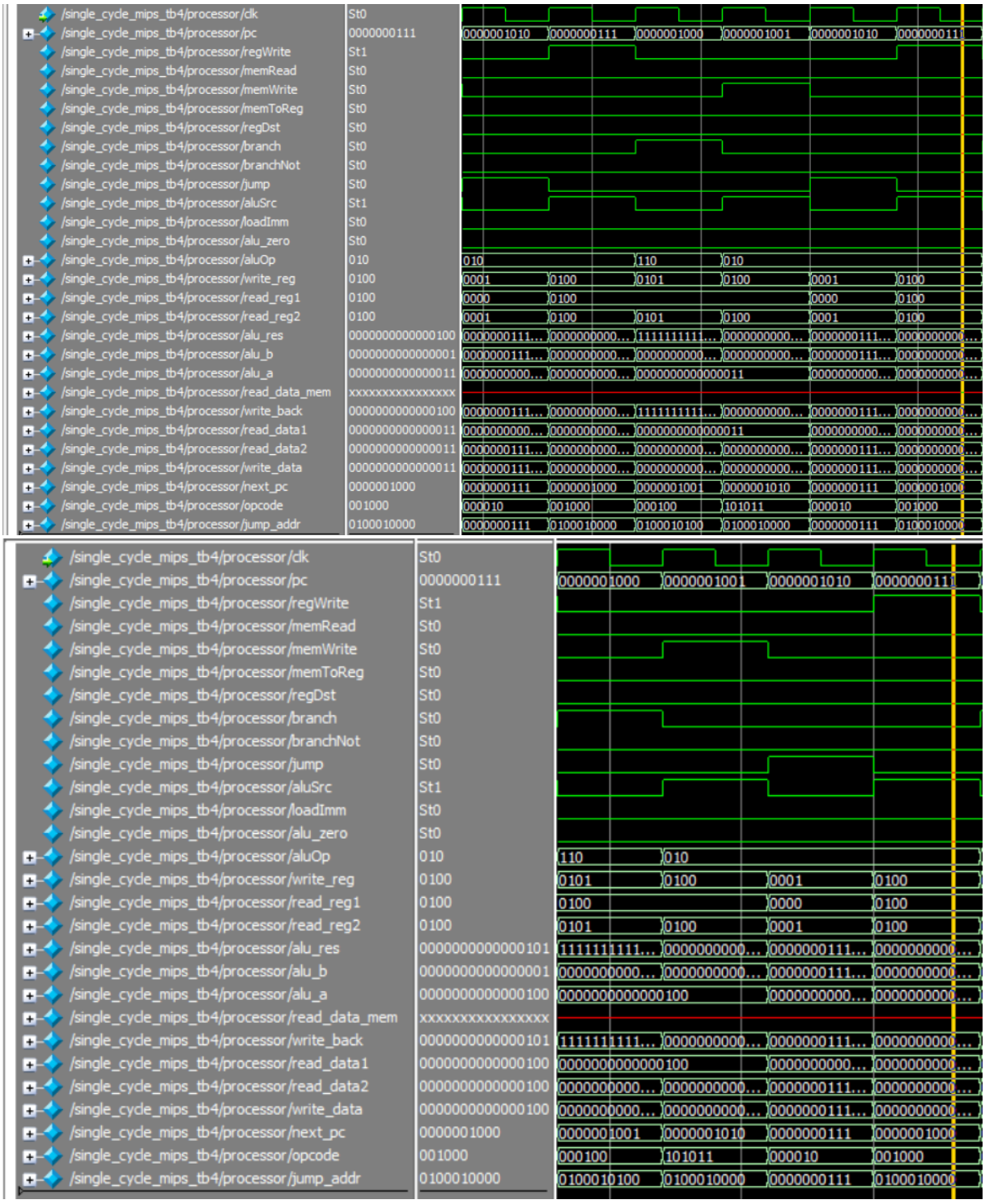
Memory

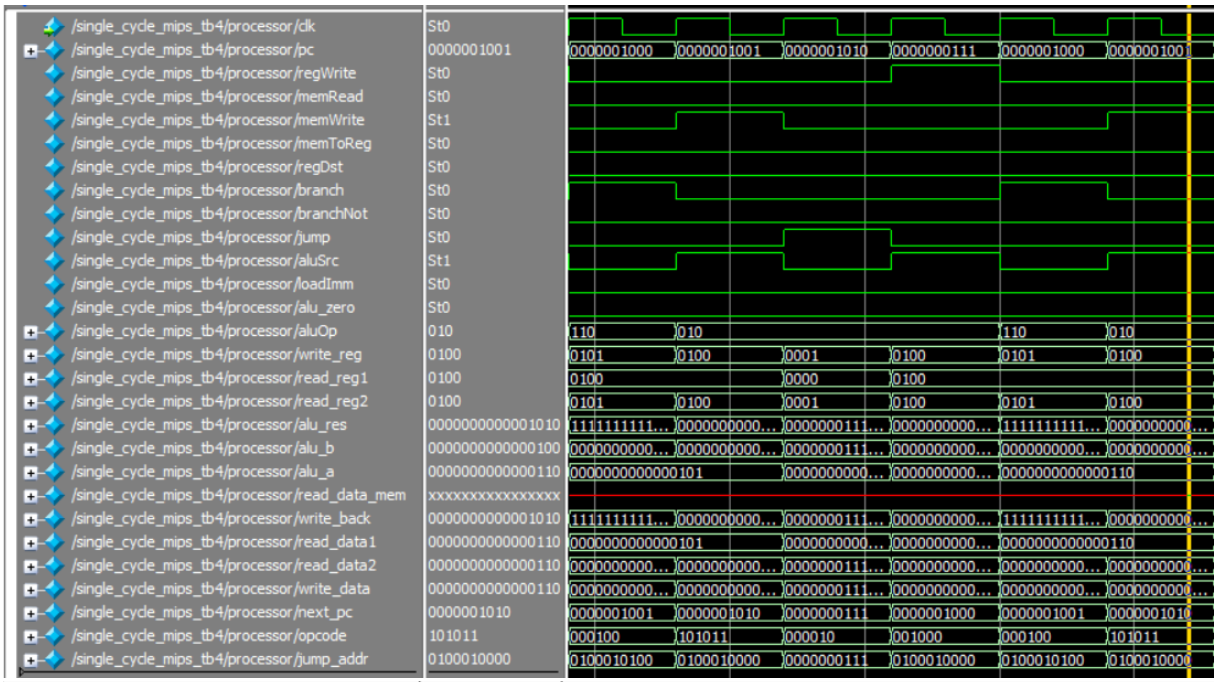
```
1   // memory data file (do not edit the following line - required for mem load use)
2   // instance=/single_cycle_mips_tb4/processor/data_mem/memory
3   // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4   0000000111111111
5   xxxxxxxxxxxxxxxx
6   xxxxxxxxxxxxxxxx
7   xxxxxxxxxxxxxxxx
8   xxxxxxxxxxxxxxxx
9   0000000000000001
10  0000000000000010
11  0000000000000011
12  0000000000000100
13  0000000000000101
14  0000000000000110
15  0000000000000111
16  0000000000001000
17  0000000000001001
18  xxxxxxxxxxxxxxxx
19  xxxxxxxxxxxxxxxx
20  xxxxxxxxxxxxxxxx
21  xxxxxxxxxxxxxxxx
```

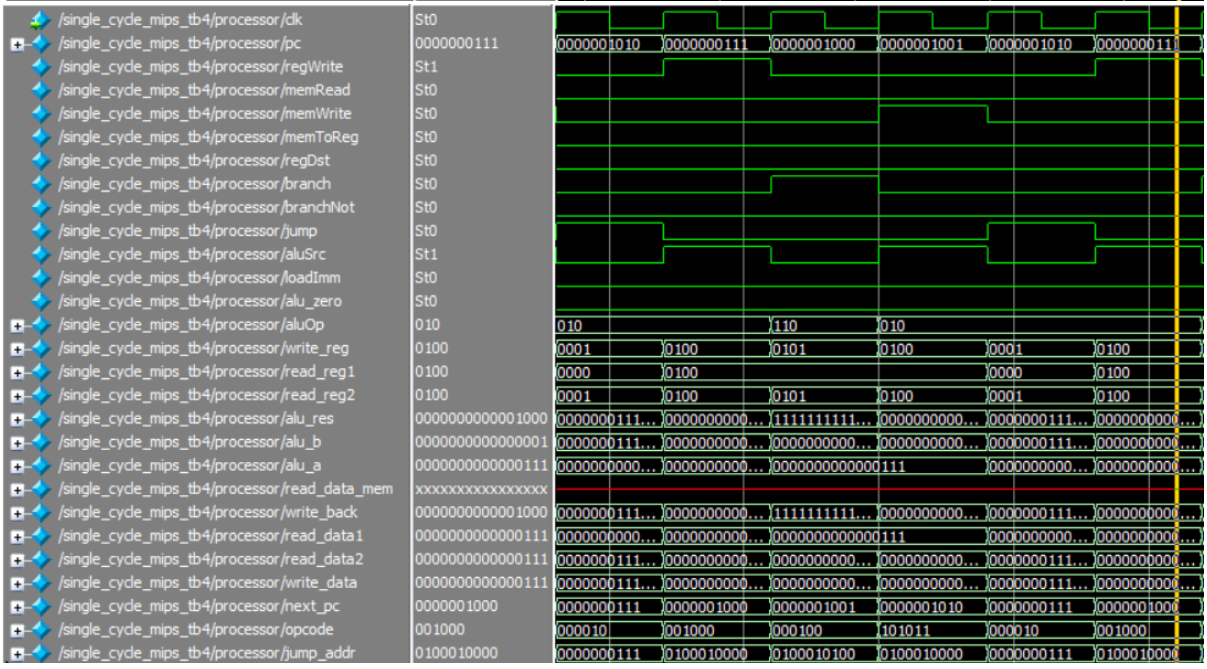
Waveform

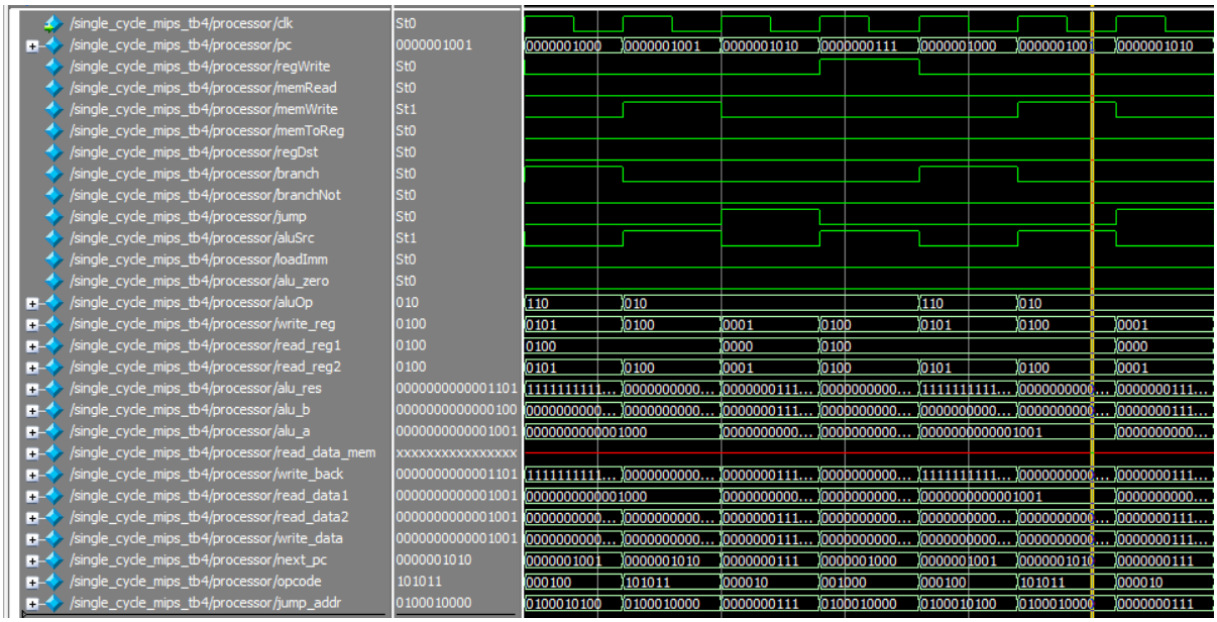


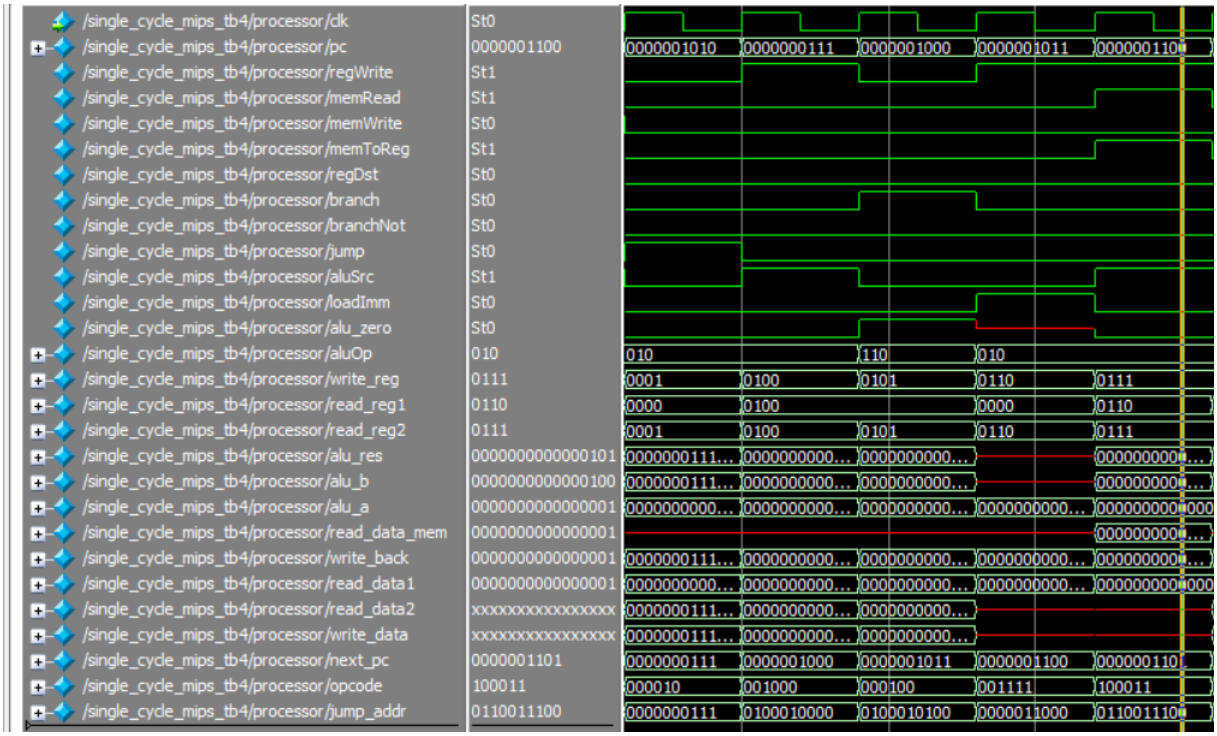

/single_cycle_mips_tb4/processor/clk St0
/single_cycle_mips_tb4/processor/pc 0000001001
/single_cycle_mips_tb4/processor/regWrite St0
/single_cycle_mips_tb4/processor/memRead St0
/single_cycle_mips_tb4/processor/memWrite St1
/single_cycle_mips_tb4/processor/memToReg St0
/single_cycle_mips_tb4/processor/regDst St0
/single_cycle_mips_tb4/processor/branch St0
/single_cycle_mips_tb4/processor/branchNot St0
/single_cycle_mips_tb4/processor/jump St0
/single_cycle_mips_tb4/processor/aluSrc St1
/single_cycle_mips_tb4/processor/loadImm St0
/single_cycle_mips_tb4/processor/alu_zero St0
/single_cycle_mips_tb4/processor/aluOp 010
/single_cycle_mips_tb4/processor/write_reg 0100
/single_cycle_mips_tb4/processor/read_reg1 0100
/single_cycle_mips_tb4/processor/read_reg2 0100
/single_cycle_mips_tb4/processor/alu_res 0000000000000110
/single_cycle_mips_tb4/processor/alu_b 0000000000000100
/single_cycle_mips_tb4/processor/alu_a 0000000000000010
/single_cycle_mips_tb4/processor/read_data_mem xxxxxxxxxxxxxxxx
/single_cycle_mips_tb4/processor/write_back 0000000000000110
/single_cycle_mips_tb4/processor/read_data1 0000000000000010
/single_cycle_mips_tb4/processor/read_data2 0000000000000010
/single_cycle_mips_tb4/processor/write_data 0000000000000010
/single_cycle_mips_tb4/processor/next_pc 0000001010
/single_cycle_mips_tb4/processor/opcode 101011
/single_cycle_mips_tb4/processor/jump_addr 0100010000

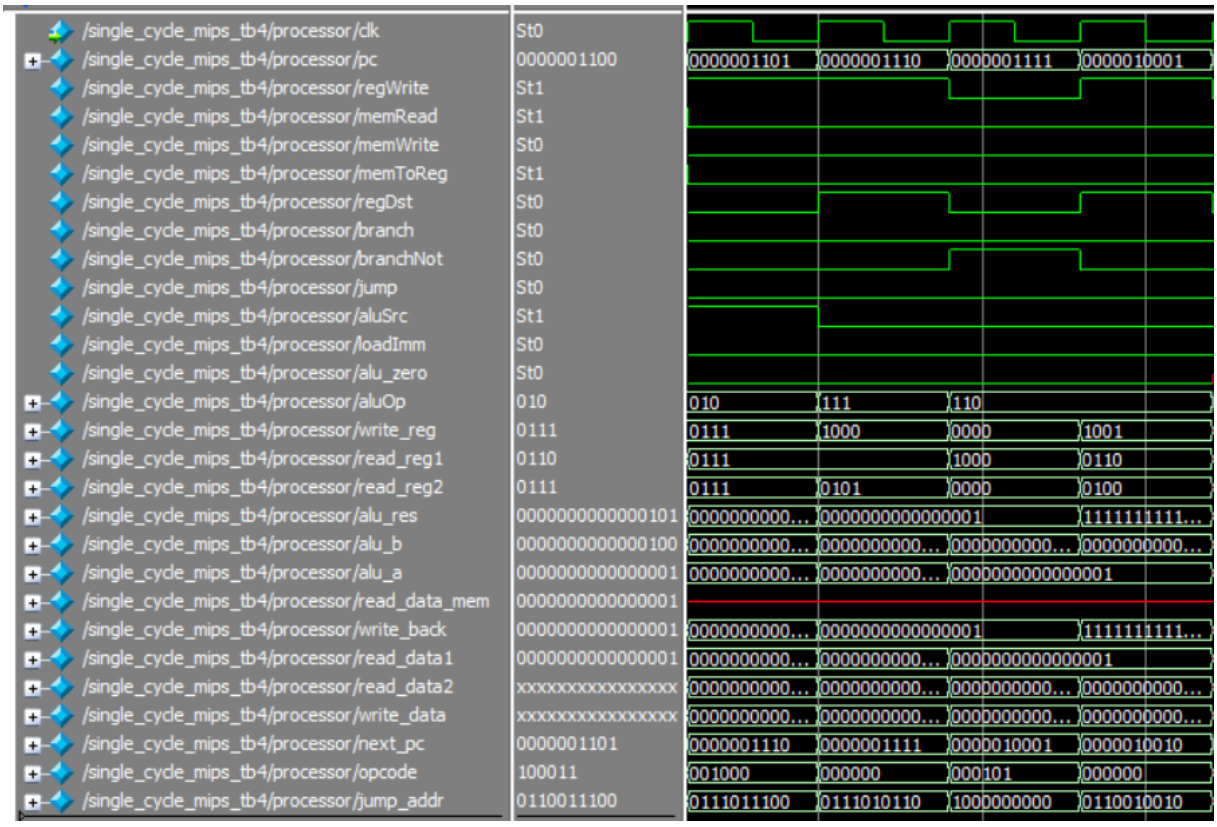

/single_cycle_mips_tb4/processor/clk St0
/single_cycle_mips_tb4/processor/pc 0000000111
/single_cycle_mips_tb4/processor/regWrite St1
/single_cycle_mips_tb4/processor/memRead St0
/single_cycle_mips_tb4/processor/memWrite St0
/single_cycle_mips_tb4/processor/memToReg St0
/single_cycle_mips_tb4/processor/regDst St0
/single_cycle_mips_tb4/processor/branch St0
/single_cycle_mips_tb4/processor/branchNot St0
/single_cycle_mips_tb4/processor/jump St0
/single_cycle_mips_tb4/processor/aluSrc St1
/single_cycle_mips_tb4/processor/loadImm St0
/single_cycle_mips_tb4/processor/alu_zero St0
/single_cycle_mips_tb4/processor/aluOp 010
/single_cycle_mips_tb4/processor/write_reg 0100
/single_cycle_mips_tb4/processor/read_reg1 0100
/single_cycle_mips_tb4/processor/read_reg2 0100
/single_cycle_mips_tb4/processor/alu_res 0000000000000100
/single_cycle_mips_tb4/processor/alu_b 0000000000000001
/single_cycle_mips_tb4/processor/alu_a 0000000000000011
/single_cycle_mips_tb4/processor/read_data_mem xxxxxxxxxxxxxxxx
/single_cycle_mips_tb4/processor/write_back 0000000000000100
/single_cycle_mips_tb4/processor/read_data1 0000000000000011
/single_cycle_mips_tb4/processor/read_data2 0000000000000011
/single_cycle_mips_tb4/processor/write_data 0000000000000011
/single_cycle_mips_tb4/processor/next_pc 0000001000
/single_cycle_mips_tb4/processor/opcode 001000
/single_cycle_mips_tb4/processor/jump_addr 0100010000
/single_cycle_mips_tb4/processor/clk St0
/single_cycle_mips_tb4/processor/pc 0000000111
/single_cycle_mips_tb4/processor/regWrite St1
/single_cycle_mips_tb4/processor/memRead St0
/single_cycle_mips_tb4/processor/memWrite St0
/single_cycle_mips_tb4/processor/memToReg St0
/single_cycle_mips_tb4/processor/regDst St0
/single_cycle_mips_tb4/processor/branch St0
/single_cycle_mips_tb4/processor/branchNot St0
/single_cycle_mips_tb4/processor/jump St0
/single_cycle_mips_tb4/processor/aluSrc St1
/single_cycle_mips_tb4/processor/loadImm St0
/single_cycle_mips_tb4/processor/alu_zero St0
/single_cycle_mips_tb4/processor/aluOp 010
/single_cycle_mips_tb4/processor/write_reg 0100
/single_cycle_mips_tb4/processor/read_reg1 0100
/single_cycle_mips_tb4/processor/read_reg2 0100
/single_cycle_mips_tb4/processor/alu_res 0000000000000101
/single_cycle_mips_tb4/processor/alu_b 0000000000000001
/single_cycle_mips_tb4/processor/alu_a 0000000000000100
/single_cycle_mips_tb4/processor/read_data_mem xxxxxxxxxxxxxxxx
/single_cycle_mips_tb4/processor/write_back 0000000000000101
/single_cycle_mips_tb4/processor/read_data1 0000000000000100
/single_cycle_mips_tb4/processor/read_data2 0000000000000100
/single_cycle_mips_tb4/processor/write_data 0000000000000100
/single_cycle_mips_tb4/processor/next_pc 0000001000
/single_cycle_mips_tb4/processor/opcode 001000
/single_cycle_mips_tb4/processor/jump_addr 0100010000


/single_cycle_mips_tb4/processor/clk St0
/single_cycle_mips_tb4/processor/pc 0000001001
/single_cycle_mips_tb4/processor/regWrite St0
/single_cycle_mips_tb4/processor/memRead St0
/single_cycle_mips_tb4/processor/memWrite St1
/single_cycle_mips_tb4/processor/memToReg St0
/single_cycle_mips_tb4/processor/regDst St0
/single_cycle_mips_tb4/processor/branch St0
/single_cycle_mips_tb4/processor/branchNot St0
/single_cycle_mips_tb4/processor/jump St0
/single_cycle_mips_tb4/processor/aluSrc St1
/single_cycle_mips_tb4/processor/loadImm St0
/single_cycle_mips_tb4/processor/alu_zero St0
/single_cycle_mips_tb4/processor/aluOp 010
/single_cycle_mips_tb4/processor/write_reg 0100
/single_cycle_mips_tb4/processor/read_reg1 0100
/single_cycle_mips_tb4/processor/read_reg2 0100
/single_cycle_mips_tb4/processor/alu_res 0000000000001010
/single_cycle_mips_tb4/processor/alu_b 0000000000000100
/single_cycle_mips_tb4/processor/alu_a 0000000000000110
/single_cycle_mips_tb4/processor/read_data_mem xxxxxxxxxxxxxxxx
/single_cycle_mips_tb4/processor/write_back 0000000000001010
/single_cycle_mips_tb4/processor/read_data1 0000000000000110
/single_cycle_mips_tb4/processor/read_data2 0000000000000110
/single_cycle_mips_tb4/processor/write_data 0000000000000110
/single_cycle_mips_tb4/processor/next_pc 0000001010
/single_cycle_mips_tb4/processor/opcode 101011
/single_cycle_mips_tb4/processor/jump_addr 0100010000


/single_cycle_mips_tb4/processor/clk St0
/single_cycle_mips_tb4/processor/pc 0000000111
/single_cycle_mips_tb4/processor/regWrite St1
/single_cycle_mips_tb4/processor/memRead St0
/single_cycle_mips_tb4/processor/memWrite St0
/single_cycle_mips_tb4/processor/memToReg St0
/single_cycle_mips_tb4/processor/regDst St0
/single_cycle_mips_tb4/processor/branch St0
/single_cycle_mips_tb4/processor/branchNot St0
/single_cycle_mips_tb4/processor/jump St0
/single_cycle_mips_tb4/processor/aluSrc St1
/single_cycle_mips_tb4/processor/loadImm St0
/single_cycle_mips_tb4/processor/alu_zero St0
/single_cycle_mips_tb4/processor/aluOp 010
/single_cycle_mips_tb4/processor/write_reg 0100
/single_cycle_mips_tb4/processor/read_reg1 0100
/single_cycle_mips_tb4/processor/read_reg2 0100
/single_cycle_mips_tb4/processor/alu_res 0000000000001000
/single_cycle_mips_tb4/processor/alu_b 0000000000000001
/single_cycle_mips_tb4/processor/alu_a 0000000000000111
/single_cycle_mips_tb4/processor/read_data_mem xxxxxxxxxxxxxxxx
/single_cycle_mips_tb4/processor/write_back 0000000000001000
/single_cycle_mips_tb4/processor/read_data1 0000000000000111
/single_cycle_mips_tb4/processor/read_data2 0000000000000111
/single_cycle_mips_tb4/processor/write_data 0000000000000111
/single_cycle_mips_tb4/processor/next_pc 0000001000
/single_cycle_mips_tb4/processor/opcode 001000
/single_cycle_mips_tb4/processor/jump_addr 0100010000


/single_cycle_mips_tb4/processor/clk
/single_cycle_mips_tb4/processor/pc
/single_cycle_mips_tb4/processor/regWrite
/single_cycle_mips_tb4/processor/memRead
/single_cycle_mips_tb4/processor/memWrite
/single_cycle_mips_tb4/processor/memToReg
/single_cycle_mips_tb4/processor/regDst
/single_cycle_mips_tb4/processor/branch
/single_cycle_mips_tb4/processor/branchNot
/single_cycle_mips_tb4/processor/jump
/single_cycle_mips_tb4/processor/aluSrc
/single_cycle_mips_tb4/processor/loadImm
/single_cycle_mips_tb4/processor/alu_zero
/single_cycle_mips_tb4/processor/aluOp
/single_cycle_mips_tb4/processor/write_reg
/single_cycle_mips_tb4/processor/read_reg1
/single_cycle_mips_tb4/processor/read_reg2
/single_cycle_mips_tb4/processor/alu_res
/single_cycle_mips_tb4/processor/alu_b
/single_cycle_mips_tb4/processor/alu_a
/single_cycle_mips_tb4/processor/read_data_mem
/single_cycle_mips_tb4/processor/write_back
/single_cycle_mips_tb4/processor/read_data1
/single_cycle_mips_tb4/processor/read_data2
/single_cycle_mips_tb4/processor/write_data
/single_cycle_mips_tb4/processor/next_pc
/single_cycle_mips_tb4/processor/opcode
/single_cycle_mips_tb4/processor/jump_addr


/single_cycle_mips_tb4/processor/clk
/single_cycle_mips_tb4/processor/pc
/single_cycle_mips_tb4/processor/regWrite
/single_cycle_mips_tb4/processor/memRead
/single_cycle_mips_tb4/processor/memWrite
/single_cycle_mips_tb4/processor/memToReg
/single_cycle_mips_tb4/processor/regDst
/single_cycle_mips_tb4/processor/branch
/single_cycle_mips_tb4/processor/branchNot
/single_cycle_mips_tb4/processor/jump
/single_cycle_mips_tb4/processor/aluSrc
/single_cycle_mips_tb4/processor/loadImm
/single_cycle_mips_tb4/processor/alu_zero
/single_cycle_mips_tb4/processor/aluOp
/single_cycle_mips_tb4/processor/write_reg
/single_cycle_mips_tb4/processor/read_reg1
/single_cycle_mips_tb4/processor/read_reg2
/single_cycle_mips_tb4/processor/alu_res
/single_cycle_mips_tb4/processor/alu_b
/single_cycle_mips_tb4/processor/alu_a
/single_cycle_mips_tb4/processor/read_data_mem
/single_cycle_mips_tb4/processor/write_back
/single_cycle_mips_tb4/processor/read_data1
/single_cycle_mips_tb4/processor/read_data2
/single_cycle_mips_tb4/processor/write_data
/single_cycle_mips_tb4/processor/next_pc
/single_cycle_mips_tb4/processor/opcode
/single_cycle_mips_tb4/processor/jump_addr


/single_cycle_mips_tb4/processor/clk
/single_cycle_mips_tb4/processor/pc
/single_cycle_mips_tb4/processor/regWrite
/single_cycle_mips_tb4/processor/memRead
/single_cycle_mips_tb4/processor/memWrite
/single_cycle_mips_tb4/processor/memToReg
/single_cycle_mips_tb4/processor/regDst
/single_cycle_mips_tb4/processor/branch
/single_cycle_mips_tb4/processor/branchNot
/single_cycle_mips_tb4/processor/jump
/single_cycle_mips_tb4/processor/aluSrc
/single_cycle_mips_tb4/processor/loadImm
/single_cycle_mips_tb4/processor/alu_zero
/single_cycle_mips_tb4/processor/aluOp
/single_cycle_mips_tb4/processor/write_reg
/single_cycle_mips_tb4/processor/read_reg1
/single_cycle_mips_tb4/processor/read_reg2
/single_cycle_mips_tb4/processor/alu_res
/single_cycle_mips_tb4/processor/alu_b
/single_cycle_mips_tb4/processor/alu_a
/single_cycle_mips_tb4/processor/read_data_mem
/single_cycle_mips_tb4/processor/write_back
/single_cycle_mips_tb4/processor/read_data1
/single_cycle_mips_tb4/processor/read_data2
/single_cycle_mips_tb4/processor/write_data
/single_cycle_mips_tb4/processor/next_pc
/single_cycle_mips_tb4/processor/opcode
/single_cycle_mips_tb4/processor/jump_addr
St0
0000001100
St1
St1
St0
St1
St0
St0
St0
St0
St1
St0
St0
010
0111
0110
0111
0000000000000101
0000000000000100
0000000000000001
0000000000000001
0000000000000001
0000000000000001
xxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxx
0000001101
100011
0110011100
0000001101
0000001110
0000001111
0000010001
0000010010
010
111
110
0111
1000
0000
1001
0111
1000
0110
0111
0101
0000
0100
0000000000...
0000000000000001
1111111111...
0000000000...
0000000000...
0000000000...
0000000000...
0000000000...
0000000000000001
0000000000...
0000000000000001
1111111111...
0000000000...
0000000000...
0000000000000001
0000000000...
0000000000...
0000000000...
0000000000...
0000000000...
0000000000...
0000000000...
0000000000...
0000001110
0000001111
0000010001
0000010010
001000
000000
000101
000000
0111011100
0111010110
1000000000
0110010010