2020-2021

MIPS Processor Top Module Design

VLSI System Design | ECE-718

Design Report

SoC Team Member(s)-

1. Ayut Ghosh (17EC8003)
2. Pawan Kumar (17EC8022)
3. Vaishnavi Matte (17EC8026)
4. Shreyan Datta (17EC8033)
5. Sandipan Chatterjee (17EC8063)

TABLE OF CONTENTS

MIPS Processor -

MIPS stands for Microprocessor without Interlocked Pipelined Stages. It is a 32-bit processor based on RISC Architecture. There are three types for MIPS architectures that are Single-cycle, Multi-cycle, and Pipelined architectures. Single Cycle Processor executes singe instruction at one time. Multi-Cycle Processor divides a single instruction into various submodules and processes them individually. Pipelined Processor processes various instruction at a single clock cycle.

The MIPS Processor Design is comprised of the sub-modules designed by the other groups. It includes ALU Design, ALU Controller Design, Combinational Block Design, Concatenate Module Design, Sequence Controller Design, Multiplexor Design, Program Counter Design, RAM Design, Register File Design, Scalable Register Design, Shift-By-Two Module Design, Sign Extend Module Design, State Register Design Module(s).

The design code is a structural implementation of the processor in Verilog HDL, by connecting all sub-modules in accordance with the Processor drawing shown in Fig 1.

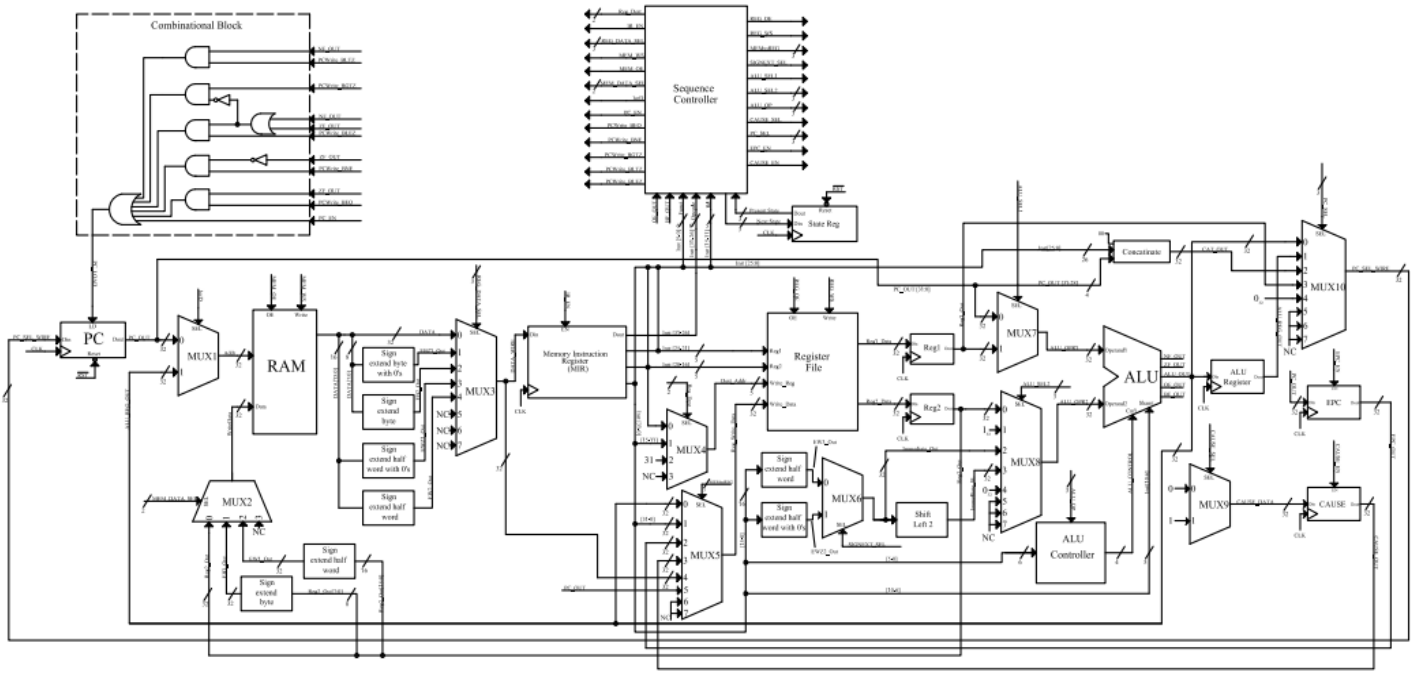


Fig 1: MIPS Processor Top Level Design

Source Code for Top Module of MIPS Processor –

```
`timescale 1ns/1ns
module risc(CLK, RST_n);
	input CLK,RST_n;
	wire OF_OUT, BF_OUT, NF_OUT, ZF_OUT, PC_EN, IorD, MEM_OE, MEM_WS, IR_EN, REG_OE, REG_WS, SIGNEXT_SEL, ALU_SEL1, EPC_EN, CAUSE_SEL, CAUSE_EN, PCWrite_BEQ, PCWrite_BNE, PCWrite_BGTZ, PCWrite_BLTZ, PCWrite_BLEZ, PC_LOAD;
	wire [2:0] present_state, next_state, PC_SEL, REG_DATA_SEL, ALU_SEL2, ALU_OP, MEMtoREG;
	wire [1:0] MEM_DATA_SEL, Reg_Dest;
	wire [31:0] Inst, PC_OUT, PC_SEL_WIRE, ALU_REG_OUT, Addr, DATA, WriteData, Reg2_Out, EB1_Out, EW1_Out, EBZ1_Out, EB2_Out, EWZ1_Out, EW2_Out, DATA_WIRE, EPC_OUT, CAUSE_OUT, Reg_Write_Data, Reg1_Data, Reg2_Data, EW3_Out, EWZ2_Out, Immediate_Out, Immediate_SL, Reg1_Out, ALU_OPR1, ALU_OPR2, ALU_OUT, CAUSE_DATA, Inst_SL, CAT_OUT, NC;
	wire [4:0] Dest_Addr;
	wire [3:0] ALU_CONTROL;
	wire [5:0] STATE;

	control CTL(Inst[31:26], present_state[2:0], OF_OUT, BF_OUT, Inst[5:0], Inst[15:11], next_state[2:0], PC_EN, PC_SEL[2:0], IorD, MEM_DATA_SEL[1:0], MEM_OE, MEM_WS, REG_DATA_SEL[2:0], IR_EN, Reg_Dest[1:0], MEMtoREG[2:0], REG_OE, REG_WS, SIGNEXT_SEL, ALU_SEL1, ALU_SEL2[2:0], ALU_OP[2:0], EPC_EN, CAUSE_SEL, CAUSE_EN, PCWrite_BEQ, PCWrite_BNE, PCWrite_BGTZ, PCWrite_BLTZ, PCWrite_BLEZ, );
	StateReg SRG(present_state[2:0], next_state[2:0], CLK, RST_n);
	comb CMB(NF_OUT, ZF_OUT, PCWrite_BLTZ, PCWrite_BGTZ, PCWrite_BLEZ, PCWrite_BNE, PCWrite_BEQ, PC_EN, PC_LOAD);
	pc PCR(PC_OUT[31:0], RST_n, CLK, PC_SEL_WIRE[31:0], PC_LOAD);
	mux2 MUX1(PC_OUT[31:0], ALU_REG_OUT[31:0], IorD, Addr[31:0]);
	RAM MEM1(DATA[31:0], Addr[7:0], WriteData[31:0], MEM_OE, MEM_WS);
	mux4 MUX2(Reg2_Out[31:0], EB1_Out[31:0], EW1_Out[31:0], NC[31:0], MEM_DATA_SEL[1:0], WriteData[31:0]);
	sign_extend8 EB1(Reg2_Out[7:0], EB1_Out[31:0]);
	sign_extend16 EW1(Reg2_Out[15:0], EW1_Out[31:0]);
	sign_extend8_zero EBZ1(DATA[7:0], EBZ1_Out[31:0]);
	sign_extend8 EB2(DATA[7:0], EB2_Out[31:0]);
	sign_extend16_zero EWZ1(DATA[15:0], EWZ1_Out[31:0]);
	sign_extend16 EW2(DATA[15:0], EW2_Out[31:0]);
	mux8 MUX3(DATA[31:0], EBZ1_Out[31:0], EB2_Out[31:0], EWZ1_Out[31:0], EW2_Out[31:0], NC[31:0], NC[31:0], NC[31:0], REG_DATA_SEL[2:0], DATA_WIRE[31:0]);
    scale_reg_en #(32)MIR(DATA_WIRE[31:0],IR_EN,CLK,Inst[31:0]);
    mux4 MUX4(Inst[20:16], Inst[15:11], 5'b11111, NC[31:0], Reg_Dest[1:0], Dest_Addr[4:0]);
    mux8 MUX5(ALU_OUT[31:0], Inst[31:0], EPC_OUT[31:0], CAUSE_OUT[31:0], DATA_WIRE[31:0], PC_OUT[31:0], NC[31:0], NC[31:0], MEMtoREG[2:0], Reg_Write_Data[31:0]);
    REG_FILE MEM2(Reg1_Data[31:0], Reg2_Data[31:0], Inst[25:21], Inst[20:16], Dest_Addr[4:0], Reg_Write_Data[31:0], REG_OE, REG_WS);
    sign_extend16 EW3(Inst[15:0], EW3_Out[31:0]);
    sign_extend16_zero EWZ2(Inst[15:0], EWZ2_Out[31:0]);
    mux2 MUX6(EW3_Out[31:0], EWZ2_Out[31:0], SIGNEXT_SEL, Immediate_Out[31:0]);
    shift_left2 SLL1(Immediate_Out[31:0], Immediate_SL[31:0]);
    scale_reg #(32)REG1(Reg1_Data[31:0], CLK, Reg1_Out[31:0]);
    scale_reg #(32)REG2(Reg2_Data[31:0], CLK, Reg2_Out[31:0]);
    mux2 MUX7(PC_OUT[31:0], Reg1_Out[31:0], ALU_SEL1, ALU_OPR1[31:0]);
    mux8 MUX8(Reg2_Out[31:0], 32'b1, Immediate_Out[31:0], Immediate_SL[31:0], 32'b0, NC[31:0], NC[31:0], NC[31:0], ALU_SEL2[2:0], ALU_OPR2[31:0]);
    alu_cont CONT(Inst[5:0], ALU_OP[2:0], ALU_CONTROL[3:0]);
    alu ALU(ALU_OPR1[31:0], ALU_OPR2[31:0], ALU_CONTROL[3:0], Inst[10:6], ALU_OUT[31:0], NF_OUT, ZF_OUT, OF_OUT, BF_OUT);
    scale_reg #(32)REGA(ALU_OUT[31:0], CLK, ALU_REG_OUT[31:0]);
    mux2 MUX9(32'b0, 32'b1, CAUSE_SEL, CAUSE_DATA[31:0]);
    scale_reg_en #(32)REGC(CAUSE_DATA[31:0], CAUSE_EN, CLK, CAUSE_OUT[31:0]);
    scale_reg_en #(32)REGE(PC_OUT[31:0], EPC_EN, CLK, EPC_OUT[31:0]);
    concatinate CAT(Inst[25:0], PC_OUT[31:28], CAT_OUT[31:0]);
    mux8 MUX10(ALU_OUT[31:0], ALU_REG_OUT[31:0], CAT_OUT[31:0], Reg1_Out[31:0], 32'b0, NC[31:0], NC[31:0], NC[31:0], PC_SEL[2:0], PC_SEL_WIRE[31:0]);

endmodule
```

Test Bench for simulation of MIPS Processor –

```
`timescale 1 ns / 1 ns          //Timescale definition
module test_bench_top_level();

   //port declaration
   reg CLK, RST_n;

   risc UUT(CLK, RST_n);    //cpu installation

   initial
   //parameters output to the simulation log file
   $monitor ("%d CLK = %b RST = %b PC_out = %d RAM_Addr = %d RAM_Data = %d RAM_WS = %b RAM_OE = %b
RAM_out = %b IR_out = %b REG_WS = %b REG_OE = %b RegData = %d RegAddr = %d A_Reg = %d B_Reg = %d
ALU_OPR1 = %d ALU_OPR2 = %d ALU_OUT = %d NF = %b ZF = %b OF = %b BF = %b EPC_EN = %d CAUSE_EN =
%d STATE = %d", $time, CLK, RST_n, UUT.PCR.Dout, UUT.MUX1.OUT, UUT.MUX2.OUT, UUT.CTL.MEM_WS,
UUT.CTL.MEM_OE, UUT.MEM1.DataOut, UUT.MIR.OUT, UUT.CTL.REG_WS, UUT.CTL.REG_OE,
UUT.MUX5.OUT, UUT.MUX4.OUT, UUT.REG1.OUT,UUT.REG2.OUT, UUT.MUX7.OUT, UUT.MUX8.OUT,
UUT.ALU.ALU_OUT, UUT.ALU.NF_OUT, UUT.ALU.ZF_OUT, UUT.ALU.OF_OUT, UUT.ALU.BF_OUT,
UUT.CTL.EPC_EN, UUT.CTL.CAUSE_EN, UUT.CTL.STATE);

   initial CLK =1'b0;
   always begin
   $write("\n");       //new line used for clarity

   #25 CLK = ~CLK; //clock generator
   end

   initial begin
   $readmemh("C:\\Users\\pkran\\RISC_complete_simu\\RAM_Data.txt", UUT.MEM1.mem);    //initialize RAM with data
from RAM_Data.txt
   end

   initial begin
   $readmemh("C:\\Users\\pkran\\RISC_complete_simu\\RegFile_Data.txt", UUT.MEM2.mem);      //initialize Register File
with data from RegFile_Data.txt
   end

   initial begin
      //$vcdpluson;     //Enable graphical viewer

      RST_n= 1'b1;
   #5  RST_n = 1'b0;

   #5  RST_n = 1'b1;

   #15000  $finish;        //give enough time for the processor to execute all instructions before terminating simulation

   end
endmodule
```

Simulation Setup:

1. **I-Verilog (For Functional Verification):**

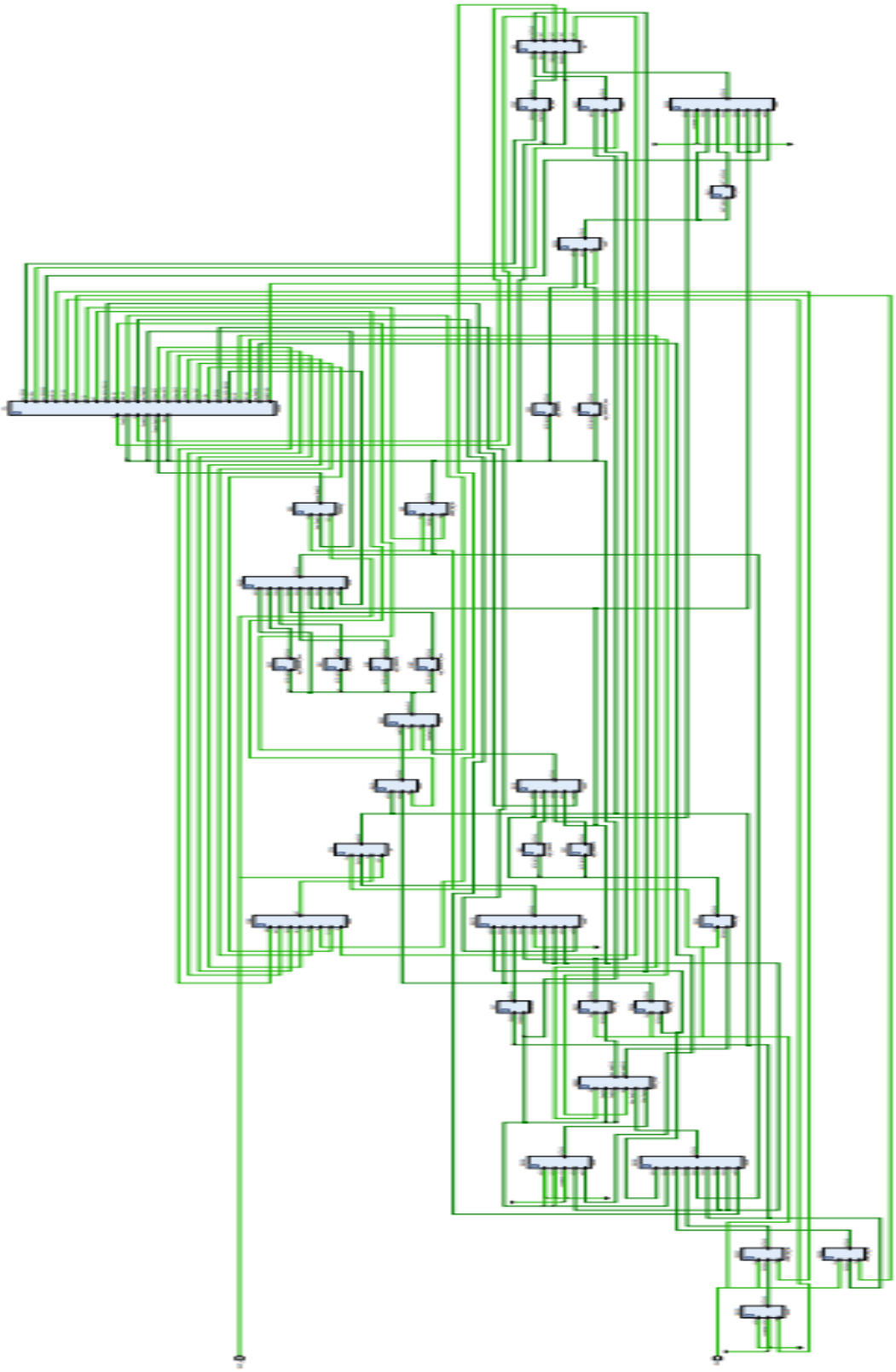
   Step 1: Download and Install everything by clicking the checkboxes from the file: " http://bleyer.org/icarus/iverilog-v11-20200824-x64_setup.exe "
   Step 2: Create a shortcut to your Desktop
   Step 3: Open the Iverilog Folder
   Step 4: Copy and Paste the Verilog Code File and Verilog Test File at your preferred place within the folder
   Step 5: Click on the address bar --> type cmd --> Terminal Opens --> Write " iverilog -o mysim complete.v RISCtb.v " --> Enter
   Step 6: Type: " vvp mysim "
   See the results!!

   _____
   Note: Code and Test Files should be saved with a .v extension
   If any problem, try writing the code in Notepad++ : https://notepad-plus-plus.org/downloads/v7.8.8/

2. **Xilinx Vivado (For Functional Verification & Schematic):**

   Step 1: Install Vivado → Open → Create a new project.
   Step 2: Upload the Source & Test Bench (.v) files.
   Step 3: Run Behavioral Simulation (For Functional Verification)
   Step 4: Create Schematic (For generating schematic of the processor)

Generated Schematic:



NOTE: High Resolution view of the schematic is attached with the email.

Analysis of the Output:

We have used 25 ports to be displayed upon simulation (in $monitor). They are described as follows: -

Signal Name	Description
CLK	System clock
RST	Active low system reset
PC_out	Program Counter output
RAM_Addr	RAM input address derived from Mux1
RAM_Data	RAM input data derived from Mux2
RAM_WS	RAM WS signal generated by the Sequence Controller
RAM_OE	RAM OE signal generated by the Sequence Controller
RAM_out	RAM data output
IR_out	Memory Instruction Register output
REG_WS	Register File WS signal generated by the Sequence Controller
REG_OE	Register File OE signal generated by the Sequence Controller
RegData	Register File write data derived from Mux5
RegAddr	Register File write address derived from Mux4
A_Reg	ALU Reg1 output
B_Reg	ALU Reg2 output
ALU_OPR1	ALU first operand derived from Mux7
ALU_OPR2	ALU second operand derived from Mux8
ALU_OUT	ALU output
NF	ALU negative flag output
ZF	ALU zero flag output
OF	ALU overflow flag output
BF	ALU invalid operation flag output
EPC_IN	Input to the EPC register
CAUSE_IN	Input to the CAUSE register
STATE	Sequence Controller state output used for debug purpose only

Addition Operation:

- ADD (R-type): Rs + Rt →Rd

Opcode	Rs	Rt	Rd	Shamt	Funct
000000	00001	00010	00011	00000	100000

RAM location: 0x000

Instruction in Hex: 0x221820

Result: 4 → R[3]

5 CLK = 0 RST = 0 PC_out = 0 RAM_Addr = 0 RAM_Data = x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000001000100001100000100000 IR_out = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx REG_WS = 0 REG_OE = 1 RegData = 1 RegAddr = X A_Reg = x B_Reg = x ALU_OPR1 = 0 ALU_OPR2 = 1 ALU_OUT = 1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_EN = 0 CAUSE_EN = 0 STATE = 0

10 CLK = 0 RST = 1 PC_out = 0 RAM_Addr = 0 RAM_Data = x RAM_WS = 0 RAM_OE = 1 RAM_out = 00000000001000100001100000100000 IR_out = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx REG_WS = 0 REG_OE = 1 RegData = 1 RegAddr = X A_Reg = x B_Reg = x ALU_OPR1 = 0 ALU_OPR2 = 1 ALU_OUT = 1 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_EN = 0 CAUSE_EN = 0 STATE = 0

25 CLK = 1 RST = 1 PC_out = 1 RAM_Addr = 1 RAM_Data = x RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001000100001100000100000 IR_out = 00000000001000100001100000100000 REG_WS = 0 REG_OE = 1 RegData = 24705 RegAddr = 2 A_Reg = x B_Reg = x ALU_OPR1 = 1 ALU_OPR2 = 24704 ALU_OUT = 24705 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_EN = 0 CAUSE_EN = 0 STATE = 1

50 CLK = 0 RST = 1 PC_out = 1 RAM_Addr = 1 RAM_Data = x RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001000100001100000100000 IR_out = 00000000001000100001100000100000 REG_WS = 0 REG_OE = 1 RegData = 24705 RegAddr = 2 A_Reg = x B_Reg = x ALU_OPR1 = 1 ALU_OPR2 = 24704 ALU_OUT = 24705 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_EN = 0 CAUSE_EN = 0 STATE = 1

75 CLK = 1 RST = 1 PC_out = 1 RAM_Addr = 1 RAM_Data = 1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001000100001100000100000 IR_out = 00000000001000100001100000100000 REG_WS = 0 REG_OE = 1 RegData = 4 RegAddr =3 A_Reg = 3 B_Reg = 1 ALU_OPR1 = 3 ALU_OPR2 = 1 ALU_OUT = 4 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_EN = 0 CAUSE_EN = 0 STATE = 4

100 CLK = 0 RST = 1 PC_out = 1 RAM_Addr = 1 RAM_Data = 1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001000100001100000100000 IR_out = 00000000001000100001100000100000 REG_WS = 0 REG_OE = 1 RegData = 4 RegAddr = 3 A_Reg = 3 B_Reg = 1 ALU_OPR1 = 3 ALU_OPR2 = 1 ALU_OUT = 4 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_EN = 0 CAUSE_EN = 0 STATE = 4

125 CLK = 1 RST = 1 PC_out = 1 RAM_Addr = 1 RAM_Data = 1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001000100001100000100000 IR_out = 00000000001000100001100000100000 REG_WS = 1 REG_OE = 1 RegData = 4 RegAddr = 3 A_Reg = 3 B_Reg = 1 ALU_OPR1 = 3 ALU_OPR2 = 1 ALU_OUT = 4 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_EN = 0 CAUSE_EN = 0 STATE = 5

150 CLK = 0 RST = 1 PC_out = 1 RAM_Addr = 1 RAM_Data = 1 RAM_WS = 0 RAM_OE = 0 RAM_out = 00000000001000100001100000100000 IR_out = 00000000001000100001100000100000 REG_WS = 1 REG_OE = 1 RegData = 4 RegAddr = 3 A_Reg = 3 B_Reg = 1 ALU_OPR1 = 3 ALU_OPR2 = 1 ALU_OUT = 4 NF = 0 ZF = 0 OF = 0 BF = 0 EPC_EN = 0 CAUSE_EN = 0 STATE = 5

Analysis: 1 + 3 → 4 → RegFile[3]. Test results show that RegData = 4 and RegAddr = 3 while the REG_WS signal goes high on the last clock cycle. The next instruction uses R[3] to verify that the value 4 actually got latched. (NOTE: The operand values in R[1] and R[2] were initially stored into the Register File before the start of simulation using the $readmemh system Verilog task).

****