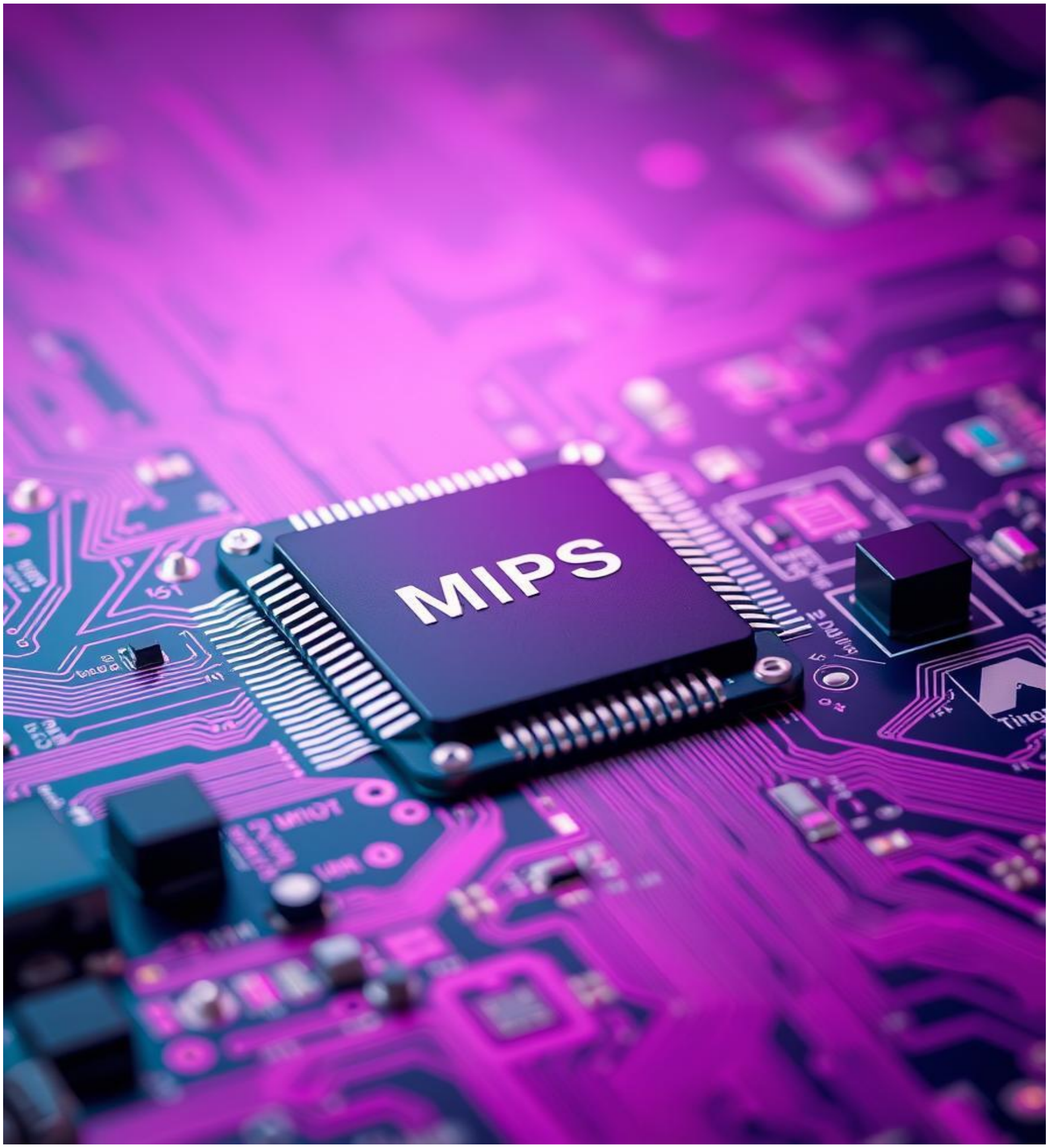
32-bit MIPS Project



Prepared by :

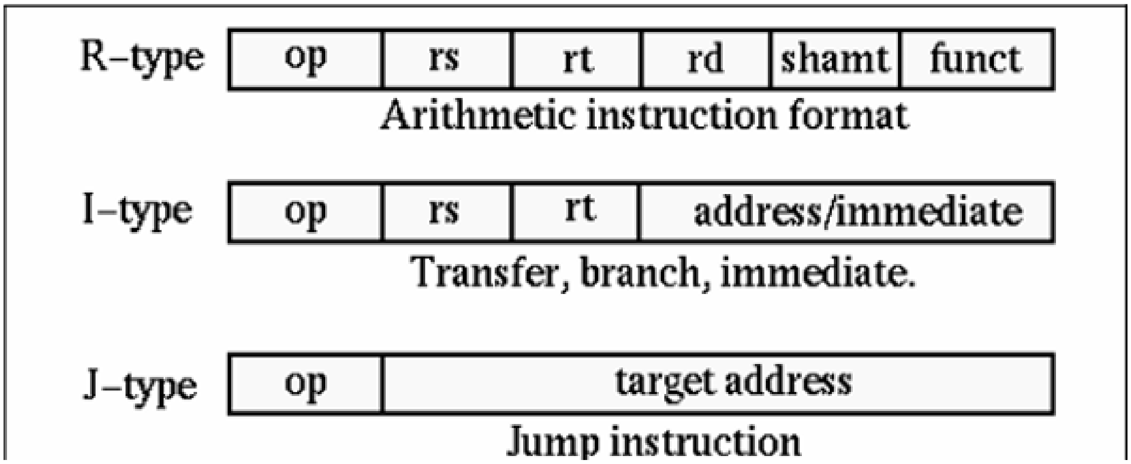
-Sarah Safwat Sayed

-Sarah Omar Abdelfatah

Abstract

- The objective of this project is to design and implement a processor on Modelsim that would run a program that is based off the Microprocessor without Interlocked Pipeline Stages (MIPS) instruction set

The MIPS instruction set can be broken down into three instruction formats. The two most common types R-type and I-type instructions form the foundation of most assembly languages. The R-type instruction consists of a label and 3 operands.

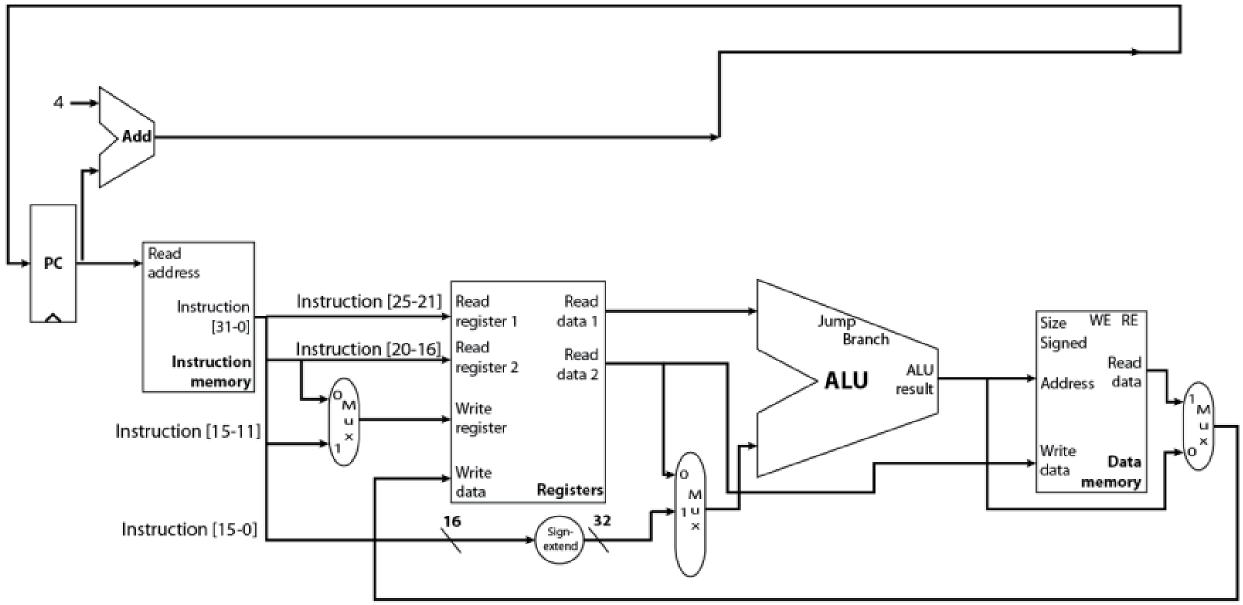


MIPS instruction set :

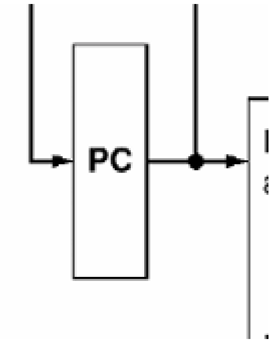
Operation	address
Add	0x00000000
Subtract	0x00000004
Multiplication	0x00000008
and	0x0000000c
or	0x00000010
nor	0x00000018
nand	0x0000001c
Shift left logical	0x00000020
Shift right logical	0x00000024
Set less than	0x00000028
Set not equal	0x00000030
Set greater than	0x00000034
Load word	0x00000038
Store word	0x0000003c

Branch equal	0x00000040

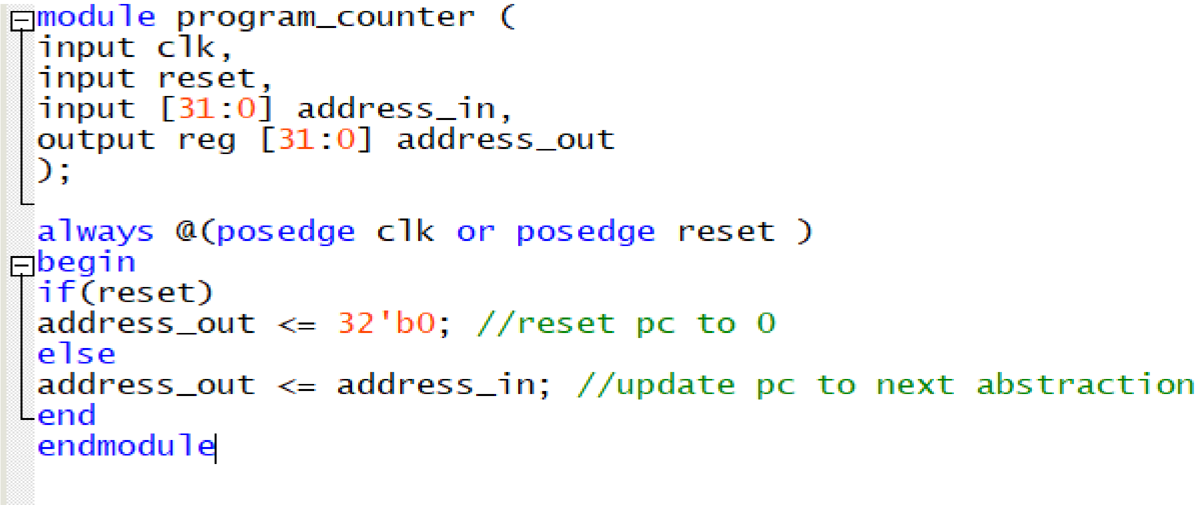
MIPS block diagram



Program counter (PC):



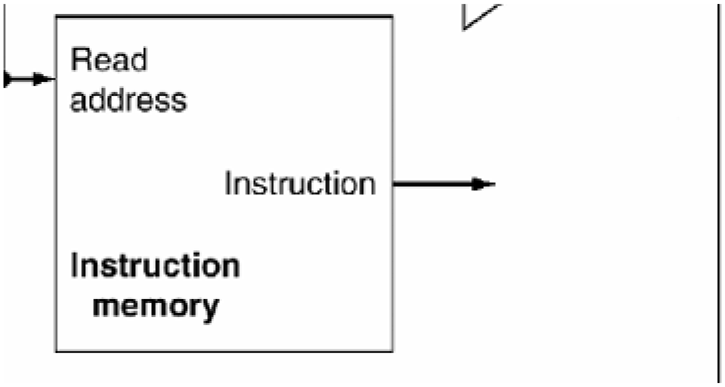
Code in Verilog :



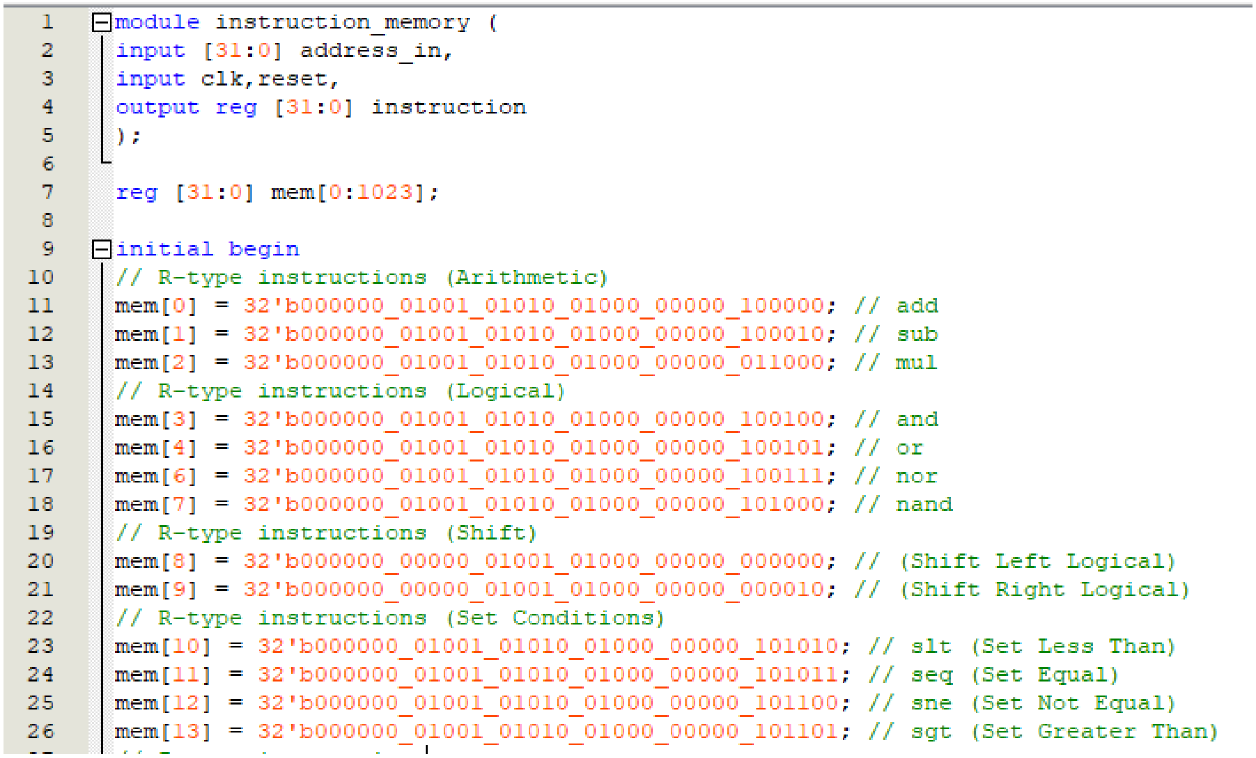
```
module program_counter (
input clk,
input reset,
input [31:0] address_in,
output reg [31:0] address_out
);

always @(posedge clk or posedge reset )
begin
if(reset)
address_out <= 32'b0; //reset pc to 0
else
address_out <= address_in; //update pc to next abstraction
end
endmodule
```

- The program counter is responsible for holding the address of the current instruction.
- In this stage the PC is also responsible for holding the address of the next instruction being input upon its incrementation from the Adder

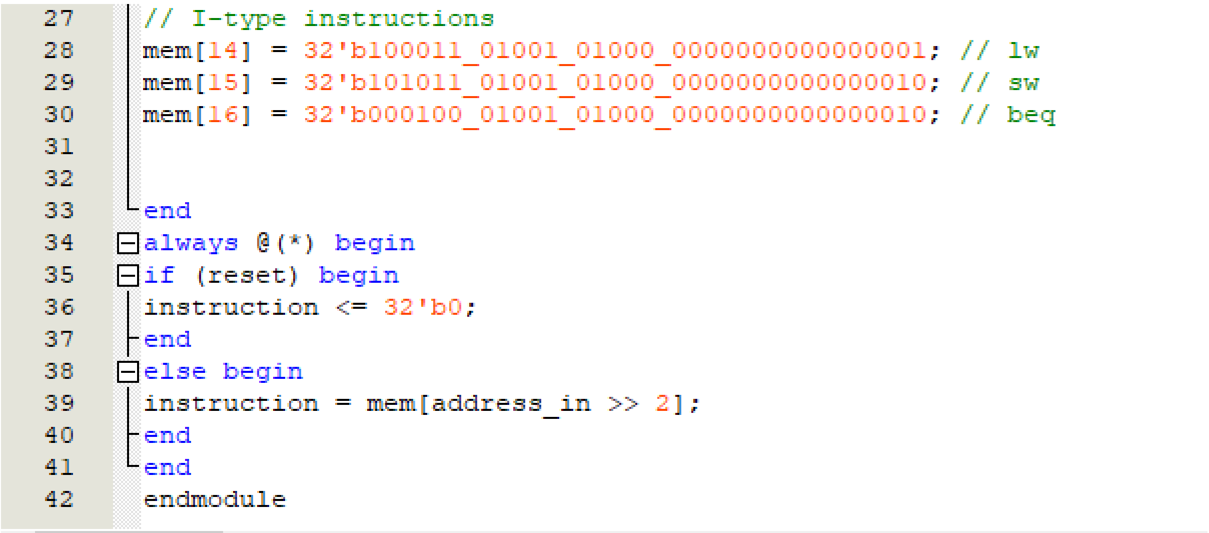
Instruction memory:



Code in Verilog :



```
module instruction_memory (
input [31:0] address_in,
input clk,reset,
output reg [31:0] instruction
);

reg [31:0] mem[0:1023];

initial begin
// R-type instructions (Arithmetic)
mem[0] = 32'b000000_01001_01010_01000_00000_100000; // add
mem[1] = 32'b000000_01001_01010_01000_00000_100010; // sub
mem[2] = 32'b000000_01001_01010_01000_00000_011000; // mul
// R-type instructions (Logical)
mem[3] = 32'b000000_01001_01010_01000_00000_100100; // and
mem[4] = 32'b000000_01001_01010_01000_00000_100101; // or
mem[6] = 32'b000000_01001_01010_01000_00000_100111; // nor
mem[7] = 32'b000000_01001_01010_01000_00000_101000; // nand
// R-type instructions (Shift)
mem[8] = 32'b000000_00000_01001_01000_00000_000000; // (Shift Left Logical)
mem[9] = 32'b000000_00000_01001_01000_00000_000010; // (Shift Right Logical)
// R-type instructions (Set Conditions)
mem[10] = 32'b000000_01001_01010_01000_00000_101010; // slt (Set Less Than)
mem[11] = 32'b000000_01001_01010_01000_00000_101011; // seq (Set Equal)
mem[12] = 32'b000000_01001_01010_01000_00000_101100; // sne (Set Not Equal)
mem[13] = 32'b000000_01001_01010_01000_00000_101101; // sgt (Set Greater Than)
```



```
// I-type instructions
mem[14] = 32'b100011_01001_01000_0000000000000001; // lw
mem[15] = 32'b101011_01001_01000_0000000000000010; // sw
mem[16] = 32'b000100_01001_01000_0000000000000010; // beq


end
always @(*) begin
if (reset) begin
instruction <= 32'b0;
end
else begin
instruction = mem[address_in >> 2];
end
end
endmodule
```

- the Instruction Memory plays a crucial role in fetching instructions from memory based on the current Program Counter (PC).

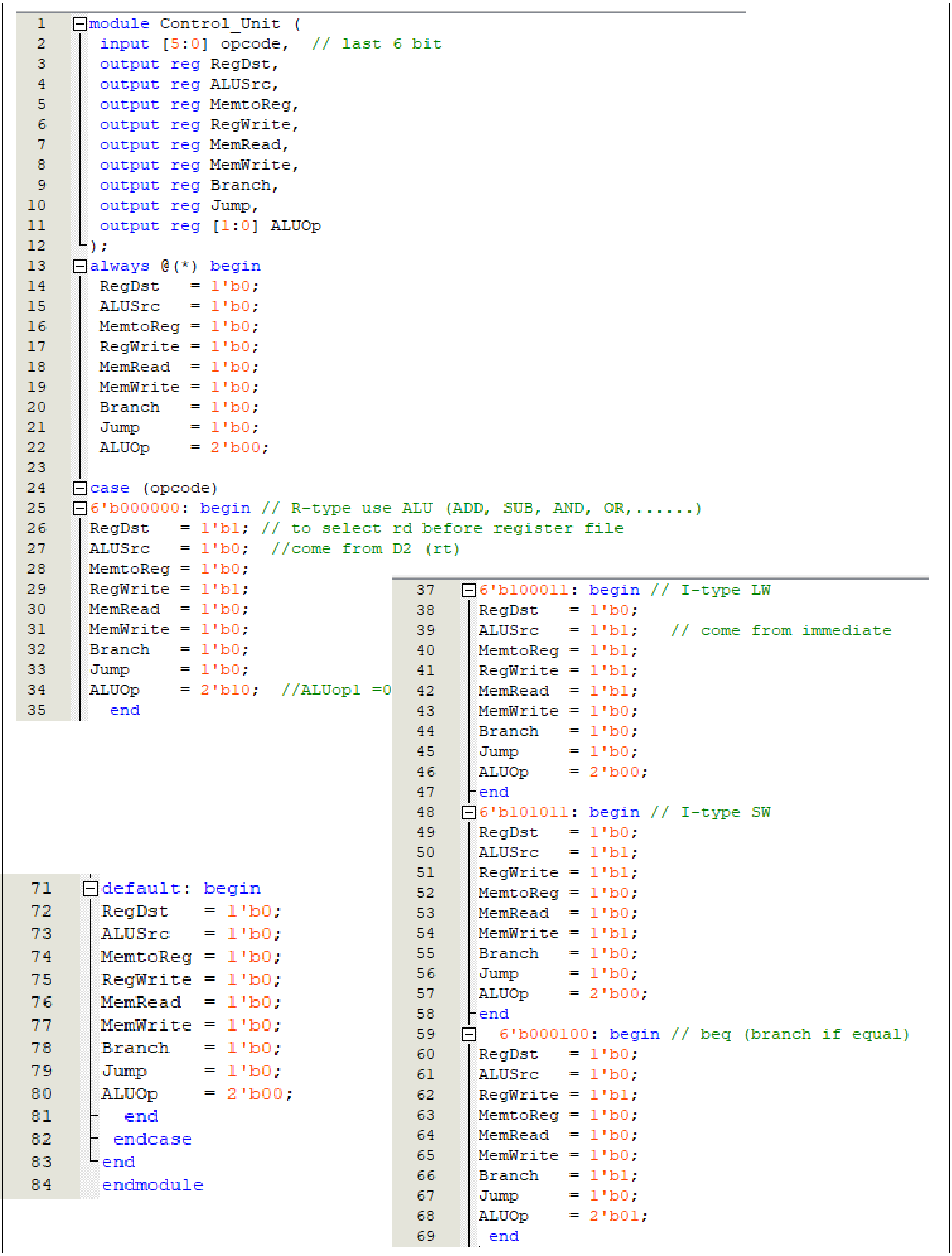
- The Instruction Memory is responsible for storing the instructions that the processor will execute and providing these instructions to the processor as needed during the execution cycle**.**

Control Unit :

The **Control Unit** is responsible for generating control signals that dictate how data moves through the processor. It interprets the **opcode** of an instruction and activates the appropriate control lines to ensure correct execution.

- **For R-Type Instructions (`000000`)** → The control unit enables register-to-register operations by setting the destination register, selecting operands from the registers, and using the function code to determine the ALU operation.
- **For Load Word (`LW - 100011`)** → The unit enables memory reading, selects an immediate offset for address calculation, and ensures the data is written back into a register.
- **For Store Word (`SW - 101011`)** → The processor computes the memory address using an immediate value and writes data from a register into memory. No register update occurs.
- **For Branch if Equal (`BEQ - 000100`)** → The ALU performs subtraction to compare two registers, and if they are equal, the branch signal is activated, modifying the program counter to execute a conditional jump.

By controlling data paths, memory access, and ALU operations, the **Control Unit** ensures proper instruction execution in the MIPS processor.



```
module Control_Unit (
 input [5:0] opcode,  // last 6 bit
 output reg RegDst,
 output reg ALUSrc,
 output reg MemtoReg,
 output reg RegWrite,
 output reg MemRead,
 output reg MemWrite,
 output reg Branch,
 output reg Jump,
 output reg [1:0] ALUOp
);
always @(*) begin
 RegDst    = 1'b0;
 ALUSrc    = 1'b0;
 MemtoReg = 1'b0;
 RegWrite = 1'b0;
 MemRead  = 1'b0;
 MemWrite = 1'b0;
 Branch    = 1'b0;
 Jump      = 1'b0;
 ALUOp     = 2'b00;

case (opcode)
6'b000000: begin // R-type use ALU (ADD, SUB, AND, OR,......)
RegDst    = 1'b1; // to select rd before register file
ALUSrc    = 1'b0;  //come from D2 (rt)
MemtoReg = 1'b0;
RegWrite = 1'b1;
MemRead  = 1'b0;
MemWrite = 1'b0;
Branch    = 1'b0;
Jump      = 1'b0;
ALUOp     = 2'b10;   //ALUop1 =0
   end
```

```
6'b100011: begin // I-type LW
RegDst    = 1'b0;
ALUSrc    = 1'b1;   // come from immediate
MemtoReg = 1'b1;
RegWrite = 1'b1;
MemRead  = 1'b1;
MemWrite = 1'b0;
Branch    = 1'b0;
Jump      = 1'b0;
ALUOp     = 2'b00;
end
6'b101011: begin // I-type SW
RegDst    = 1'b0;
ALUSrc    = 1'b1;
RegWrite = 1'b1;
MemtoReg = 1'b0;
MemRead  = 1'b0;
MemWrite = 1'b1;
Branch    = 1'b0;
Jump      = 1'b0;
ALUOp     = 2'b00;
end
  6'b000100: begin // beq (branch if equal)
RegDst    = 1'b0;
ALUSrc    = 1'b0;
RegWrite = 1'b1;
MemtoReg = 1'b0;
MemRead  = 1'b0;
MemWrite = 1'b0;
Branch    = 1'b1;
Jump      = 1'b0;
ALUOp     = 2'b01;
 end
```

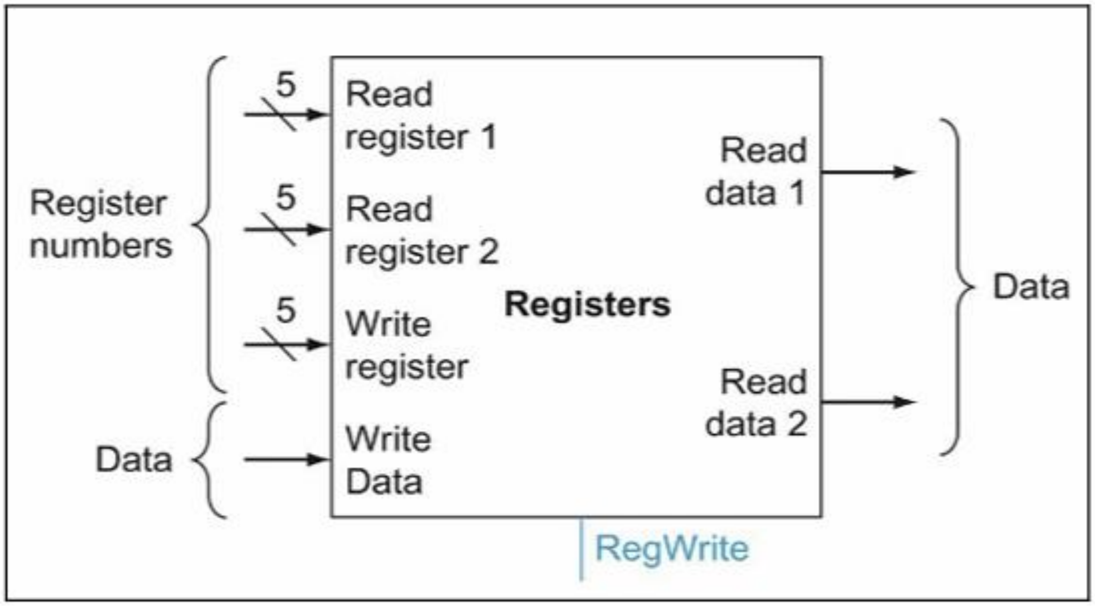
```
default: begin
RegDst    = 1'b0;
ALUSrc    = 1'b0;
MemtoReg = 1'b0;
RegWrite = 1'b0;
MemRead  = 1'b0;
MemWrite = 1'b0;
Branch    = 1'b0;
Jump      = 1'b0;
ALUOp     = 2'b00;
   end
 endcase
end
endmodule
```

Register file :

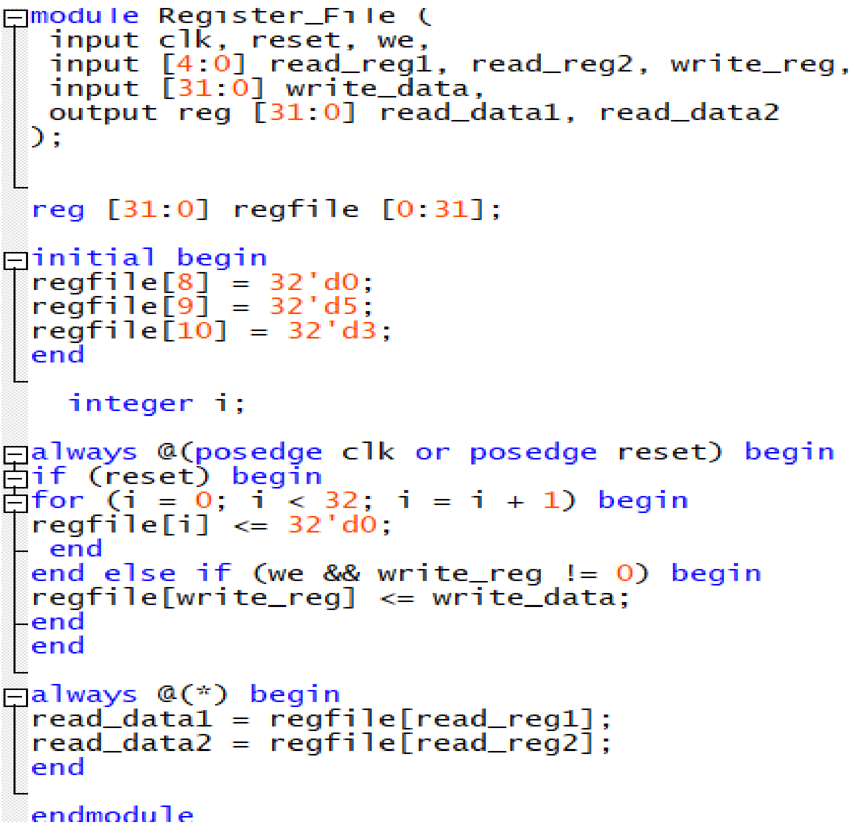
the Register File is a crucial component that stores and provides fast access to the processor's registers. These registers are used to hold data temporarily during computation and are accessed by various instructions like arithmetic operations, memory load/store, and branching.

The **Register File** supports two types of operations:

- **Read**: The ability to read data from one or two registers.
- **Write**: The ability to write data to a register



Code in Verilog :

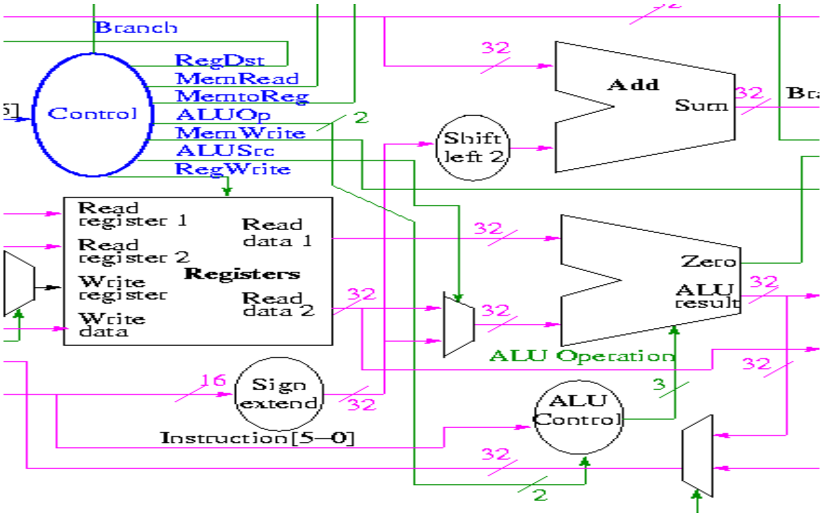


```
module Register_File (
 input clk, reset, we,
 input [4:0] read_reg1, read_reg2, write_reg,
 input [31:0] write_data,
 output reg [31:0] read_data1, read_data2
);


reg [31:0] regfile [0:31];

initial begin
regfile[8] = 32'd0;
regfile[9] = 32'd5;
regfile[10] = 32'd3;
end

  integer i;

always @(posedge clk or posedge reset) begin
if (reset) begin
for (i = 0; i < 32; i = i + 1) begin
regfile[i] <= 32'd0;
 end
end else if (we && write_reg != 0) begin
regfile[write_reg] <= write_data;
end
end

always @(*) begin
read_data1 = regfile[read_reg1];
read_data2 = regfile[read_reg2];
end

endmodule
```

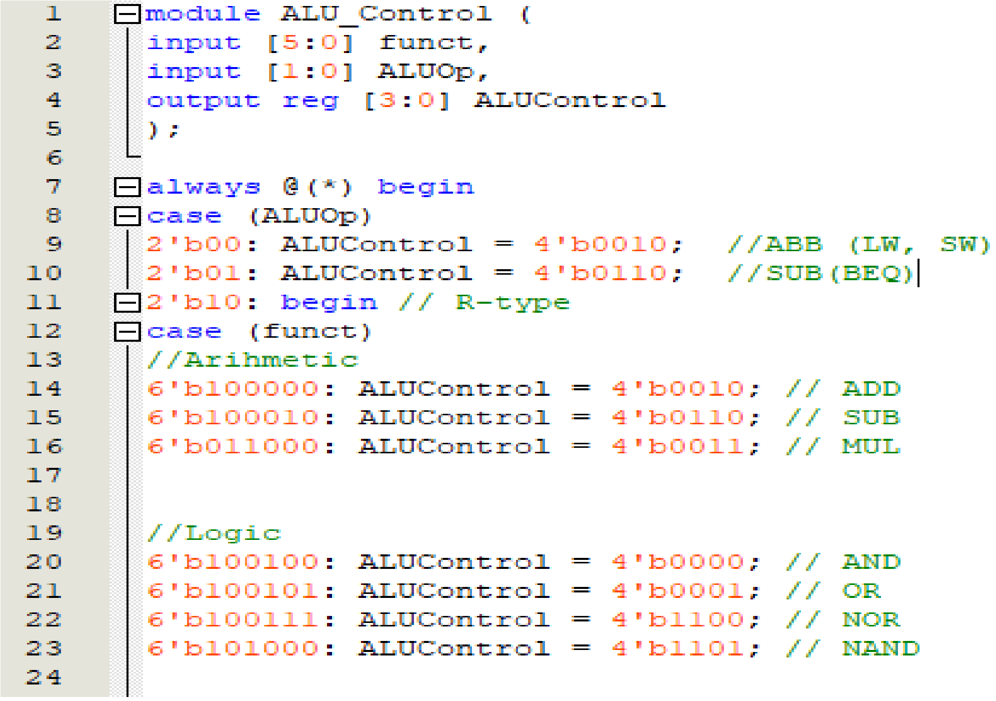
ALU Control:

- **ALU Control** determines the operation for the ALU based on the **ALUOp** signal and the **funct** field for R-type instructions.
- For **LW and SW** the ALU performs **addition (0010)**, while **BEQ** use **subtraction (0110)**.
- R-type instructions are decoded using the **funct field**, enabling arithmetic, logical, shift, and comparison operations.
- Unrecognized instructions default to **1111 (No Operation)** to prevent unintended execution.

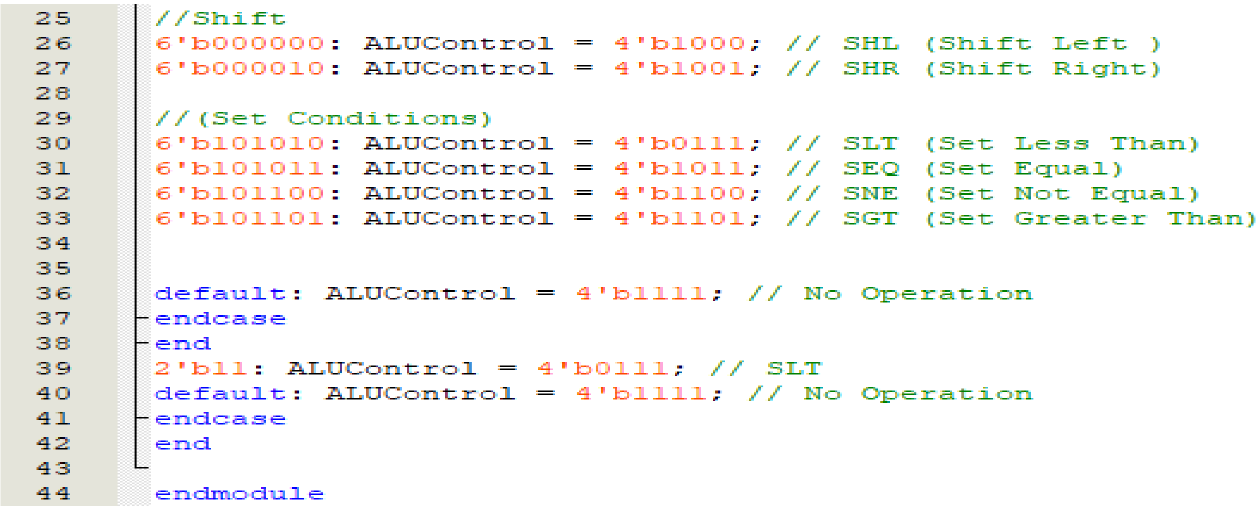
The module is purely **combinational**, using a **case statement** to generate the correct ALU control signal dynamically .



Verilog code :



```verilog
module ALU_Control (
input [5:0] funct,
input [1:0] ALUOp,
output reg [3:0] ALUControl
);

always @(*) begin
case (ALUOp)
2'b00: ALUControl = 4'b0010;  //ABB (LW, SW)
2'b01: ALUControl = 4'b0110;  //SUB(BEQ)
2'b10: begin // R-type
case (funct)
//Arihmetic
6'b100000: ALUControl = 4'b0010; // ADD
6'b100010: ALUControl = 4'b0110; // SUB
6'b011000: ALUControl = 4'b0011; // MUL


//Logic
6'b100100: ALUControl = 4'b0000; // AND
6'b100101: ALUControl = 4'b0001; // OR
6'b100111: ALUControl = 4'b1100; // NOR
6'b101000: ALUControl = 4'b1101; // NAND

```



```verilog
//Shift
6'b000000: ALUControl = 4'b1000; // SHL (Shift Left )
6'b000010: ALUControl = 4'b1001; // SHR (Shift Right)

//(Set Conditions)
6'b101010: ALUControl = 4'b0111; // SLT (Set Less Than)
6'b101011: ALUControl = 4'b1011; // SEQ (Set Equal)
6'b101100: ALUControl = 4'b1100; // SNE (Set Not Equal)
6'b101101: ALUControl = 4'b1101; // SGT (Set Greater Than)


default: ALUControl = 4'b1111; // No Operation
endcase
end
2'b11: ALUControl = 4'b0111; // SLT
default: ALUControl = 4'b1111; // No Operation
endcase
end

endmodule
```

ALU:

This **32-bit ALU** executes arithmetic, logic, shift, and comparison operations based on a **4-bit ALUControl signal**, making it suitable for MIPS-based processors.

1. Arithmetic Operations :

-ADD (`4'b0010`) → Uses an adder to compute `A + B`.

-SUB (`4'b0110`) → Uses the same adder but with `B` inverted to perform `A - B`.

-MUL (`4'b0011`) → Direct multiplication `A * B`.

Handles **carry-out (cout)** and **overflow (v)** for correctness.

2. Logical Operations :

AND (`4'b0000`) → `A & B` (bit masking).

OR (`4'b0001`) → `A | B` (bit setting).

NOR (`4'b1100`) → `~(A | B).`

NAND (`4'b1101`) → `~(A & B).`

3. Shift Operations:

SHL (`4'b1000`) → Left shift `B << A`.

SHR (`4'b1001`) → Right shift `B >> A`

4. Comparison Operations :

SLT (`4'b0111`) → `1` if `A > B`, else `0`.

SEQ (`4'b1011`) → `1` if `A == B`, else `0`.

SNE (`4'b1100`) → `1` if `A ≠ B`, else `0`.

SGT (`4'b1101`) → `1` if `A < B`, else `0`.

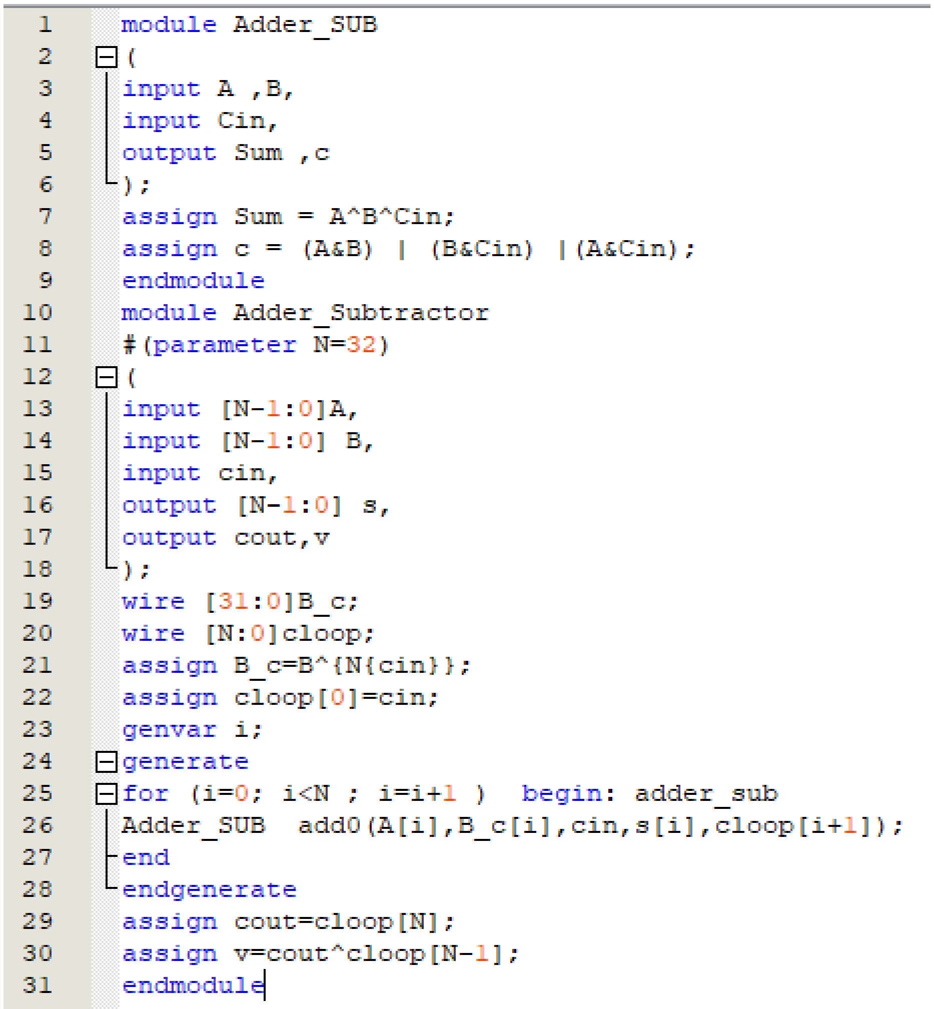
Supports **branching instructions in MIPS .**

5. Zero Flag :

Set to `1` when `ALUResult == 32'b0`, useful for **BEQ instruction.**

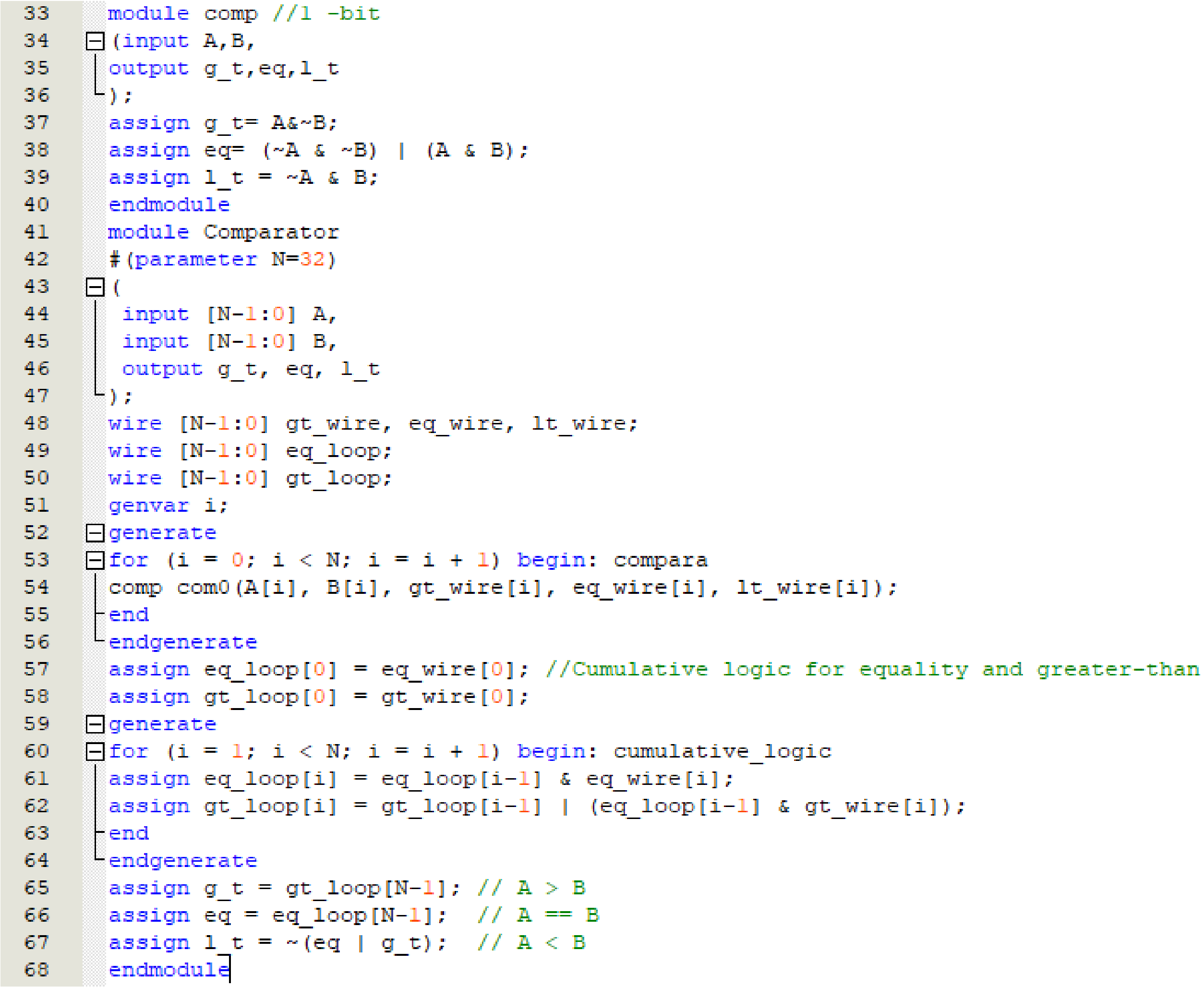
Verilog code

- Adder_Subtractor For 32-bit :



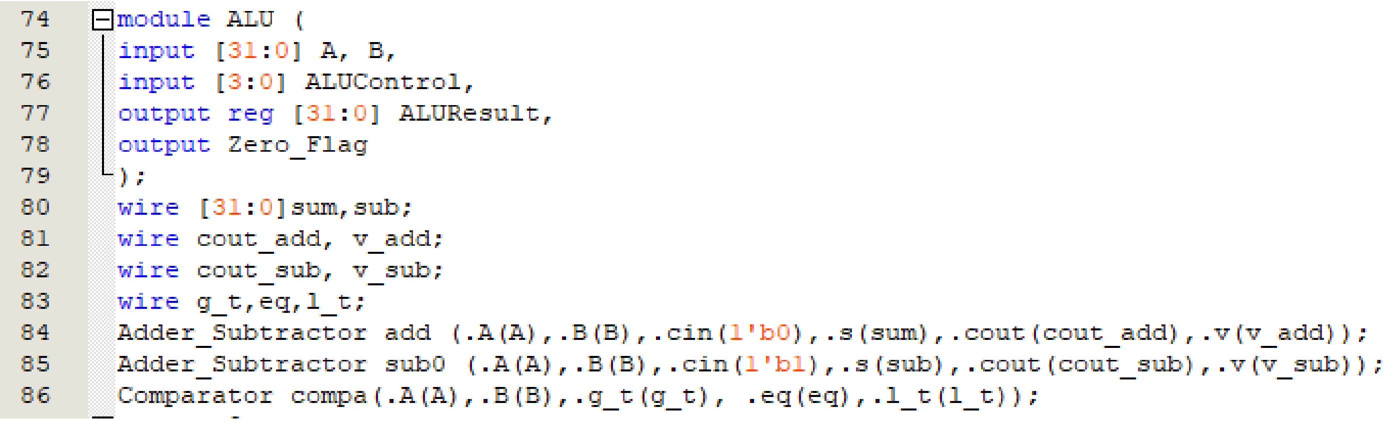
```verilog
module Adder_SUB
(
input A ,B,
input Cin,
output Sum ,c
);
assign Sum = A^B^Cin;
assign c = (A&B)  |  (B&Cin)  |(A&Cin);
endmodule
module Adder_Subtractor
#(parameter N=32)
(
input [N-1:0]A,
input [N-1:0] B,
input cin,
output [N-1:0] s,
output cout,v
);
wire [31:0]B_c;
wire [N:0]cloop;
assign B_c=B^{N{cin}};
assign cloop[0]=cin;
genvar i;
generate
for (i=0; i<N ; i=i+1 )  begin: adder_sub
Adder_SUB  add0(A[i],B_c[i],cin,s[i],cloop[i+1]);
end
endgenerate
assign cout=cloop[N];
assign v=cout^cloop[N-1];
endmodule
```

- Comparator For 32-bit :

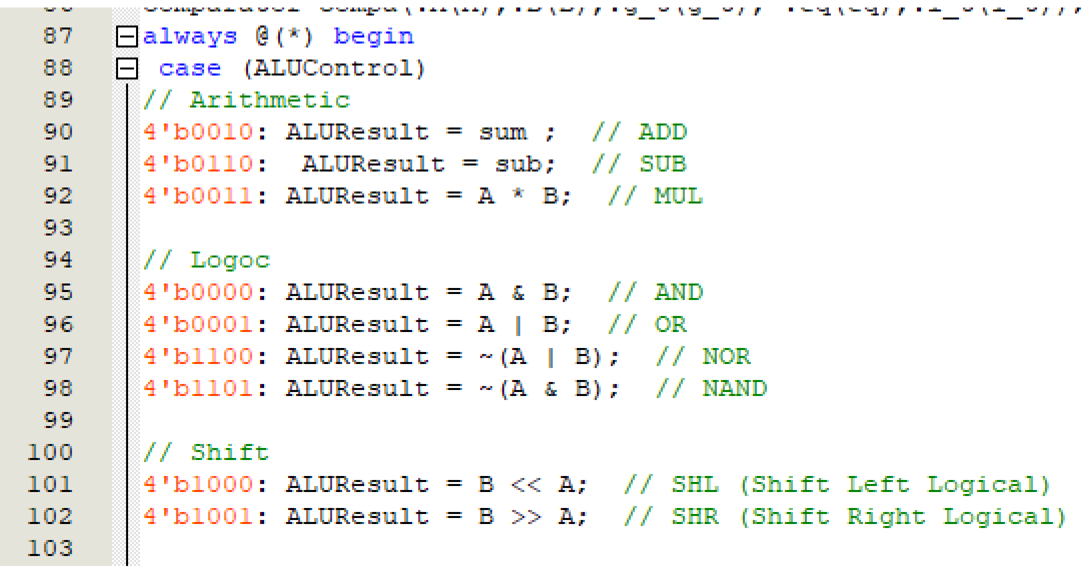


```
module comp //1 -bit
(input A,B,
output g_t,eq,l_t
);
assign g_t= A&~B;
assign eq= (~A & ~B) | (A & B);
assign l_t = ~A & B;
endmodule
module Comparator
#(parameter N=32)
(
 input [N-1:0] A,
 input [N-1:0] B,
 output g_t, eq, l_t
);
wire [N-1:0] gt_wire, eq_wire, lt_wire;
wire [N-1:0] eq_loop;
wire [N-1:0] gt_loop;
genvar i;
generate
for (i = 0; i < N; i = i + 1) begin: compara
comp com0(A[i], B[i], gt_wire[i], eq_wire[i], lt_wire[i]);
end
endgenerate
assign eq_loop[0] = eq_wire[0]; //Cumulative logic for equality and greater-than
assign gt_loop[0] = gt_wire[0];
generate
for (i = 1; i < N; i = i + 1) begin: cumulative_logic
assign eq_loop[i] = eq_loop[i-1] & eq_wire[i];
assign gt_loop[i] = gt_loop[i-1] | (eq_loop[i-1] & gt_wire[i]);
end
endgenerate
assign g_t = gt_loop[N-1]; // A > B
assign eq = eq_loop[N-1];  // A == B
assign l_t = ~(eq | g_t);  // A < B
endmodule
```

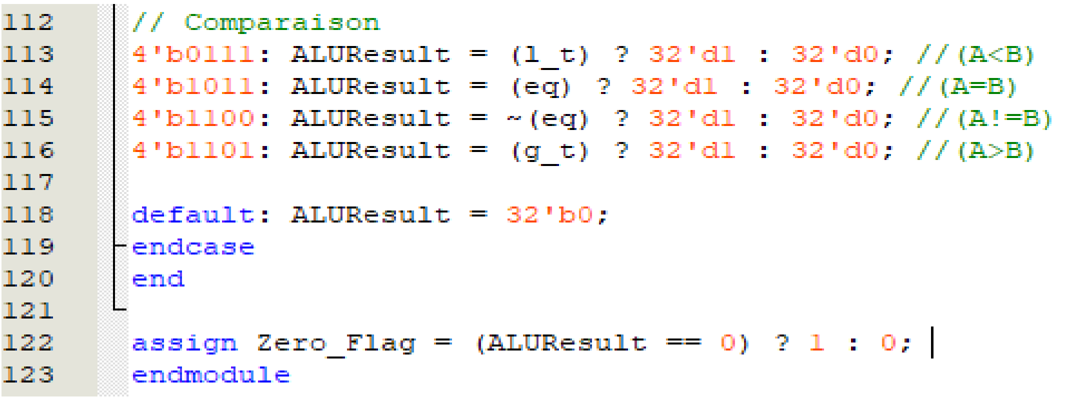
- ALU TOP MODULE :



```
module ALU (
input [31:0] A, B,
input [3:0] ALUControl,
output reg [31:0] ALUResult,
output Zero_Flag
);
wire [31:0]sum,sub;
wire cout_add, v_add;
wire cout_sub, v_sub;
wire g_t,eq,l_t;
Adder_Subtractor add (.A(A),.B(B),.cin(1'b0),.s(sum),.cout(cout_add),.v(v_add));
Adder_Subtractor sub0 (.A(A),.B(B),.cin(1'b1),.s(sub),.cout(cout_sub),.v(v_sub));
Comparator compa(.A(A),.B(B),.g_t(g_t), .eq(eq),.l_t(l_t));
```



```
always @(*) begin
 case (ALUControl)
// Arithmetic
4'b0010: ALUResult = sum ;  // ADD
4'b0110:  ALUResult = sub;  // SUB
4'b0011: ALUResult = A * B;  // MUL

// Logoc
4'b0000: ALUResult = A & B;  // AND
4'b0001: ALUResult = A | B;  // OR
4'b1100: ALUResult = ~(A | B);  // NOR
4'b1101: ALUResult = ~(A & B);  // NAND

// Shift
4'b1000: ALUResult = B << A;  // SHL (Shift Left Logical)
4'b1001: ALUResult = B >> A;  // SHR (Shift Right Logical)

```



```
// Comparaison
4'b0111: ALUResult = (l_t) ? 32'd1 : 32'd0; //(A<B)
4'b1011: ALUResult = (eq) ? 32'd1 : 32'd0; //(A=B)
4'b1100: ALUResult = ~(eq) ? 32'd1 : 32'd0; //(A!=B)
4'b1101: ALUResult = (g_t) ? 32'd1 : 32'd0; //(A>B)

default: ALUResult = 32'b0;
endcase
end

assign Zero_Flag = (ALUResult == 0) ? 1 : 0;
endmodule
```

Data Memory :

-The **Data Memory** module stores and retrieves 32-bit words using **word-aligned addressing**. Since each word is **4 bytes**, we ignore the **lowest two bits (`address[1:0]`)** to ensure proper alignment. Instead, we use `address[11:2]`, which shifts the address right by **2 bits**, effectively dividing by 4.

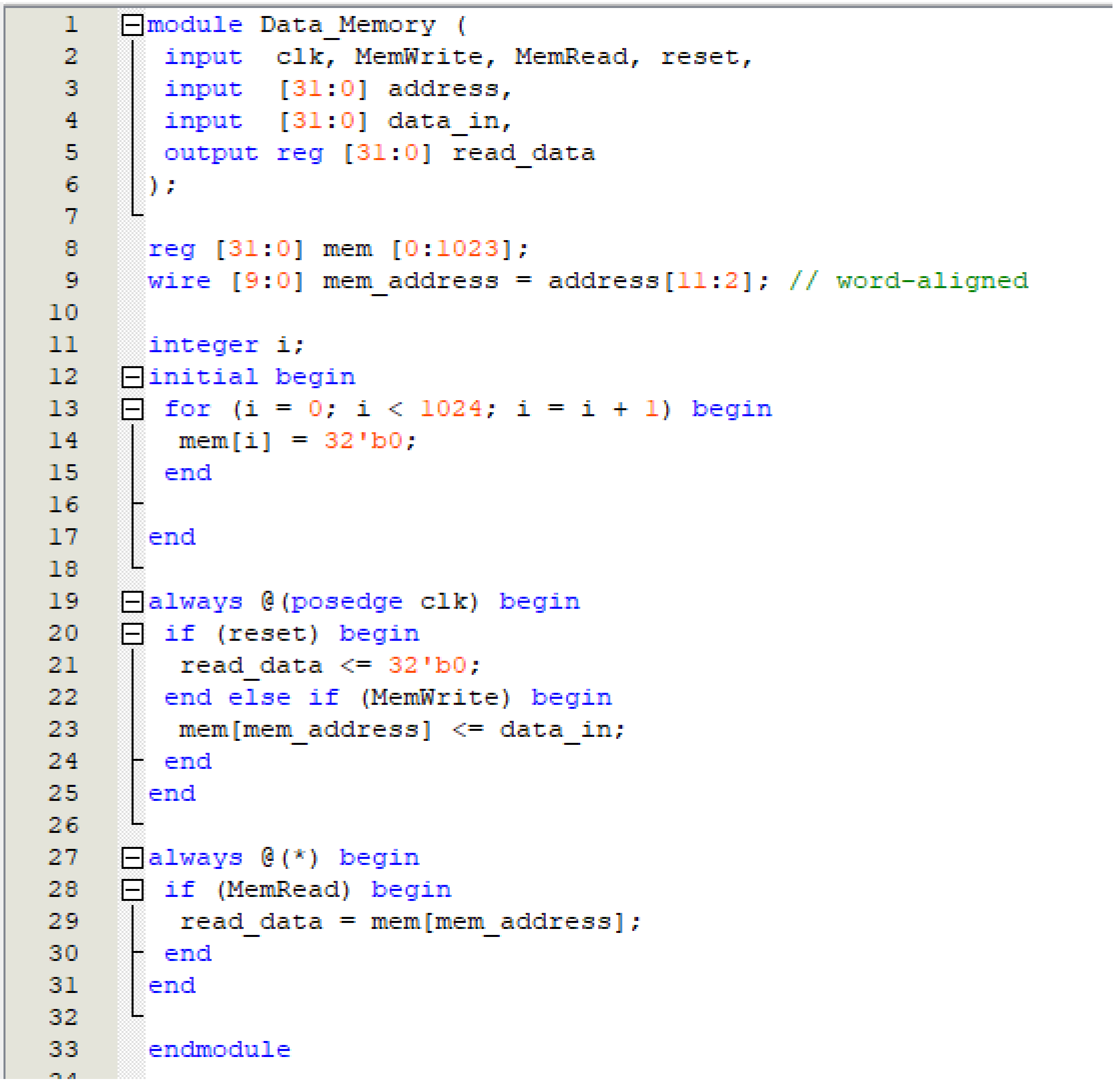
-The **Data Memory** module handles **load (LW)** and **store (SW)** operations. It consists of **1024 words** (each 32-bit) and uses **word-aligned addressing**.

- **Reads (`MemRead = 1`)** → Data is fetched from memory and assigned to `read_data`.

- **Writes (`MemWrite = 1`)** → On the rising edge of `clk`, `data_in` is stored at the specified address.
- **Reset (`reset = 1`)** → Clears `read_data` to `0`.

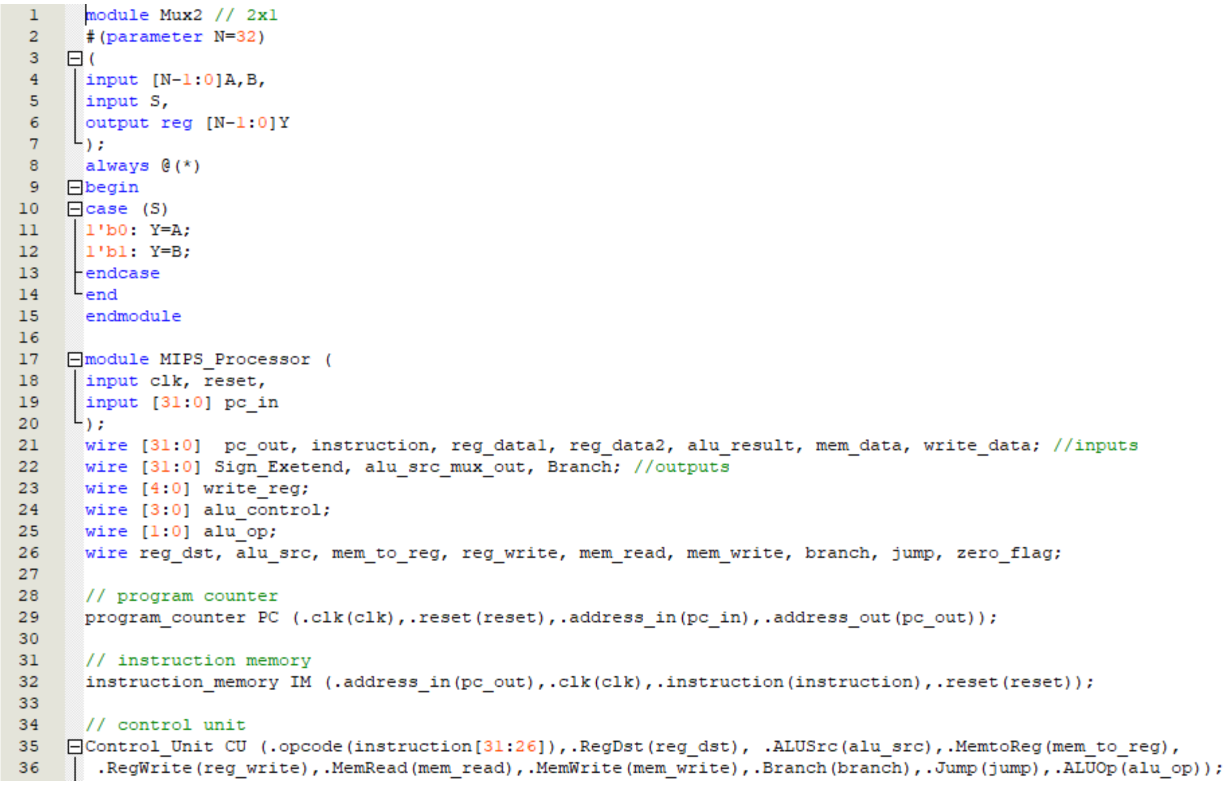
It ensures proper memory access during program execution.

Verilog Code :

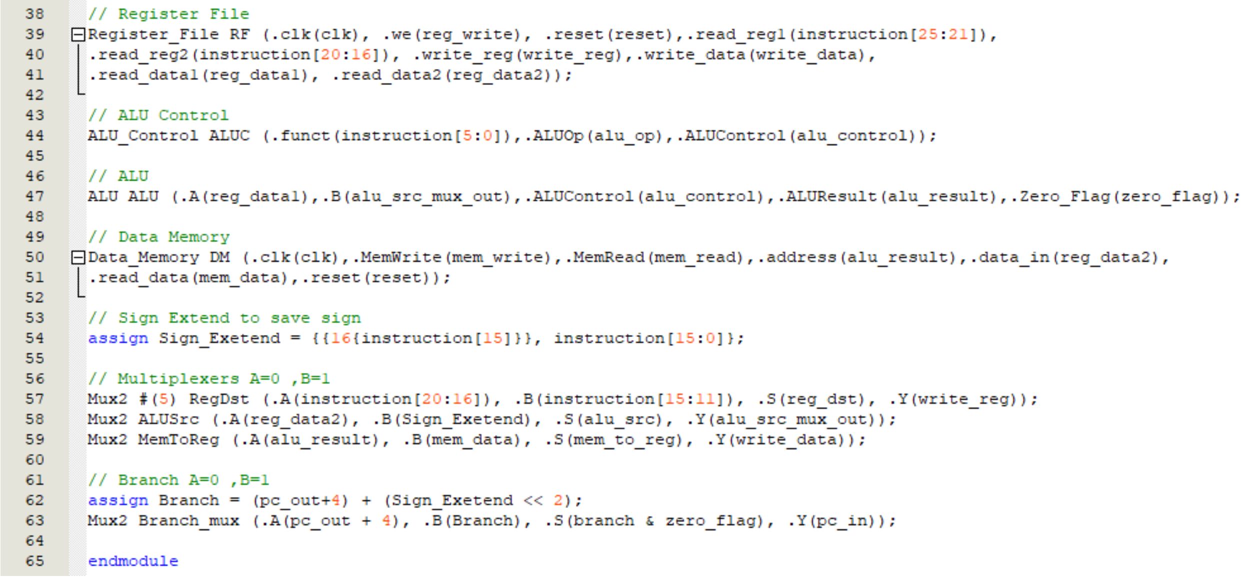


```
module Data_Memory (
 input  clk, MemWrite, MemRead, reset,
 input  [31:0] address,
 input  [31:0] data_in,
 output reg [31:0] read_data
);

reg [31:0] mem [0:1023];
wire [9:0] mem_address = address[11:2]; // word-aligned

integer i;
initial begin
 for (i = 0; i < 1024; i = i + 1) begin
  mem[i] = 32'b0;
 end

end

always @(posedge clk) begin
 if (reset) begin
  read_data <= 32'b0;
 end else if (MemWrite) begin
  mem[mem_address] <= data_in;
 end
end

always @(*) begin
 if (MemRead) begin
  read_data = mem[mem_address];
 end
end

endmodule
```

TOP Module:



```verilog
module Mux2 // 2x1
#(parameter N=32)
(
input [N-1:0]A,B,
input S,
output reg [N-1:0]Y
);
always @(*)
begin
case (S)
1'b0: Y=A;
1'b1: Y=B;
endcase
end
endmodule

module MIPS_Processor (
input clk, reset,
input [31:0] pc_in
);
wire [31:0]  pc_out, instruction, reg_data1, reg_data2, alu_result, mem_data, write_data; //inputs
wire [31:0] Sign_Exetend, alu_src_mux_out, Branch; //outputs
wire [4:0] write_reg;
wire [3:0] alu_control;
wire [1:0] alu_op;
wire reg_dst, alu_src, mem_to_reg, reg_write, mem_read, mem_write, branch, jump, zero_flag;

// program counter
program_counter PC (.clk(clk),.reset(reset),.address_in(pc_in),.address_out(pc_out));

// instruction memory
instruction_memory IM (.address_in(pc_out),.clk(clk),.instruction(instruction),.reset(reset));

// control unit
Control_Unit CU (.opcode(instruction[31:26]),.RegDst(reg_dst), .ALUSrc(alu_src),.MemtoReg(mem_to_reg),
 .RegWrite(reg_write),.MemRead(mem_read),.MemWrite(mem_write),.Branch(branch),.Jump(jump),.ALUOp(alu_op));
```



```verilog
// Register File
Register_File RF (.clk(clk), .we(reg_write), .reset(reset),.read_reg1(instruction[25:21]),
.read_reg2(instruction[20:16]), .write_reg(write_reg),.write_data(write_data),
.read_data1(reg_data1), .read_data2(reg_data2));

// ALU Control
ALU_Control ALUC (.funct(instruction[5:0]),.ALUOp(alu_op),.ALUControl(alu_control));

// ALU
ALU ALU (.A(reg_data1),.B(alu_src_mux_out),.ALUControl(alu_control),.ALUResult(alu_result),.Zero_Flag(zero_flag));

// Data Memory
Data_Memory DM (.clk(clk),.MemWrite(mem_write),.MemRead(mem_read),.address(alu_result),.data_in(reg_data2),
.read_data(mem_data),.reset(reset));

// Sign Extend to save sign
assign Sign_Exetend = {{16{instruction[15]}}, instruction[15:0]};

// Multiplexers A=0 ,B=1
Mux2 #(5) RegDst (.A(instruction[20:16]), .B(instruction[15:11]), .S(reg_dst), .Y(write_reg));
Mux2 ALUSrc (.A(reg_data2), .B(Sign_Exetend), .S(alu_src), .Y(alu_src_mux_out));
Mux2 MemToReg (.A(alu_result), .B(mem_data), .S(mem_to_reg), .Y(write_data));

// Branch A=0 ,B=1
assign Branch = (pc_out+4) + (Sign_Exetend << 2);
Mux2 Branch_mux (.A(pc_out + 4), .B(Branch), .S(branch & zero_flag), .Y(pc_in));

endmodule
```

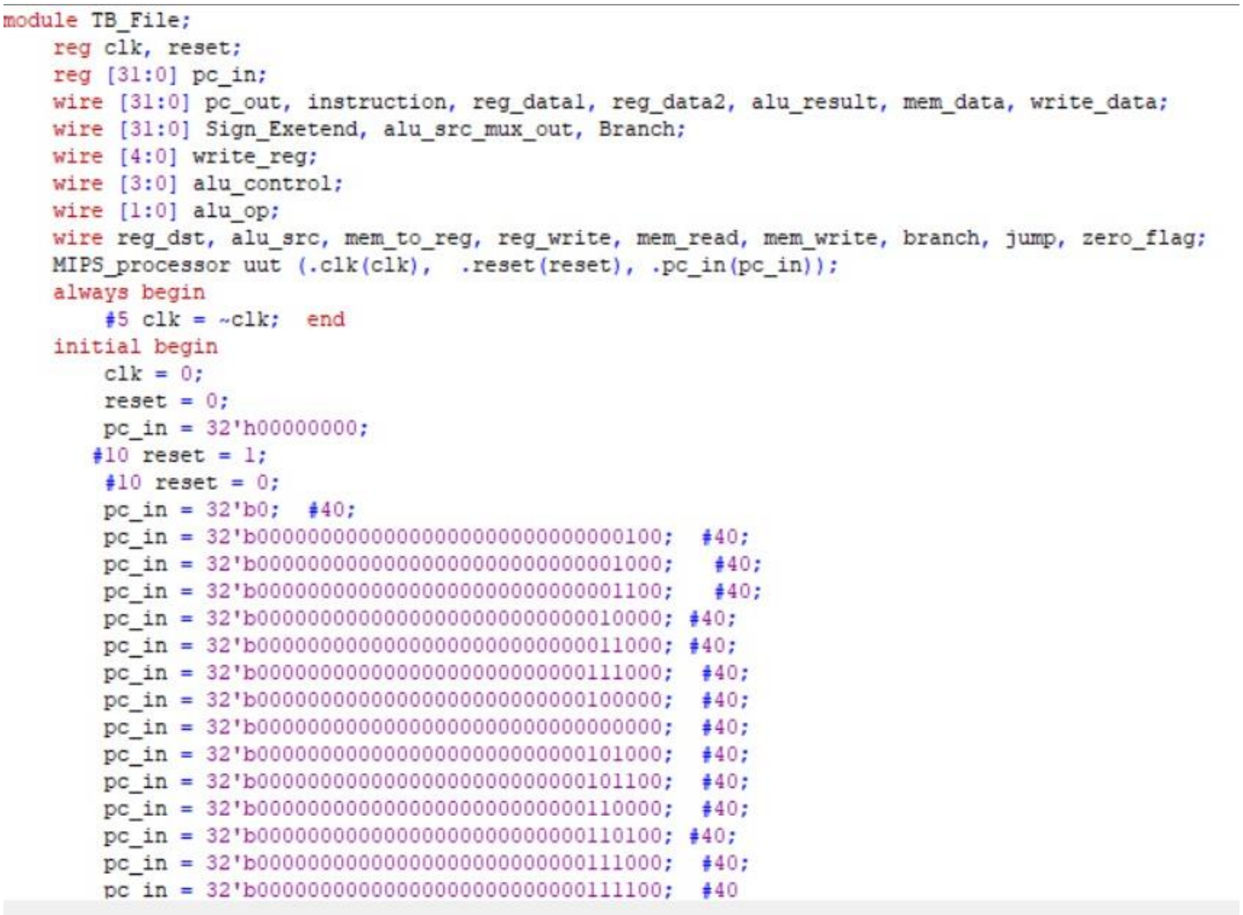
Testbench :

A **testbench** verifies module functionality by simulating inputs and checking outputs. It typically:

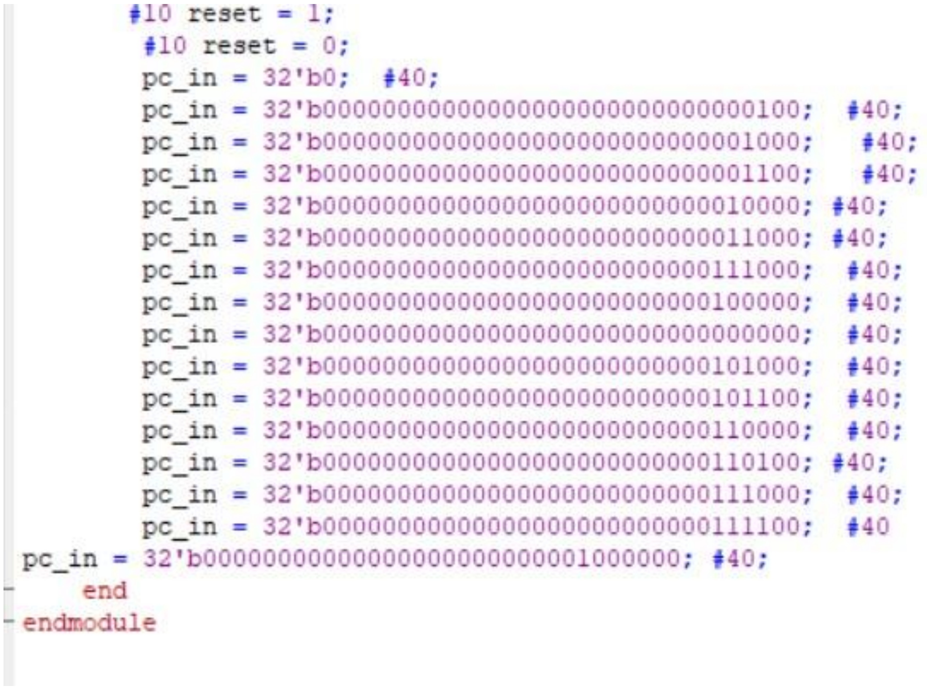
- Generates **clock and reset** signals.
- Applies test values to inputs.
- Observes and verifies outputs.

For MIPS, it tests operations like **ALU calculations, memory access, and control signals** to ensure correct behavior.

Testbench code:



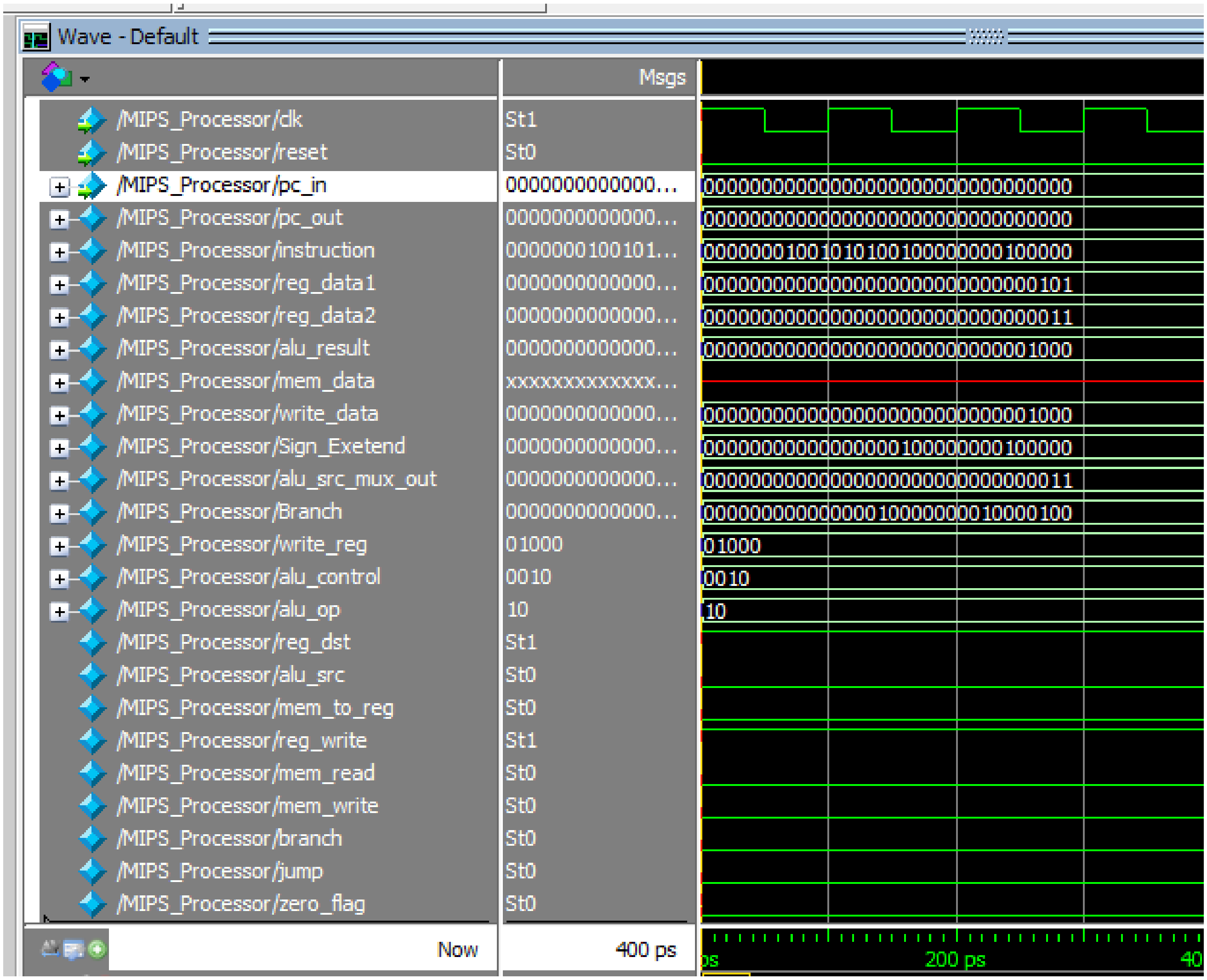
```
module TB_File;
    reg clk, reset;
    reg [31:0] pc_in;
    wire [31:0] pc_out, instruction, reg_data1, reg_data2, alu_result, mem_data, write_data;
    wire [31:0] Sign_Exetend, alu_src_mux_out, Branch;
    wire [4:0] write_reg;
    wire [3:0] alu_control;
    wire [1:0] alu_op;
    wire reg_dst, alu_src, mem_to_reg, reg_write, mem_read, mem_write, branch, jump, zero_flag;
    MIPS_processor uut (.clk(clk),  .reset(reset), .pc_in(pc_in));
    always begin
        #5 clk = ~clk;  end
    initial begin
        clk = 0;
        reset = 0;
        pc_in = 32'h00000000;
       #10 reset = 1;
        #10 reset = 0;
        pc_in = 32'b0;  #40;
        pc_in = 32'b00000000000000000000000000000100;  #40;
        pc_in = 32'b00000000000000000000000000001000;   #40;
        pc_in = 32'b00000000000000000000000000001100;   #40;
        pc_in = 32'b00000000000000000000000000010000; #40;
        pc_in = 32'b00000000000000000000000000011000; #40;
        pc_in = 32'b00000000000000000000000000111000;  #40;
        pc_in = 32'b00000000000000000000000000100000;  #40;
        pc_in = 32'b00000000000000000000000000000000;  #40;
        pc_in = 32'b00000000000000000000000000101000;  #40;
        pc_in = 32'b00000000000000000000000000101100;  #40;
        pc_in = 32'b00000000000000000000000000110000;  #40;
        pc_in = 32'b00000000000000000000000000110100; #40;
        pc_in = 32'b00000000000000000000000000111000;  #40;
        pc_in = 32'b00000000000000000000000000111100;  #40
```



```
        #10 reset = 1;
         #10 reset = 0;
         pc_in = 32'b0;  #40;
         pc_in = 32'b00000000000000000000000000000100;  #40;
         pc_in = 32'b00000000000000000000000000001000;   #40;
         pc_in = 32'b00000000000000000000000000001100;   #40;
         pc_in = 32'b00000000000000000000000000010000; #40;
         pc_in = 32'b00000000000000000000000000011000; #40;
         pc_in = 32'b00000000000000000000000000111000;  #40;
         pc_in = 32'b00000000000000000000000000100000;  #40;
         pc_in = 32'b00000000000000000000000000000000;  #40;
         pc_in = 32'b00000000000000000000000000101000;  #40;
         pc_in = 32'b00000000000000000000000000101100;  #40;
         pc_in = 32'b00000000000000000000000000110000;  #40;
         pc_in = 32'b00000000000000000000000000110100; #40;
         pc_in = 32'b00000000000000000000000000111000;  #40;
         pc_in = 32'b00000000000000000000000000111100;  #40
pc_in = 32'b00000000000000000000000001000000; #40;
     end
endmodule
```

Timing For each instruction :

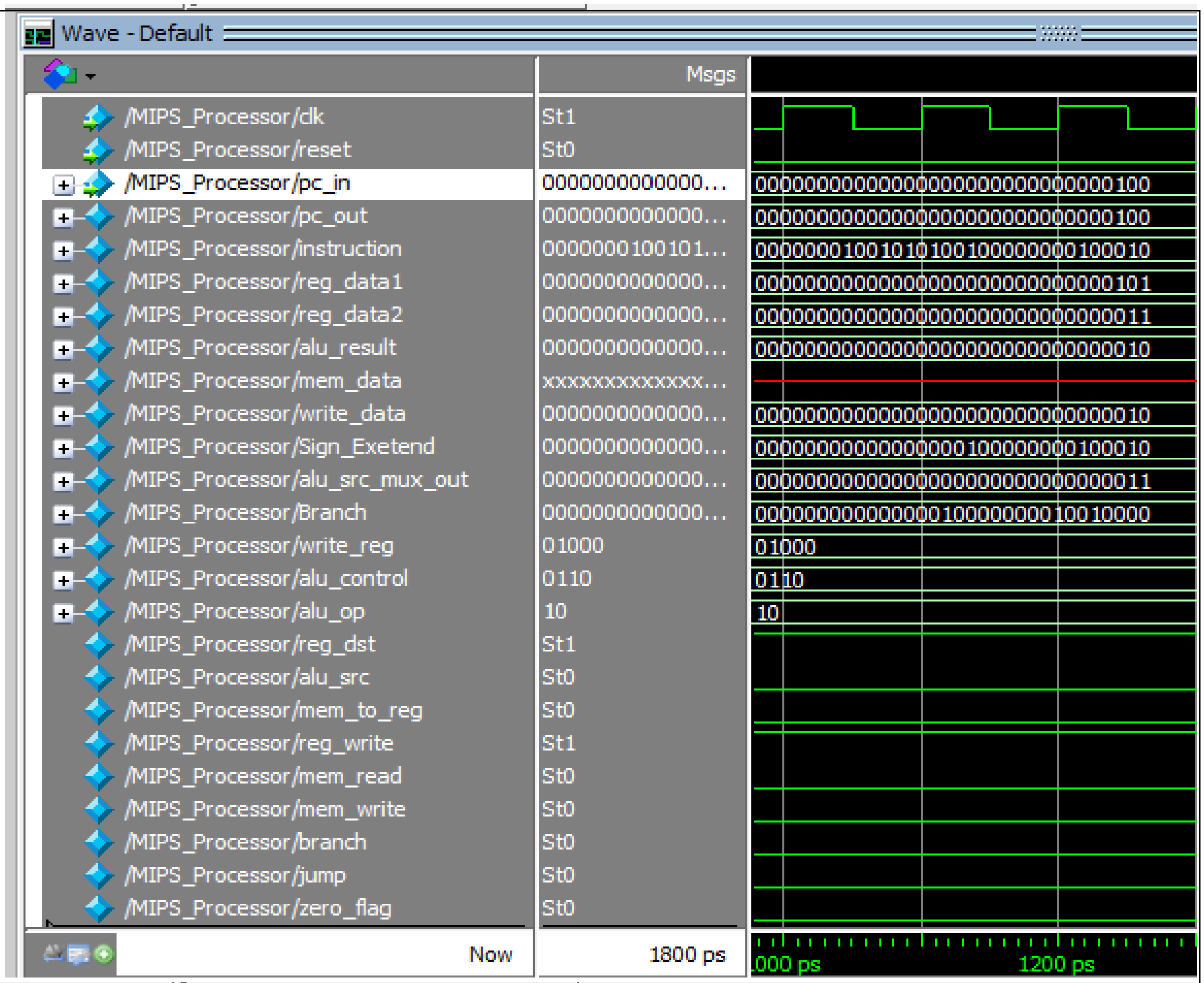
1)ADD:



Wave - Default

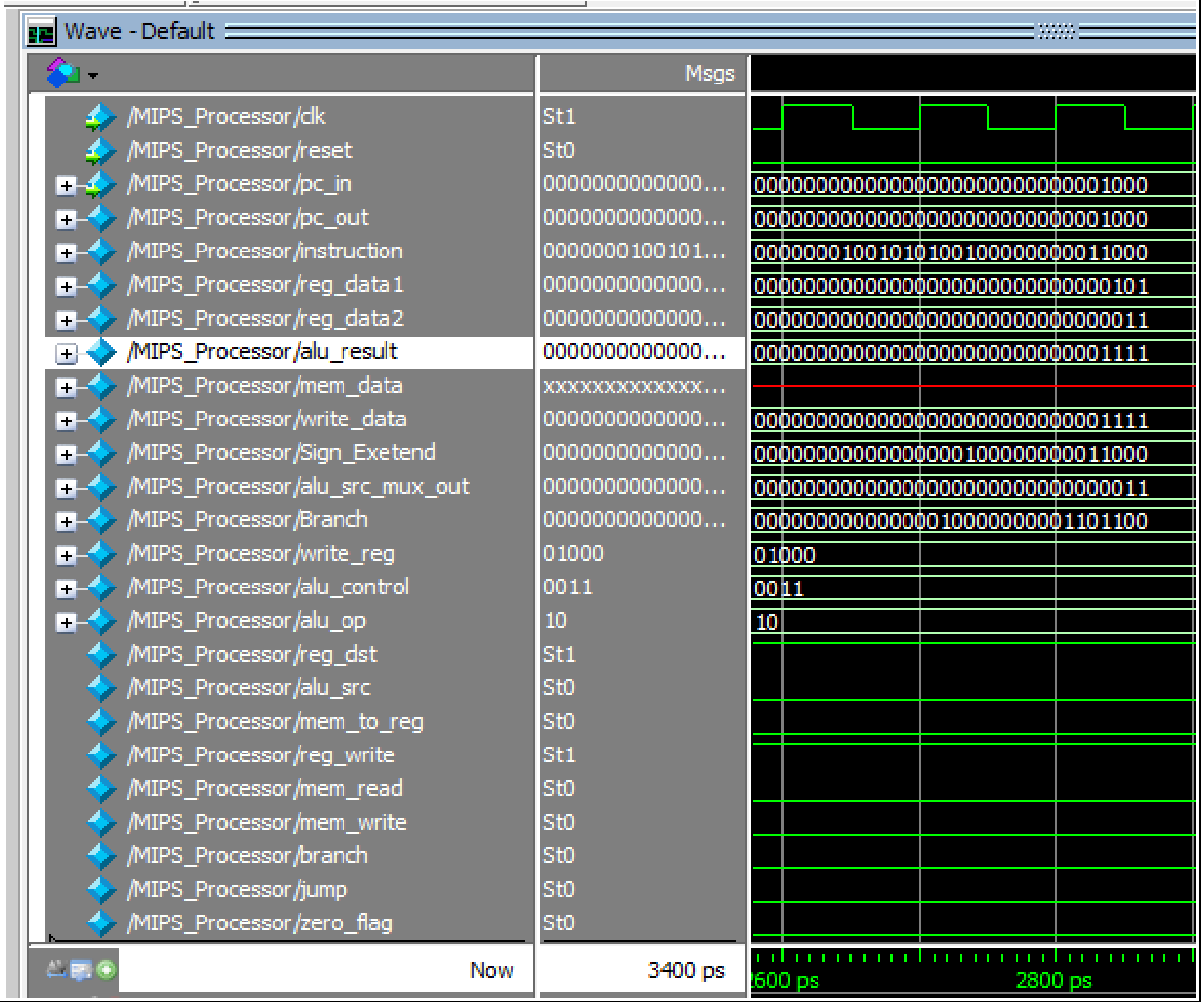
```
                                  Msgs
/MIPS_Processor/clk               St1
/MIPS_Processor/reset             St0
/MIPS_Processor/pc_in             0000000000000...   00000000000000000000000000000000
/MIPS_Processor/pc_out            0000000000000...   00000000000000000000000000000000
/MIPS_Processor/instruction       0000000100101...   00000001001010100100000000100000
/MIPS_Processor/reg_data1         0000000000000...   00000000000000000000000000000101
/MIPS_Processor/reg_data2         0000000000000...   00000000000000000000000000000011
/MIPS_Processor/alu_result        0000000000000...   00000000000000000000000000001000
/MIPS_Processor/mem_data          xxxxxxxxxxxxx...
/MIPS_Processor/write_data        0000000000000...   00000000000000000000000000001000
/MIPS_Processor/Sign_Exetend      0000000000000...   00000000000000000100000000100000
/MIPS_Processor/alu_src_mux_out   0000000000000...   00000000000000000000000000000011
/MIPS_Processor/Branch            0000000000000...   00000000000000010000000010000100
/MIPS_Processor/write_reg         01000              01000
/MIPS_Processor/alu_control       0010               0010
/MIPS_Processor/alu_op            10                 10
/MIPS_Processor/reg_dst           St1
/MIPS_Processor/alu_src           St0
/MIPS_Processor/mem_to_reg        St0
/MIPS_Processor/reg_write         St1
/MIPS_Processor/mem_read          St0
/MIPS_Processor/mem_write         St0
/MIPS_Processor/branch            St0
/MIPS_Processor/jump              St0
/MIPS_Processor/zero_flag         St0
                            Now   400 ps
```

2) SUB:



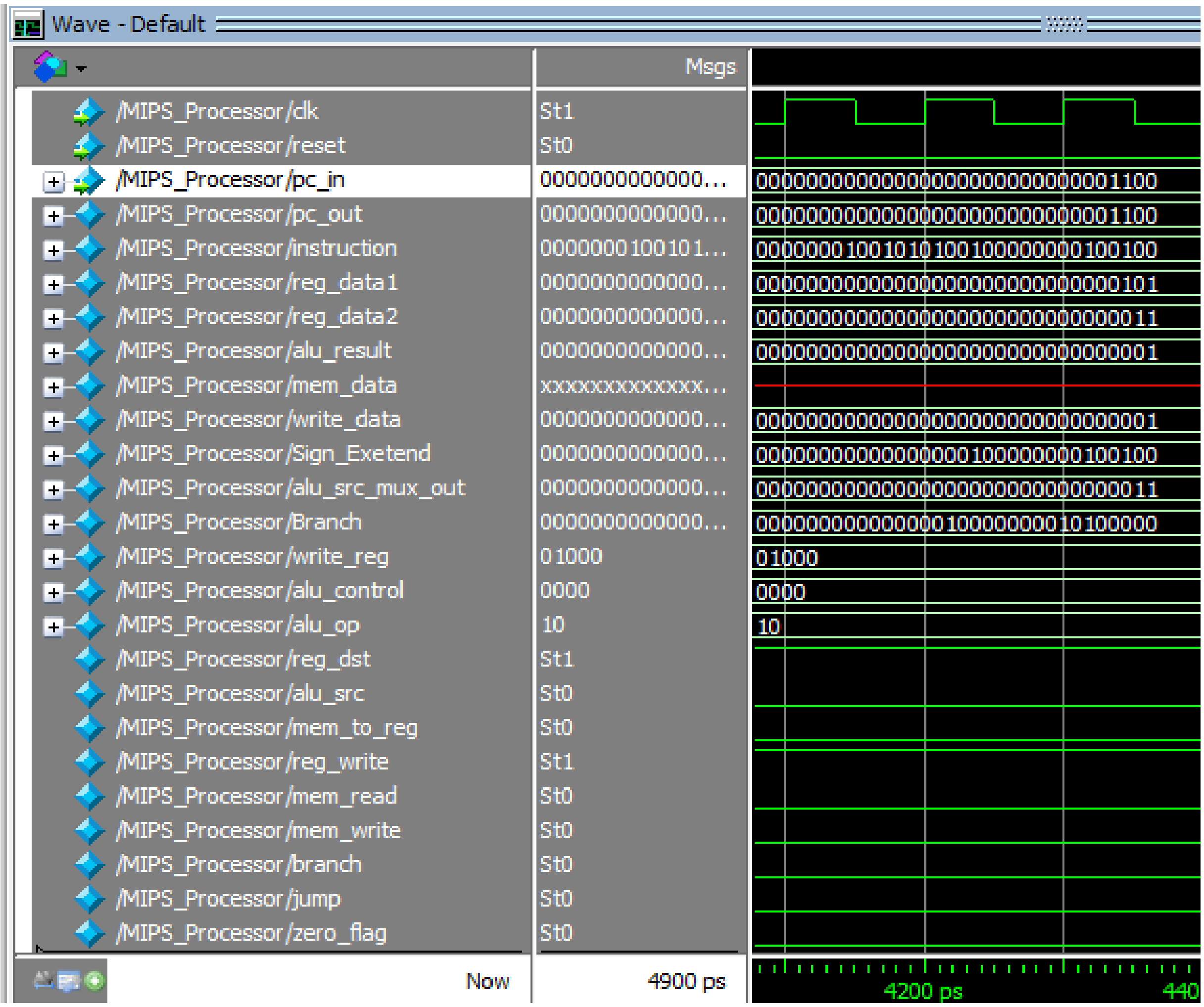
	Msgs	
/MIPS_Processor/clk	St1	
/MIPS_Processor/reset	St0	
/MIPS_Processor/pc_in	0000000000000...	00000000000000000000000000000100
/MIPS_Processor/pc_out	0000000000000...	00000000000000000000000000000100
/MIPS_Processor/instruction	0000000100101...	00000001001010100100000000100010
/MIPS_Processor/reg_data1	0000000000000...	00000000000000000000000000000101
/MIPS_Processor/reg_data2	0000000000000...	00000000000000000000000000000011
/MIPS_Processor/alu_result	0000000000000...	00000000000000000000000000000010
/MIPS_Processor/mem_data	xxxxxxxxxxxxx...	
/MIPS_Processor/write_data	0000000000000...	00000000000000000000000000000010
/MIPS_Processor/Sign_Exetend	0000000000000...	00000000000000000100000000100010
/MIPS_Processor/alu_src_mux_out	0000000000000...	00000000000000000000000000000011
/MIPS_Processor/Branch	0000000000000...	00000000000000010000000010010000
/MIPS_Processor/write_reg	01000	01000
/MIPS_Processor/alu_control	0110	0110
/MIPS_Processor/alu_op	10	10
/MIPS_Processor/reg_dst	St1	
/MIPS_Processor/alu_src	St0	
/MIPS_Processor/mem_to_reg	St0	
/MIPS_Processor/reg_write	St1	
/MIPS_Processor/mem_read	St0	
/MIPS_Processor/mem_write	St0	
/MIPS_Processor/branch	St0	
/MIPS_Processor/jump	St0	
/MIPS_Processor/zero_flag	St0	
Now	1800 ps	000 ps 1200 ps

3) MUL:



	Msgs	
/MIPS_Processor/clk	St1	
/MIPS_Processor/reset	St0	
/MIPS_Processor/pc_in	0000000000000...	00000000000000000000000000001000
/MIPS_Processor/pc_out	0000000000000...	00000000000000000000000000001000
/MIPS_Processor/instruction	0000000100101...	00000001001010100100000000011000
/MIPS_Processor/reg_data1	0000000000000...	00000000000000000000000000000101
/MIPS_Processor/reg_data2	0000000000000...	00000000000000000000000000000011
/MIPS_Processor/alu_result	0000000000000...	00000000000000000000000000001111
/MIPS_Processor/mem_data	xxxxxxxxxxxxx...	
/MIPS_Processor/write_data	0000000000000...	00000000000000000000000000001111
/MIPS_Processor/Sign_Exetend	0000000000000...	00000000000000000100000000011000
/MIPS_Processor/alu_src_mux_out	0000000000000...	00000000000000000000000000000011
/MIPS_Processor/Branch	0000000000000...	00000000000000010000000001101100
/MIPS_Processor/write_reg	01000	01000
/MIPS_Processor/alu_control	0011	0011
/MIPS_Processor/alu_op	10	10
/MIPS_Processor/reg_dst	St1	
/MIPS_Processor/alu_src	St0	
/MIPS_Processor/mem_to_reg	St0	
/MIPS_Processor/reg_write	St1	
/MIPS_Processor/mem_read	St0	
/MIPS_Processor/mem_write	St0	
/MIPS_Processor/branch	St0	
/MIPS_Processor/jump	St0	
/MIPS_Processor/zero_flag	St0	
Now	3400 ps	600 ps 2800 ps

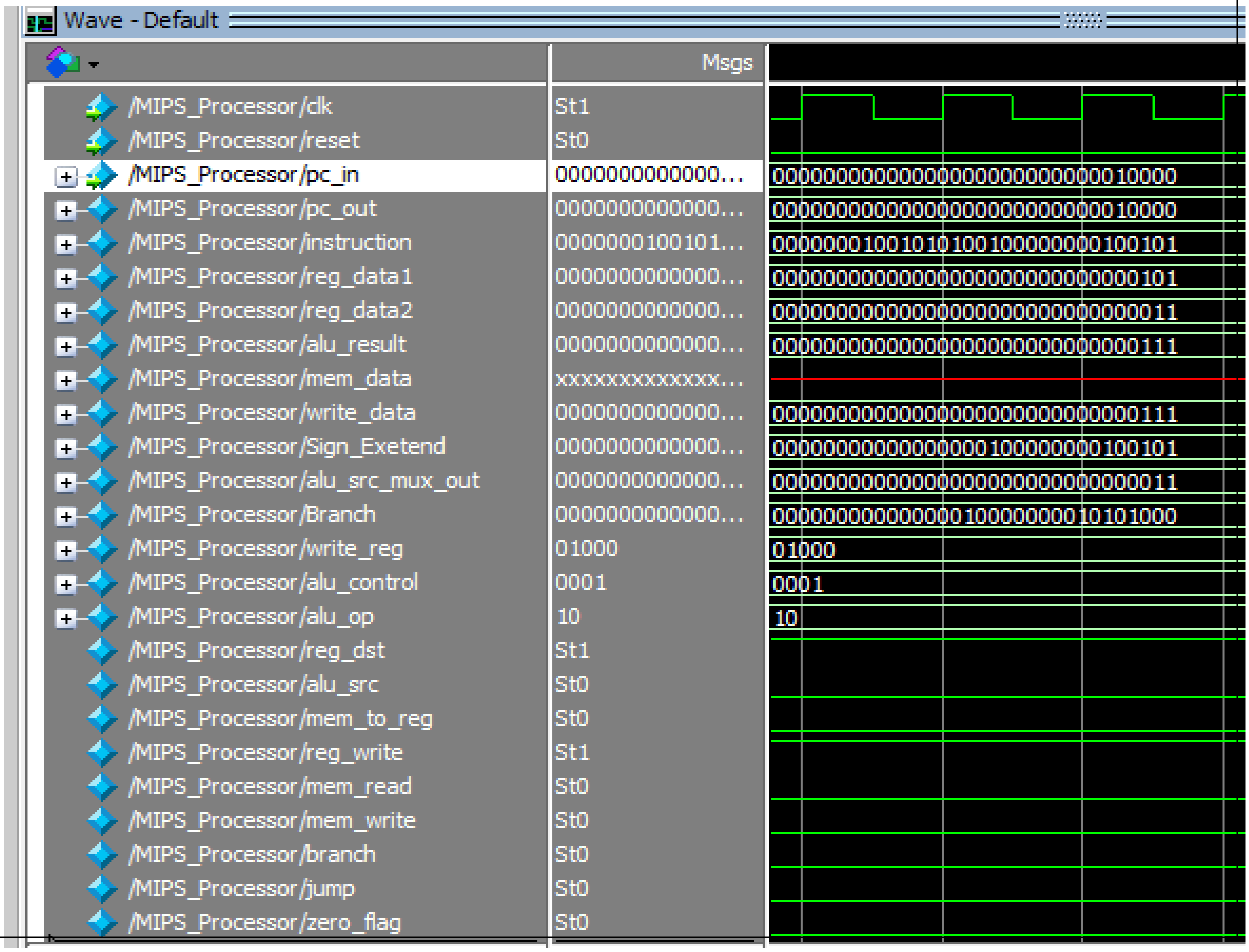
4)AND:



Wave - Default

	Msgs	
/MIPS_Processor/clk	St1	
/MIPS_Processor/reset	St0	
/MIPS_Processor/pc_in	0000000000000...	00000000000000000000000000001100
/MIPS_Processor/pc_out	0000000000000...	00000000000000000000000000001100
/MIPS_Processor/instruction	0000000100101...	00000001001010100100000000100100
/MIPS_Processor/reg_data1	0000000000000...	00000000000000000000000000000101
/MIPS_Processor/reg_data2	0000000000000...	00000000000000000000000000000011
/MIPS_Processor/alu_result	0000000000000...	00000000000000000000000000000001
/MIPS_Processor/mem_data	xxxxxxxxxxxxx...	
/MIPS_Processor/write_data	0000000000000...	00000000000000000000000000000001
/MIPS_Processor/Sign_Exetend	0000000000000...	00000000000000000100000000100100
/MIPS_Processor/alu_src_mux_out	0000000000000...	00000000000000000000000000000011
/MIPS_Processor/Branch	0000000000000...	00000000000000010000000010100000
/MIPS_Processor/write_reg	01000	01000
/MIPS_Processor/alu_control	0000	0000
/MIPS_Processor/alu_op	10	10
/MIPS_Processor/reg_dst	St1	
/MIPS_Processor/alu_src	St0	
/MIPS_Processor/mem_to_reg	St0	
/MIPS_Processor/reg_write	St1	
/MIPS_Processor/mem_read	St0	
/MIPS_Processor/mem_write	St0	
/MIPS_Processor/branch	St0	
/MIPS_Processor/jump	St0	
/MIPS_Processor/zero_flag	St0	
Now	4900 ps	4200 ps 440

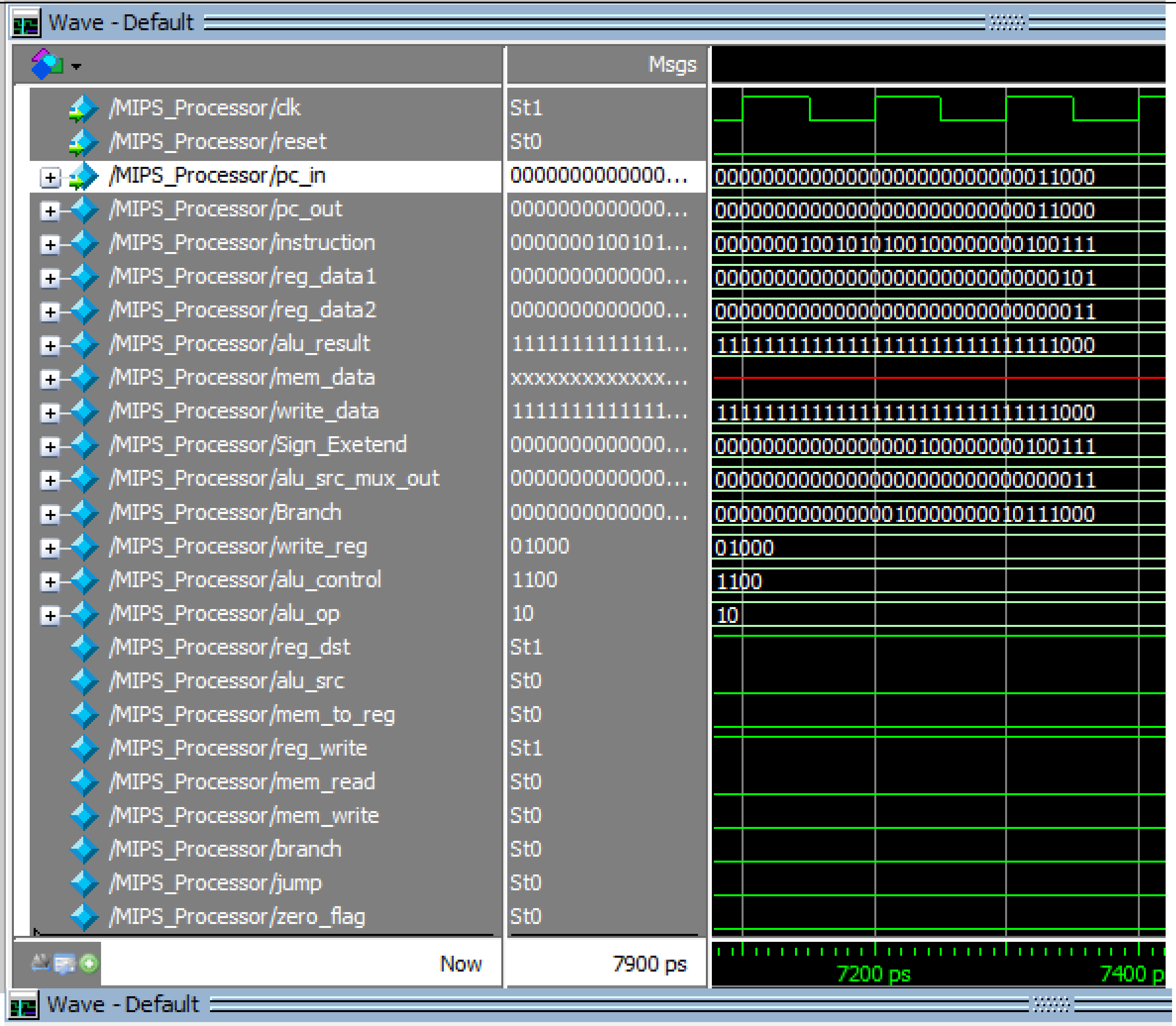
5)OR:



Wave - Default

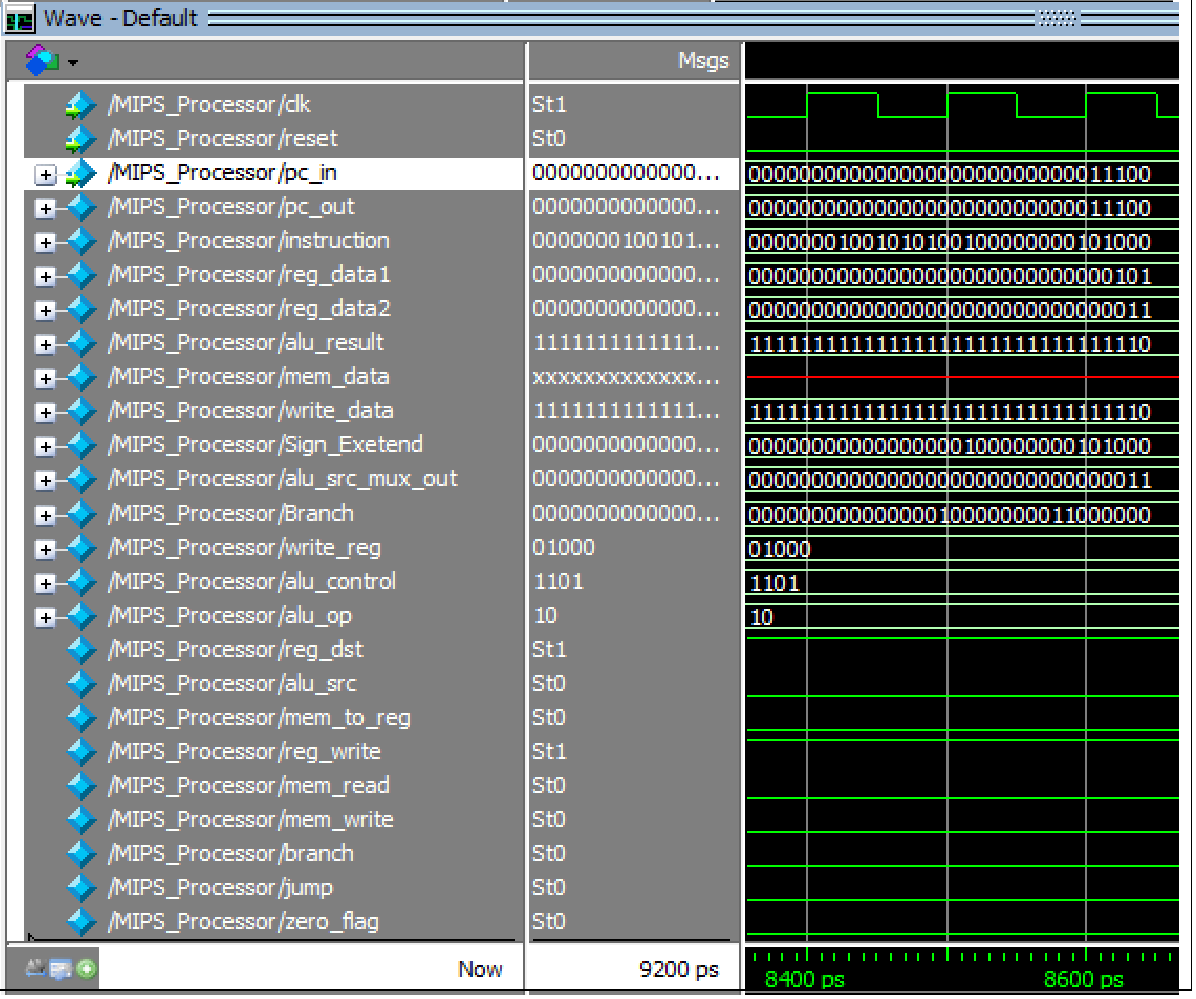
	Msgs	
/MIPS_Processor/clk	St1	
/MIPS_Processor/reset	St0	
/MIPS_Processor/pc_in	0000000000000...	00000000000000000000000000010000
/MIPS_Processor/pc_out	0000000000000...	00000000000000000000000000010000
/MIPS_Processor/instruction	0000000100101...	00000001001010100100000000100101
/MIPS_Processor/reg_data1	0000000000000...	00000000000000000000000000000101
/MIPS_Processor/reg_data2	0000000000000...	00000000000000000000000000000011
/MIPS_Processor/alu_result	0000000000000...	00000000000000000000000000000111
/MIPS_Processor/mem_data	xxxxxxxxxxxxx...	
/MIPS_Processor/write_data	0000000000000...	00000000000000000000000000000111
/MIPS_Processor/Sign_Exetend	0000000000000...	00000000000000000100000000100101
/MIPS_Processor/alu_src_mux_out	0000000000000...	00000000000000000000000000000011
/MIPS_Processor/Branch	0000000000000...	00000000000000010000000010101000
/MIPS_Processor/write_reg	01000	01000
/MIPS_Processor/alu_control	0001	0001
/MIPS_Processor/alu_op	10	10
/MIPS_Processor/reg_dst	St1	
/MIPS_Processor/alu_src	St0	
/MIPS_Processor/mem_to_reg	St0	
/MIPS_Processor/reg_write	St1	
/MIPS_Processor/mem_read	St0	
/MIPS_Processor/mem_write	St0	
/MIPS_Processor/branch	St0	
/MIPS_Processor/jump	St0	
/MIPS_Processor/zero_flag	St0	

6)NOR:



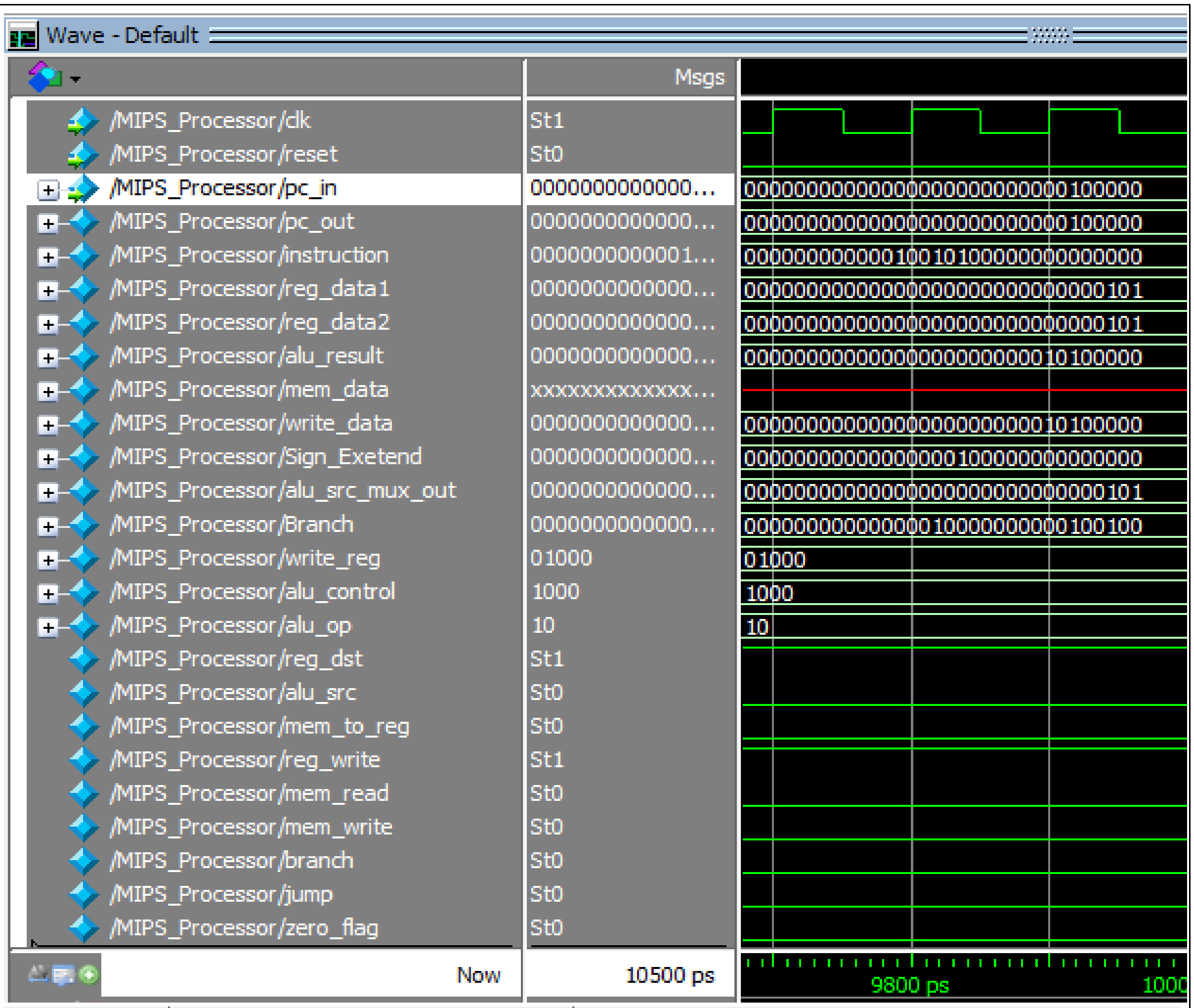
```
Wave - Default
                                       Msgs
/MIPS_Processor/clk                    St1
/MIPS_Processor/reset                  St0
/MIPS_Processor/pc_in                  0000000000000...   00000000000000000000000000011000
/MIPS_Processor/pc_out                 0000000000000...   00000000000000000000000000011000
/MIPS_Processor/instruction            0000000100101...   00000001001010100100000000100111
/MIPS_Processor/reg_data1              0000000000000...   00000000000000000000000000000101
/MIPS_Processor/reg_data2              0000000000000...   00000000000000000000000000000011
/MIPS_Processor/alu_result             1111111111111...   11111111111111111111111111111000
/MIPS_Processor/mem_data               xxxxxxxxxxxxx...
/MIPS_Processor/write_data             1111111111111...   11111111111111111111111111111000
/MIPS_Processor/Sign_Exetend           0000000000000...   00000000000000000100000000100111
/MIPS_Processor/alu_src_mux_out        0000000000000...   00000000000000000000000000000011
/MIPS_Processor/Branch                 0000000000000...   00000000000000010000000010111000
/MIPS_Processor/write_reg              01000              01000
/MIPS_Processor/alu_control            1100               1100
/MIPS_Processor/alu_op                 10                 10
/MIPS_Processor/reg_dst                St1
/MIPS_Processor/alu_src                St0
/MIPS_Processor/mem_to_reg             St0
/MIPS_Processor/reg_write              St1
/MIPS_Processor/mem_read               St0
/MIPS_Processor/mem_write              St0
/MIPS_Processor/branch                 St0
/MIPS_Processor/jump                   St0
/MIPS_Processor/zero_flag              St0
                                  Now  7900 ps             7200 ps        7400 p
```

7)NAND:



```
Wave - Default
                                       Msgs
/MIPS_Processor/clk                    St1
/MIPS_Processor/reset                  St0
/MIPS_Processor/pc_in                  0000000000000...   00000000000000000000000000011100
/MIPS_Processor/pc_out                 0000000000000...   00000000000000000000000000011100
/MIPS_Processor/instruction            0000000100101...   00000001001010100100000000101000
/MIPS_Processor/reg_data1              0000000000000...   00000000000000000000000000000101
/MIPS_Processor/reg_data2              0000000000000...   00000000000000000000000000000011
/MIPS_Processor/alu_result             1111111111111...   11111111111111111111111111111110
/MIPS_Processor/mem_data               xxxxxxxxxxxxx...
/MIPS_Processor/write_data             1111111111111...   11111111111111111111111111111110
/MIPS_Processor/Sign_Exetend           0000000000000...   00000000000000000100000000101000
/MIPS_Processor/alu_src_mux_out        0000000000000...   00000000000000000000000000000011
/MIPS_Processor/Branch                 0000000000000...   00000000000000010000000011000000
/MIPS_Processor/write_reg              01000              01000
/MIPS_Processor/alu_control            1101               1101
/MIPS_Processor/alu_op                 10                 10
/MIPS_Processor/reg_dst                St1
/MIPS_Processor/alu_src                St0
/MIPS_Processor/mem_to_reg             St0
/MIPS_Processor/reg_write              St1
/MIPS_Processor/mem_read               St0
/MIPS_Processor/mem_write              St0
/MIPS_Processor/branch                 St0
/MIPS_Processor/jump                   St0
/MIPS_Processor/zero_flag              St0
                                  Now  9200 ps             8400 ps        8600 ps
```

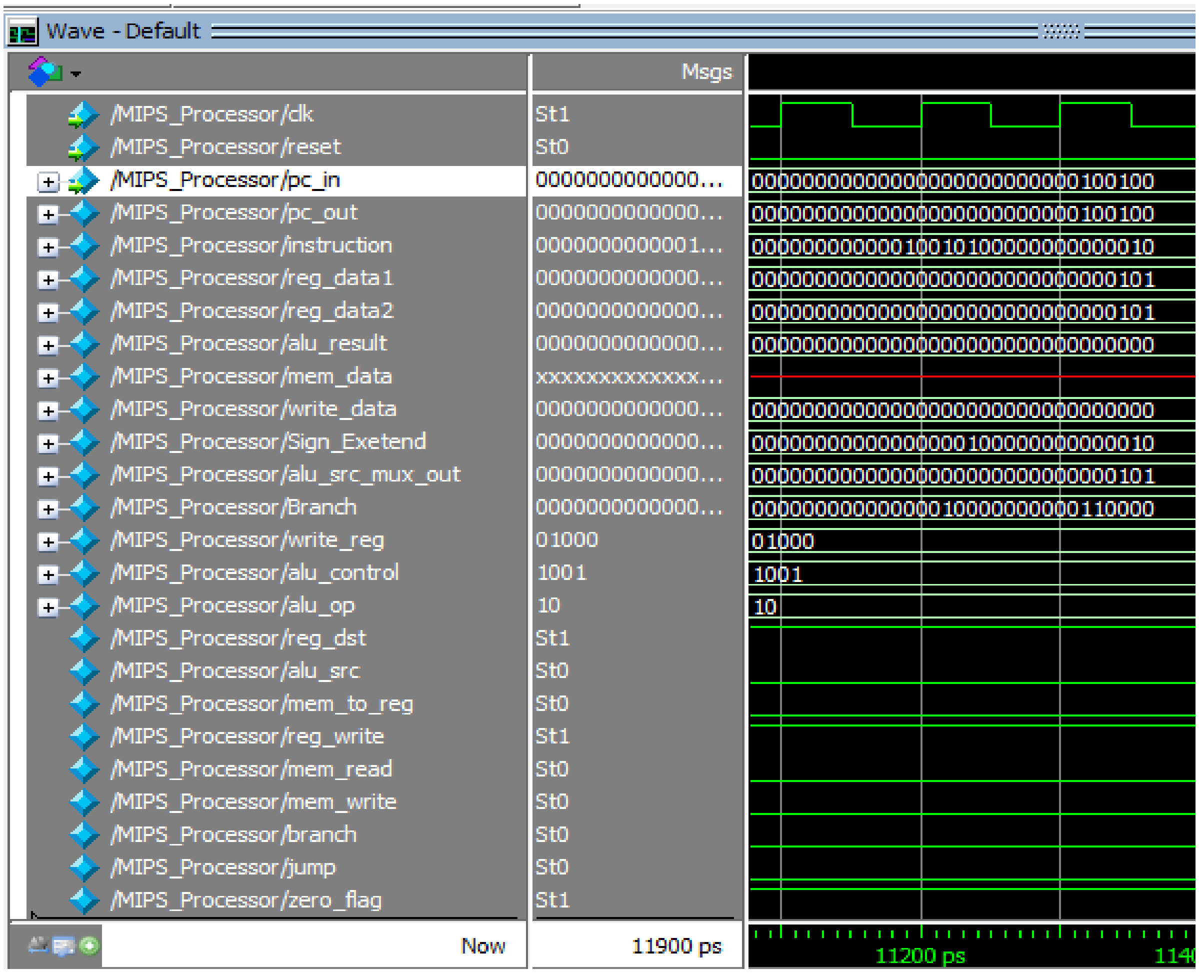
8)SHL:



Wave - Default

	Msgs	
/MIPS_Processor/clk	St1	
/MIPS_Processor/reset	St0	
/MIPS_Processor/pc_in	0000000000000...	00000000000000000000000000100000
/MIPS_Processor/pc_out	0000000000000...	00000000000000000000000000100000
/MIPS_Processor/instruction	0000000000001...	00000000000010010100000000000000
/MIPS_Processor/reg_data1	0000000000000...	00000000000000000000000000000101
/MIPS_Processor/reg_data2	0000000000000...	00000000000000000000000000000101
/MIPS_Processor/alu_result	0000000000000...	00000000000000000000000010100000
/MIPS_Processor/mem_data	xxxxxxxxxxxxx...	
/MIPS_Processor/write_data	0000000000000...	00000000000000000000000010100000
/MIPS_Processor/Sign_Exetend	0000000000000...	00000000000000000100000000000000
/MIPS_Processor/alu_src_mux_out	0000000000000...	00000000000000000000000000000101
/MIPS_Processor/Branch	0000000000000...	00000000000000010000000000100100
/MIPS_Processor/write_reg	01000	01000
/MIPS_Processor/alu_control	1000	1000
/MIPS_Processor/alu_op	10	10
/MIPS_Processor/reg_dst	St1	
/MIPS_Processor/alu_src	St0	
/MIPS_Processor/mem_to_reg	St0	
/MIPS_Processor/reg_write	St1	
/MIPS_Processor/mem_read	St0	
/MIPS_Processor/mem_write	St0	
/MIPS_Processor/branch	St0	
/MIPS_Processor/jump	St0	
/MIPS_Processor/zero_flag	St0	
Now	10500 ps	

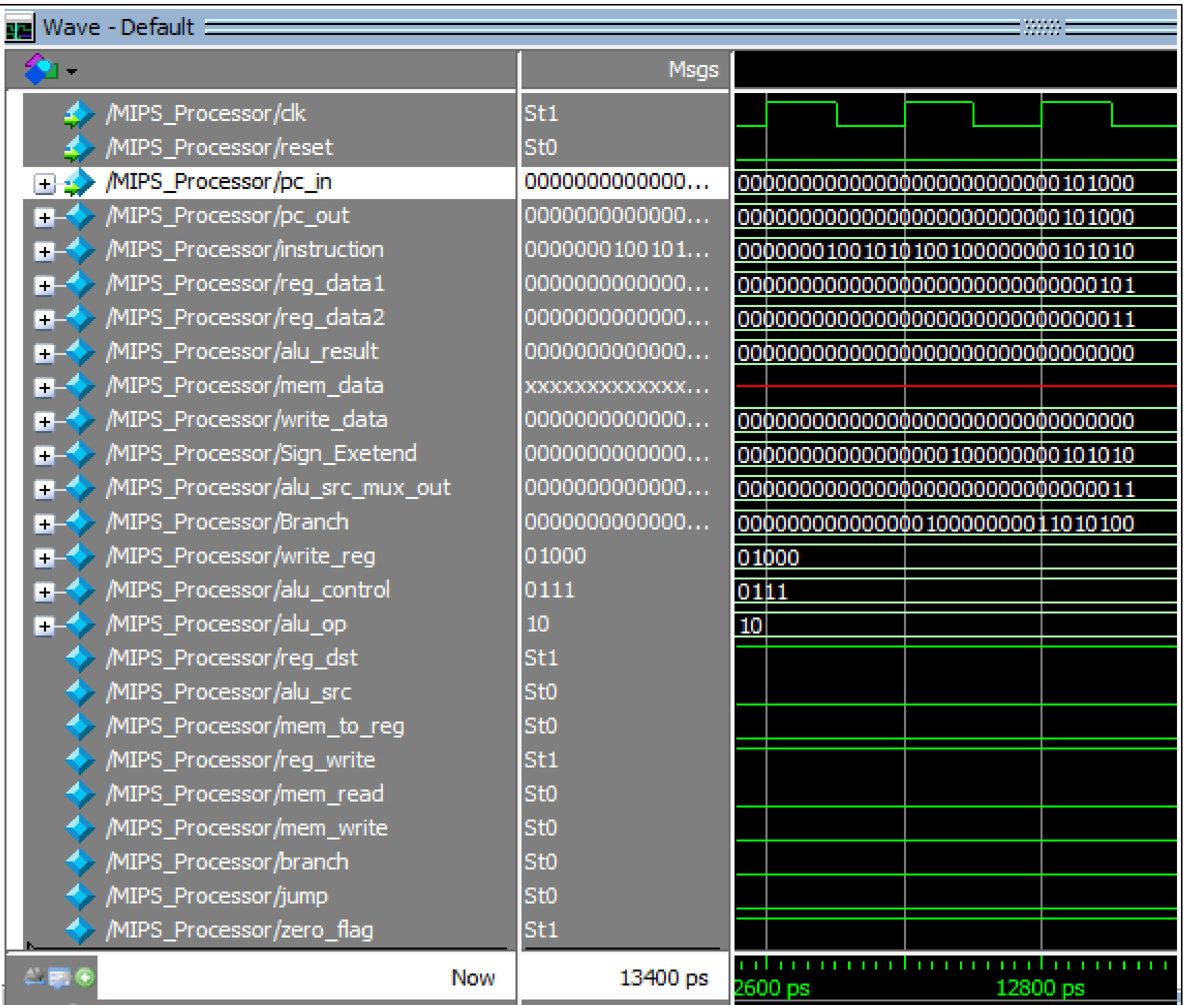
9)SHR:



Wave - Default

	Msgs	
/MIPS_Processor/clk	St1	
/MIPS_Processor/reset	St0	
/MIPS_Processor/pc_in	0000000000000...	00000000000000000000000000100100
/MIPS_Processor/pc_out	0000000000000...	00000000000000000000000000100100
/MIPS_Processor/instruction	0000000000001...	00000000000010010100000000000010
/MIPS_Processor/reg_data1	0000000000000...	00000000000000000000000000000101
/MIPS_Processor/reg_data2	0000000000000...	00000000000000000000000000000101
/MIPS_Processor/alu_result	0000000000000...	00000000000000000000000000000000
/MIPS_Processor/mem_data	xxxxxxxxxxxxx...	
/MIPS_Processor/write_data	0000000000000...	00000000000000000000000000000000
/MIPS_Processor/Sign_Exetend	0000000000000...	00000000000000000100000000000010
/MIPS_Processor/alu_src_mux_out	0000000000000...	00000000000000000000000000000101
/MIPS_Processor/Branch	0000000000000...	00000000000000010000000000110000
/MIPS_Processor/write_reg	01000	01000
/MIPS_Processor/alu_control	1001	1001
/MIPS_Processor/alu_op	10	10
/MIPS_Processor/reg_dst	St1	
/MIPS_Processor/alu_src	St0	
/MIPS_Processor/mem_to_reg	St0	
/MIPS_Processor/reg_write	St1	
/MIPS_Processor/mem_read	St0	
/MIPS_Processor/mem_write	St0	
/MIPS_Processor/branch	St0	
/MIPS_Processor/jump	St0	
/MIPS_Processor/zero_flag	St1	
Now	11900 ps	

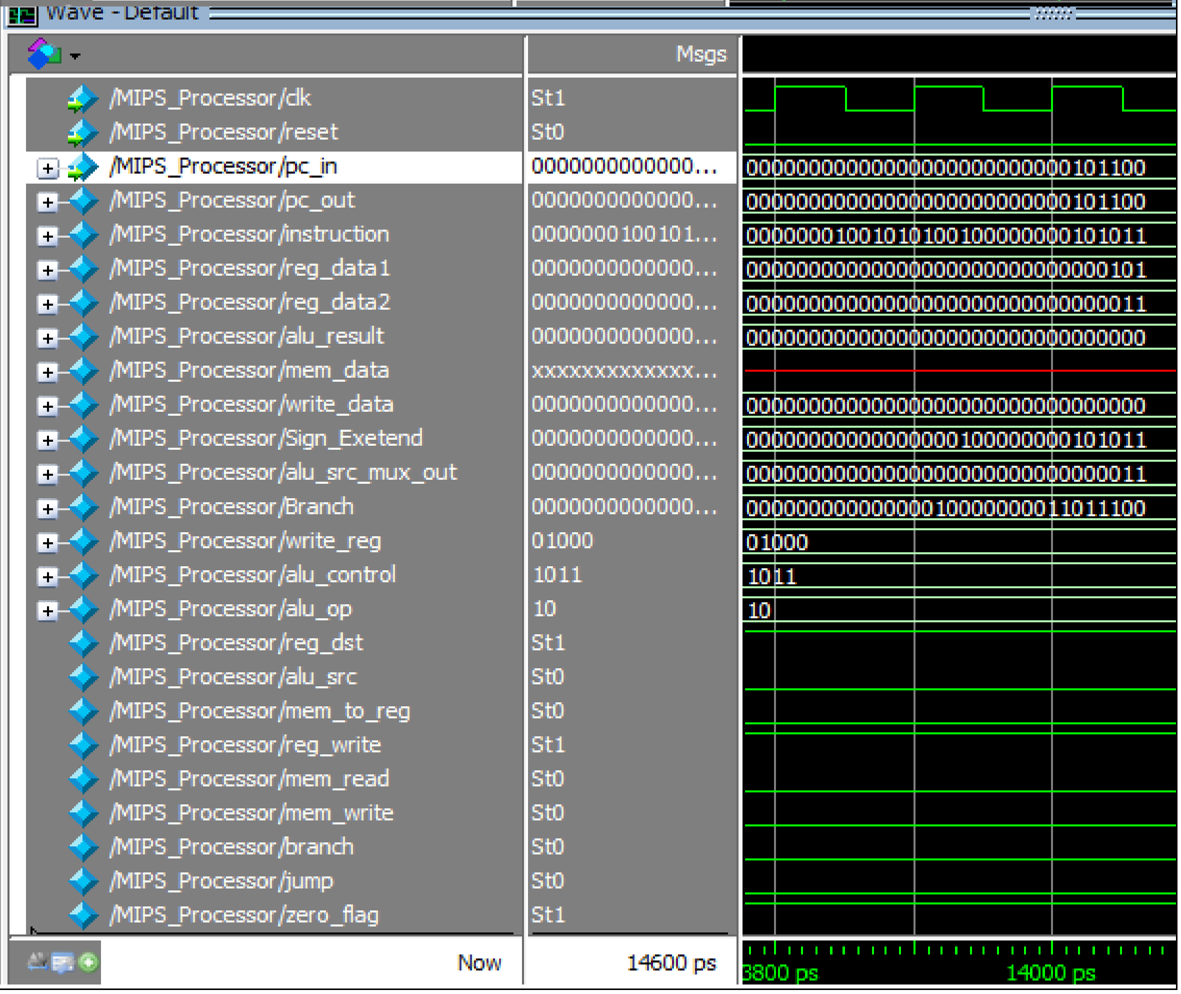
10)SLT :



Wave - Default

	Msgs	
/MIPS_Processor/clk	St1	
/MIPS_Processor/reset	St0	
/MIPS_Processor/pc_in	0000000000000...	00000000000000000000000000101000
/MIPS_Processor/pc_out	0000000000000...	00000000000000000000000000101000
/MIPS_Processor/instruction	0000000100101...	00000001001010100100000000101010
/MIPS_Processor/reg_data1	0000000000000...	00000000000000000000000000000101
/MIPS_Processor/reg_data2	0000000000000...	00000000000000000000000000000011
/MIPS_Processor/alu_result	0000000000000...	00000000000000000000000000000000
/MIPS_Processor/mem_data	xxxxxxxxxxxxx...	
/MIPS_Processor/write_data	0000000000000...	00000000000000000000000000000000
/MIPS_Processor/Sign_Exetend	0000000000000...	00000000000000000100000000101010
/MIPS_Processor/alu_src_mux_out	0000000000000...	00000000000000000000000000000011
/MIPS_Processor/Branch	0000000000000...	00000000000000010000000011010100
/MIPS_Processor/write_reg	01000	01000
/MIPS_Processor/alu_control	0111	0111
/MIPS_Processor/alu_op	10	10
/MIPS_Processor/reg_dst	St1	
/MIPS_Processor/alu_src	St0	
/MIPS_Processor/mem_to_reg	St0	
/MIPS_Processor/reg_write	St1	
/MIPS_Processor/mem_read	St0	
/MIPS_Processor/mem_write	St0	
/MIPS_Processor/branch	St0	
/MIPS_Processor/jump	St0	
/MIPS_Processor/zero_flag	St1	
Now	13400 ps	2600 ps 12800 ps

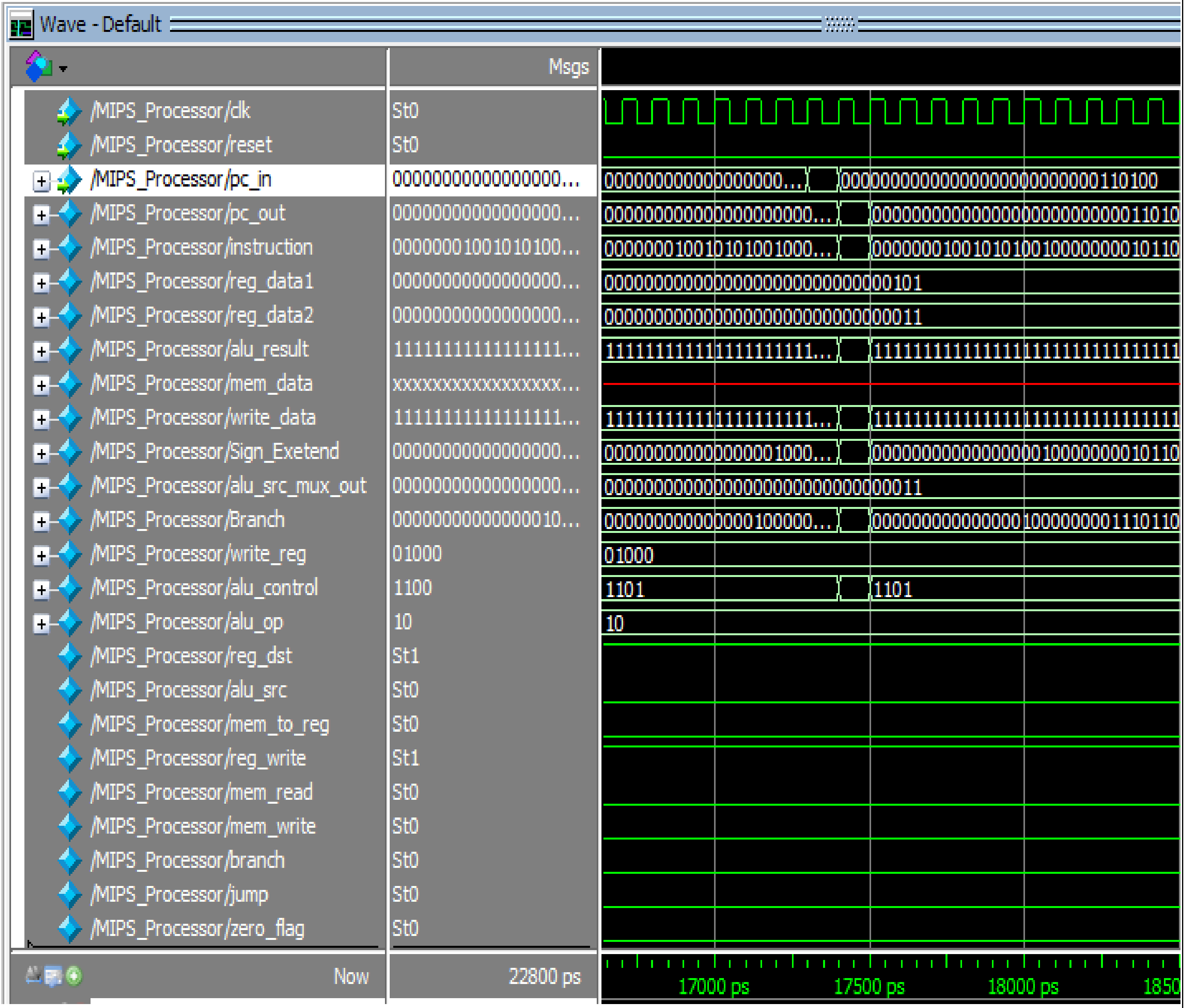
11)SEQ :



Wave - Default

	Msgs	
/MIPS_Processor/clk	St1	
/MIPS_Processor/reset	St0	
/MIPS_Processor/pc_in	0000000000000...	00000000000000000000000000101100
/MIPS_Processor/pc_out	0000000000000...	00000000000000000000000000101100
/MIPS_Processor/instruction	0000000100101...	00000001001010100100000000101011
/MIPS_Processor/reg_data1	0000000000000...	00000000000000000000000000000101
/MIPS_Processor/reg_data2	0000000000000...	00000000000000000000000000000011
/MIPS_Processor/alu_result	0000000000000...	00000000000000000000000000000000
/MIPS_Processor/mem_data	xxxxxxxxxxxxx...	
/MIPS_Processor/write_data	0000000000000...	00000000000000000000000000000000
/MIPS_Processor/Sign_Exetend	0000000000000...	00000000000000000100000000101011
/MIPS_Processor/alu_src_mux_out	0000000000000...	00000000000000000000000000000011
/MIPS_Processor/Branch	0000000000000...	00000000000000010000000011011100
/MIPS_Processor/write_reg	01000	01000
/MIPS_Processor/alu_control	1011	1011
/MIPS_Processor/alu_op	10	10
/MIPS_Processor/reg_dst	St1	
/MIPS_Processor/alu_src	St0	
/MIPS_Processor/mem_to_reg	St0	
/MIPS_Processor/reg_write	St1	
/MIPS_Processor/mem_read	St0	
/MIPS_Processor/mem_write	St0	
/MIPS_Processor/branch	St0	
/MIPS_Processor/jump	St0	
/MIPS_Processor/zero_flag	St1	
Now	14600 ps	3800 ps 14000 ps

12) SNE & 13) SGT (B=3 & A=5)

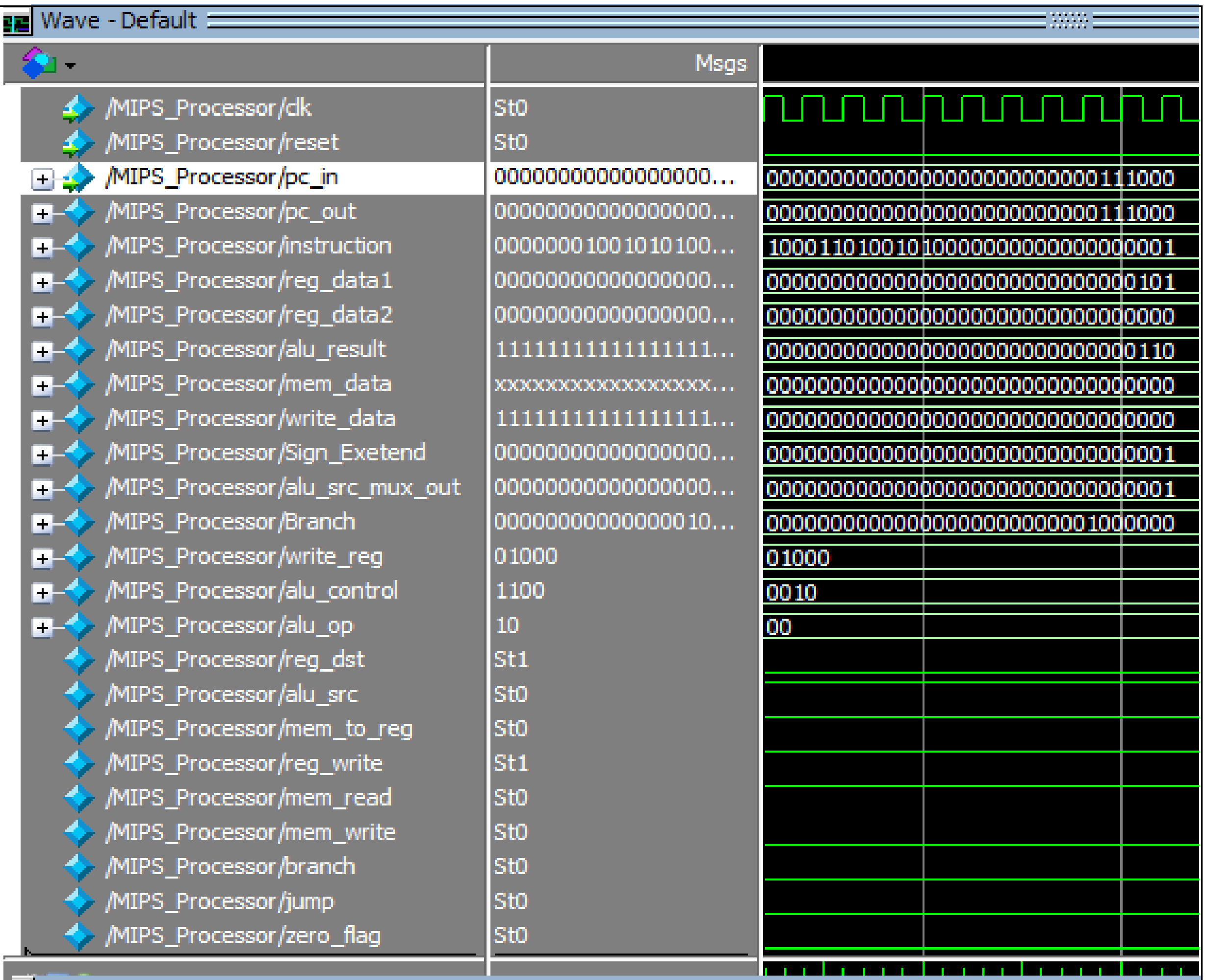


Wave - Default

	Msgs		
/MIPS_Processor/clk	St0		
/MIPS_Processor/reset	St0		
/MIPS_Processor/pc_in	00000000000000000...	000000000000000000...	00000000000000000000000000110100
/MIPS_Processor/pc_out	00000000000000000...	000000000000000000000...	0000000000000000000000000011010
/MIPS_Processor/instruction	00000001001010100...	000000010010101001000...	0000000100101010010000000010110
/MIPS_Processor/reg_data1	00000000000000000...	00000000000000000000000000000101	
/MIPS_Processor/reg_data2	00000000000000000...	00000000000000000000000000000011	
/MIPS_Processor/alu_result	11111111111111111...	111111111111111111111...	1111111111111111111111111111111
/MIPS_Processor/mem_data	xxxxxxxxxxxxxxxxx...		
/MIPS_Processor/write_data	11111111111111111...	111111111111111111111...	1111111111111111111111111111111
/MIPS_Processor/Sign_Exetend	00000000000000000...	000000000000000001000...	0000000000000000010000000010110
/MIPS_Processor/alu_src_mux_out	00000000000000000...	00000000000000000000000000000011	
/MIPS_Processor/Branch	00000000000000010...	0000000000000000100000...	00000000000000001000000001110110
/MIPS_Processor/write_reg	01000	01000	
/MIPS_Processor/alu_control	1100	1101	1101
/MIPS_Processor/alu_op	10	10	
/MIPS_Processor/reg_dst	St1		
/MIPS_Processor/alu_src	St0		
/MIPS_Processor/mem_to_reg	St0		
/MIPS_Processor/reg_write	St1		
/MIPS_Processor/mem_read	St0		
/MIPS_Processor/mem_write	St0		
/MIPS_Processor/branch	St0		
/MIPS_Processor/jump	St0		
/MIPS_Processor/zero_flag	St0		

Now 22800 ps

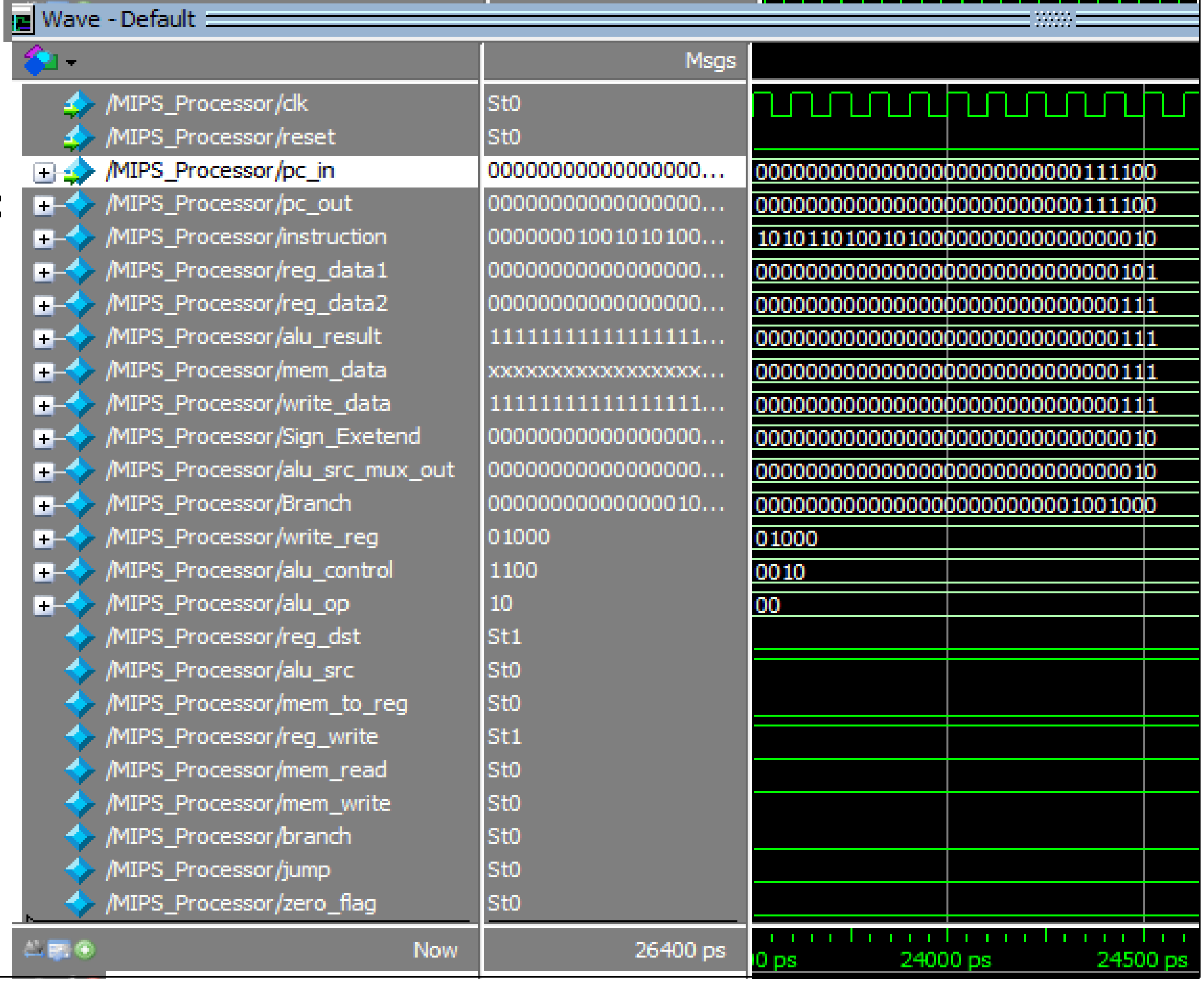
14)LW :



Wave - Default

	Msgs	
/MIPS_Processor/clk	St0	
/MIPS_Processor/reset	St0	
/MIPS_Processor/pc_in	0000000000000000...	00000000000000000000000000111000
/MIPS_Processor/pc_out	0000000000000000...	00000000000000000000000000111000
/MIPS_Processor/instruction	00000001001010100...	10001101001010000000000000000001
/MIPS_Processor/reg_data1	0000000000000000...	00000000000000000000000000000101
/MIPS_Processor/reg_data2	0000000000000000...	00000000000000000000000000000000
/MIPS_Processor/alu_result	1111111111111111...	00000000000000000000000000000110
/MIPS_Processor/mem_data	xxxxxxxxxxxxxxxx...	00000000000000000000000000000000
/MIPS_Processor/write_data	1111111111111111...	00000000000000000000000000000000
/MIPS_Processor/Sign_Exetend	0000000000000000...	00000000000000000000000000000001
/MIPS_Processor/alu_src_mux_out	0000000000000000...	00000000000000000000000000000001
/MIPS_Processor/Branch	0000000000000010...	00000000000000000000000001000000
/MIPS_Processor/write_reg	01000	01000
/MIPS_Processor/alu_control	1100	0010
/MIPS_Processor/alu_op	10	00
/MIPS_Processor/reg_dst	St1	
/MIPS_Processor/alu_src	St0	
/MIPS_Processor/mem_to_reg	St0	
/MIPS_Processor/reg_write	St1	
/MIPS_Processor/mem_read	St0	
/MIPS_Processor/mem_write	St0	
/MIPS_Processor/branch	St0	
/MIPS_Processor/jump	St0	
/MIPS_Processor/zero_flag	St0	

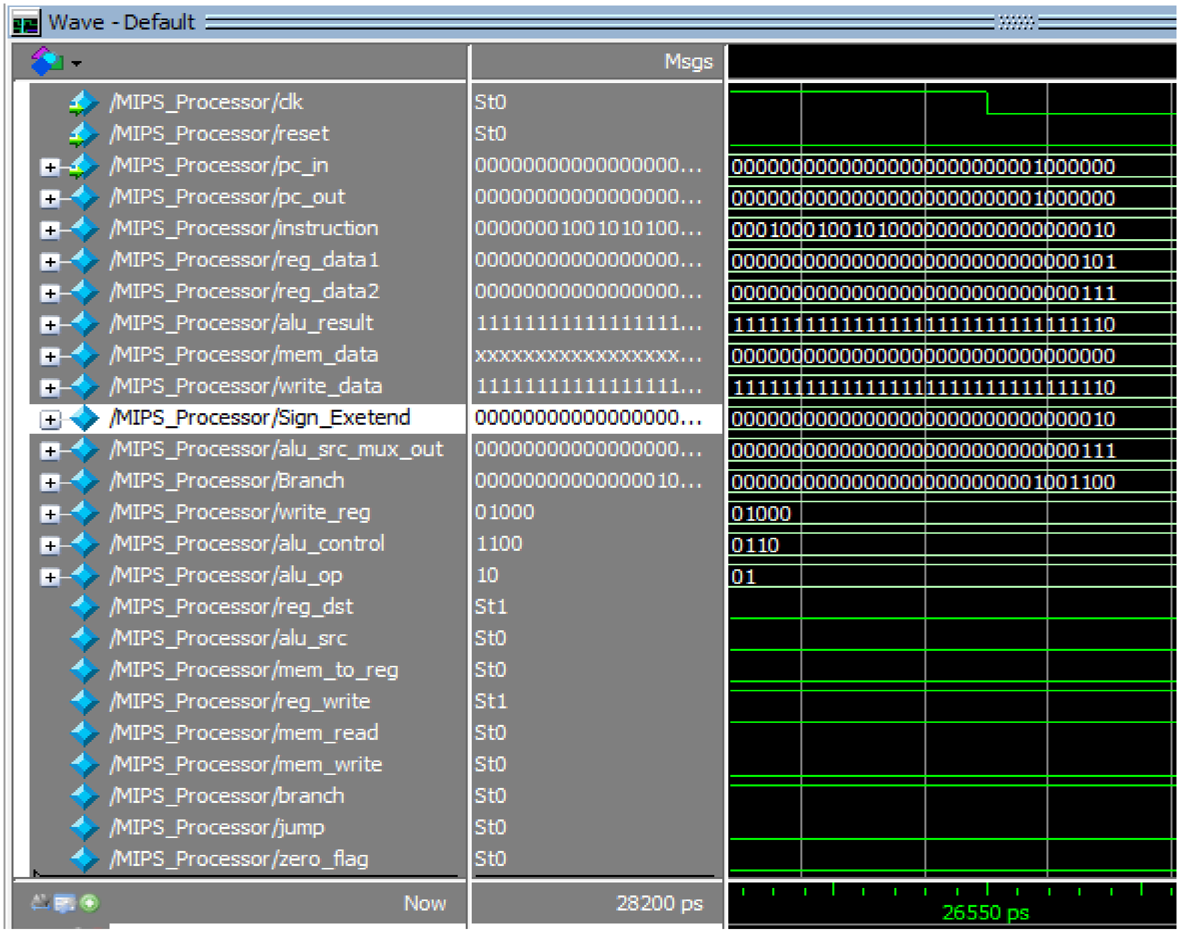
15) SW:



Wave - Default

	Msgs	
/MIPS_Processor/clk	St0	
/MIPS_Processor/reset	St0	
/MIPS_Processor/pc_in	0000000000000000...	00000000000000000000000000111100
/MIPS_Processor/pc_out	0000000000000000...	00000000000000000000000000111100
/MIPS_Processor/instruction	00000001001010100...	10101101001010000000000000000010
/MIPS_Processor/reg_data1	0000000000000000...	00000000000000000000000000000101
/MIPS_Processor/reg_data2	0000000000000000...	00000000000000000000000000000111
/MIPS_Processor/alu_result	1111111111111111...	00000000000000000000000000000111
/MIPS_Processor/mem_data	xxxxxxxxxxxxxxxx...	00000000000000000000000000000111
/MIPS_Processor/write_data	1111111111111111...	00000000000000000000000000000111
/MIPS_Processor/Sign_Exetend	0000000000000000...	00000000000000000000000000000010
/MIPS_Processor/alu_src_mux_out	0000000000000000...	00000000000000000000000000000010
/MIPS_Processor/Branch	0000000000000010...	00000000000000000000000001001000
/MIPS_Processor/write_reg	01000	01000
/MIPS_Processor/alu_control	1100	0010
/MIPS_Processor/alu_op	10	00
/MIPS_Processor/reg_dst	St1	
/MIPS_Processor/alu_src	St0	
/MIPS_Processor/mem_to_reg	St0	
/MIPS_Processor/reg_write	St1	
/MIPS_Processor/mem_read	St0	
/MIPS_Processor/mem_write	St0	
/MIPS_Processor/branch	St0	
/MIPS_Processor/jump	St0	
/MIPS_Processor/zero_flag	St0	
Now	26400 ps	

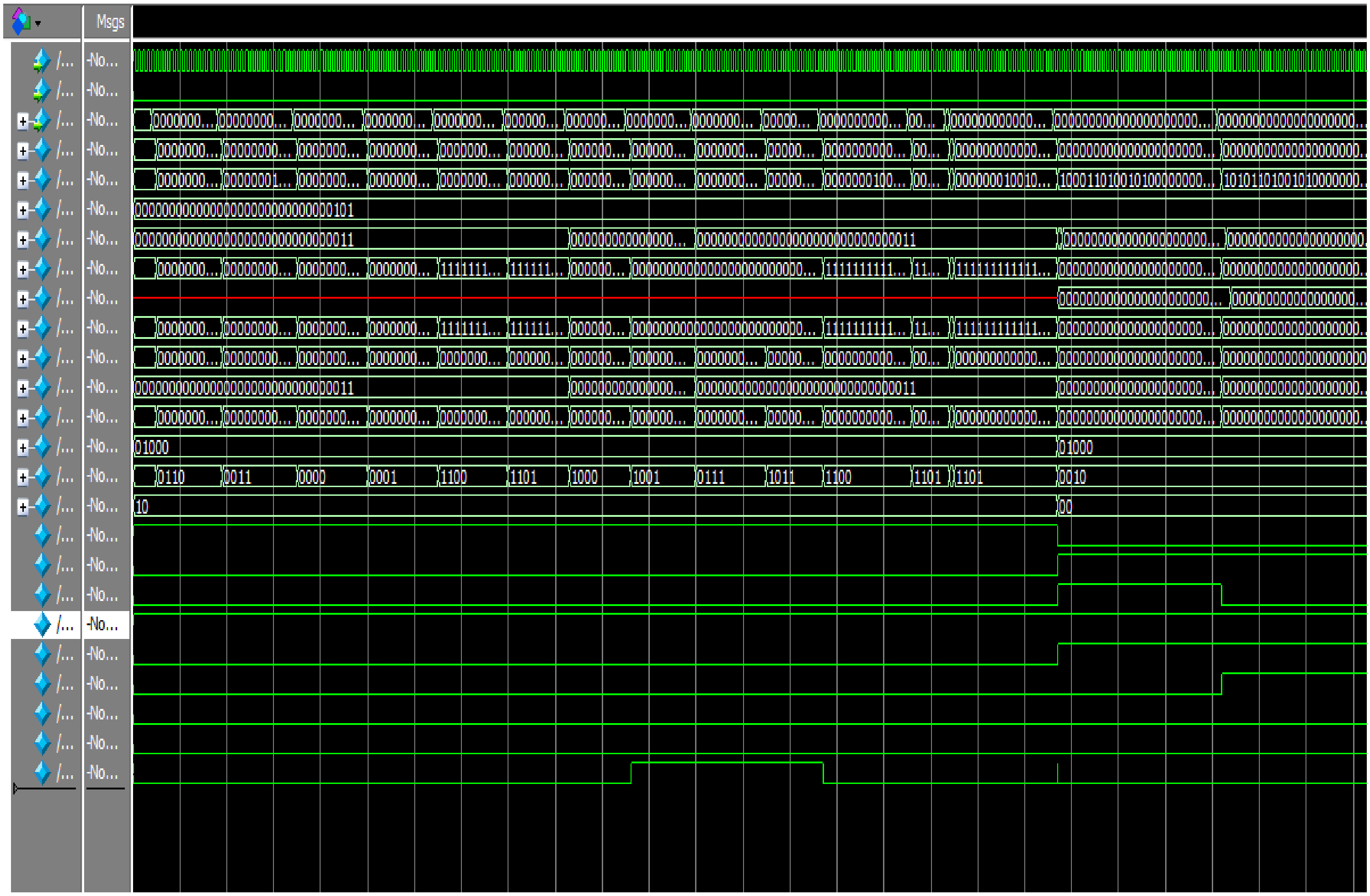
16)BEQ:



Wave - Default

```
                                   Msgs
/MIPS_Processor/clk                St0
/MIPS_Processor/reset              St0
/MIPS_Processor/pc_in              00000000000000000...   00000000000000000000000001000000
/MIPS_Processor/pc_out             00000000000000000...   00000000000000000000000001000000
/MIPS_Processor/instruction        00000001001010100...   00010001001010000000000000000010
/MIPS_Processor/reg_data1          00000000000000000...   00000000000000000000000000000101
/MIPS_Processor/reg_data2          00000000000000000...   00000000000000000000000000000111
/MIPS_Processor/alu_result         11111111111111111...   11111111111111111111111111111110
/MIPS_Processor/mem_data           xxxxxxxxxxxxxxxxx...   00000000000000000000000000000000
/MIPS_Processor/write_data         11111111111111111...   11111111111111111111111111111110
/MIPS_Processor/Sign_Exetend       00000000000000000...   00000000000000000000000000000010
/MIPS_Processor/alu_src_mux_out    00000000000000000...   00000000000000000000000000000111
/MIPS_Processor/Branch             0000000000000010...    00000000000000000000000001001100
/MIPS_Processor/write_reg          01000                  01000
/MIPS_Processor/alu_control        1100                   0110
/MIPS_Processor/alu_op             10                     01
/MIPS_Processor/reg_dst            St1
/MIPS_Processor/alu_src            St0
/MIPS_Processor/mem_to_reg         St0
/MIPS_Processor/reg_write          St1
/MIPS_Processor/mem_read           St0
/MIPS_Processor/mem_write          St0
/MIPS_Processor/branch             St0
/MIPS_Processor/jump               St0
/MIPS_Processor/zero_flag          St0

Now                                28200 ps               26550 ps
```

TESTBENCH OVERVIEW:



```
      Msgs
/...  -No...
/...  -No...
/...  -No...  0000000... 00000000... 0000000... 0000000... 0000000... 000000... 000000... 0000000... 0000000... 00000... 0000000000... 00... 000000000000... 0000000000000000000000... 0000000000000000000000...
/...  -No...  0000000... 00000000... 0000000... 0000000... 0000000... 000000... 000000... 000000... 0000000... 00000... 0000000000... 00... 000000000000... 0000000000000000000000... 0000000000000000000000...
/...  -No...  0000000... 00000001... 0000000... 0000000... 0000000... 000000... 000000... 000000... 0000000... 00000... 0000000100... 00... 000000010010... 1000110100101000000000... 1010110100101000000...
/...  -No...  00000000000000000000000000000101
/...  -No...  00000000000000000000000000000011 000000000000000... 00000000000000000000000000000011 0000000000000000000000... 0000000000000000000000...
/...  -No...  0000000... 00000000... 0000000... 0000000... 1111111... 111111... 000000... 0000000000000000000000000... 1111111111... 11... 111111111111... 0000000000000000000000... 0000000000000000000000...
/...  -No...  0000000000000000000000... 0000000000000000000...
/...  -No...  0000000... 00000000... 0000000... 0000000... 1111111... 111111... 000000... 0000000000000000000000000... 1111111111... 11... 111111111111... 0000000000000000000000... 0000000000000000000000...
/...  -No...  0000000... 00000000... 0000000... 0000000... 0000000... 000000... 000000... 000000... 0000000... 00000... 0000000000... 00... 000000000000... 0000000000000000000000... 0000000000000000000000
/...  -No...  00000000000000000000000000000011 000000000000000... 00000000000000000000000000000011 0000000000000000000000... 0000000000000000000000...
/...  -No...  0000000... 00000000... 0000000... 0000000... 0000000... 000000... 000000... 000000... 0000000... 00000... 0000000000... 00... 000000000000... 0000000000000000000000... 0000000000000000000000...
/...  -No...  01000 01000
/...  -No...  0110 0011 0000 0001 1100 1101 1000 1001 0111 1011 1100 1101 1101 0010
/...  -No...  10 00
/...  -No...
/...  -No...
/...  -No...
/...  -No...
/...  -No...
/...  -No...
/...  -No...
/...  -No...
/...  -No...
```