DIGITAL SYSTEM DESIGN WITH HDL
LAB 7'S REPORT

Name: Nguyễn Tuấn Dũng

ID: 21520746

Class: CE213.O11.MTCL

SIMPLE PROCESSOR DESIGN

Section 1. Theory

1. Simple Processor Design based on Datapath and Control Unit follow MIPS architecture.

The combination of Datapath and Control Unit is fundamental to the operation of a processor, and their design follows the principles of the MIPS (Microprocessor without Interlocked Pipeline Stages) architecture. Let's break down each component:

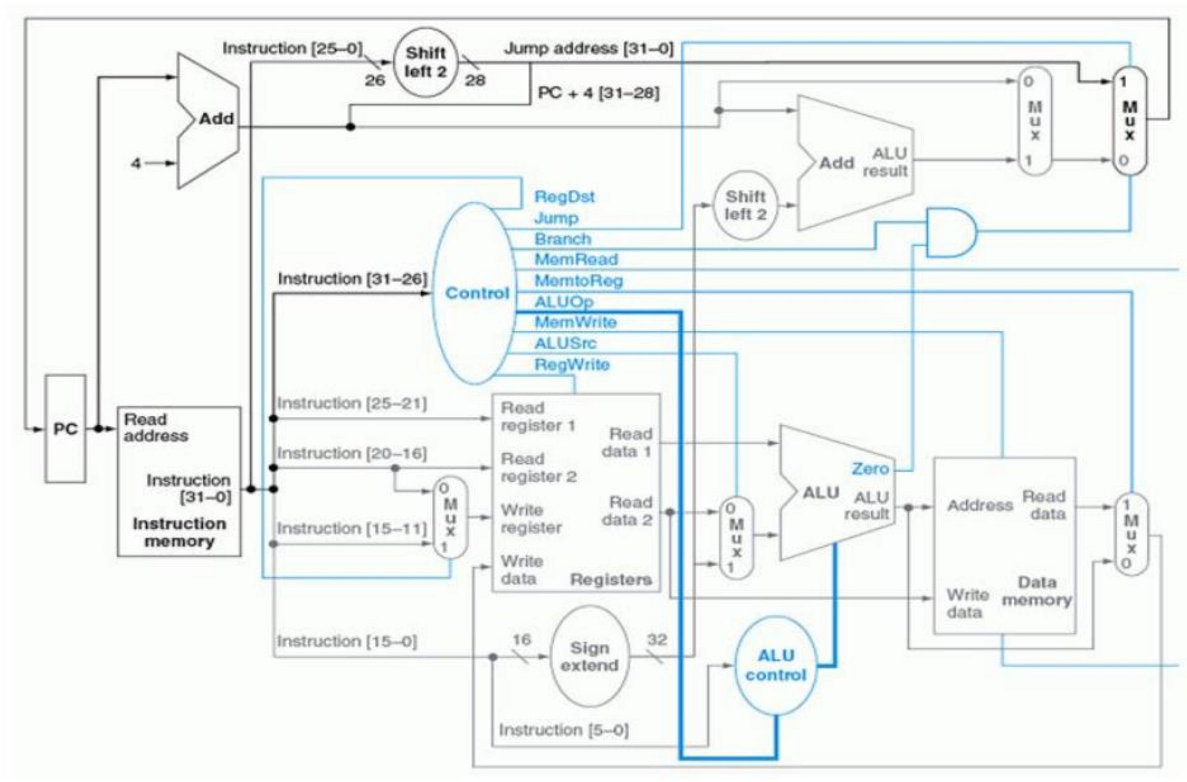
1. **Datapath:**
    - The Datapath is responsible for executing instructions and performing arithmetic and logic operations. In the context of MIPS, it consists of the following key components:
        - **Registers:** A set of 32 registers used to store data temporarily.
        - **ALU (Arithmetic Logic Unit):** Performs arithmetic and logic operations on data.
        - **MUX (Multiplexer):** Selects between different data sources.
        - **Memory Unit:** Manages data transfer between the processor and memory.
        - **PC (Program Counter):** Holds the address of the next instruction.
        - **Sign-extend Unit:** Extends immediate values for arithmetic operations.
        - **Shifters:** Perform bit-shifting operations.
    - The Datapath connects these components in a way that allows data to flow through the processor during instruction execution.
2. **Control Unit:**

- The Control Unit manages the Datapath and ensures that instructions are executed in the correct sequence. It generates control signals to activate or deactivate specific components of the Datapath based on the type of instruction being executed. For MIPS architecture, the Control Unit includes:
    - **Instruction Decoder:** Identifies the type of instruction and determines how to execute it.
    - **ALU Control Unit:** Determines the specific operation to be performed by the ALU.
    - **PC Control Unit:** Manages the updating of the Program Counter.
    - **Memory Control Unit:** Controls read and write operations to memory.
    - **MUX Control Unit:** Manages the selection of data sources.
- The Control Unit interprets the opcode of the instruction and generates the necessary control signals to coordinate the Datapath components.

3. **MIPS Architecture:**
    - MIPS is a RISC architecture, which means it uses a reduced set of simple instructions that can be executed in a single clock cycle. The focus is on simplicity and efficiency.
    - Instructions are typically divided into three formats: R-type (register), I-type (immediate), and J-type (jump).
    - The instruction format and the encoding of the instructions are considered in the design of both Datapath and Control Unit.

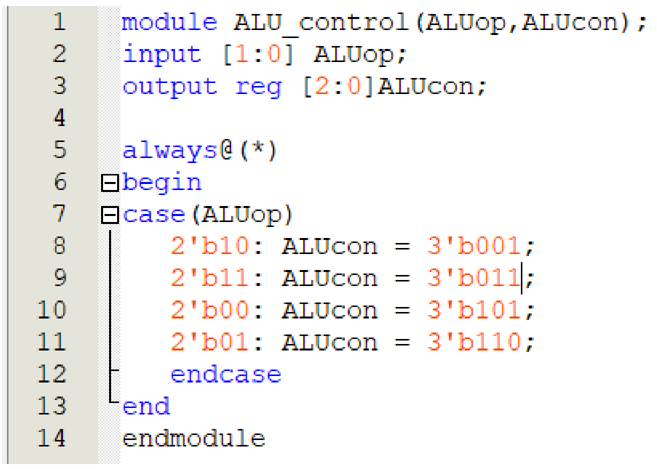
Section 2. Assignment

1. Complete connection circuit of simple Processor from simple DATAPATH (lab 4) and simple Control Unit (lab 5).

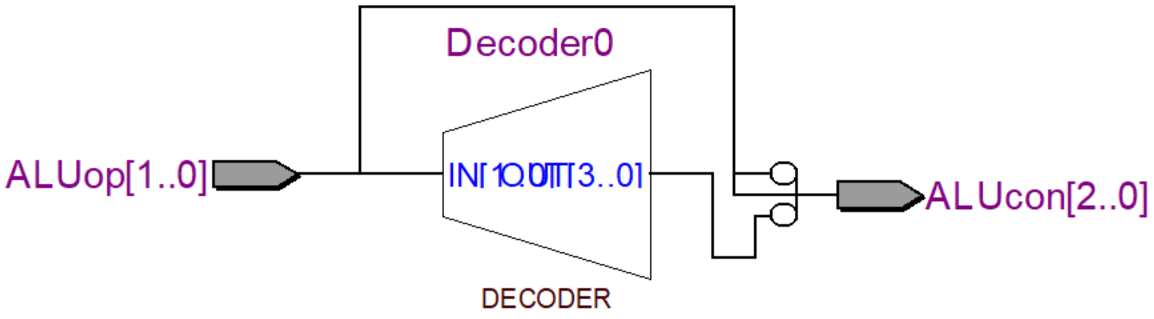


MODULE ALU_Control:

1.Verilog Code:

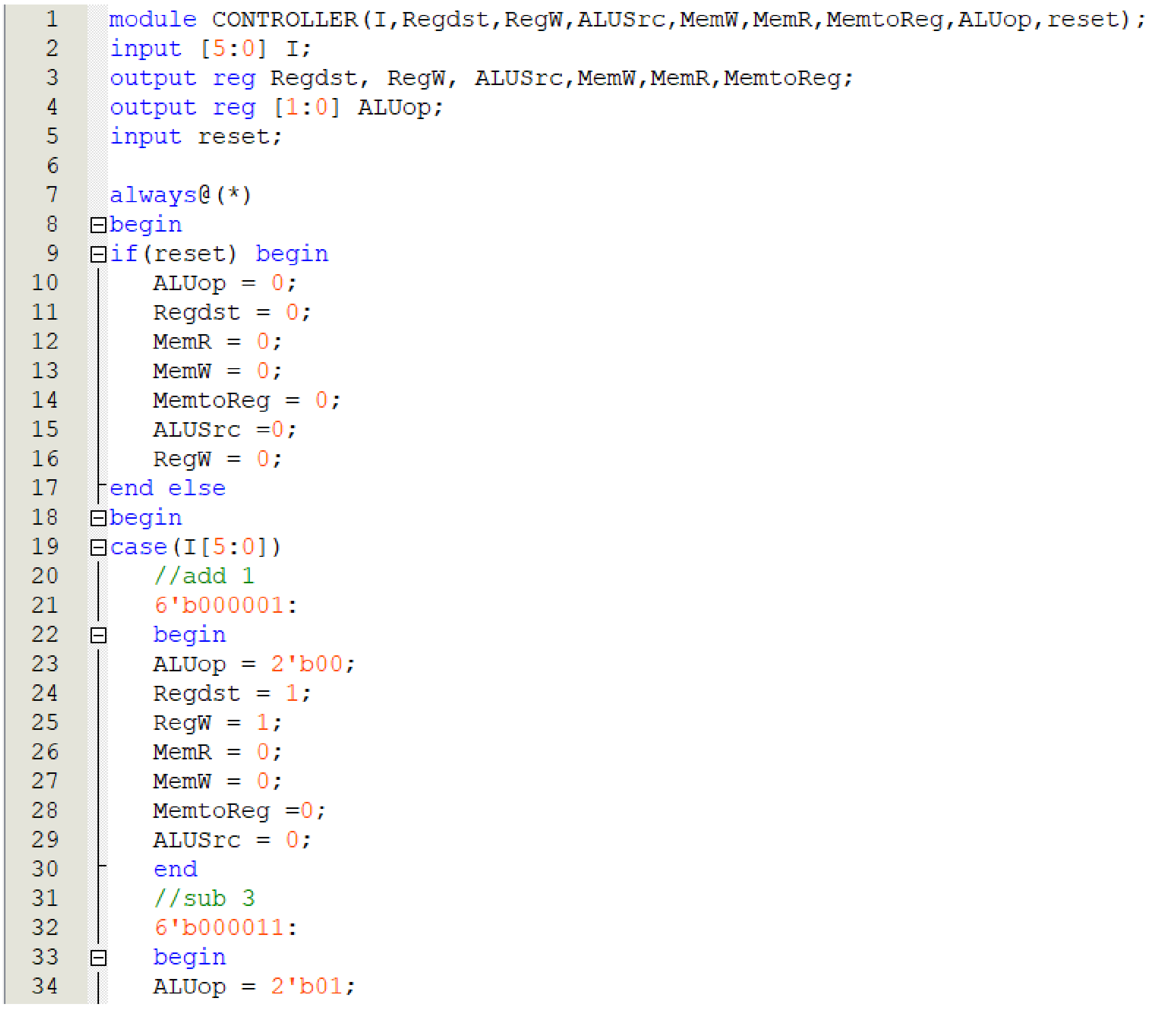

```
module ALU_control(ALUop,ALUcon);
input [1:0] ALUop;
output reg [2:0]ALUcon;

always@(*)
begin
case(ALUop)
    2'b10: ALUcon = 3'b001;
    2'b11: ALUcon = 3'b011;
    2'b00: ALUcon = 3'b101;
    2'b01: ALUcon = 3'b110;
    endcase
end
endmodule
```

2.RTL

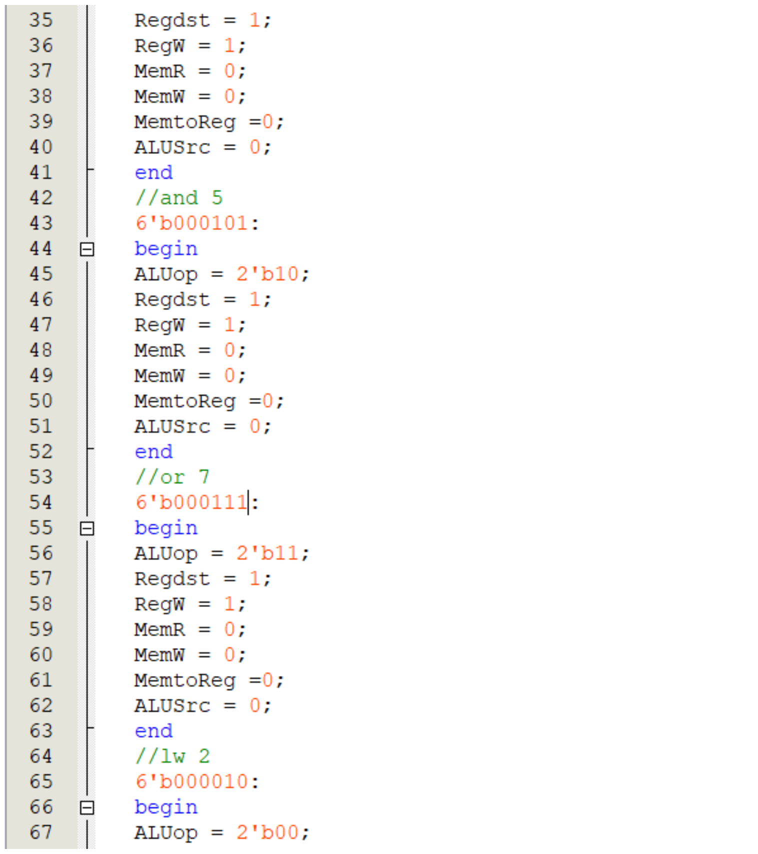


MODULE Control Unit:

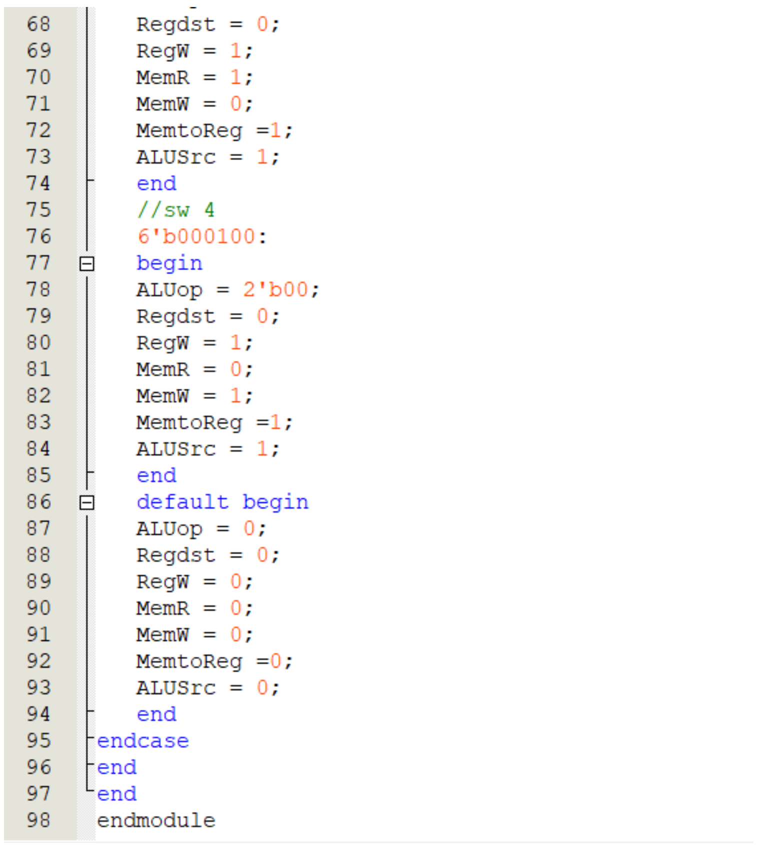
1.Verilog



```
module CONTROLLER(I,Regdst,RegW,ALUSrc,MemW,MemR,MemtoReg,ALUop,reset);
input [5:0] I;
output reg Regdst, RegW, ALUSrc,MemW,MemR,MemtoReg;
output reg [1:0] ALUop;
input reset;

always@(*)
begin
if(reset) begin
    ALUop = 0;
    Regdst = 0;
    MemR = 0;
    MemW = 0;
    MemtoReg = 0;
    ALUSrc =0;
    RegW = 0;
end else
begin
case(I[5:0])
    //add 1
    6'b000001:
    begin
    ALUop = 2'b00;
    Regdst = 1;
    RegW = 1;
    MemR = 0;
    MemW = 0;
    MemtoReg =0;
    ALUSrc = 0;
    end
    //sub 3
    6'b000011:
    begin
    ALUop = 2'b01;
```

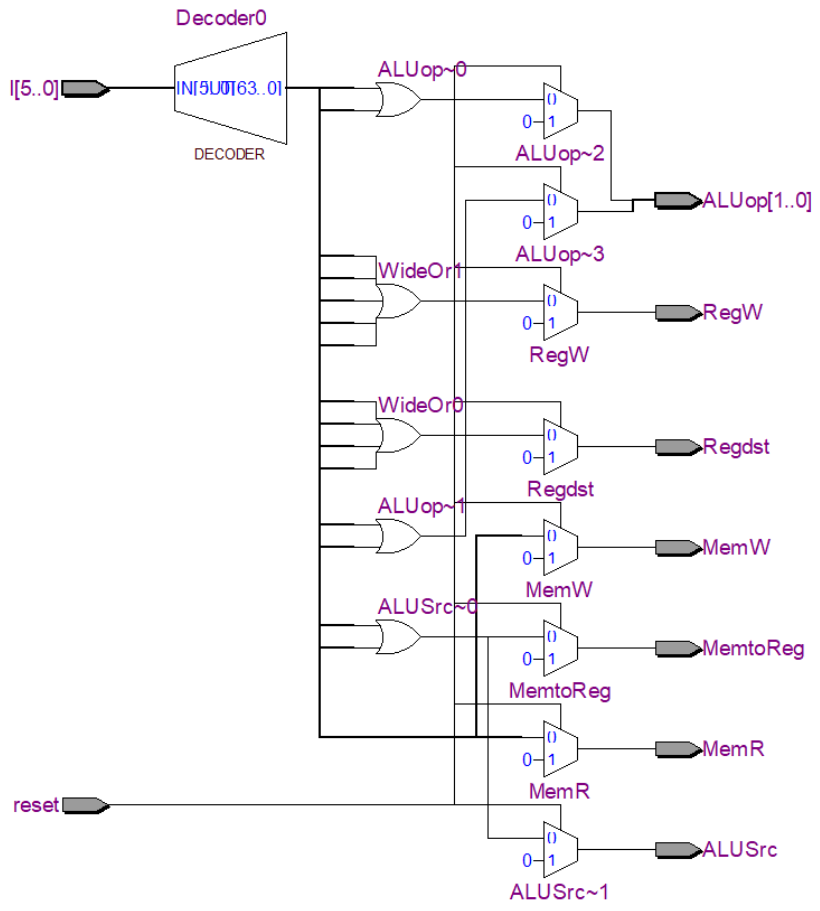


```verilog
    Regdst = 1;
    RegW = 1;
    MemR = 0;
    MemW = 0;
    MemtoReg =0;
    ALUSrc = 0;
    end
    //and 5
    6'b000101:
    begin
    ALUop = 2'b10;
    Regdst = 1;
    RegW = 1;
    MemR = 0;
    MemW = 0;
    MemtoReg =0;
    ALUSrc = 0;
    end
    //or 7
    6'b000111:
    begin
    ALUop = 2'b11;
    Regdst = 1;
    RegW = 1;
    MemR = 0;
    MemW = 0;
    MemtoReg =0;
    ALUSrc = 0;
    end
    //lw 2
    6'b000010:
    begin
    ALUop = 2'b00;
```



```verilog
    Regdst = 0;
    RegW = 1;
    MemR = 1;
    MemW = 0;
    MemtoReg =1;
    ALUSrc = 1;
    end
    //sw 4
    6'b000100:
    begin
    ALUop = 2'b00;
    Regdst = 0;
    RegW = 1;
    MemR = 0;
    MemW = 1;
    MemtoReg =1;
    ALUSrc = 1;
    end
    default begin
    ALUop = 0;
    Regdst = 0;
    RegW = 0;
    MemR = 0;
    MemW = 0;
    MemtoReg =0;
    ALUSrc = 0;
    end
endcase
end
end
endmodule
```

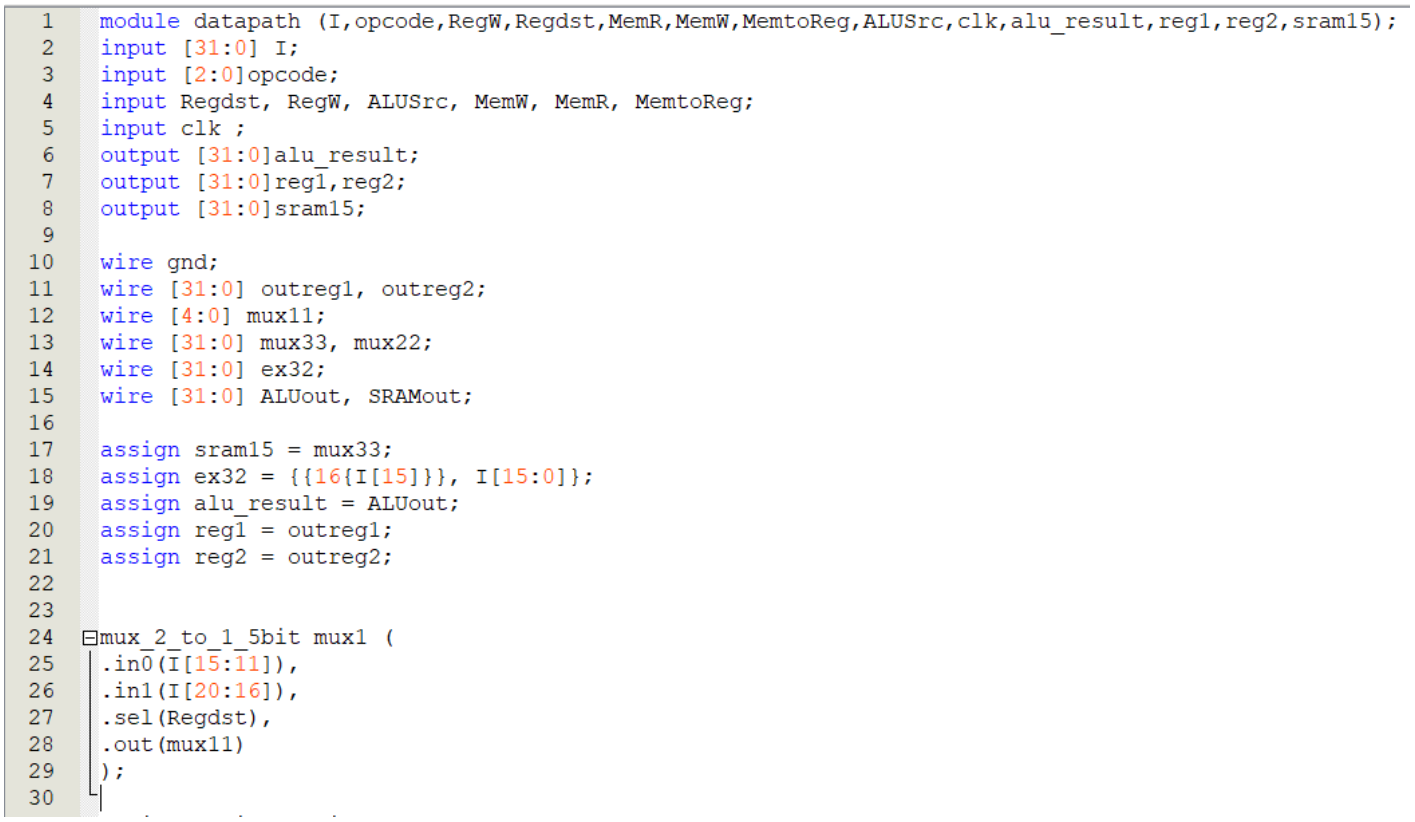
2.RTL



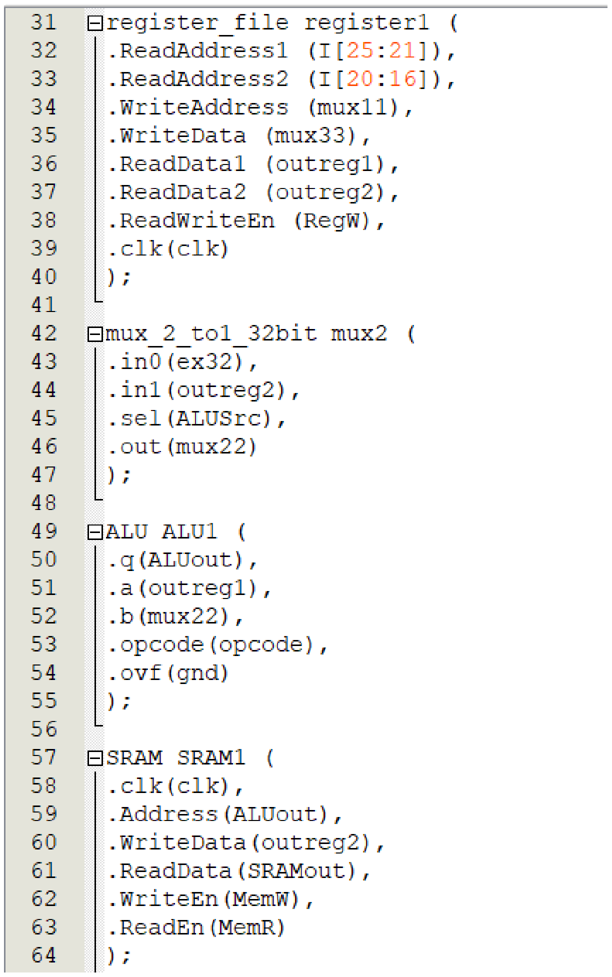
Operation	add	sub	and	or	lw	sw
Opcode	1	3	5	7	2	4
ALUop	0	1	2	3	0	0
ALUcontrol	5	6	1	3	5	5
Regdst	1	1	1	1	0	0
RegWrite	1	1	1	1	1	1
MemRead	0	0	0	0	1	0
MemWrite	0	0	0	0	0	1
MemtoReg	0	0	0	0	1	1
ALUSrc	0	0	0	0	1	1

MODULE DATAPATH:

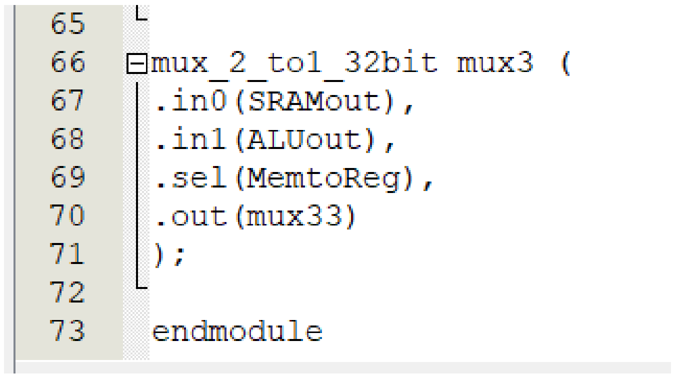
1.VERILOG CODE



```
module datapath (I,opcode,RegW,Regdst,MemR,MemW,MemtoReg,ALUSrc,clk,alu_result,reg1,reg2,sram15);
input [31:0] I;
input [2:0]opcode;
input Regdst, RegW, ALUSrc, MemW, MemR, MemtoReg;
input clk ;
output [31:0]alu_result;
output [31:0]reg1,reg2;
output [31:0]sram15;

wire gnd;
wire [31:0] outreg1, outreg2;
wire [4:0] mux11;
wire [31:0] mux33, mux22;
wire [31:0] ex32;
wire [31:0] ALUout, SRAMout;

assign sram15 = mux33;
assign ex32 = {{16{I[15]}}, I[15:0]};
assign alu_result = ALUout;
assign reg1 = outreg1;
assign reg2 = outreg2;


mux_2_to_1_5bit mux1 (
.in0(I[15:11]),
.in1(I[20:16]),
.sel(Regdst),
.out(mux11)
);

```

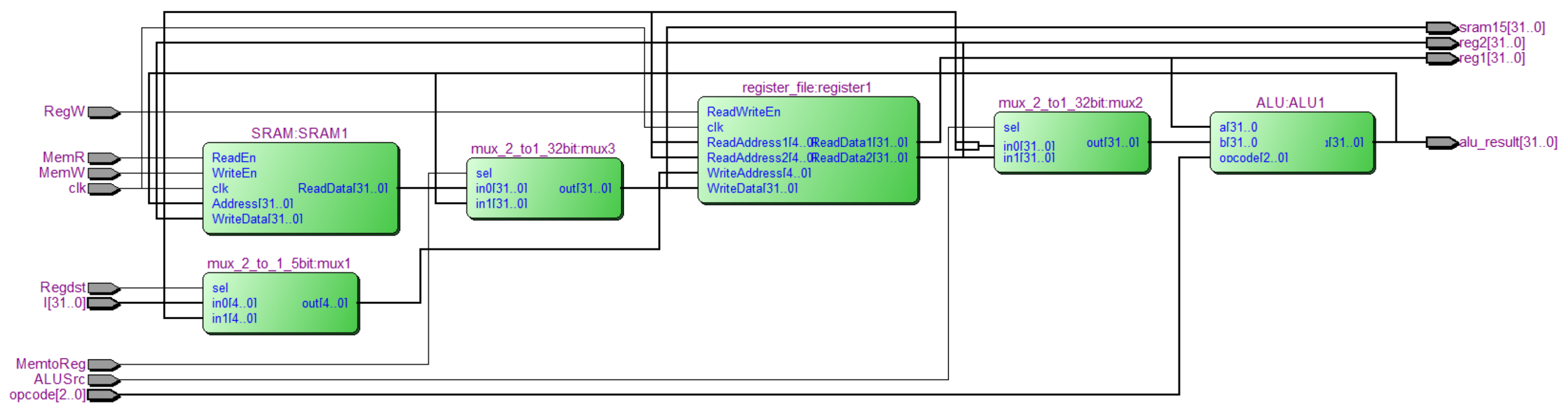


```verilog
register_file register1 (
.ReadAddress1 (I[25:21]),
.ReadAddress2 (I[20:16]),
.WriteAddress (mux11),
.WriteData (mux33),
.ReadData1 (outreg1),
.ReadData2 (outreg2),
.ReadWriteEn (RegW),
.clk(clk)
);

mux_2_to1_32bit mux2 (
.in0(ex32),
.in1(outreg2),
.sel(ALUSrc),
.out(mux22)
);

ALU ALU1 (
.q(ALUout),
.a(outreg1),
.b(mux22),
.opcode(opcode),
.ovf(gnd)
);

SRAM SRAM1 (
.clk(clk),
.Address(ALUout),
.WriteData(outreg2),
.ReadData(SRAMout),
.WriteEn(MemW),
.ReadEn(MemR)
);
```



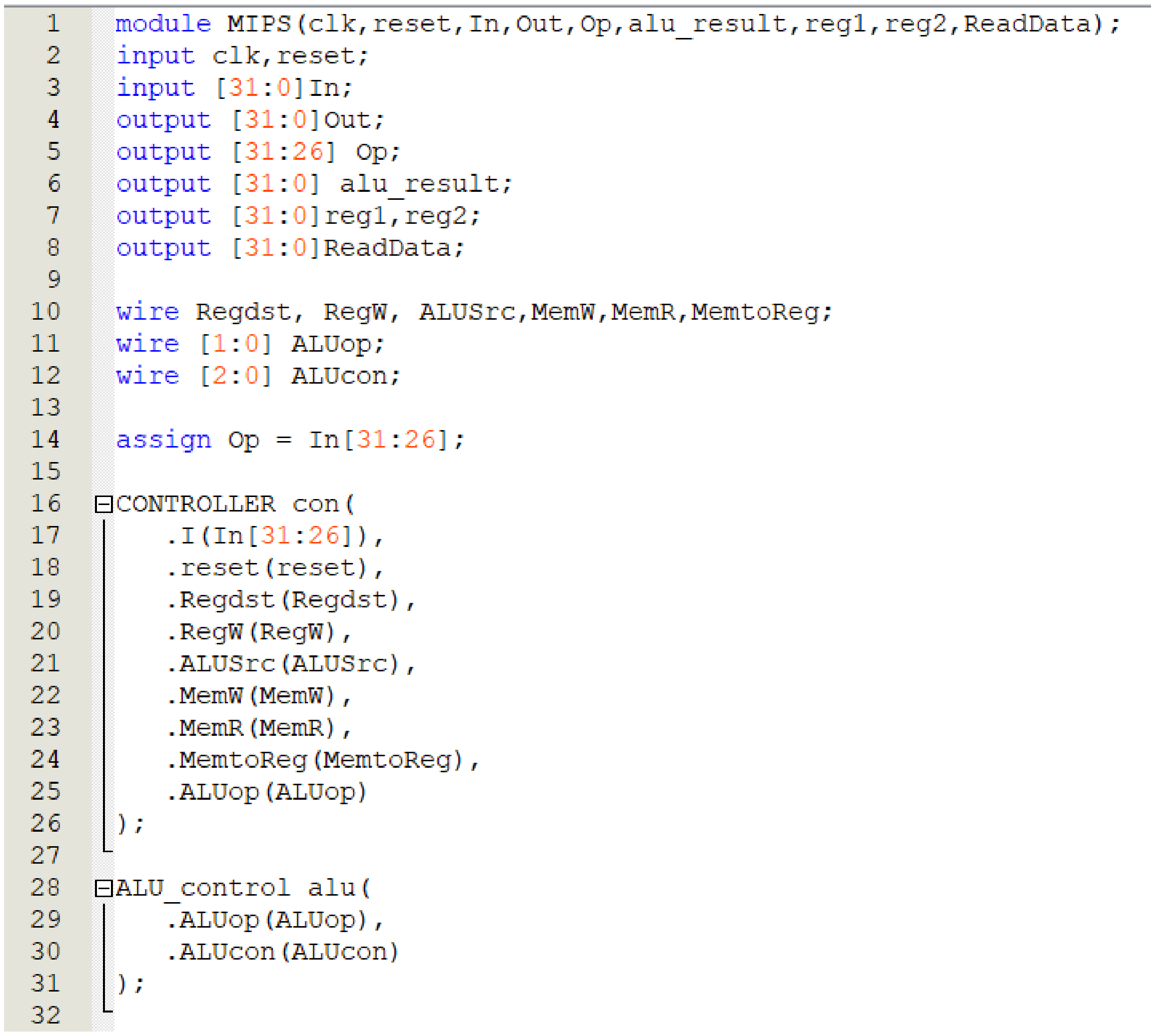
```verilog

mux_2_to1_32bit mux3 (
.in0(SRAMout),
.in1(ALUout),
.sel(MemtoReg),
.out(mux33)
);

endmodule
```

2.RTL VIEWER

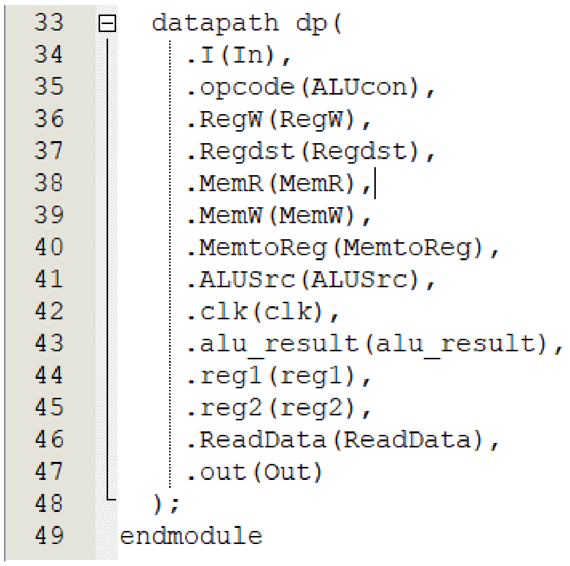


MODULE MIPS

1.Verilog code

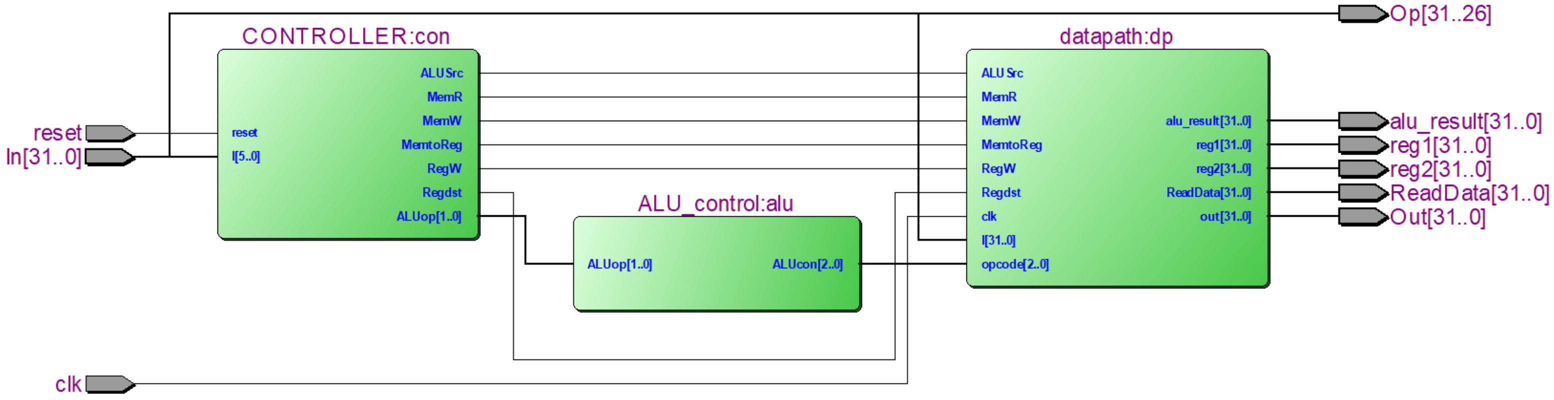


```
module MIPS(clk,reset,In,Out,Op,alu_result,reg1,reg2,ReadData);
input clk,reset;
input [31:0]In;
output [31:0]Out;
output [31:26] Op;
output [31:0] alu_result;
output [31:0]reg1,reg2;
output [31:0]ReadData;

wire Regdst, RegW, ALUSrc,MemW,MemR,MemtoReg;
wire [1:0] ALUop;
wire [2:0] ALUcon;

assign Op = In[31:26];

CONTROLLER con(
    .I(In[31:26]),
    .reset(reset),
    .Regdst(Regdst),
    .RegW(RegW),
    .ALUSrc(ALUSrc),
    .MemW(MemW),
    .MemR(MemR),
    .MemtoReg(MemtoReg),
    .ALUop(ALUop)
);

ALU_control alu(
    .ALUop(ALUop),
    .ALUcon(ALUcon)
);

```

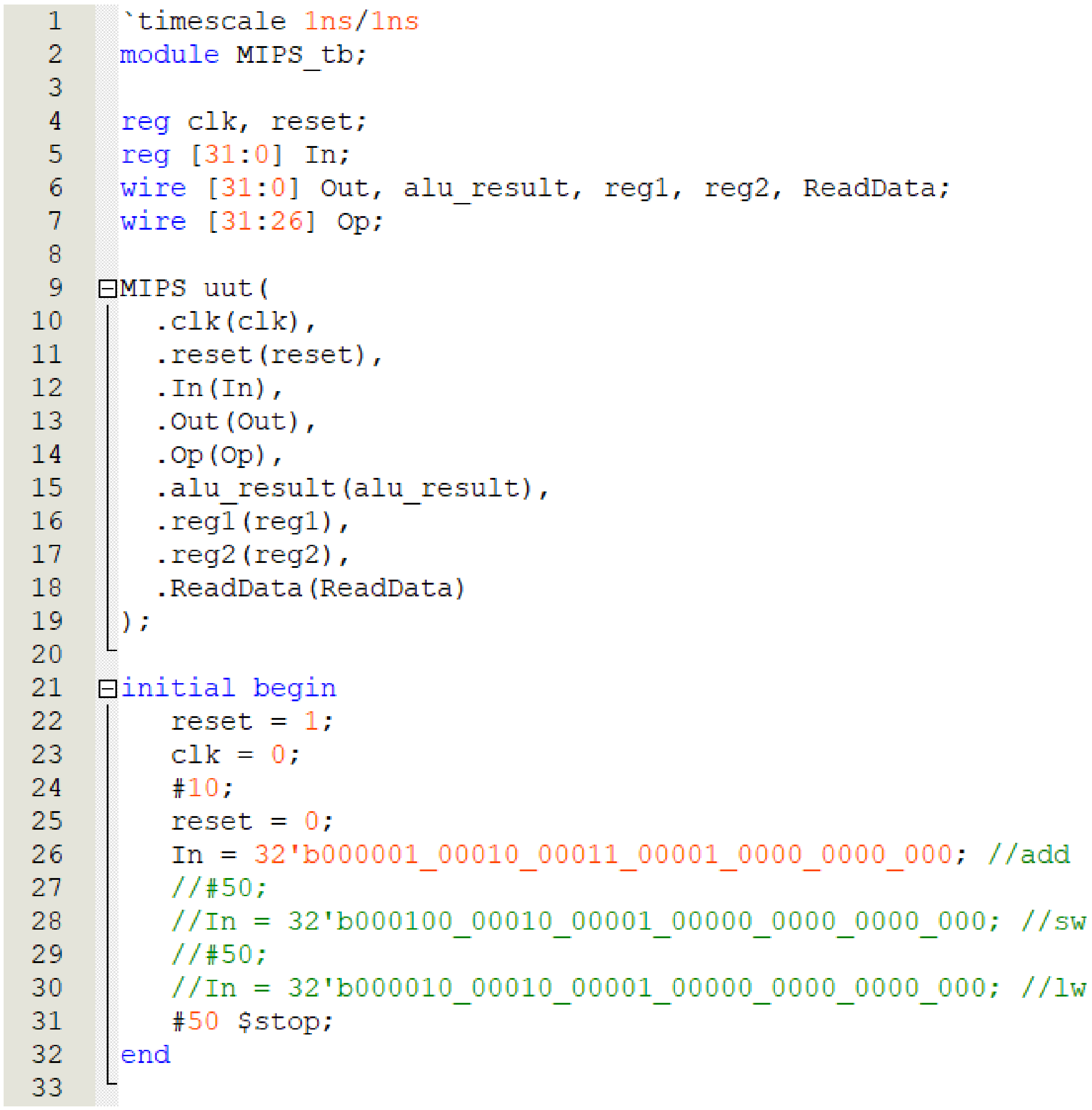


```
    datapath dp(
      .I(In),
      .opcode(ALUcon),
      .RegW(RegW),
      .Regdst(Regdst),
      .MemR(MemR),
      .MemW(MemW),
      .MemtoReg(MemtoReg),
      .ALUSrc(ALUSrc),
      .clk(clk),
      .alu_result(alu_result),
      .reg1(reg1),
      .reg2(reg2),
      .ReadData(ReadData),
      .out(Out)
    );
endmodule
```

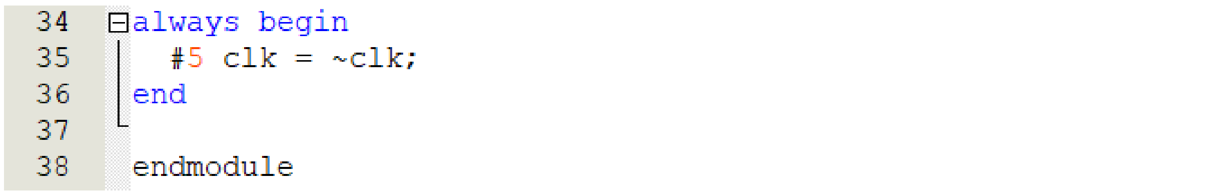
2.RTL



3.TestBench



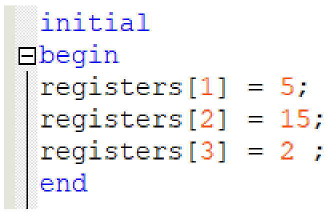
```verilog
`timescale 1ns/1ns
module MIPS_tb;

reg clk, reset;
reg [31:0] In;
wire [31:0] Out, alu_result, reg1, reg2, ReadData;
wire [31:26] Op;

MIPS uut(
  .clk(clk),
  .reset(reset),
  .In(In),
  .Out(Out),
  .Op(Op),
  .alu_result(alu_result),
  .reg1(reg1),
  .reg2(reg2),
  .ReadData(ReadData)
);

initial begin
   reset = 1;
   clk = 0;
   #10;
   reset = 0;
   In = 32'b000001_00010_00011_00001_0000_0000_000; //add
   //#50;
   //In = 32'b000100_00010_00001_00000_0000_0000_000; //sw
   //#50;
   //In = 32'b000010_00010_00001_00000_0000_0000_000; //lw
   #50 $stop;
end

```



```verilog
always begin
  #5 clk = ~clk;
end

endmodule
```

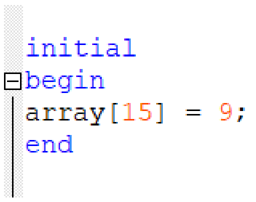
4.Waveform (**$1 address is 1, $2 address is 2, $3 address is 3**)

REGISTER



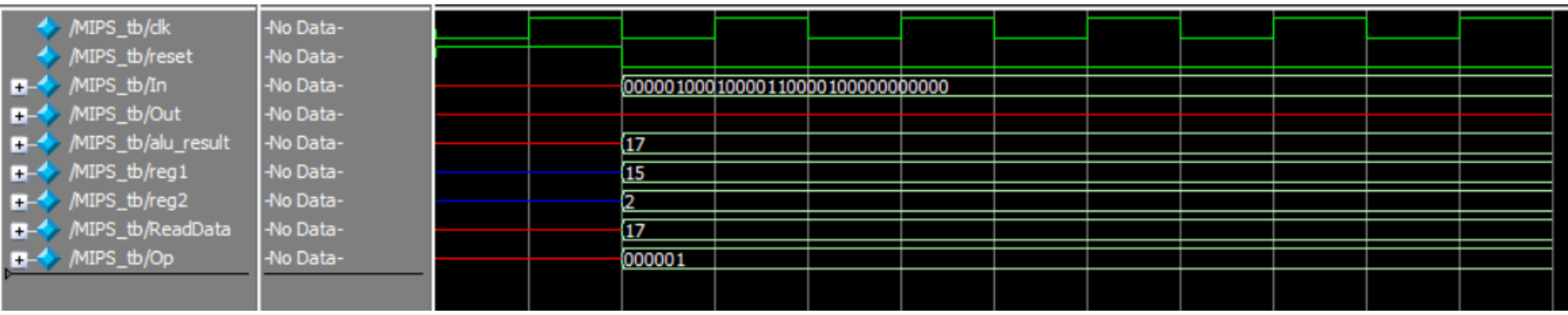
```
initial
begin
registers[1] = 5;
registers[2] = 15;
registers[3] = 2 ;
end
```

SRAM

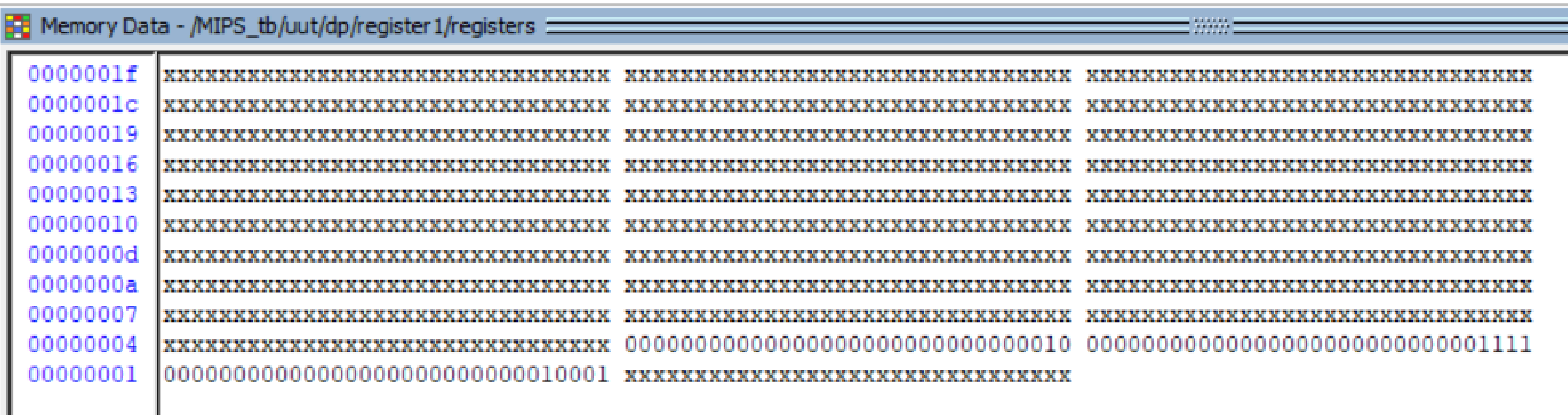


```
initial
begin
array[15] = 9;
end
```

add $1, $2, $3



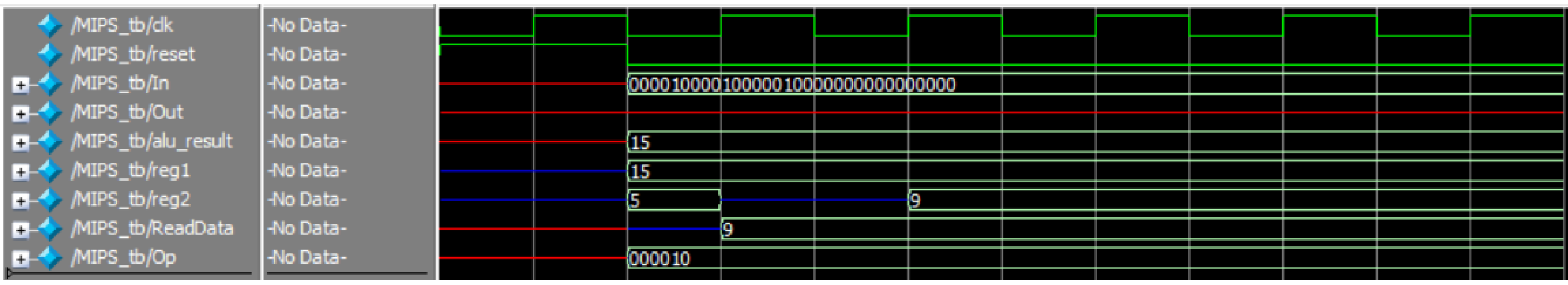
```
/MIPS_tb/clk          -No Data-
/MIPS_tb/reset        -No Data-
/MIPS_tb/In           -No Data-   00000100010000110000100000000000
/MIPS_tb/Out          -No Data-
/MIPS_tb/alu_result   -No Data-   17
/MIPS_tb/reg1         -No Data-   15
/MIPS_tb/reg2         -No Data-   2
/MIPS_tb/ReadData     -No Data-   17
/MIPS_tb/Op           -No Data-   000001
```

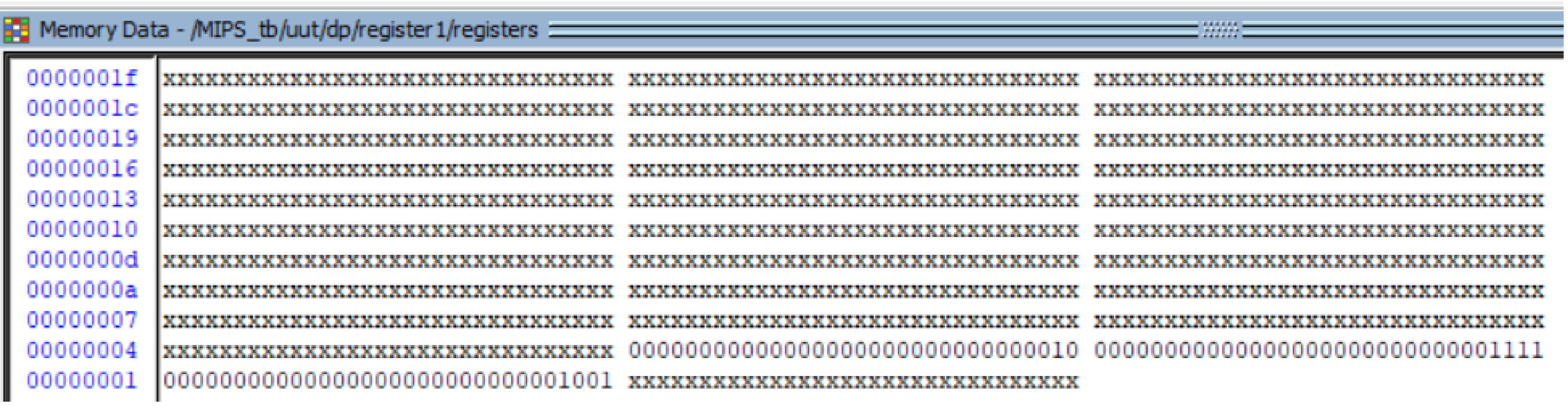


```
Memory Data - /MIPS_tb/uut/dp/register1/registers

0000001f xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001c xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000016 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000013 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000010 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000d xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000a xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000007 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000004 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx 00000000000000000000000000000010 00000000000000000000000000001111
00000001 00000000000000000000000000010001 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Explanation: Reg1 is $2 and Reg2 is $3 , alu_result is the value of $1. Reg1 + Reg2 = 15 + 2 = 17. In the image above show that 17 store in $1.

lw $1, 0($2)

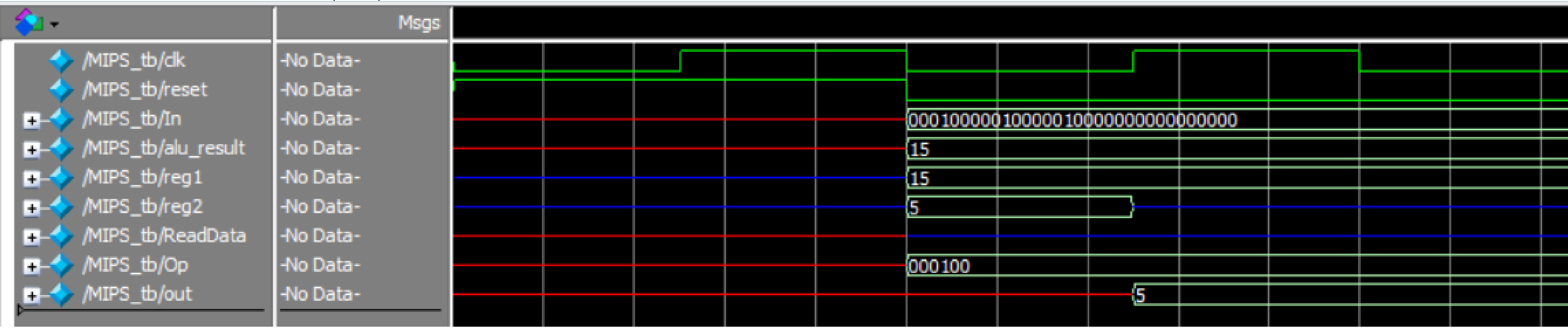


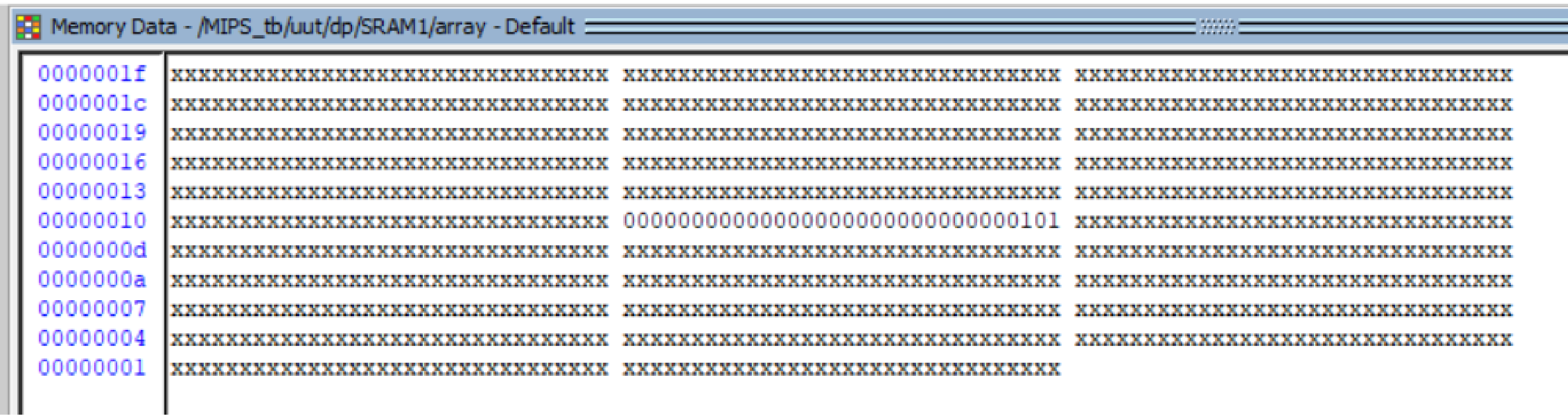

Memory Data - /MIPS_tb/uut/dp/register1/registers

```
0000001f xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001c xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000016 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000013 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000010 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000d xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000a xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000007 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000004 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx 00000000000000000000000000000010 00000000000000000000000000001111
00000001 00000000000000000000000000001001 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Explanation: Reg2 = $1 and Reg1 = $2. Load the data of sram[reg[1]+0] (9) to the regs[2] so the value of regs[1] is not changed. The value of Reg1 is the address of the SRAM bar and stores the value of address Reg1 in SRAM into Reg2 (address value).

sw $1, 0($2)





```
Memory Data - /MIPS_tb/uut/dp/SRAM1/array - Default
0000001f xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001c xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000016 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000013 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000010 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx 00000000000000000000000000000101 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000d xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000a xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000007 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000004 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000001 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Explanation: Reg2 = $1 and Reg1 = $2. Store the value of regs[2] (5) to the sram[reg[1]]. The value of Reg1 is the address of the SRAM bar and save the value of Reg2 to that address.