EE-475L: Computer Architecture

Lab Report 6

Submitted by

Marwa Waseem

2019-EE-7

Submitted to

Sir Ali Imran

Department of Electrical Engineering

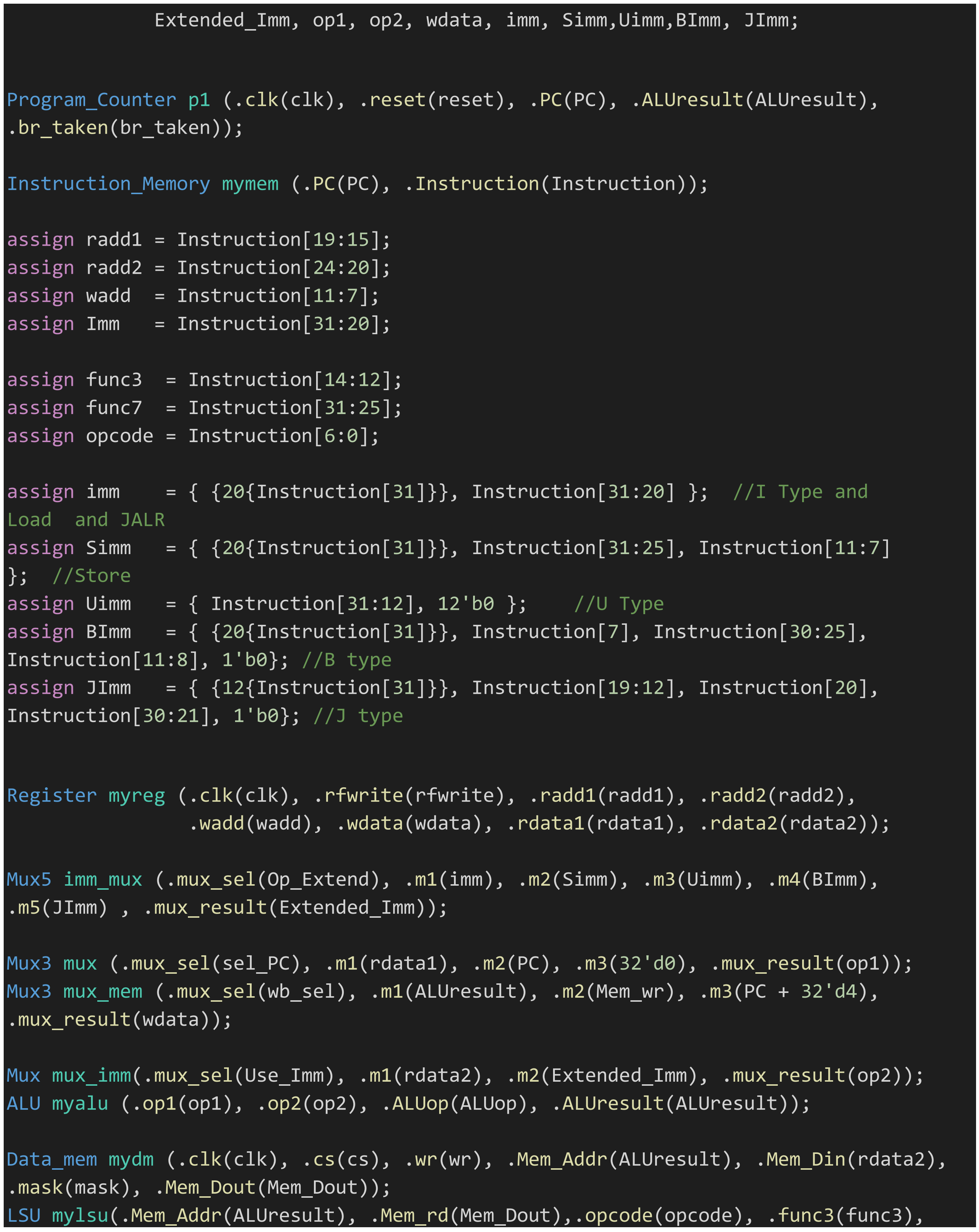
University of Engineering and Technology, Lahore, Pakistan.

Top Module

```
/*--------------------------------------------------------------------
  Top Module for Project 1
--------------------------------------------------------------------*/
module Project_1(input clk, reset);

logic rfwrite, Use_Imm;
logic [1:0] sel_PC, wb_sel;
logic [2:0] func3, br_type, Op_Extend;
logic [3:0] ALUop;
logic [6:0] func7, opcode;


Datapath_1 d1 (.clk(clk), .reset(reset), .rfwrite(rfwrite), .ALUop(ALUop),
.Op_Extend(Op_Extend) , .Use_Imm(Use_Imm) ,
                .sel_PC(sel_PC) , .wb_sel(wb_sel) , .func3(func3), .func7(func7),
.opcode(opcode) , .br_type(br_type));

controller_1 c1 (.func3(func3), .opcode(opcode), .func7(func7), .rfwrite(rfwrite),
.ALUop(ALUop) ,
                   .Use_Imm(Use_Imm), .sel_PC(sel_PC) , .Op_Extend(Op_Extend),
.wb_sel(wb_sel), .br_type(br_type));

endmodule
```

Datapath

```
/*--------------------------------------------------------------------
  Top Module for Datapath
--------------------------------------------------------------------*/
module Datapath_1(input logic clk, reset, rfwrite, Use_Imm,
                  input logic [1:0] sel_PC, wb_sel,
                  input logic [2:0] Op_Extend, br_type,
                  input logic [3:0] ALUop,
                  output logic [2:0] func3,
                  output logic [6:0] func7, opcode);

logic cs, wr;
logic [3:0] mask;
logic [4:0] radd1, radd2, wadd;
logic [31:0] rdata1, rdata2, PC, Instruction, ALUresult, Mem_wr, Mem_Dout,
```



```
             Extended_Imm, op1, op2, wdata, imm, Simm,Uimm,BImm, JImm;


Program_Counter p1 (.clk(clk), .reset(reset), .PC(PC), .ALUresult(ALUresult),
.br_taken(br_taken));

Instruction_Memory mymem (.PC(PC), .Instruction(Instruction));

assign radd1 = Instruction[19:15];
assign radd2 = Instruction[24:20];
assign wadd  = Instruction[11:7];
assign Imm   = Instruction[31:20];

assign func3  = Instruction[14:12];
assign func7  = Instruction[31:25];
assign opcode = Instruction[6:0];

assign imm    = { {20{Instruction[31]}}, Instruction[31:20] };  //I Type and
Load  and JALR
assign Simm   = { {20{Instruction[31]}}, Instruction[31:25], Instruction[11:7]
};  //Store
assign Uimm   = { Instruction[31:12], 12'b0 };    //U Type
assign BImm   = { {20{Instruction[31]}}, Instruction[7], Instruction[30:25],
Instruction[11:8], 1'b0}; //B type
assign JImm   = { {12{Instruction[31]}}, Instruction[19:12], Instruction[20],
Instruction[30:21], 1'b0}; //J type


Register myreg (.clk(clk), .rfwrite(rfwrite), .radd1(radd1), .radd2(radd2),
                .wadd(wadd), .wdata(wdata), .rdata1(rdata1), .rdata2(rdata2));

Mux5 imm_mux (.mux_sel(Op_Extend), .m1(imm), .m2(Simm), .m3(Uimm), .m4(BImm),
.m5(JImm) , .mux_result(Extended_Imm));

Mux3 mux (.mux_sel(sel_PC), .m1(rdata1), .m2(PC), .m3(32'd0), .mux_result(op1));
Mux3 mux_mem (.mux_sel(wb_sel), .m1(ALUresult), .m2(Mem_wr), .m3(PC + 32'd4),
.mux_result(wdata));

Mux mux_imm(.mux_sel(Use_Imm), .m1(rdata2), .m2(Extended_Imm), .mux_result(op2));
ALU myalu (.op1(op1), .op2(op2), .ALUop(ALUop), .ALUresult(ALUresult));

Data_mem mydm (.clk(clk), .cs(cs), .wr(wr), .Mem_Addr(ALUresult), .Mem_Din(rdata2),
.mask(mask), .Mem_Dout(Mem_Dout));
LSU mylsu(.Mem_Addr(ALUresult), .Mem_rd(Mem_Dout),.opcode(opcode), .func3(func3),
          .cs(cs), .wr(wr), .mask(mask), .Mem_wr(Mem_wr));

Branch_cond mybr (.rdata1(rdata1), .rdata2(rdata2), .br_type(br_type),
.br_taken(br_taken));



endmodule
```

Controller

```
/*------------------------------------------------------------------
  Module for Controller
------------------------------------------------------------------*/
module controller_1(
                    input logic [2:0] func3,
                    input logic [6:0] opcode, func7,
                    output logic rfwrite, Use_Imm,
                    output logic [1:0] sel_PC, wb_sel,
                    output logic[2:0] Op_Extend, br_type,
                    output logic[3:0] ALUop);

always_comb begin
        rfwrite = 1'b0;
        Use_Imm = 1'b0;
        sel_PC = 2'b00;
        wb_sel=2'b00;
        Op_Extend = 2'b00;
        ALUop = 4'b0000;


        case(opcode)
            7'b0110011: rfwrite = 1; //R Type Instruction

            7'b0010011: begin //I Type Instruction
                            Use_Imm   = 1;
                            rfwrite   = 1;
```

```
                        end

        7'b0000011: begin //Load Word
                        Use_Imm    = 1;
                        rfwrite    = 1;
                        wb_sel     = 1;
                        Op_Extend = 0;
                        end
        7'b0100011: begin //Store Word
                        Use_Imm    = 1;
                        Op_Extend = 1;
                        end
        7'b0110111: begin    // LUI Instruction
                        rfwrite    = 1;
                        Use_Imm    = 1;
                        Op_Extend = 2;
                        sel_PC = 2;
                    end
        7'b0010111: begin    // AUIPC Type Instruction
                        rfwrite    = 1;
                        Use_Imm    = 1;
                        Op_Extend = 2;
                        sel_PC = 1;
                    end
        7'b1100011: begin    //Branch Instruction
                        Use_Imm =1;
                        Op_Extend = 3;
                        sel_PC = 1;
                        case(func3)
                        3'b000: br_type = 3'd0;    //BEQ
                        3'b001: br_type = 3'd1;    //BNE
                        3'b100: br_type = 3'd2;    //BLT
                        3'b101: br_type = 3'd3;    //BGE
                        3'b110: br_type = 3'd4;    //BLTU
                        3'b111: br_type = 3'd5;    //BGEU
                        endcase
                    end
        7'b1101111: begin   //Jump instruction
                        rfwrite = 1;
                        Use_Imm =1;
                        Op_Extend = 4;   //Sign extension
                        wb_sel = 2;
                        sel_PC= 1;
                        br_type =3'd6;
```

```verilog
                        end
            7'b1100111: begin  //JALR
                            rfwrite=1;
                            Use_Imm =1;
                            Op_Extend = 0;
                            wb_sel = 2;
                            sel_PC = 0;
                            br_type =3'd6;
                        end

        endcase


        if (opcode == 7'b0110011 || opcode == 7'b0010011) begin
        case(func3)
            3'b000: begin
                    if (func7 == 7'b0000000)
                        ALUop = 4'd0;               // add
                    else if (func7 == 7'b0100000)
                        ALUop = 4'd1;               // subtract
                    end
            3'b001: begin
                    ALUop = 4'd2;                       // sll
                    end
            3'b010: ALUop = 4'd3;                   // slt
            3'b011: ALUop = 4'd4;                       // sltu
            3'b100: ALUop = 4'd5;                       // xor
            3'b101: begin
                    if (func7 == 7'b0000000)
                        ALUop = 4'd6;               // srl
                    else if (func7 == 7'b0100000)
                        ALUop = 4'd7;               // sra
                    end
            3'b110: ALUop = 4'd8;                   // or
            3'b111: ALUop = 4'd9;                       // and
            default: ALUop = 0;
        endcase
    end

end
endmodule
```

Branch Condition

```
module Branch_cond( input logic [31:0] rdata1, rdata2,
                    input logic [2:0] br_type,
                    output logic br_taken);   //br_taken = br_result

always_comb begin
    case(br_type)
    3'd0 : br_taken <= rdata1 ==rdata2;                           //BEQ
    3'd1 : br_taken <= rdata1 !=rdata2;                           //BNE
    3'd2 : br_taken <= $signed(rdata1) < $signed(rdata2);         //BLT
    3'd3 : br_taken <= $signed(rdata1) >= $signed(rdata2);        //BGE
    3'd4 : br_taken <= $unsigned(rdata1) < $unsigned(rdata2);     //BLTU
    3'd5 : br_taken <= $unsigned(rdata1) >= $unsigned(rdata2);     //BGEU
    3'd6 : br_taken <= 1;                                          //JAL, JALR
    default : br_taken = 0;
    endcase
end

endmodule
```

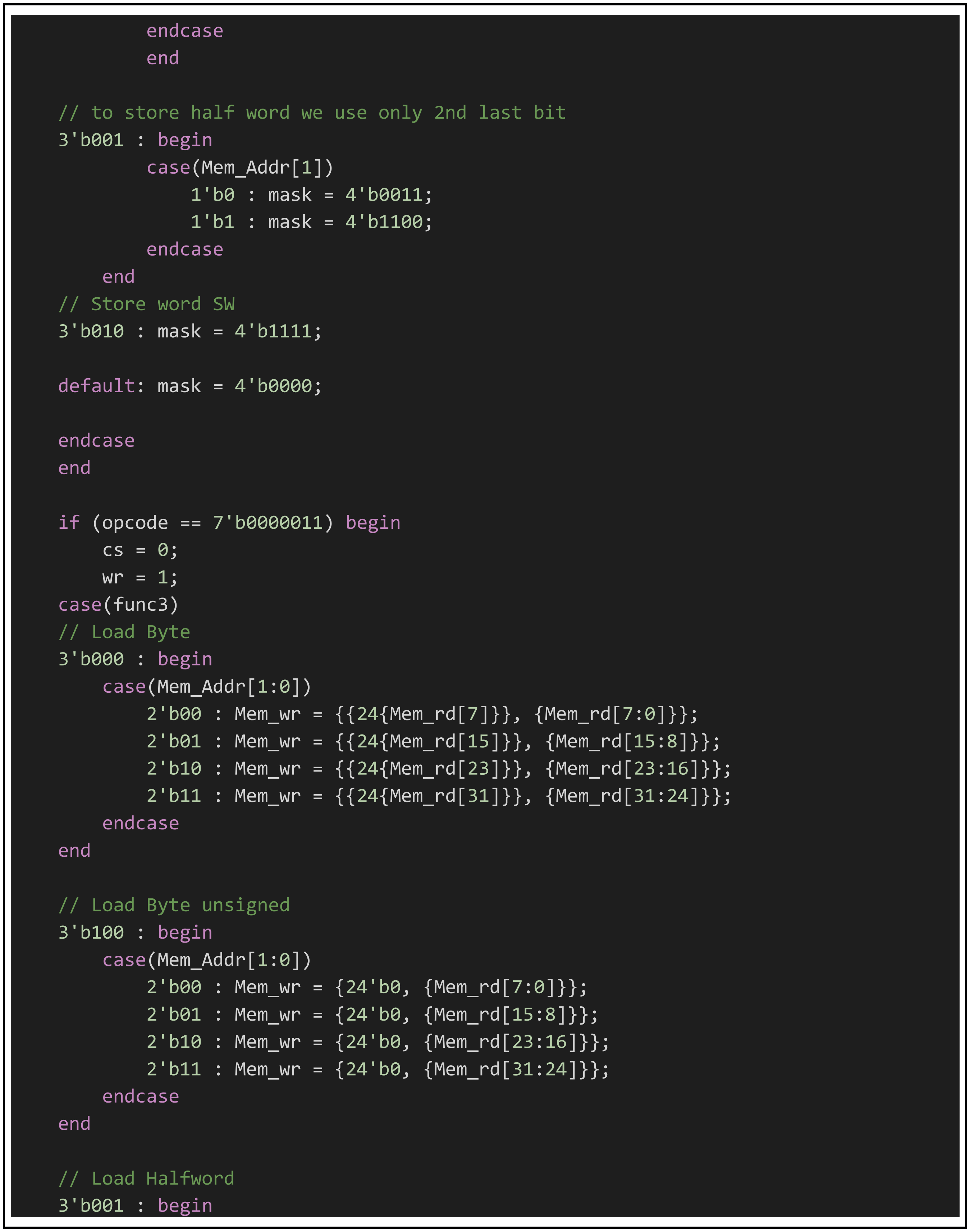
Data memory

```

/*-----------------------------------------------------------------
  Module for Data Memory
-------------------------------------------------------------------*/
module Data_mem(input logic clk, cs, wr,
                input logic  [31:0] Mem_Addr, Mem_Din,
                input logic  [3:0] mask,
                output logic [31:0] Mem_Dout);

logic [31:0] Data_File [31:0];


initial begin
    $readmemh("data.txt", Data_File);
    end

//Write operation (Store Data) synchronous
always_ff @(negedge clk)
begin
```

```
    if(!cs && !wr) begin
        if (mask[0])
            Data_File[Mem_Addr[31:2]][7:0] = Mem_Din[7:0];
        if (mask[1])
            Data_File[Mem_Addr[31:2]][15:8] = Mem_Din[15:8];
        if (mask[2])
            Data_File[Mem_Addr[31:2]][23:16] = Mem_Din[23:16];
        if (mask[3])
            Data_File[Mem_Addr[31:2]][31:24] = Mem_Din[31:24];
    end
end

assign Mem_Dout = (!cs & wr) ? Data_File[Mem_Addr[31:2]] : 32'b0;  // asynchronous
read

endmodule
```

Load Store Unit

```
// LOAD STORE UNIT

module LSU(input logic [31:0] Mem_Addr, Mem_rd,
           input logic [6:0] opcode,
           input logic [2:0] func3,
           output logic cs, wr,
           output logic [3:0] mask,
           output logic [31:0] Mem_wr);

//wrMem_wr Operation
always_comb begin
    if(opcode == 7'b0100011) begin
        cs = 0;
        wr = 0;
    case(func3)
    //to store a byte SB we use last 2 bits
     3'b000 : begin
            case(Mem_Addr[1:0])
                2'b00 : mask = 4'b0001;
                2'b01 : mask = 4'b0010;
                2'b10 : mask = 4'b0100;
                2'b11 : mask = 4'b1000;
```



```
        endcase
        end

// to store half word we use only 2nd last bit
3'b001 : begin
        case(Mem_Addr[1])
            1'b0 : mask = 4'b0011;
            1'b1 : mask = 4'b1100;
        endcase
    end
// Store word SW
3'b010 : mask = 4'b1111;

default: mask = 4'b0000;

endcase
end

if (opcode == 7'b0000011) begin
    cs = 0;
    wr = 1;
case(func3)
// Load Byte
3'b000 : begin
    case(Mem_Addr[1:0])
        2'b00 : Mem_wr = {{24{Mem_rd[7]}}, {Mem_rd[7:0]}};
        2'b01 : Mem_wr = {{24{Mem_rd[15]}}, {Mem_rd[15:8]}};
        2'b10 : Mem_wr = {{24{Mem_rd[23]}}, {Mem_rd[23:16]}};
        2'b11 : Mem_wr = {{24{Mem_rd[31]}}, {Mem_rd[31:24]}};
    endcase
end

// Load Byte unsigned
3'b100 : begin
    case(Mem_Addr[1:0])
        2'b00 : Mem_wr = {24'b0, {Mem_rd[7:0]}};
        2'b01 : Mem_wr = {24'b0, {Mem_rd[15:8]}};
        2'b10 : Mem_wr = {24'b0, {Mem_rd[23:16]}};
        2'b11 : Mem_wr = {24'b0, {Mem_rd[31:24]}};
    endcase
end

// Load Halfword
3'b001 : begin
```

```
        case(Mem_Addr[1])
            1'b0 : Mem_wr = {{16{Mem_rd[15]}}, {Mem_rd[15:0]}};
            1'b1 : Mem_wr = {{16{Mem_rd[31]}}, {Mem_rd[31:16]}};
        endcase
    end

    // Load halfword unsigned
    3'b101 : begin
        case(Mem_Addr[1])
            1'b0 : Mem_wr = {16'b0, {Mem_rd[15:0]}};
            1'b1 : Mem_wr = {16'b0, {Mem_rd[31:16]}};
        endcase
    end

    // Load Word
    3'b010: Mem_wr = Mem_rd;
    default: Mem_wr = 0;
    endcase
    end
end
endmodule
```

Instruction Memory

```
/*-----------------------------------------------------------------
  Module for Instruction Memory
------------------------------------------------------------------*/
module Instruction_Memory(input logic [31:0] PC,
                          output logic[31:0] Instruction);

//64 Instructions each 8-bit wide
logic [7:0] Instruction_Files [63:0];

initial begin
    $readmemh("ins.txt", Instruction_Files);
    end

always_comb begin
    Instruction =
{Instruction_Files[PC],Instruction_Files[PC+1],Instruction_Files[PC+2],Instruction_Fi
les[PC+3]};
    end
endmodule
```

Register Files

```
/*-----------------------------------------------------------------
  Module for Register Files
------------------------------------------------------------------*/
module Register(input logic clk, rfwrite,
                input logic [4:0] radd1, radd2, wadd,
                input logic[31:0] wdata,
                output logic[31:0] rdata1, rdata2);

//32 Registers each 32-bit wide
logic [31:0] Register_Files [31:0];

initial begin
    $readmemh("reg.txt", Register_Files);
    end

//Synchronous Write
always_ff @ (posedge clk) begin
    if (rfwrite)
        Register_Files [wadd] = (|wadd) ? wdata : 32'b0;
    end
//Asynchronous Read
always_comb begin
    rdata1 = (|radd1) ? Register_Files[radd1] : 32'b0;
    rdata2 = (|radd2) ? Register_Files[radd2] : 32'b0;
    end
endmodule
```

Program Counter

```
/*----------------------------------------------------------------
  Module for Program Counter
------------------------------------------------------------------*/
module Program_Counter(input logic clk, reset,
                       input logic [31:0] ALUresult,
                       input logic br_taken,
                       output logic [31:0] PC);

always_ff @(posedge clk) begin
    if(reset)
        PC <= 32'd0;
    else
        PC <= br_taken ? ALUresult :    PC+ 32'd4;
    end
endmodule
```

Mux 2x1

```
/*----------------------------------------------------------------
  Module for Multiplexer 2x1
------------------------------------------------------------------*/
module Mux(input mux_sel,
           input[31:0] m1, m2,
           output reg[31:0] mux_result);

always_comb begin
    case (mux_sel)
        0: mux_result <= m1;
        1: mux_result <= m2;
    endcase
    end
endmodule
```

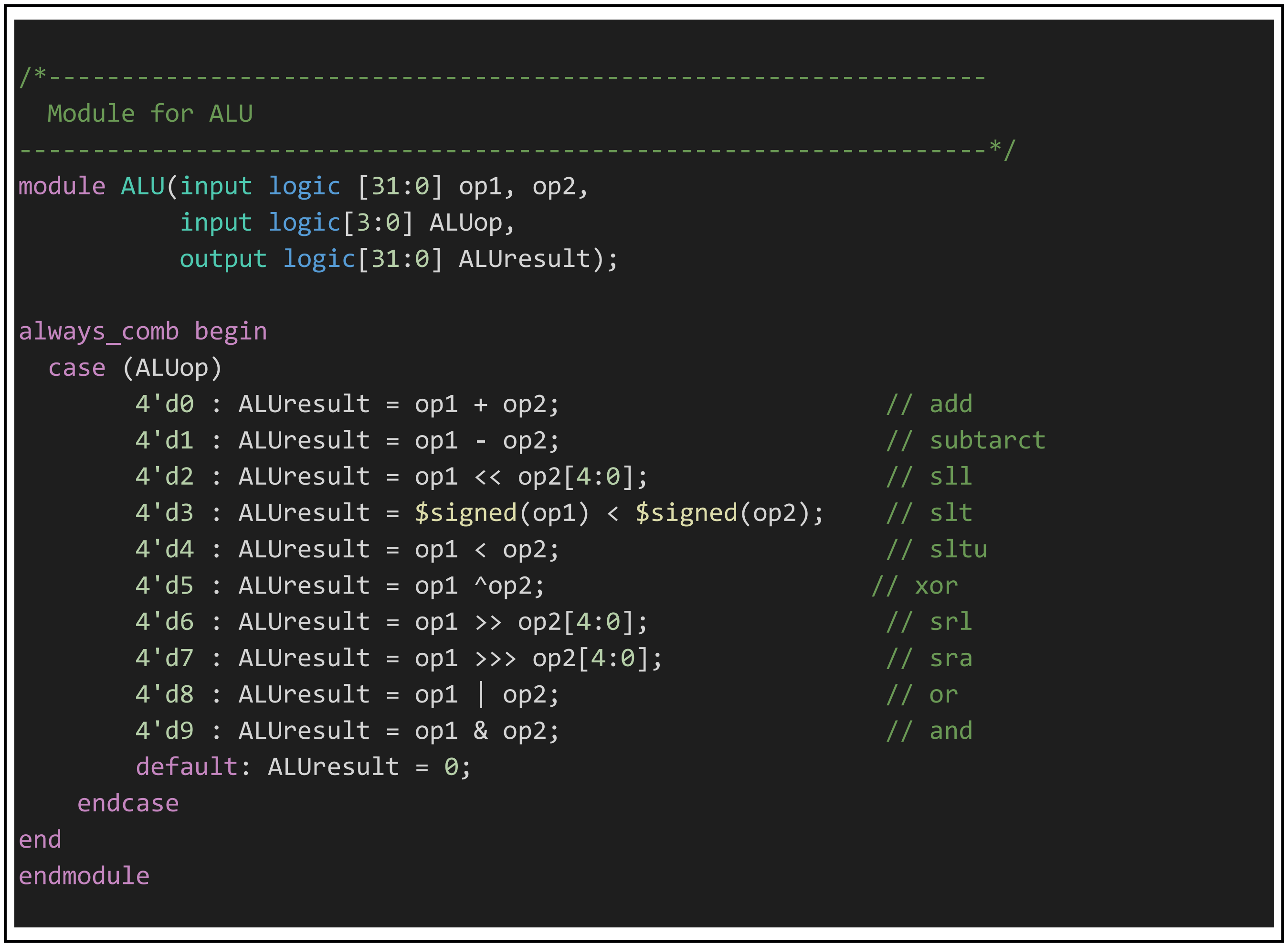
Mux 3x1

```
/*----------------------------------------------------------------
  Module for Multiplexer 3x1
------------------------------------------------------------------*/
module Mux3(input [1:0] mux_sel,
           input[31:0] m1, m2, m3,
           output reg[31:0] mux_result);

always_comb begin
    case (mux_sel)
        0: mux_result <= m1;
        1: mux_result <= m2;
        2: mux_result <= m3;
    endcase
    end
endmodule
```

Mux 5x1

```
/*----------------------------------------------------------------
  Module for Multiplexer 5x1
------------------------------------------------------------------*/
module Mux5(input[2:0] mux_sel,
           input[31:0] m1, m2,m3,m4,m5,
           output reg[31:0] mux_result);

always_comb begin
    case (mux_sel)
        3'd0: mux_result <= m1;
        3'd1: mux_result <= m2;
        3'd2: mux_result <= m3;
        3'd3: mux_result <= m4;
        3'd4: mux_result <= m5;
    endcase
    end
endmodule
```

ALU



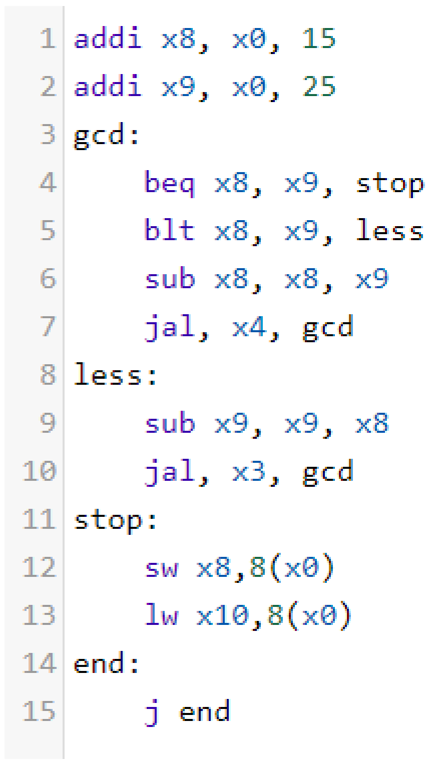
```
/*----------------------------------------------------------------
  Module for ALU
------------------------------------------------------------------*/
module ALU(input logic [31:0] op1, op2,
           input logic[3:0] ALUop,
           output logic[31:0] ALUresult);

always_comb begin
  case (ALUop)
        4'd0 : ALUresult = op1 + op2;                      // add
        4'd1 : ALUresult = op1 - op2;                      // subtarct
        4'd2 : ALUresult = op1 << op2[4:0];                // sll
        4'd3 : ALUresult = $signed(op1) < $signed(op2);    // slt
        4'd4 : ALUresult = op1 < op2;                      // sltu
        4'd5 : ALUresult = op1 ^op2;                      // xor
        4'd6 : ALUresult = op1 >> op2[4:0];                // srl
        4'd7 : ALUresult = op1 >>> op2[4:0];               // sra
        4'd8 : ALUresult = op1 | op2;                      // or
        4'd9 : ALUresult = op1 & op2;                      // and
        default: ALUresult = 0;
    endcase
end
endmodule
```

TestBench:

```
/*----------------------------------------------------------------
  Testbench
------------------------------------------------------------------*/
module tb_lab6 ();

logic clk , reset;

Project_1 uut (.clk(clk), .reset(reset));
parameter T = 10; // Clock Period
/*----------------------------------------------------------------
  Clock Generator
------------------------------------------------------------------*/
initial
begin
clk = 0;
forever #(T/2) clk=~clk;
end

initial
begin
reset = 1;
@(posedge clk);
reset = 0;
#110;
end
/*----------------------------------------------------------------
  Reset Sequence
------------------------------------------------------------------*/

endmodule
```

Assembly code using Venus & Machine code(ins.txt) using Venus:

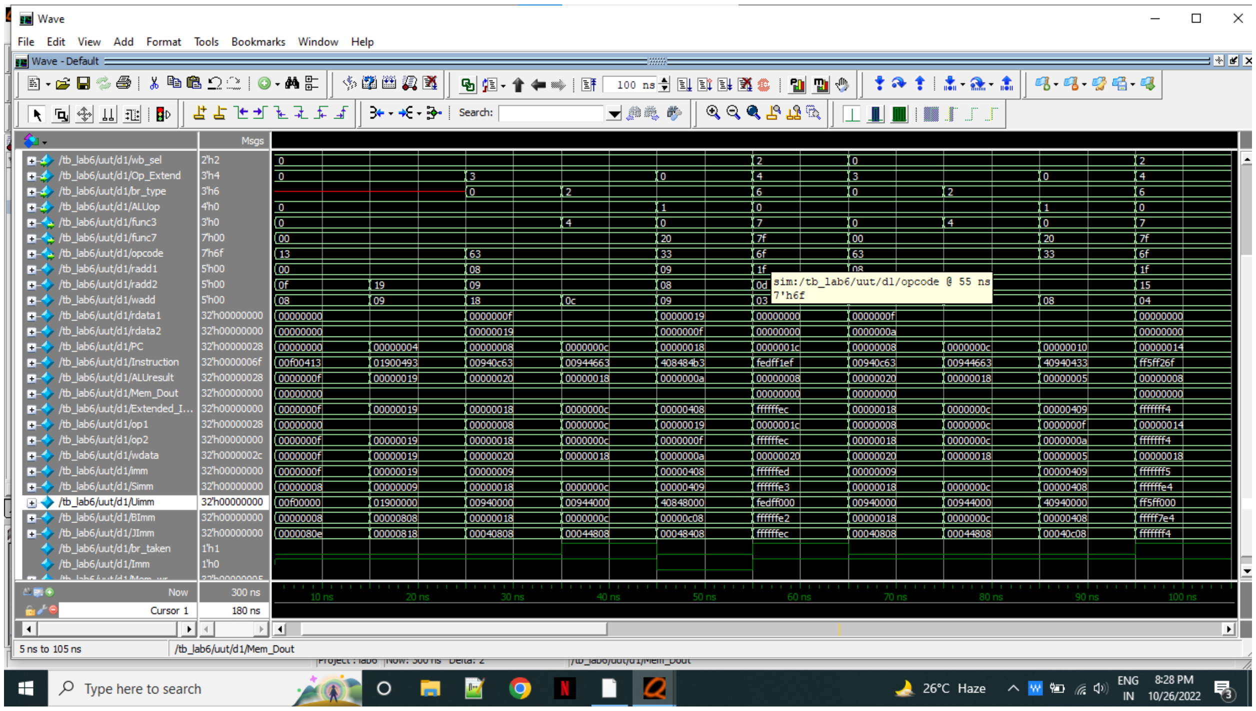
00f00413
01900493
00940c63
00944663
40940433
ff5ff26f
408484b3
fedff1ef
00802423
00802503
0000006f



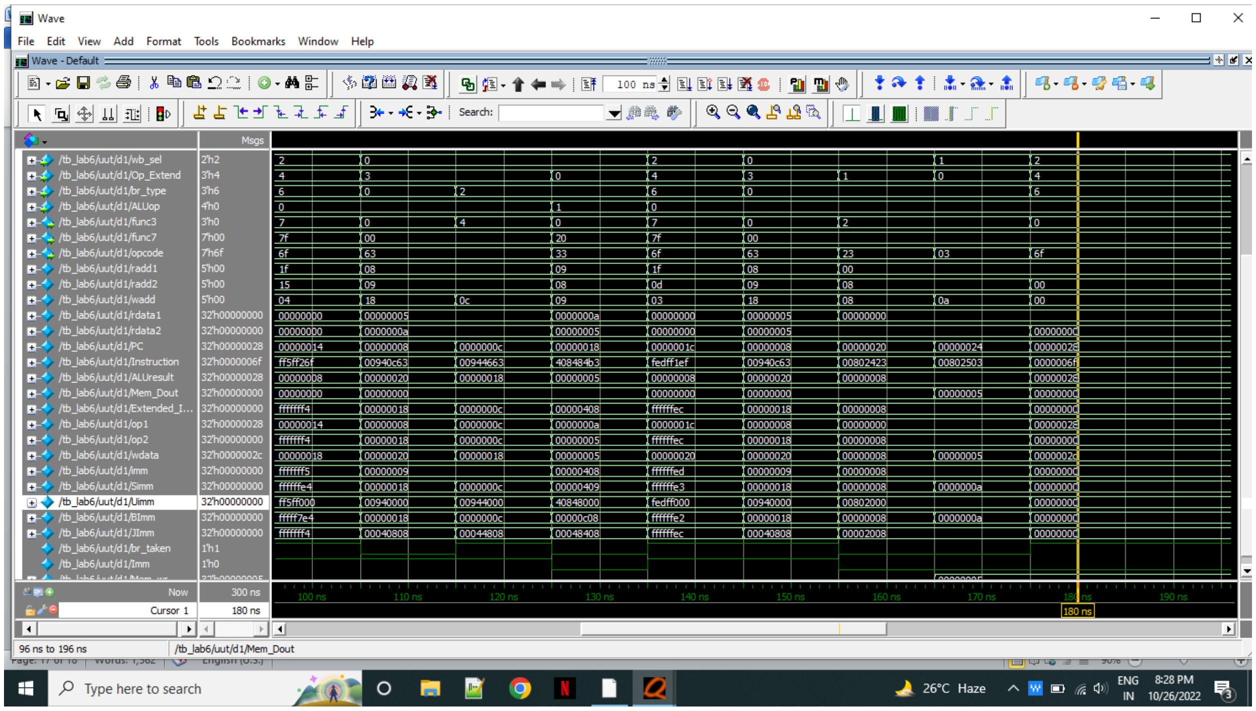
```
addi x8, x0, 15
addi x9, x0, 25
gcd:
    beq x8, x9, stop
    blt x8, x9, less
    sub x8, x8, x9
    jal, x4, gcd
less:
    sub x9, x9, x8
    jal, x3, gcd
stop:
    sw x8,8(x0)
    lw x10,8(x0)
end:
    j end
```

Questasim simulations and memory files:

Simulation waveforms:

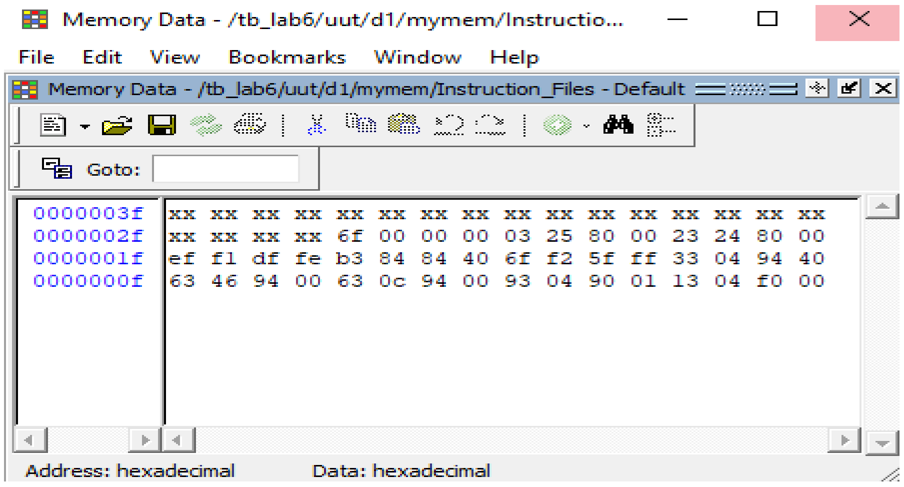


```
Wave - Default                                   Msgs
/tb_lab6/uut/d1/wb_sel                           2'h2
/tb_lab6/uut/d1/Op_Extend                        3'h4
/tb_lab6/uut/d1/br_type                          3'h6
/tb_lab6/uut/d1/ALUop                            4'h0
/tb_lab6/uut/d1/func3                            3'h0
/tb_lab6/uut/d1/func7                            7'h00
/tb_lab6/uut/d1/opcode                           7'h6f
/tb_lab6/uut/d1/radd1                            5'h00
/tb_lab6/uut/d1/radd2                            5'h00
/tb_lab6/uut/d1/wadd                             5'h00
/tb_lab6/uut/d1/rdata1                           32'h00000000
/tb_lab6/uut/d1/rdata2                           32'h00000000
/tb_lab6/uut/d1/PC                               32'h00000028
/tb_lab6/uut/d1/Instruction                      32'h0000006f
/tb_lab6/uut/d1/ALUresult                        32'h00000028
/tb_lab6/uut/d1/Mem_Dout                         32'h00000000
/tb_lab6/uut/d1/Extended_I...                    32'h00000000
/tb_lab6/uut/d1/op1                              32'h00000028
/tb_lab6/uut/d1/op2                              32'h00000000
/tb_lab6/uut/d1/wdata                            32'h0000002c
/tb_lab6/uut/d1/imm                              32'h00000000
/tb_lab6/uut/d1/Simm                             32'h00000000
/tb_lab6/uut/d1/Uimm                             32'h00000000
/tb_lab6/uut/d1/BImm                             32'h00000000
/tb_lab6/uut/d1/JImm                             32'h00000000
/tb_lab6/uut/d1/br_taken                         1'h1
/tb_lab6/uut/d1/Imm                              1'h0
Now                                              300 ns
Cursor 1                                         180 ns

sim:/tb_lab6/uut/d1/opcode @ 55 ns
7'h6f

5 ns to 105 ns        /tb_lab6/uut/d1/Mem_Dout
```



```
Wave - Default                                   Msgs
/tb_lab6/uut/d1/wb_sel                           2'h2
/tb_lab6/uut/d1/Op_Extend                        3'h4
/tb_lab6/uut/d1/br_type                          3'h6
/tb_lab6/uut/d1/ALUop                            4'h0
/tb_lab6/uut/d1/func3                            3'h0
/tb_lab6/uut/d1/func7                            7'h00
/tb_lab6/uut/d1/opcode                           7'h6f
/tb_lab6/uut/d1/radd1                            5'h00
/tb_lab6/uut/d1/radd2                            5'h00
/tb_lab6/uut/d1/wadd                             5'h00
/tb_lab6/uut/d1/rdata1                           32'h00000000
/tb_lab6/uut/d1/rdata2                           32'h00000000
/tb_lab6/uut/d1/PC                               32'h00000028
/tb_lab6/uut/d1/Instruction                      32'h0000006f
/tb_lab6/uut/d1/ALUresult                        32'h00000028
/tb_lab6/uut/d1/Mem_Dout                         32'h00000000
/tb_lab6/uut/d1/Extended_I...                    32'h00000000
/tb_lab6/uut/d1/op1                              32'h00000028
/tb_lab6/uut/d1/op2                              32'h00000000
/tb_lab6/uut/d1/wdata                            32'h0000002c
/tb_lab6/uut/d1/imm                              32'h00000000
/tb_lab6/uut/d1/Simm                             32'h00000000
/tb_lab6/uut/d1/Uimm                             32'h00000000
/tb_lab6/uut/d1/BImm                             32'h00000000
/tb_lab6/uut/d1/JImm                             32'h00000000
/tb_lab6/uut/d1/br_taken                         1'h1
/tb_lab6/uut/d1/Imm                              1'h0
Now                                              300 ns
Cursor 1                                         180 ns

96 ns to 196 ns       /tb_lab6/uut/d1/Mem_Dout
```

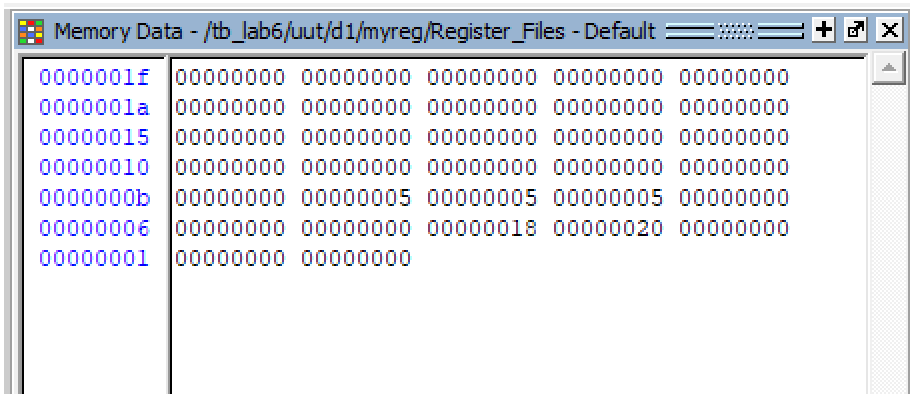
Instruction file:


Memory Data - /tb_lab6/uut/d1/mymem/Instructio...
File Edit View Bookmarks Window Help
Memory Data - /tb_lab6/uut/d1/mymem/Instruction_Files - Default
Goto:

```
0000003f xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx
0000002f xx xx xx xx 6f 00 00 00 03 25 80 00 23 24 80 00
0000001f ef f1 df fe b3 84 84 40 6f f2 5f ff 33 04 94 40
0000000f 63 46 94 00 63 0c 94 00 93 04 90 01 13 04 f0 00
```

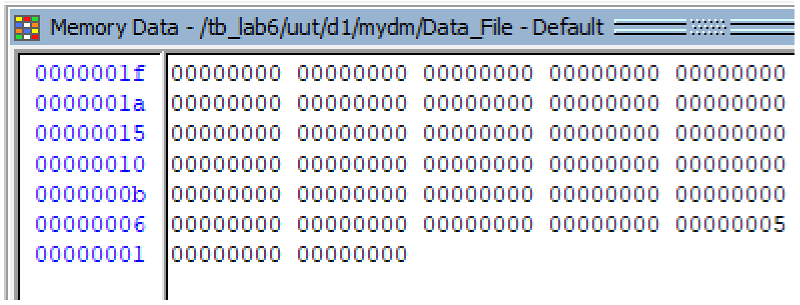
Address: hexadecimal
Data: hexadecimal

Register file:


Memory Data - /tb_lab6/uut/d1/myreg/Register_Files - Default

```
0000001f 00000000 00000000 00000000 00000000 00000000
0000001a 00000000 00000000 00000000 00000000 00000000
00000015 00000000 00000000 00000000 00000000 00000000
00000010 00000000 00000000 00000000 00000000 00000000
0000000b 00000000 00000005 00000005 00000005 00000000
00000006 00000000 00000000 00000018 00000020 00000000
00000001 00000000 00000000
```

Data file:



```
0000001f 00000000 00000000 00000000 00000000 00000000
0000001a 00000000 00000000 00000000 00000000 00000000
00000015 00000000 00000000 00000000 00000000 00000000
00000010 00000000 00000000 00000000 00000000 00000000
0000000b 00000000 00000000 00000000 00000000 00000000
00000006 00000000 00000000 00000000 00000000 00000005
00000001 00000000 00000000
```