CSE4117 MICROPROCESSORS

ASSIGNMENT #4 Report

150119679 - Muhammedcan PİRİNÇÇİ

150117061 - İsra Nur ALPEREN

150116836 - Enes Nadir SANLI

❖ Overview of the Project

In this assignment we added some new modules and using all the previous modules we completed the given work. But to be able to apply previous components to this work we would make some modifications on them. We increased the total instruction bit for one value and this value keeps the information of whether the instruction is L-type or R-type. Some new signals inside the control unit “MA, WD, RD”. New modules are rom, program counter, and register controller. We used a register controller module three times and those are taking the input signal of MA, WD, and RD.

For the program counter simulation we wrote basic instructions to see if every module works fine. In this example we use instructions that write, read to ram(R - type) and also classic L type.

4'b0000 : data = 12'b000000000001; // 0 # add, 1 -> acc = 1

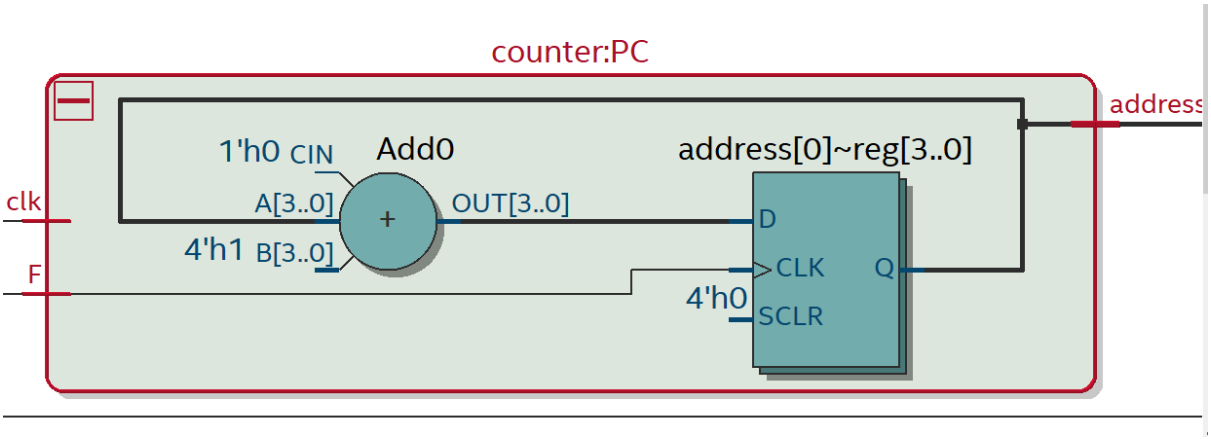
4'b0001 : data = 12'b000001100000; // 1 # add, 01100000 -> acc = 01100001

4'b0010 : data = 12'b111000100000; // 2 # storer, 100000 -> addr[100000] = 01100001

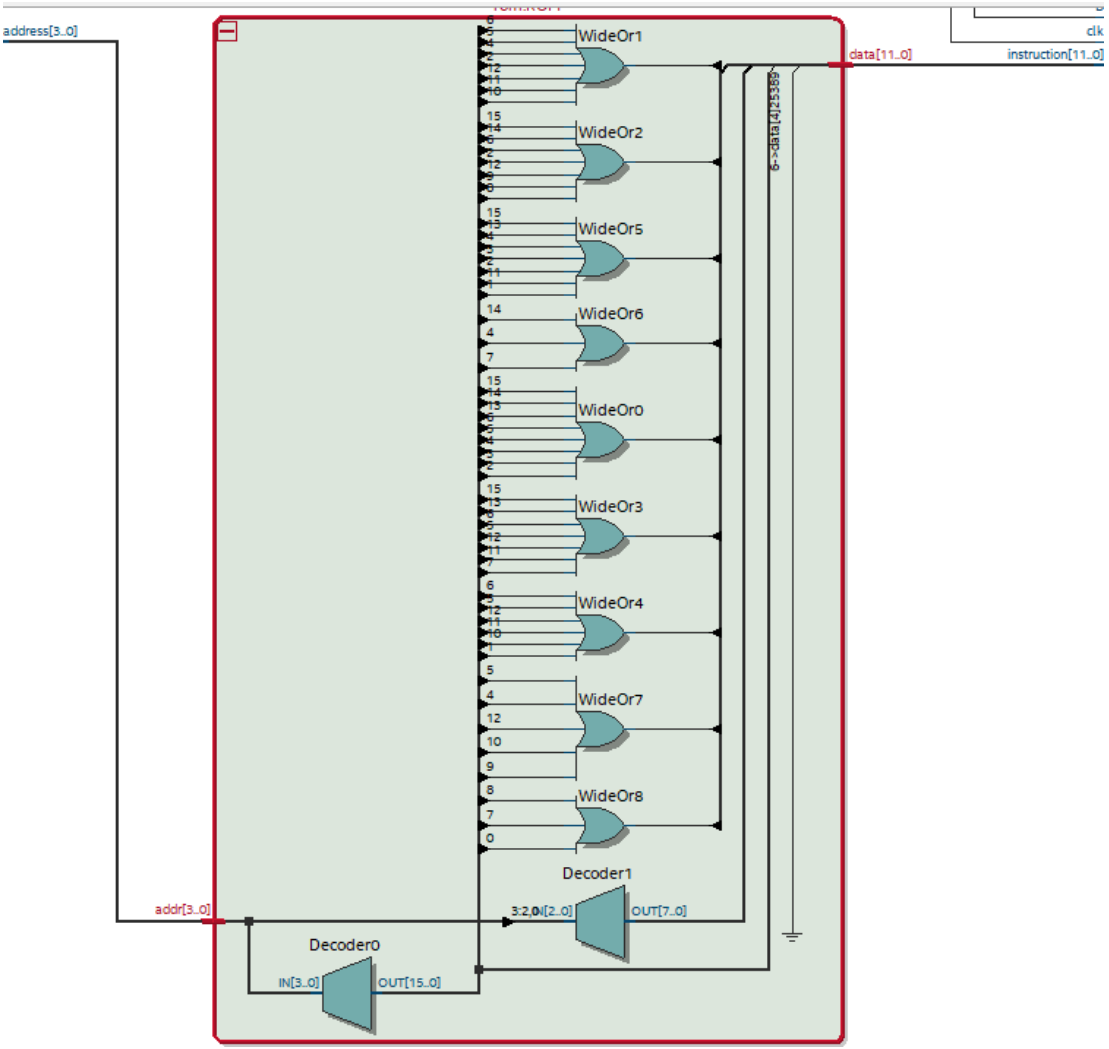
4'b0011 : data = 12'b100000100000; // 3 # addr, 100000 -> acc = (01100001 + 01100001) = 11000010

❖ RTL Views and Test Screenshots

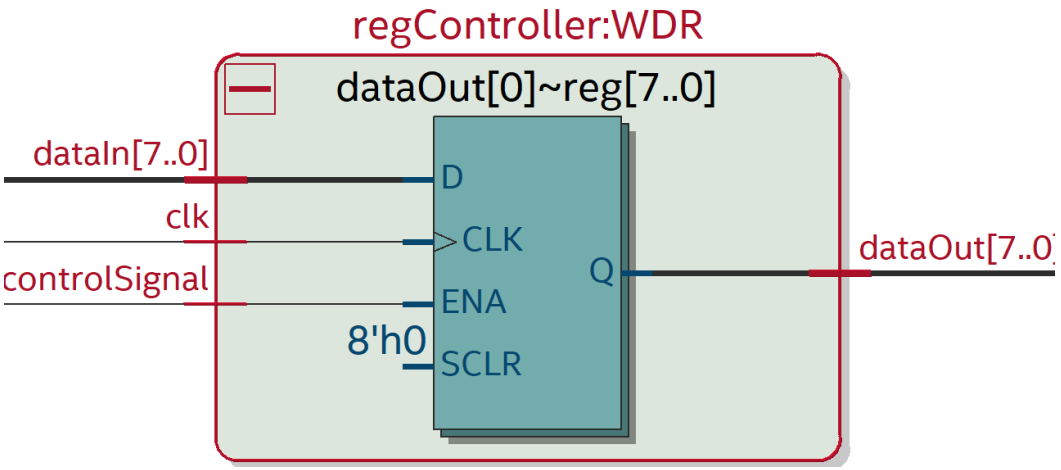
Program Counter:



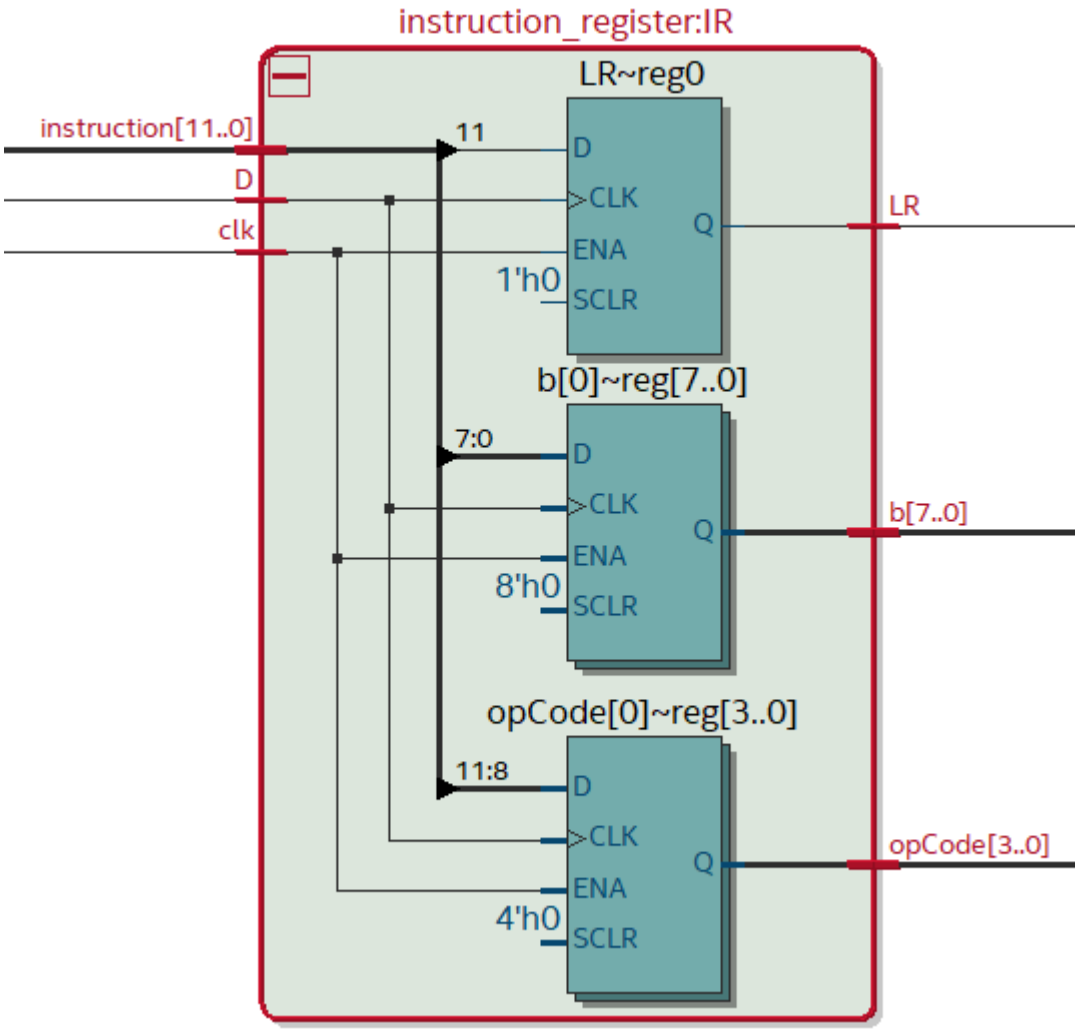
Rom:



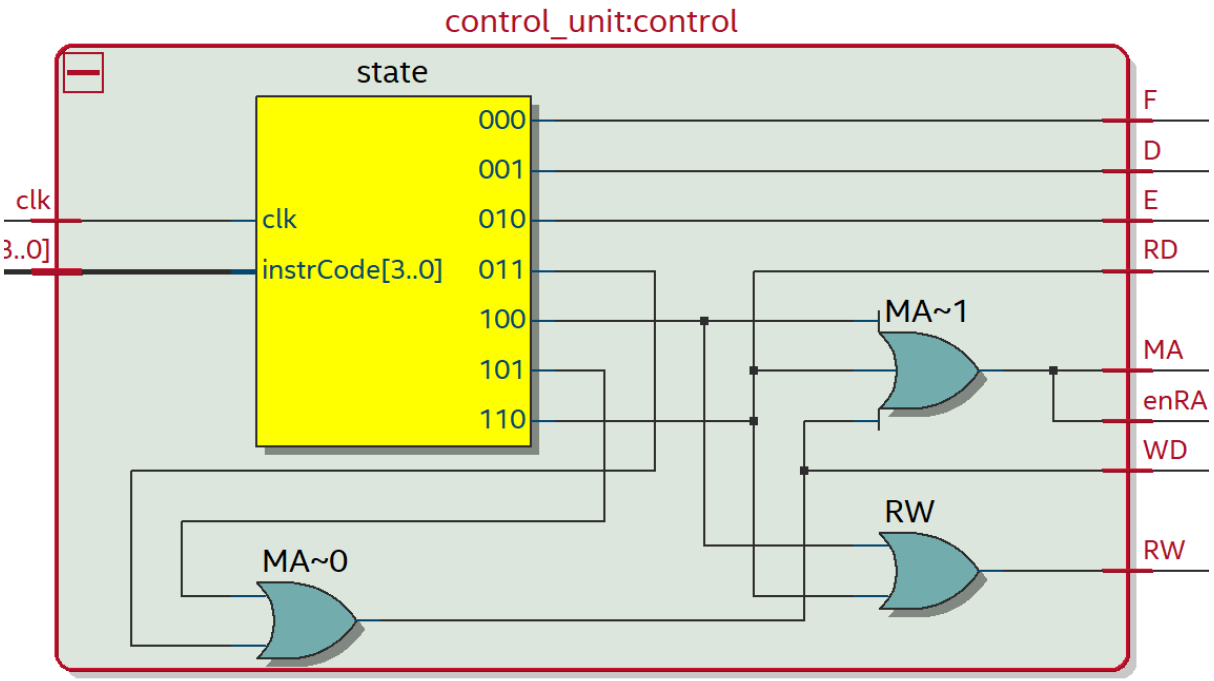
Register Controller:



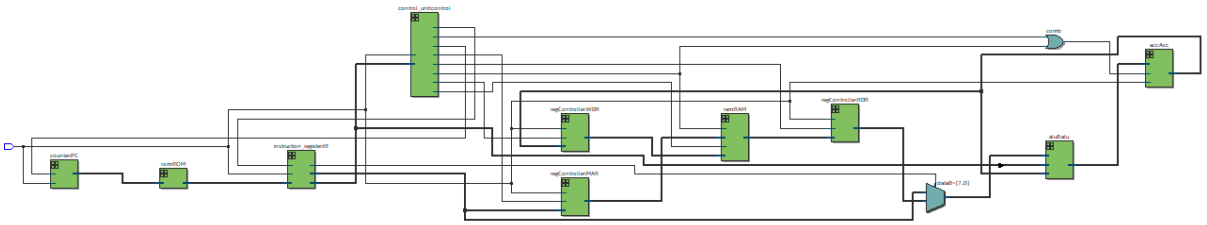
Instruction Register:



Control Unit:

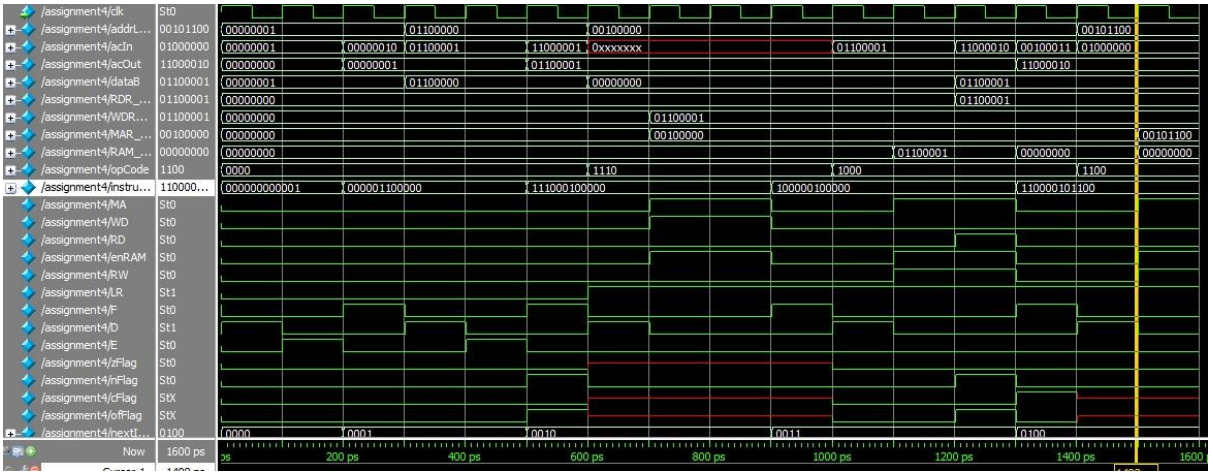


Assignment 4:



Test Results:

We made our simulation on the ModelSim simulation tool. The results were as we expected.



❖ New Verilog Files

counter.v file:

```
module counter( clk, F, address);
input clk, F;
output reg [3:0] address;

initial address = 4'b1111;

always@(posedge F)
begin
address = address + 1;
end

endmodule
```

rom.v file:

```
module rom(addr,data);
input [3:0] addr;
output reg [11:0] data;

always@ (*)
    case(addr)
        4'b0000 : data = 12'b000000000001; // 0
        4'b0001 : data = 12'b000001100000; // 1
        4'b0010 : data = 12'b111000100000; // 2
        4'b0011 : data = 12'b100000100000; // 3
        4'b0100 : data = 12'b110000101100; // 4
        4'b0101 : data = 12'b110101000100; // 5
        4'b0110 : data = 12'b111101010000; // 6
        4'b0111 : data = 12'b000100001001; // 7
        4'b1000 : data = 12'b001000000001; // 8
        4'b1001 : data = 12'b001000000100; // 9
        4'b1010 : data = 12'b010001000100; // a
        4'b1011 : data = 12'b010101100000; // b
        4'b1100 : data = 12'b011111000100; // c
        4'b1101 : data = 12'b100100100000; // d
        4'b1110 : data = 12'b101010001000; // e
        default : data = 12'b101100100000; // f
    endcase

endmodule
```

regController.v file:

```
module regController(clk, controlSignal, dataIn, dataOut);
input clk, controlSignal;
input [7:0] dataIn;
output reg [7:0] dataOut;
initial dataOut = 8'b0;

always@(*)
begin
dataOut = ( clk ==1 & controlSignal==1) ? dataIn: dataOut;
end

endmodule
```

assignment4.v file:

```
module assignment4(clk);

input clk;
wire[7:0] addrLiteral, acIn, acOut, dataB, RDR_Out, WDR_Out, MAR_Out, RAM_Out ;
wire[3:0] opCode;
wire[11:0] instruction;
wire  MA, WD, RD, enRAM, RW, LR, F, D, E, zFlag, nFlag, cFlag, ofFlag;
wire[3:0] nextInstrAdd;

instruction_register IR(clk, instruction, D, opCode, addrLiteral, LR);
control_unit control(clk,opCode, F , D, E,  MA, WD, RD, enRAM, RW);
counter PC( clk, F, nextInstrAdd);
rom ROM(nextInstrAdd,instruction);
ram RAM(RAM_Out, WDR_Out, MAR_Out,enRAM, RW);
regController MAR(clk, MA, addrLiteral, MAR_Out);
regController WDR(clk, WD, acOut, WDR_Out);
regController RDR(clk, RD, RAM_Out, RDR_Out);
assign dataB = LR ? RDR_Out: addrLiteral;//////////////////////////

alu8 alu( acIn, dataB, acOut, zFlag, nFlag, cFlag, ofFlag, opCode );
acc Acc(clk, acIn, (E | RW) , acOut);


endmodule
```

control_unit.v file:

```
module control_unit(clk, instrCode, F, D, E, MA, WD, RD, enRAM, RW);

input clk;
input [3:0] instrCode;
output F, D, E, MA, WD, RD, enRAM, RW;
reg [2:0] state;


initial state = 3'b000;

always @(posedge clk )
begin

   if((clk == 1) && (state == 3'b000)) // F state to D state
      state = 3'b001;  // D state

   // L type
   else if((state == 3'b001) && ~instrCode[3]) // D state to  E
      state = 3'b010; // E state - alu

   else if(state == 3'b010) // E state to F state
      state = 3'b000; // F state

   // Write
   else if((state == 3'b001) & instrCode[3] & (instrCode[2] & instrCode[1] & ~instrCode[0])) // D state to  WR - storer (if we use fsr we have to add it)
      state = 3'b011; // WR state

   else if(state == 3'b011) // WR to Execute store state
      state = 3'b101; // Execute store state

   else if(state == 3'b101) //  Execute store state to
      state = 3'b000; // F state

   // Read Ram
   else if((state == 3'b001) && instrCode[3]) // D state to  RR
      state = 3'b100; //RR

   else if(state == 3'b100) // RR to Execute load state
      state = 3'b110; // Execute store state

   else if(state == 3'b110) //  Execute store state to F state
      state = 3'b000; // F state

end



assign F = (state == 3'b000) ? 1 : 0;
assign D = (state == 3'b001) ? 1 : 0;
assign E = (state == 3'b010) ? 1 : 0;
assign RD = (state == 3'b110) ? 1 : 0;
assign RW = ((state == 3'b100) || (state == 3'b110)) ? 1 : 0;
assign WD = ((state == 3'b011) || (state == 3'b101)) ? 1 : 0;
assign enRAM = ((state == 3'b011) || (state == 3'b101) || (state == 3'b100) || (state == 3'b110)) ? 1 : 0;
assign MA = ((state == 3'b011) || (state == 3'b101) || (state == 3'b100) || (state == 3'b110)) ? 1 : 0;


endmodule
```