A Simplified MIPS Processor in Verilog

PC (1/1)

```
module MIPSPC(clk, newPC, PC);
        parameter DELAY_T = 10;
        parameter MIPS_PC_WIDTH_m1 = 7;

        input clk;
        input [MIPS_PC_WIDTH_m1:0] newPC;
        output [MIPS_PC_WIDTH_m1:0] PC;
        reg [MIPS_PC_WIDTH_m1:0] currPC;
        initial
        begin
                currPC = 0;
        end
        always @(posedge clk)
        begin
                #DELAY_T
                currPC = newPC;
        end
        assign PC = currPC;
endmodule
```

Instruction Memory (1/4)

```
module IM(CSB,WRB,ABUS,DATABUS);
        parameter DELAY_T = 10;
        parameter IM_DATA_W_m1 = 31;
        parameter IM_ADDR_W_m1 = 7;
        parameter IM_ADDR_MAX_m1 = 255;
        parameter IM_DATA_Z = 32'bzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz;

        input CSB;    // active low chip select
        input WRB;  // active low write control
        input [IM_ADDR_W_m1:0] ABUS;      // address bus
        inout [IM_DATA_W_m1:0] DATABUS;   // data bus

        //** internal signals
        reg   [IM_DATA_W_m1:0] DATABUS_driver;
        wire  [IM_DATA_W_m1:0] DATABUS = DATABUS_driver;
        reg   [IM_DATA_W_m1:0] ram[0:IM_ADDR_MAX_m1];

        integer i;
```

Instruction Memory (2/4)

```
module IM(CSB,WRB,ABUS,DATABUS);
    initial    //initialize all RAM cells to 0 at startup
    begin
        DATABUS_driver = IM_DATA_Z;
        $display($time," Reading MIPS program");
        for (i=0; i <= IM_ADDR_MAX_m1; i = i + 1)
            ram[i] = 0;
        //insert code here
```

Instruction Memory (3/4)

```
module IM(CSB,WRB,ABUS,DATABUS);
                ram[0]  = 32'b00100000000000000000000000000000;  // addi $0, $0, 0
                ram[1]  = 32'b00100000001000010000000000000001;  // addi $1, $1, 1
                ram[2]  = 32'b00100000010000100000000000000010;  // addi $2, $2, 2
                ram[3]  = 32'b00100000011000110000000000000011;  // addi $3, $3, 3
                ram[4]  = 32'b00100000100001000000000000000100;  // addi $4, $4, 4
                ram[5]  = 32'b00100000101001010000000000000101;  // addi $5, $5, 5
                ram[6]  = 32'b00000000000000000000000000000000;  // nop
                ram[7]  = 32'b00000000000000000000000000000000;  // nop
                ram[8]  = 32'b00000000000000000000000000000000;  // nop
                ram[9]  = 32'b00000000000000000000000000000000;  // nop
                ram[10] = 32'b00010000100000110000000000000010;  // beq $4, $3, 2
                ram[11] = 32'b10101100011000100000000000000001;  // sw $2, 1($3)
                ram[12] = 32'b10001100100000101000000000000000;  // lw $5, 0($4)
                ram[13] = 32'b00000000100001010001100000100000;  //add $3, $4, $5
        end
```

Instruction Memory (4/4)

```
module IM(CSB,WRB,ABUS,DATABUS);
        always @(CSB or WRB or ABUS)
        begin
        if (CSB == 1'b0)
                begin
                if (WRB == 1'b1) //Reading from sram (data valid after 10ns)
                        begin
                                #10 DATABUS_driver =  ram[ABUS];
                                $display($time," Reading %m ABUS=%b
                                        DATA=%b",ABUS,DATABUS_driver);
                        end
                end
        else //sram unselected, stop driving bus after 10ns
                begin
                        DATABUS_driver <=  #DELAY_T IM_DATA_Z;
                end
        end
endmodule
```

Register File (1/1)

```
module MIPSREG(clk, RegWrite, ReadAddr1, ReadAddr2, WriteAddr, ReadData1, ReadData2, WriteData);
        parameter DELAY_T = 10;
        parameter MIPS_REG_ADDR_W_m1 = 4;
        parameter MIPS_REG_DATA_W_m1 = 31;
        parameter MIPS_REG_NUM_m1 = 31;

        input clk, RegWrite;
        input [MIPS_REG_ADDR_W_m1:0] ReadAddr1, ReadAddr2, WriteAddr;
        input [MIPS_REG_DATA_W_m1:0] ReadData1, ReadData2, WriteData;
        reg [MIPS_REG_DATA_W_m1:0] regs [0:MIPS_REG_NUM_m1];

        integer i;

        initial    //initialize all RAM cells to 0 at startup
        begin
                for (i=0; i <= MIPS_REG_NUM_m1; i = i + 1)
                        regs[i] = 0;
        end
        always @(posedge clk)
        begin
                if (RegWrite == 1'b1)
                begin
                        #DELAY_T
                        $display($time," writing %m regindex=%b val=%b",WriteAddr,WriteData);
                        regs[WriteAddr] = WriteData;
                end
        end
        assign ReadData1 = regs[ReadAddr1];
        assign ReadData2 = regs[ReadAddr2];
endmodule
```

ALU (1/1)

```
module MIPSALU (ALUctl, A, B, ALUOut, Zero);
        input [3:0] ALUctl;
        input [31:0] A,B;
        output reg [31:0] ALUOut;
        output Zero;

        assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0
        always @(ALUctl, A, B) //reevaluate if these change
        case (ALUctl)
                0: ALUOut <= A & B;
                1: ALUOut <= A | B;
                2: ALUOut <= A + B;
                6: ALUOut <= A - B;
                7: ALUOut <= A < B ? 1:0;
                12: ALUOut <= ~(A | B); // result is nor
                default: ALUOut <= 0; //default to 0, should not happen;
        endcase
endmodule
```

Data Memory (1/2)

```
module DM(MemRead, MemWrite, ABUS, DIN, DATABUS);
        parameter DELAY_T = 10;
        parameter DM_DATA_W_m1 = 31;
        parameter DM_ADDR_W_m1 = 7;
        parameter DM_ADDR_MAX_m1 = 255;
        parameter DM_DATA_Z = 32'bzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz;

        input MemRead;
        input MemWrite;
        input [DM_ADDR_W_m1:0] ABUS;      // address bus
        input [DM_DATA_W_m1:0] DIN;       // data in bus
        output [DM_DATA_W_m1:0] DATABUS;   // data out bus

        //** internal signals
        reg   [DM_DATA_W_m1:0] DATABUS_driver;
        wire  [DM_DATA_W_m1:0] DATABUS = DATABUS_driver;
        reg   [DM_DATA_W_m1:0] ram[0:DM_ADDR_MAX_m1];

        integer i;
```

Data Memory (2/2)

```
module DM(MemRead, MemWrite, ABUS, DIN, DATABUS);
        initial   //initialize all RAM cells to junk at startup
        begin
                DATABUS_driver = 0;
                for (i=0; i <= DM_ADDR_MAX_m1; i = i + 1)
                        ram[i] = i*10 + 1;
        end
        always @(MemRead or MemWrite or ABUS or DIN)
        begin
                #DELAY_T DATABUS_driver =  ram[ABUS];
                $display($time," Reading %m ABUS=%b
                        DATA=%b",ABUS,DATABUS_driver);
                if (MemWrite == 1'b1)
                begin
                        #30
                        if (MemWrite == 1'b1)
                         begin
                                $display($time," Writing %m ABUS=%b
                                        DATA=%b",ABUS,DIN);
                                ram[ABUS] = DIN;
                         end
                end
        endendmodule
```

MUX (1/1)

```
module STwoToOne32 (sel, in0, in1, out);
        input sel;
        input [31:0] in0, in1;
        output reg [31:0] out;

        always @(sel, in0, in1)
                if (sel == 0)
                        out <= in0;
                else
                        out <= in1;
endmodule

module STwoToOne5 (sel, in0, in1, out);
        input sel;
        input [4:0] in0, in1;
        output reg [4:0] out;

        always @(sel, in0, in1)
                if (sel == 0)
                        out <= in0;
                else
                        out <= in1;
endmodule
```

Sign Extend (1/1)

```
module SignExtend (in, out);
        input [15:0] in;
        output [31:0] out;

        assign out[15:0] = in[15:0];
        assign out[31:16] = in[15];
endmodule
```

Next PC (1/1)

```
module getNextPC (PCSrc, currPC, offset, out);
        parameter MIPS_PC_WIDTH_m1 = 7;
        input PCSrc;
        input [MIPS_PC_WIDTH_m1:0] offset;
        input [MIPS_PC_WIDTH_m1:0] currPC;
        output reg [MIPS_PC_WIDTH_m1:0] out;

        always @(PCSrc, currPC, offset)
                if (PCSrc == 0)
                        out <= currPC + 1;
                else
                        out <= currPC + 1 + offset;
endmodule
```

Control (1/4)

```
module MIPSCtrl (instr, RegDst, ALUSrc, MemToReg, RegWrite, MemWrite, MemRead,
branch, ALUCtrl);
        input [31:0] instr;
        output reg RegDst, ALUSrc, MemToReg, RegWrite, MemWrite, MemRead, branch;
        output reg [3:0] ALUCtrl;

        always @(instr) //reevaluate if these change
                if (instr[31:26] == 6'b000000) // R-type
                        begin
                                RegDst <= 1;
                                ALUSrc <= 0;
                                MemToReg <= 0;
                                RegWrite <= 1;
                                MemRead <= 0;
                                MemWrite <= 0;
                                branch <= 0;
                                case (instr[5:0])
                                        32: ALUCtrl <= 4'b0010;
                                        34: ALUCtrl <= 4'b0110;
                                        default: ALUCtrl <= 0; //should not happen
                                endcase
                        end
```

Control (2/4)

```
module MIPSCtrl (instr, RegDst, ALUSrc, MemToReg, RegWrite, MemWrite, MemRead, branch,
ALUCtrl);
                else if (instr[31:26] == 6'b100011) // lw
                        begin
                                RegDst <= 0;
                                ALUSrc <= 1;
                                MemToReg <= 1;
                                RegWrite <= 1;
                                MemRead <= 1;
                                MemWrite <= 0;
                                branch <= 0;
                                ALUCtrl <= 4'b0010;
                        end
                else if (instr[31:26] == 6'b101011) // sw
                        begin
                                RegDst <= 0;
                                ALUSrc <= 1;
                                MemToReg <= 0;
                                RegWrite <= 0;
                                MemRead <= 0;
                                MemWrite <= 1;
                                branch <= 0;
                                ALUCtrl <= 4'b0010;
                        end
```

Control (3/4)

```
module MIPSCtrl (instr, RegDst, ALUSrc, MemToReg, RegWrite, MemWrite, MemRead, branch,
ALUCtrl);
                else if (instr[31:26] == 6'b000100) //beq
                        begin
                                RegDst <= 0;
                                ALUSrc <= 0;
                                MemToReg <= 0;
                                RegWrite <= 0;
                                MemRead <= 0;
                                MemWrite <= 0;
                                branch <= 1;
                                ALUCtrl <= 4'b0110;
                        end
                else if (instr[31:26] == 6'b001000) //addi
                        begin
                                RegDst <= 0;
                                ALUSrc <= 1;
                                MemToReg <= 0;
                                RegWrite <= 1;
                                MemRead <= 0;
                                MemWrite <= 0;
                                branch <= 0;
                                ALUCtrl <= 4'b0010;
                        end
```

Control (4/4)

```
module MIPSCtrl (instr, RegDst, ALUSrc, MemToReg, RegWrite, MemWrite, MemRead, branch,
ALUCtrl);
                    else if (instr[31:26] == 6'b000001) //lwr
                            begin
                                    RegDst <= 1;
                                    ALUSrc <= 0;
                                    MemToReg <= 1;
                                    RegWrite <= 1;
                                    MemRead <= 1;
                                    MemWrite <= 0;
                                    branch <= 0;
                                    ALUCtrl <= 4'b0110;
                            end
                    else
                            begin
                                    RegDst <= 0;
                                    ALUSrc <= 0;
                                    MemToReg <= 0;
                                    RegWrite <= 0;
                                    MemRead <= 0;
                                    MemWrite <= 0;
                                    branch <= 0;
                                    ALUCtrl <= 4'b0000;
                            end
endmodule
```

Test Bench (1/2)

```
module test_bench ();
        reg osc;
        initial begin
                osc = 0;
        end

        always begin
                #100 osc = ~osc;
        end

        wire clk;
        assign clk = osc;

        parameter DM_DATA_W_m1 = 31;
        parameter DM_ADDR_W_m1 = 7;
        parameter IM_DATA_W_m1 = 31;
        parameter IM_ADDR_W_m1 = 7;
        wire RegDst;
        wire ALUSrc;
        wire MemToReg;
        wire RegWrite;
        wire MemWrite;
        wire MemRead;
        wire branch, PCSrc;
        wire [3:0] ALUCtrlSig;
```

Test Bench (2/2)

```
module test_bench ();
        wire [IM_ADDR_W_m1:0] newIADDR, iABUS;
        MIPSPC PC(clk, newIADDR, iABUS);

        wire [IM_DATA_W_m1:0] iDATABUS;
        IM instMem(1'b0,1'b1, iABUS, iDATABUS);

        wire [4:0] RFWriteAddr;
        wire [31:0] ReadData1, ReadData2, WriteData;
        MIPSREG RegFile(clk, RegWrite, iDATABUS[25:21], iDATABUS[20:16], RFWriteAddr, ReadData1, ReadData2,
                                        WriteData);

        wire [31:0] ALUSndInput, ALUOut;
        wire Zero;
        MIPSALU ALU(ALUCtrlSig, ReadData1, ALUSndInput, ALUOut, Zero);

        wire [DM_DATA_W_m1:0] dDATABUS;
        DM dataMem(MemRead, MemWrite, ALUOut, ReadData2, dDATABUS);

        wire [31:0] extended32;
        SignExtend SignExt(iDATABUS[15:0], extended32[31:0]);

        assign PCSrc = Zero & branch;
        getNextPC NextPC(PCSrc, iABUS, iDATABUS[7:0], newIADDR);

        STwoToOne5 TwoToOne5_1(RegDst, iDATABUS[20:16], iDATABUS[15:11], RFWriteAddr);
        STwoToOne32 TwoToOne32_1(ALUSrc, ReadData2, extended32, ALUSndInput);
        STwoToOne32 TwoToOne32_2(MemToReg, ALUOut, dDATABUS, WriteData);

        MIPSCtrl Control(iDATABUS, RegDst, ALUSrc, MemToReg, RegWrite, MemWrite, MemRead, branch, ALUCtrlSig);
endmodule
```

Running

- Set the clock to 200ps
- Every time you click run simulation, another instruction is executed
- View wire details from the wave form
- Use the memory viewer to look at the registers and data/instruction memory