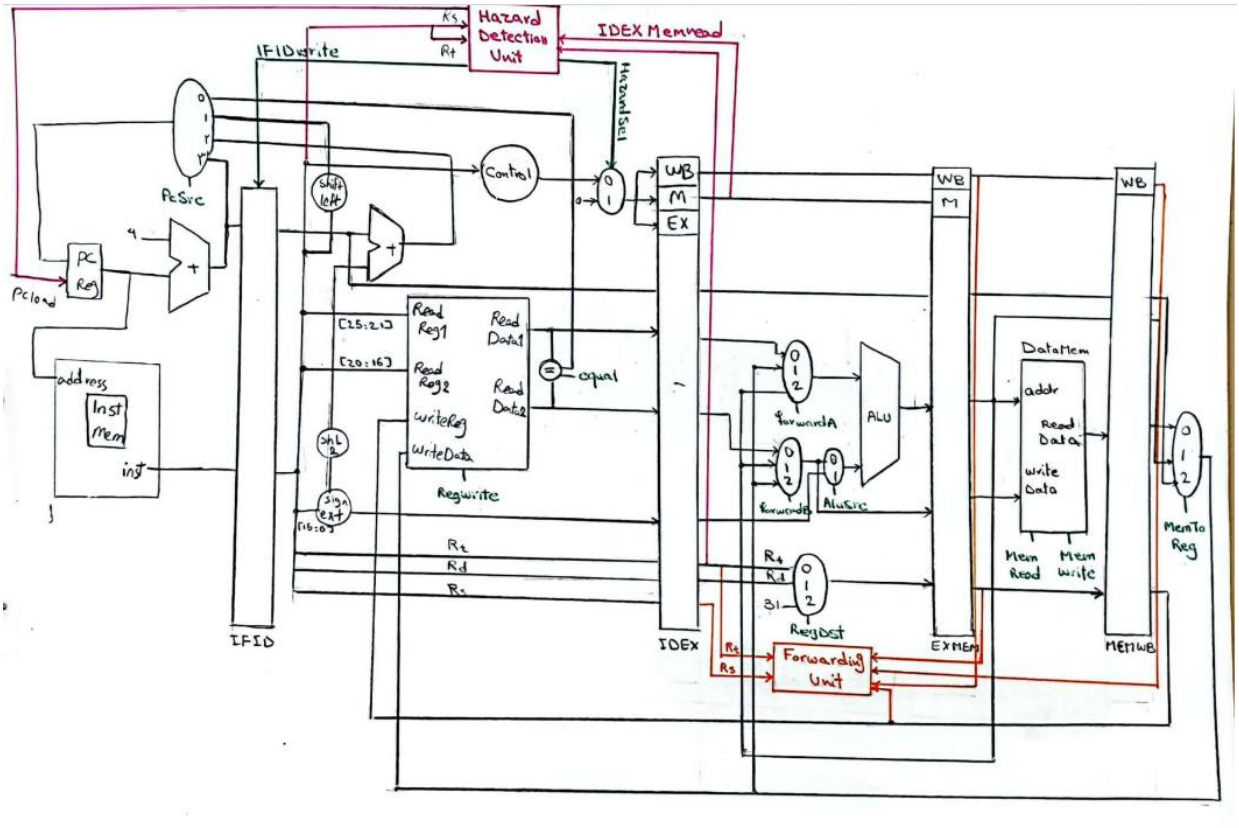
Computer Assignment3

“PipeLine Mips Processor”

Ava Mirmohamadmahdi 810199501

Nesa Abbasimoghadam 810199457

DataPath



DataPath Verilog

```
module DataPath(instMemAddress, dataMemAddress, dataMemWriteData, IFIDInstructionOut, RtIDEX, RsIDEX, RdEXMEM, RdMEMWB,
                equal, MemReadIDEXOut, dataMemWrite, dataMemRead, RegWriteEXMEM, RegWriteMEMWB, instruction,
                dataMemReadData, AluOp, RegDst, PCSrc, MemToReg, forwardA, forwardB, HazardSel, ALUSrc,
                RegWrite, clk, rst, MemWrite, MemRead, PcLoad, IFIDLoad, Flush);

    output [31:0] instMemAddress, dataMemAddress, dataMemWriteData;
    output [31:0] IFIDInstructionOut;
    output [4:0] RtIDEX, RsIDEX;
    output [4:0] RdEXMEM;
    output [4:0] RdMEMWB;
    output equal;
    output MemReadIDEXOut;
    output dataMemWrite, dataMemRead;
    output RegWriteEXMEM, RegWriteMEMWB;
    input [31:0] instruction, dataMemReadData;
    input [2:0] AluOp;
    input [1:0] RegDst, PCSrc, MemToReg;
    input [1:0] forwardA, forwardB;
    input HazardSel, ALUSrc, RegWrite, clk, rst;
    input MemWrite, MemRead;
    input PcLoad, IFIDLoad;
    input Flush;

    wire [31:0] pcOut;
    wire [31:0] muxPcSrcOut;
    Register #(32) PcReg(pcOut, muxPcSrcOut, rst, PcLoad, clk);

    wire [31:0] adderPcOut;
    Adder AdderPc(adderPcOut, 32'd4, pcOut);

    wire [31:0] IFIDAdderOut;
    IFIDReg IFID(IFIDInstructionOut, IFIDAdderOut, instruction, adderPcOut, IFIDLoad, Flush, clk, rst);

    wire [27:0] shiftJOut;
    wire [31:0] jOut;
    assign shiftJOut = {IFIDInstructionOut [25:0] ,2'b00};
    assign jOut = {IFIDAdderOut [31:28], shiftJOut};

    wire [31:0] shift2Beq;
    wire [31:0] signExtended;
    assign signExtended = {{16{IFIDInstructionOut[15]}}, IFIDInstructionOut[15:0]};
    assign shift2Beq = signExtended << 2;
    wire [31:0] adderBeqOut;
    wire [31:0] readData1, readData2;

    Mux4To1 #(32) MuxPcSrc(muxPcSrcOut, readData1, jOut, adderBeqOut, adderPcOut, PCSrc);

    Adder BeqAdder(adderBeqOut, IFIDAdderOut, shift2Beq);

    wire [31:0] muxMemToRegOut;
    wire [4:0] writeRegMEMWB;

    RegFile RG(readData1, readData2, muxMemToRegOut, IFIDInstructionOut[25:21], IFIDInstructionOut[20:16],
               writeRegMEMWB, RegWriteMEMWB, rst, clk);

    assign equal = (readData1 == readData2);

    wire [31:0] readData1IDEX, readData2IDEX, signExtendIDEX;
    wire [31:0] adderPcOutIDEX;
    wire [4:0] RdIDEX;

    wire [2:0] AluOpIDEX;
    wire [1:0] RegDstIDEX, MemToRegIDEX;
    wire AluSrcIDEX, RegWriteIDEX, MemReadIDEX, MemWriteIDEX;

    wire [10:0] muxHazardOut;
    wire [10:0] hazard;
    assign hazard = {MemToReg/*10-9*/, MemWrite/*8*/, MemRead/*7*/, RegDst/*6-5*/, RegWrite/*4*/, ALUSrc/*3*/, AluOp/*2-0*/};
```

```
    Mux2To1 #(11) MuxHazard(muxHazardOut, 11'd0, hazard, HazardSel);
    IDEX IDEXreg(AluSrcIDEX, RegWriteIDEX, MemReadIDEX, MemWriteIDEX, RegDstIDEX, MemToRegIDEX, AluOpIDEX,
            RtIDEX, RsIDEX, RdIDEX, adderPcOutIDEX, readData1IDEX, readData2IDEX, signExtendIDEX, clk,
            rst, muxHazardOut[3], muxHazardOut[4], muxHazardOut[7], muxHazardOut[8], muxHazardOut[6:5],
            muxHazardOut[10:9], muxHazardOut[2:0], IFIDInstructionOut[20:16],
            IFIDInstructionOut[25:21], IFIDInstructionOut[15:11], adderPcOut, readData1, readData2, signExtended);

    assign MemReadIDEXOut = MemReadIDEX;

    wire [4:0] muxRegDstOut;
    Mux3To1 #(5) MuxRegDst(muxRegDstOut, RtIDEX, RdIDEX, 5'd31, RegDstIDEX);

    wire [31:0] muxBOut, muxAluSrcOut;
    Mux2To1 #(32) MuxAluSrc(muxAluSrcOut, muxBOut, signExtendIDEX, AluSrcIDEX);

    wire [31:0] muxAOut;
    wire [31:0] aluResultEXMEM;

    Mux3To1 #(32) MuxForwardA(muxAOut, readData1IDEX, muxMemToRegOut, aluResultEXMEM, forwardA);

    Mux3To1 #(32) MuxForwardB(muxBOut, readData2IDEX, muxMemToRegOut, aluResultEXMEM, forwardB);

    wire [31:0] aluResult;
    ALU Alu(aluResult, muxAluSrcOut, muxAOut, AluOpIDEX);

    wire [31:0] adderPcOutEXMEM;
    wire [31:0] muxBEXMEM;
    wire [4:0] RegDstEXMEM;
    wire [1:0] MemToRegEXMEM;
    wire MemReadEXMEM, MemWriteEXMEM;
    EXMEM EXMEMReg(RegWriteEXMEM, MemReadEXMEM, MemWriteEXMEM, MemToRegEXMEM, RegDstEXMEM, adderPcOutEXMEM,
            aluResultEXMEM, muxBEXMEM, clk, rst, RegWriteIDEX, MemReadIDEX, MemWriteIDEX, MemToRegIDEX, muxRegDstOut,
            adderPcOutIDEX, aluResult, muxBOut);

    assign dataMemRead = MemReadEXMEM;
    assign dataMemWrite = MemWriteEXMEM;
    assign RdEXMEM = RegDstEXMEM;

    wire [1:0] MemToRegMEMWB;

    wire [31:0] memoryDataMEMWB;
    wire [31:0] aluResultMEMWB;
    wire [31:0] adderPcOutMEMWB;

    MEMWB MEMWBReg(RegWriteMEMWB, MemToRegMEMWB, writeRegMEMWB, memoryDataMEMWB, aluResultMEMWB, adderPcOutMEMWB, clk, rst,
            RegWriteEXMEM, MemToRegEXMEM, RegDstEXMEM, dataMemReadData, aluResultEXMEM, adderPcOutEXMEM);

    assign RdMEMWB = writeRegMEMWB;

    Mux3To1 #(32) MuxMemToReg(muxMemToRegOut, aluResultMEMWB, memoryDataMEMWB, adderPcOutMEMWB, MemToRegMEMWB);

    assign instMemAddress = pcOut;
    assign dataMemAddress = aluResultEXMEM;
    assign dataMemWriteData = muxBEXMEM;

endmodule
```

Controller

	Flush	ALUop	Memwrite	MemRead	ALUsrc	Regwrite	MemToReg	RegDst	Pcsrc
RType	0	10	0	0	0	1	00	01	11
Addi	0	00	0	0	1	1	00	00	11
sw	0	00	1	0	1	0	00	00	11
Lw	0	00	0	1	1	1	01	00	11
Jump	1	00	0	0	0	0	00	00	01
JumpR	0	00	0	0	0	0	00	00	00
Jal	0	00	0	0	0	0	10	10	01
Beq	equal	00	0	0	0	0	00	00	10 or 11
Slti	0	11	0	0	1	1	00	00	11

AlU Controller Verilog

```
module AluController(ALUoperation, ALUop, func);
    output reg [2:0] ALUoperation;
    input [1:0] ALUop;
    input [5:0] func;
    parameter[5:0] And = 6'b100100, Or = 6'b100101, Add =  6'b100000, Sub = 6'b100010,  Slt = 6'b101010;
    always@(ALUop, func) begin
        ALUoperation = 3'b010;
        if(ALUop == 2'd0) // lw or sw
            ALUoperation = 3'b010;
        else if(ALUop == 2'd1) // beq
            ALUoperation = 3'b011;
        else if(ALUop == 2'd2) begin
            case(func)
                And: ALUoperation = 3'b000;
                Or:  ALUoperation = 3'b001;
                Add: ALUoperation = 3'b010;
                Sub: ALUoperation = 3'b011;
                Slt: ALUoperation = 3'b111;  // slt & slti
                default:   ALUoperation = 3'b000;
            endcase
        end
        else
            ALUoperation = 3'b111;
    end
endmodule
```

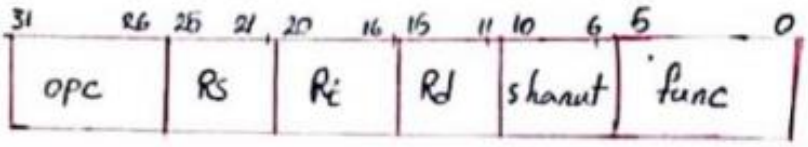
Controller Verilog

```
module Controller(MemToReg, PCSrc, RegDst, ALUoperation, ALUSrc, RegWrite, MemWrite, MemRead, Flush,
                  opc, func, equal);
    output reg [1:0] MemToReg, PCSrc, RegDst;
    output [2:0] ALUoperation;
    output reg ALUSrc, RegWrite, MemWrite, MemRead, Flush;
    input [5:0] opc, func;
    input equal;
    reg [1:0] ALUop;
    reg [3:0] state;
    AluController ALUCtrl(ALUoperation, ALUop, func);

    parameter [5:0] RType = 6'b000000, Lw = 6'b100011, Sw = 6'b101011, Beq = 6'b000100, Addi = 6'b001000,
        Jump =6'b000010, Jal = 6'b000011, JumpR = 6'b000110, Slti = 6'b001010;

        always @(opc, equal)begin
        {Flush, ALUop, MemWrite, MemRead, ALUSrc, RegWrite, MemToReg, RegDst, PCSrc} = 13'd3;
        case (opc)
            RType : begin
                RegDst = 2'b01;
                RegWrite = 1'b1;
                ALUop = 2'b10;
                state = 4'd1;
            end

            Addi: begin
                RegWrite = 1'b1;
                ALUSrc = 1'b1;
                state = 4'd5;
            end

            Sw : begin
                ALUSrc = 1'b1;
                MemWrite = 1'b1;
                state = 4'd3;
            end
            Lw: begin
                ALUSrc = 1'b1;
                MemToReg = 2'b01;
                RegWrite = 1'b1;
                MemRead = 1'b1;
                state = 4'd2;
            end

            Jump: begin
                PCSrc = 2'b01;
                Flush = 1'b1;
                state = 4'd6;
            end

            JumpR: begin
                PCSrc = 2'b00;
                state = 4'd8;
            end

            Jal: begin
                RegDst = 2'b10;
                MemToReg = 2'b10;
                PCSrc = 2'b01;
                state = 4'd7;
            end
```

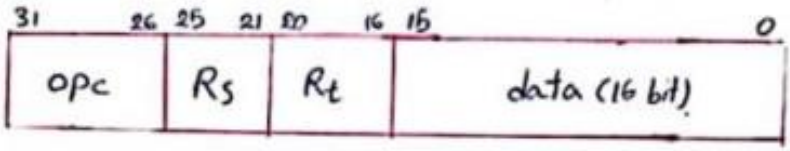
```
            Beq : begin
            PCSrc = equal ? 2'b10 : 2'b11;
            Flush = equal;
                state = 4'd4;
            end

            Slti: begin
                RegWrite = 1'b1;
                ALUSrc = 1'b1;
                RegDst = 2'b00;
                ALUop = 2'b11;
                MemToReg = 2'b00;
                state = 4'd9;
            end
        endcase
    end

endmodule
```

Instruction Format

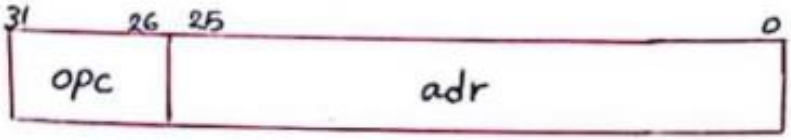
RT (add, sub, and, or, slt)



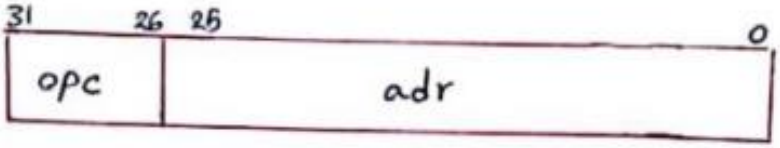
addi



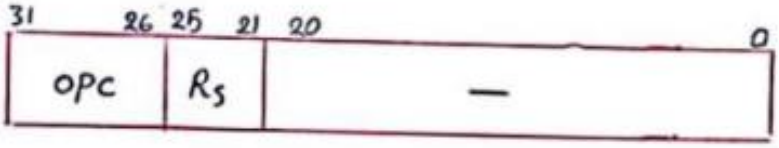
j



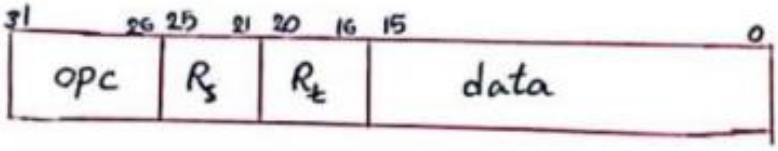
jal



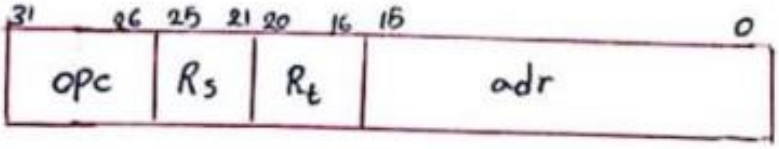
jr



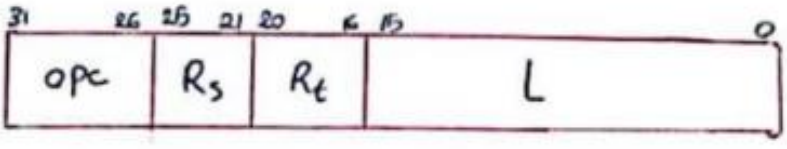
slti



lw / sw



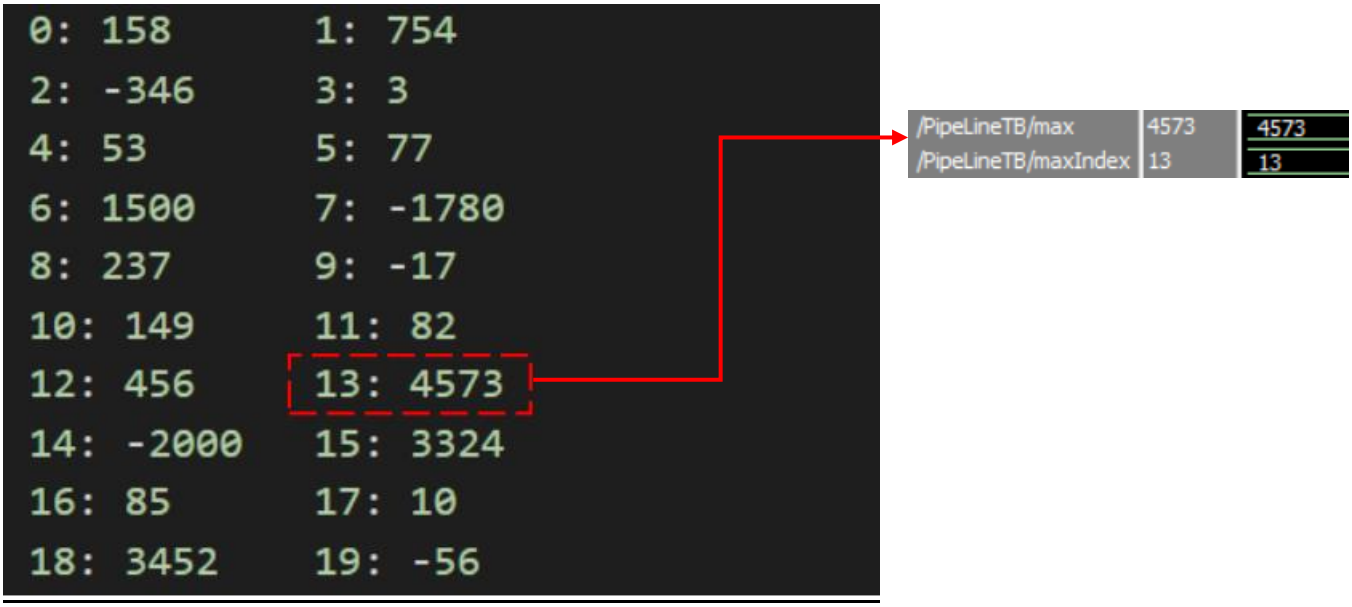
beq



Assembly Instruction

```
          LW R1, 1000(R0)
          ADDI R2, R1, 0
          ADDI R3, R0, 1
          ADDI R8, R0, 0
          ADDI R4, R0, 20
          NOP
          NOP
          NOP
loop :    BEQ R3, R4, afterLoop
          ADD R5, R0, 0
          ADD R5, R5, R5
          ADD R5, R5, R5
          LW R6, 1000(R5)
          SLT R7, R2, R6
          NOP
          NOP
          NOP
          BEQ R7, R0, endLoop
          ADD R2, R6, R0
          ADD R8, R3, R0
endLoop : ADDI R3, R3, 1
          NOP
          NOP
          NOP
          J loop
afterLoop :
          SW R2, 2000(R0)
          SW R8, 2004(R0)
```

20 Random Numbers



Waveforms