MIPS CPU Project Report

1. Instruction Analysis

00221820	add: R3, R1, R2	
AC010000	sw: R1, 0(R0)	
8C240000	lw R4, 0(R1)	
10210001	beq R1, R1, +8	//Branch taken
00001820	add R3, R0, R0	//This instruction will be skipped because of branch taken
00411822	sub R3, R2, R1	

All the instruction was assigned at the "Instruction.tex"

2. Simulation

The register file was be initially by assigned

R0 = 0;

R1 = 8;

R2 = 20

The instruction memory was be initially assigned

Mem[0] = 5;

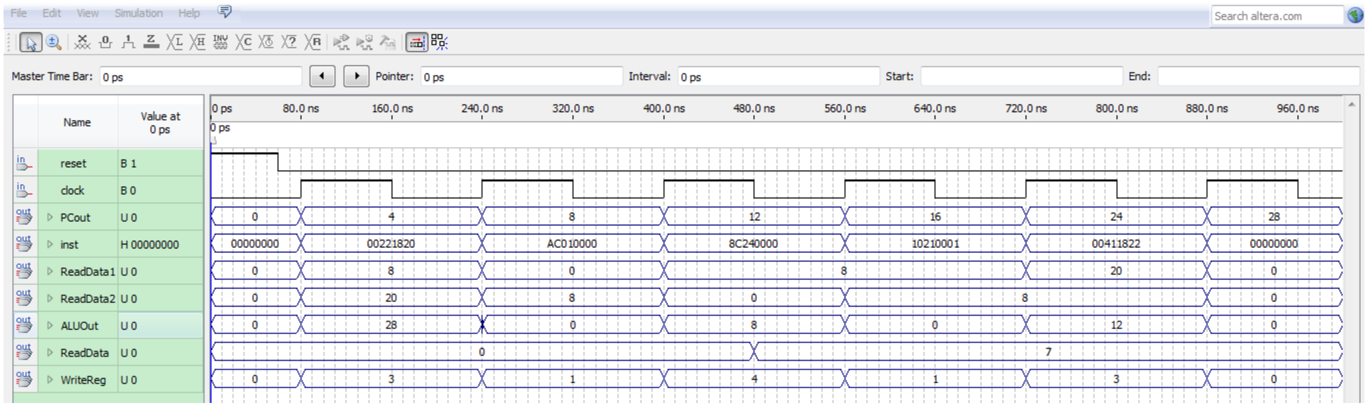
Mem[1] = 6;

Mem[2] = 7;

So we could conclude all the signals as follow:

clock	1	2	3	4	5
PCout	4	8	12	16	24
inst	00221820	AC010000	8C240000	10210001	00411822
ReadData1	8	0	8	8	20
ReadData2	20	8	0	8	8
ALUout	28	0	8	0	12
ReadData	-	-	7	-	-
WriteReg	R3	R1	R4	R1	R3

Note the 3rd clock cycle is LW instruction, so the ReadData from DataMemory was 7, ALUout = 8 means the word address was Mem[2]. And all of the other computing results were right. The waveform is as follow:



3. Source Design Code

3.1 PC

```
module PC(clock, reset, PCin, PCout);

 input clock, reset;
 input [31:0] PCin;

 output reg [31:0] PCout;

 always @(posedge clock) begin
        if (reset == 1)
                PCout <= 0;
        else
                PCout <= PCin + 4;
 end
endmodule
```

3.2 Instruction Memory

```
module InstMem(clock, address, inst);

 input clock;
 input [31:0] address;

 output reg [31:0]     inst;

 reg [31:0] Mem [0:127];

 initial begin
        $readmemh("Instruction.txt", Mem, 0, 5);
 end

 always @( posedge clock) begin
        inst <= Mem[address[31:2]];
 end
endmodule
```

3.3 Main Control

```
module MainControl(
 input [5:0] Opcode,

 output reg RegDst, RegWrite, ALUSrc,
 output reg MemtoReg, MemRead, MemWrite,
 output reg Branch,
	output reg [1:0] ALUOp);

	always @(*) begin
		case(Opcode)
			0: begin
				RegDst		<= 1;
				ALUSrc		<= 0;
				MemtoReg		<= 0;
				RegWrite	<= 1;
				MemRead		<= 0;
				MemWrite		<= 0;
				Branch		<= 0;
				ALUOp			<= 2'b10;;
			end
			35: begin
				RegDst		<= 0;
				ALUSrc		<= 1;
				MemtoReg		<= 1;
				RegWrite	<= 1;
				MemRead		<= 1;
				MemWrite		<= 0;
				Branch		<= 0;
				ALUOp			<= 2'b00;;
			end
			43: begin
				RegDst		<= 0;
				ALUSrc		<= 1;
				MemtoReg		<= 0;
				RegWrite	<= 0;
				MemRead		<= 0;
				MemWrite		<= 1;
				Branch		<= 0;
				ALUOp			<= 2'b00;
			end
			4: begin
				RegDst		<= 0;
				ALUSrc		<= 0;
				MemtoReg		<= 0;
				RegWrite	<= 0;
				MemRead		<= 0;
				MemWrite		<= 0;
				Branch		<= 1;
				ALUOp			<= 2'b01;
			end
		endcase
	end
endmodule
```

3.4 Mux1 (For instMem to RegFile)

```
module Mux1(inst20_16, inst15_11, RegDst, WriteReg);

 input [20:16] inst20_16;
 input [15:11] inst15_11;
 input RegDst;

 output reg [4:0] WriteReg;

 always @ (RegDst, inst20_16, inst15_11) begin
        case(RegDst)
               0 : WriteReg <= inst20_16;
               1 : WriteReg <= inst15_11;
        endcase
 end
endmodule
```

3.5 ALU Control

```
 module ALUControl (ALUOp, FuncCode, ALUCtl);

 input [1:0] ALUOp;
 input [5:0] FuncCode;
 output reg [3:0] ALUCtl;

 always @(ALUOp, FuncCode) begin
 if(ALUOp == 0)
        ALUCtl <= 2;        //LW and SW use add
 else if(ALUOp == 1)
        ALUCtl <= 6;         // branch use subtract
 else
        case(FuncCode)
               32: ALUCtl <= 2; //add
               34: ALUCtl <= 6; //subtract
               36: ALUCtl <= 0; //and
               37: ALUCtl <= 1; //or
               39: ALUCtl <= 12; //nor
               42: ALUCtl <= 7; //slt
               default: ALUCtl <= 15; //should not happen
        endcase
 end
endmodule
```

3.6 Mux2

```
module Mux2 (ALUSrc, ReadData2, Extend32, ALU_B);

 input ALUSrc;
 input [31:0] ReadData2,Extend32;

 output reg [31:0] ALU_B;

 always @(ALUSrc, ReadData2, Extend32) begin
        case (ALUSrc)
               0: ALU_B <= ReadData2 ;
               1: ALU_B <= Extend32;
        endcase
 end
endmodule
```

3.7 Register File

```
 module RegFile(clock, RegWrite, ReadReg1, ReadReg2, WriteReg, WriteData, ReadData1, ReadData2);

 input clock;
 input RegWrite;

 input [4:0] ReadReg1, ReadReg2, WriteReg;
 input [31:0] WriteData;

 output [31:0] ReadData1, ReadData2;

 reg [31:0] reg_mem [0:31];
 initial begin
        reg_mem[0] <= 0;
        reg_mem[1] <= 8;
        reg_mem[2] <= 20;
 end
 assign ReadData1 = reg_mem[ReadReg1];
 assign ReadData2 = reg_mem[ReadReg2];

 always @(posedge clock) begin
        if (RegWrite == 1)
               reg_mem[WriteReg] = WriteData;
 end
endmodule
```

3.8 ALU

```
 module ALU (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);

 always @(ALUCtl, A, B) begin
        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);
               default: ALUOut <= 0;
        endcase
 end
endmodule
```

3.9 Sign Extend

```
 module SignExtend (inst15_0, Extend32);

 input [15:0] inst15_0;
 output reg [31:0] Extend32;

 always @(inst15_0) begin
        Extend32[31:0] <= inst15_0[15:0];
 end
endmodule
```

3.10 Shift Left 2

```
 module ShiftLeft2 (ShiftIn, ShiftOut);

 input [31:0] ShiftIn;
 output reg [31:0] ShiftOut;

 always @(ShiftIn) begin
        ShiftOut = ShiftIn << 2;
 end

endmodule
```

3.11 Add_ALU

```
 module Add_ALU(PCout, ShiftOut, Add_ALUOut);
 input [31:0] PCout;
 input [31:0] ShiftOut;

 output reg [31:0] Add_ALUOut;

 always @(*) begin
        Add_ALUOut <= PCout + ShiftOut;
 end
endmodule
```

3.12 Mux4 (For Branch)

```
 module Mux4 (PCout, Add_ALUOut, AndGateOut, PCin);

 input [31:0] PCout, Add_ALUOut;
 input AndGateOut;

 output reg [31:0] PCin;

 initial begin
        PCin <= 0;
 end

 always @(*) begin
        case (AndGateOut)
               0: PCin <= PCout ;
               1: PCin <= Add_ALUOut;
        endcase
 end
endmodule
```

3.13 Data Memory

```
module DataMemory (clock, address, MemWrite, MemRead, WriteData, ReadData);

 input clock;
 input [6:0] address;
 input MemWrite, MemRead;
 input [31:0] WriteData;

 output reg [31:0] ReadData;

 reg [31:0] Mem[0:127]; //32 bits memory with 128 entries

 initial begin
        Mem[0] = 5;
        Mem[1] = 6;
        Mem[2] = 7;
 end

 always @ (posedge clock) begin

        if (MemWrite == 1)
                Mem[address[6:2]] <= WriteData;
 end

 always @(negedge clock) begin
        if (MemRead == 1)
                ReadData <= Mem[address[6:2]];
 end
 endmodule
```

3.14 Mux3 (For ALU)

```
module Mux3 (ReadData, ALUOut, MemtoReg, WriteData_Reg);

 input [31:0] ReadData, ALUOut;
 input MemtoReg;

 output reg [31:0] WriteData_Reg;

 always @(*) begin
        case (MemtoReg)
                0: WriteData_Reg <= ALUOut ;
                1: WriteData_Reg <= ReadData;
        endcase
 end
endmodule
```

3.15 And Gate

```
module AndGate(Branch, Zero, AndGateOut);
 input Branch;
 input Zero;
 output reg AndGateOut;

 always @(*) begin
        AndGateOut <= Branch && Zero;
 end
endmodule
```

3.16 MipsCPU

```
module MipsCPU(clock, reset,
                    PCin,PCout,
                    inst,
                    RegDst, RegWrite, ALUSrc, MemtoReg, MemRead, MemWrite, Branch,
                    ALUOp,
                    WriteReg,
                    ReadData1, ReadData2,
                    Extend32,
                    ALU_B,
                    ShiftOut,
                    ALUCtl,
                    Zero,
                    ALUOut,
                    Add_ALUOut,
                    AndGateOut,
                    ReadData,
                    WriteData_Reg);

 input clock;
 input reset;

 //Connection of PC
 output wire [31:0] PCin, PCout;
 PC pc_0(
         //inputs
         .clock(clock),
         .reset(reset),
         .PCin(PCin),
         //outputs
         .PCout(PCout)
 );

 //Connection of InstMem
 output wire [31:0] inst;
 InstMem instmem_0(
         //inputs
         .clock(clock),
         .address(PCin),
         //outputs
         .inst(inst)
 );

 //Connection of MainControl
 output wire RegDst, RegWrite, ALUSrc, MemtoReg, MemRead, MemWrite, Branch;
 output wire [1:0] ALUOp;
 MainControl main_control_0(
         //inputs
         .Opcode(inst[31:26]),
         //outputs
         .RegDst(RegDst),
         .RegWrite(RegWrite),
         .ALUSrc(ALUSrc),
         .MemtoReg(MemtoReg),
         .MemRead(MemRead),
         .MemWrite(MemWrite),
         .Branch(Branch),
         .ALUOp(ALUOp)
 );

 //Connection of the Mux between InstMem and RegisterFile
 output wire [4:0]    WriteReg;
 Mux1 mux1_0(
         //inputs
         .inst20_16(inst[20:16]),
         .inst15_11(inst[15:11]),
         .RegDst(RegDst),
         //outputs
         .WriteReg(WriteReg)
 );

 //Connection of RegFile
 output wire [31:0] ReadData1, ReadData2;
 RegFile regfile_0(
         //inputs
         .clock(clock),
         .ReadReg1(inst[25:21]),
         //**********************************************************************
         .ReadReg2(inst[20:16]),
         .RegWrite(RegWrite),
         .WriteReg(WriteReg),
         .WriteData(WriteData_Reg),
         //outputs
         .ReadData1(ReadData1),
         .ReadData2(ReadData2)
 );

 //Connection of SignExtend
 output wire [31:0] Extend32;
 SignExtend sign_extend_0(
         //inputs
         .inst15_0(inst[15:0]),
 		//outputs
 		.Extend32(Extend32)
 	);

 	//Connection of Mux2
 	output wire [31:0] ALU_B;
 	Mux2 mux2_0(
 		//inputs
 		.ALUSrc(ALUSrc),
 		.ReadData2(ReadData2),
 		.Extend32(Extend32),
 		//outputs
 		.ALU_B(ALU_B)
 	);

 	//Connection of ShiftLeft2
 	output wire [31:0] ShiftOut;
 	ShiftLeft2 shift_left2_0(
 		//inputs
 		.ShiftIn(Extend32),
 		//outputs
 		.ShiftOut(ShiftOut)
 	);

 	//Connection of ALUControl
 	output wire [3:0] ALUCtl;
 	ALUControl alu_control_0(
 		//inputs
 		.ALUOp(ALUOp),
 		.FuncCode(inst[5:0]),
 		//outputs
 		.ALUCtl(ALUCtl)
 	);

 	//Connection of ALU
 	output wire Zero;
 	output wire [31:0] ALUOut;
 	ALU alu_0(
 		//inputs
 		.A(ReadData1),
 		.B(ALU_B),
 		.ALUCtl(ALUCtl),
 		//outputs
 		.ALUOut(ALUOut),
 		.Zero(Zero)
 	);
```

```
//Connection of Add_ALU
output wire [31:0] Add_ALUOut;
Add_ALU add_alu_0(
	//inputs
	.PCout(PCout),
	.ShiftOut(ShiftOut),
	//outputs
	.Add_ALUOut(Add_ALUOut)
);

//Connection of AndGate
output wire AndGateOut;
AndGate and_gate_0(
	//inputs
	.Branch(Branch),
	.Zero(Zero),
	//outputs
	.AndGateOut(AndGateOut)
);

//Connection of Mux4
Mux4 mux4_0(
	//inputs
	.PCout(PCout),
	.Add_ALUOut(Add_ALUOut),
	.AndGateOut(AndGateOut),
	//outputs
	.PCin(PCin)
);

//Connection of DataMemory
output wire [31:0] ReadData;
DataMemory    data_memory_0(
	//inputs
	.clock(clock),
	.address(ALUOut),
	.MemWrite(MemWrite),
	.MemRead(MemRead),
	.WriteData(ReadData2),
	//outputs
	.ReadData(ReadData)
);

//Connection of Mux3
output wire[31:0] WriteData_Reg;
Mux3 mu3_0(
//inputs
```

```
 .ReadData(ReadData),
 .ALUOut(ALUOut),
 //outputs
 .WriteData_Reg(WriteData_Reg)
 );
endmodule
```