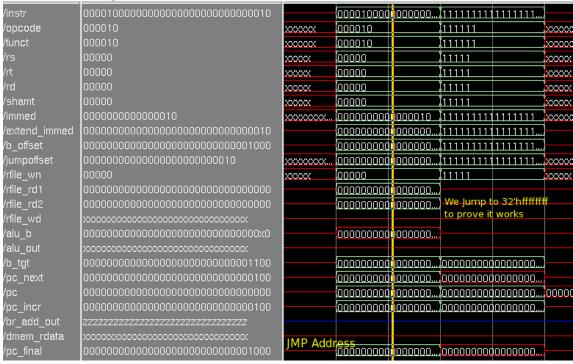
Lab Report

In this lab, we modify a MIPS single-cycle processor's datapath and control signals to handle the JUMP, ADDI, BEQ, and BNE instructions.

1. Waveform diagram of the JUMP instruction.



2. Waveform diagram of the ADDI instruction.



3. Waveform diagram of the BEQ instruction.

JT/instr	000100000110010000000000000000010	10001100	100010000q	100100	(11111111111	1111111	
JT/opcode	000100	100011	000100		111111		∞
JT/funct	000010	001000	000010		111111		∞
JT/rs	00011	00000	00011		11111		∞
JT/rt	00100	00100			11111		XX
JT/rd	00000	00000			11111		XX
JT/sh a mt	00000	00000			11111		xx
JT/immed	000000000000010	00000000	10000000000	000010	111111111	1111111	xx
JT/extend_immed	000000000000000000000000000000000000000	00000000	10000000000	000000	111111111	1111111	}
JT/b_offset	000000000000000000000000000000000000000	00000000	10000000000	000000	111111111	1111111	}—
JT/jumpoffset	00011001000000000000000010	00000001	000110010	000000	111111111	1111111	xx
JT/rfile_wn	00x00	00100	00x00		11111		XX
JT/rfile_rd1	000000000000000000000000000000000000000	00000000	10000000000	000000	\vdash		\vdash
JT/rfile_rd2	000000000000000000000000000000000000000	_	0000000000	000000	\vdash		\vdash
JT/rfile_wd	xxxxxxxxxxxx	00000000	}				
JT/b_tgt	00000000000000000000000000010100	00000000	10000000000	000000	0000000000	0000000	}—
JT/pc_next	00000000000000000000000000010100	00000000	100000000000	000000	000000000	0000000	
JT/pc	00000000000000000000000000001000	00000000	10000000000	000000	0000000000	0000000	00
JT/pc_incr	0000000000000000000000000001100	00000000	10000000000	000000	0000000000	0000000	}—
JT/pc_final	000000000000000000000000000010100	00000000	1000000000	000000	000000000	0000000	
JT/ALU/ctl	110	010	110		XXX		
JT/ALU/a	000000000000000000000000000000000000000	00000000	10000000000	000000	h to 30	or cecece	\vdash
JT/ALU/b	000000000000000000000000000000000000000	00000000	100000000000	000000	Jump to 32 to show it	- CONTINUE	
JT/ALU/result	000000000000000000000000000000000000000	00000000	10000000000	0000000	branching.	-	
JT/ALU/zero	1 A and B are the same so zero is set.						

4. Waveform diagram of the BNE instruction.

T/instr	000101000110010000000000000000010	P010000 1000101001100100 11111111111111
		0010000 0001010001100100 111111111
Γ/opcode 	000101	001000 000101 1111111
T/funct	000010	000110 000010 1111111
Γ/rs	00011	00011 11111
Γ/rt	00100	00011 00100 111111
T/rd	00000	00000 111111
T/sh a mt	00000	00000 111111
T/immed	000000000000010	0000000
T/extend_immed	000000000000000000000000000000000000000	0000000
T/b_offset	00000000000000000000000000001000	0000000
T/jumpoffset	00011001000000000000000010	0001100 0001100 0000000011111111
T/rfile_wn	00x00	00011 00x00 111111
T/rfile_rd1	000000000000000000000000000000000000000	0000000 (0000000 000000000)
T/rfile_rd2	000000000000000000000000000000000000000	0000000 1000000000000000
T/rfile_wd	xxxxxxxxxxxx	0000000
T/b_tgt	00000000000000000000000000011000	000000
T/pc_next	00000000000000000000000000011000	0000000
Г/рс	00000000000000000000000000001100	0000000
T/pc_incr	00000000000000000000000000010000	000000
T/pc_final	00000000000000000000000000011000	0000000
T/ALU/ctl	110	010 110 xxx
T/ALU/a	000000000000000000000000000000000000000	0000000 1000000000000000
T/ALU/b	000000000000000000000000000000000000000	0000000
T/ALU/result	00000000000000000000000000000110	0000000
Γ/ALU/zero	0 Result shows zero (not equal)	

Assembly Program

```
LW r3, 8(r0)

LW r4, 8(r0)

ADDI r4, r4, 6

BEQ r3, r4, DONE; LOOP

ADDI r3, r3, 1

JUMP LOOP

ADDI r3, r3, 1; DONE

BNE r3, r4, FINAL

JUNK-INSTRUCTION:

FINAL:
```

Waveform diagram of the test assembly program. The Signals have been truncated because they make the image too big. You can look at the previous figure to see the names of the signals.

T/instr	[xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx		(100011000	0000011		0000100	100100000	0000100
T/opcode	xxxxxx	xxxxxx	100011	14/				וחח
T/funct	Ixxxxx	xxxxxx	001000	VV	LW		1000100	
Γ/rs	xxxx	xxxxx	00000				00100	
Γ/rt	lxxxx	xxxxx	00011		.00100			
T/rd	xxxxx	xxxxx	00000					
T/sh a mt	Ixxxx	xxxxx	00000					
T/immed	Ixxxxxxxxxxx	xxxxxxxxx	000000000	0001000			1000000000	0000100
T/extend imm					000000000	001000		
T/b offset	xxxxxxxxxxxxxxxxxxxxxxxxxx00				00000000		1000000000	
_ T/jumpoffset	l:::::::::::::::::::::::::::::::::::::	xxxxxxxxx	000000000		.0000000010		.001000010	
T/rfile wn	xxxxx	xxxxx	00011		.00100			
 T/rfile_rd1	l:::::::::::::::::::::::::::::::::::::			000000000	000000000	000000	1000000000	00000000
T/rfile rd2	l:::::::::::::::::::::::::::::::::::::						000000000	
T/rfile wd			(000000000	000000000	000000000	000010	.000000000	
T/alu b					00000000			
T/alu out	1,000,000,000,000,000,000,000,000,000				000000000			
T/b_tgt					.000000000			
T/pc_next					.000000000			
Г/pc	1,000,000,000,000,000,000,000,000,000				.000000000			innnnnnn
T/pc incr							000000000	
T/br add out	72272777777777							
T/dmem rdata			(000000000	000000000	000000000	000010	,	
T/pc_final							1 1000000000	0000000



000100000	1100100	001000	00011000°	11 0000 <u>1</u> 1	00000000	00000011	0000011001	00001000		11 <mark>.</mark> 000010	1000000000	00000100	0000110010
000100	BEQ	001000	ADDI	00001	O IMP	00010	00 BEQ	001000	ADDI	000010	IMP	000100	BEQ
000010		000001		00001	1	(0000	10	000001		000011		000010	
00011				(00000		0001				(00000		00011	
00100		00011		(00000		00100		00011		(00000		00100	
10000												В	ranch is
00000												ta	aken
000000000	0000010	1000000	000000000	00000	00000000	011 (0000)	2000000000	10 000000	1000000000	000000	1000000001	1 000000	0000000001
000000000	00000000	1000000	000000000	00000	00000000	000 00001	2000000000	00000000	1000000000	000000,	1000000000	000000	000000000
000000000	00000000	1000000	000000000	00000	00000000	000 00001	2000000000	00000000	1000000000	000000,00	1000000000	000000	1000000000
000110010	00000000	000110	001100000	00000,00	00000000	0000001	100100000	00000110	000 100000	000000,	1000000000	00000110	0010000000
00x00		00011		(00000		00x00		00011		(00000		00x00	
00000000	100000000	0000000	00000011	(00000	00000000	000100001	0000000000	100000000000000000000000000000000000000	000000100	(000000	1000000000	00000000	000000000
00000000	00000000	1000000	00000000					00000000					
			000000000						000000000				
000000000	00000000	1000000	000000000	00000	00000000	000 00000	0000000000	000000	000000000	000000	000000000	000000	000000000
	11111111							11000000					1111111111
					00000000			00			000000000		
								00000000					
								00000000					
								000000					
								<u> </u>				11111111	
	1000 AD		000010	MP	1000100	BEQ	001000 /	ADDI	000101		1111111		XXXXXXX
	0001		000011		1000010		1000001				1111111		XXXXXX
<u> </u>			00000			ranch ta					11111		XXXXX
000			00000		100100JU	mps to	00011		00100		111111		XXXXX
000					A	DDi			Branch ta		11111		XXXXX
000									Jumps to	32'hffffffff	11111		XXXXX
<u>000</u>	000000000	00001	000000000	00000011	10000000	<u>000000010</u>	100000000	00000001				11111111	XXXXXXXX
000	2000000000	00000}	00000000	00000000	10000000	<u>oqoooooo</u>	00000000	<u> </u>	000000000	000001	111111111	11111111	
								<u> </u>			111111111		
000	011000110	00000	000000000	00000000	0001100	100000000	00011000	11000000	000110010	00000000	11111111	11111111	xxxxxxx
000	011	X	00000		00x00		00011		00x00		11111		XXXXX
000	0000000000	00000 🕻	000000000	00000000	0000000	00000000	οφοσοσσσο	Ф00110	000000000	00000000			
000	000000000	00000	00000000	00000000	10000000	00000000	οφοσοσσσοι	000110	000000000	00000000			
000	000000000	00000					00000000	00000000					
							laggaggg	0000000	00000000	0000000			
<u>(1</u> 01	000000000	00000	<u>00000000</u> 01	<u>,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,</u>	<u>,00000000</u>		···		<u>(r-r-r-r-r-r-r-r-r</u>	7-7-7-7-7-3118			
	000000000 000000000		000000000	00000000				000000000					
000	000000000	00000			0000000	00000000	00000000		000000000	00000000)	000000000	100000000	}
000 000	000000000	00000 00000	00000000	00000000	0000000	000000000 000000000	. 000000000 . 00000000	00000000	000000000 000000000)00000000))00000000			
000 000	000000000 000000000 000000000	00000 00000 00000	000000000 000000000	00000000	0000000 0000000 0000000	000000000 000000000 000000000		00000000 000000000	000000000 000000000 000000000)00000000)00000000)00000000	000000000	00000000	

Modified ROM32 Code

```
module rom32(address, data_out);
input [31:0] address;
output [31:0] data_out;
reg [31:0] data_out;

parameter BASE_ADDRESS = 25'd0; // address that applies to this memory
wire [4:0] mem_offset;
```

```
wire address_select;
assign mem_offset = address[6:2]; // drop 2 LSBs to get word offset
assign address_select = (address[31:7] == BASE_ADDRESS); // address decoding
always @(address_select or mem_offset)
 if ((address \% 4) != 0) $display($time, " rom32 error: unaligned address \%d", address);
 if (address_select == 1)
 begin
   case (mem_offset)
     /*
      // JUMP
      5'd0 : data_out = { 6'd2, 26'd2};
                                                      // jmp
       5'd1 : data_out = { 6'd35, 5'd0, 5'd3, 16'd8 }; // lw $3, 8($0) r3=2
       5'd2 : data_out = { 8'hff, 8'hff, 8'hff, 8'hff }; // lw $4, 20($0) r4=5
     /*
      // ADDT
       5'd0 : data_out = { 6'd35, 5'd0, 5'd3, 16'd8 }; // lw $3, 8($0) r3=2
       5'd1 : data_out = { 6'd8, 5'd3, 5'd3, 16'd6 }; // addi $3, $3, 2
     /*
      // BEQ
       5'd0 : data_out = { 6'd35, 5'd0, 5'd3, 16'd8 }; // lw $3, 8($0) r3=2
       5'd1 : data_out = { 6'd35, 5'd0, 5'd4, 16'd8 }; // lw $4, 8($0) r4=2
       5'd2 : data_out = { 6'd4, 5'd3, 5'd4, 16'd2 }; // beq r3, r4
       5'd3 : data_out = { 6'd35, 5'd0, 5'd3, 16'd8 }; //
       5'd4 : data out = { 8'hee, 8'hee, 8'hee, 8'hee }: //
       5'd5 : data_out = { 8'hff, 8'hff, 8'hff, 8'hff }; // Target of beq
      // BNE
       5'd0 : data_out = { 6'd35, 5'd0, 5'd3, 16'd8}; // lw $3, 8($0) r3=2
       5'd2 : data_out = { 6'd8, 5'd3, 5'd3, 16'd6 };
                                                     // addi $3, $3, 6 r3=8
       5'd3 : data_out = { 6'd5, 5'd3, 5'd4, 16'd2};
                                                     // bne r3, r4
       5'd4 : data_out = { 6'd35, 5'd0, 5'd3, 16'd8 }; //
       5'd5 : data_out = { 8'hee, 8'hee, 8'hee, 8'hee }; //
       5'd6 : data_out = { 8'hff, 8'hff, 8'hff, 8'hff }; // Target of bne
       // Test Program
       5'd0 : data_out = { 6'd35, 5'd0, 5'd3, 16'd8}; // lw $3, 8($0) (r3=2)
       5'd1 : data_out = { 6'd35, 5'd0, 5'd4, 16'd8}; // lw $4, 8($0) (r4=2)
       5'd2 : data_out = { 6'd8, 5'd4, 5'd4, 16'd4 }; // addi $4, $4, 6 (r4=6)
       5'd3 : data_out = { 6'd4, 5'd3, 5'd4, 16'd2 };  // beq $3, $4, DONE ; LOOP
                                                     // addi $3, $3, 1
       5'd4 : data_out = { 6'd8, 5'd3, 5'd3, 16'd1 };
       5'd5 : data_out = { 6'd2, 26'd3 };
                                                      // JUMP LOOP
       5'd6 : data_out = { 6'd8, 5'd3, 5'd3, 16'd1 }; // addi $3, $3, 1 ; DONE
       5'd7 : data_out = { 6'd5, 5'd3, 5'd4, 16'd1 }; // bne $3, $4, FINAL
       5'd8 : data_out = { 8'hee, 8'hee, 8'hee, 8'hee }; //
       5'd9 : data_out = { 8'hff, 8'hff, 8'hff, 8'hff }; // FINAL:
       // add more cases here as desired
       default data_out = 32'hxxxx;
   endcase
   display(time, " reading data: rom32[h] => h", address, data_out);
end
```

endmodule

Modified Datapath Code

```
module mips_single(clk, reset);
   input clk, reset;
   // instruction bus
   wire [31:0] instr;
   // break out important fields from instruction
   wire [5:0] opcode, funct;
   wire [4:0] rs, rt, rd, shamt;
   wire [15:0] immed;
   wire [31:0] extend_immed, b_offset;
   wire [25:0] jumpoffset;
   assign opcode = instr[31:26];
   assign rs = instr[25:21];
   assign rt = instr[20:16];
   assign rd = instr[15:11];
   assign shamt = instr[10:6];
   assign funct = instr[5:0];
   assign immed = instr[15:0];
   assign jumpoffset = instr[25:0];
   // sign-extender
   assign extend_immed = { {16{immed[15]}}, immed };
   // branch offset shifter
   assign b_offset = extend_immed << 2;</pre>
   // datapath signals
   wire [4:0] rfile wn:
   wire [31:0] rfile_rd1, rfile_rd2, rfile_wd, alu_b, alu_out, b_tgt, pc_next,
               pc, pc_incr, br_add_out, dmem_rdata, pc_final;
   // control signals
   wire RegWrite, Branch, PCSrc, RegDst, MemtoReg, MemRead, MemWrite, ALUSrc, Zero, Jump, sub_zero;
   wire [1:0] ALUOp;
   wire [2:0] Operation;
   // module instantiations
   reg32 PC(clk, reset, pc_final, pc);
   add32 PCADD(pc, 32'd4, pc_incr);
   add32 BRADD(pc_incr, b_offset, b_tgt);
   reg_file RFILE(clk, RegWrite, rs, rt, rfile_wn, rfile_rd1, rfile_rd2, rfile_wd);
   alu ALU(Operation, rfile_rd1, alu_b, alu_out, Zero);
   rom32 IMEM(pc, instr);
   mem32 DMEM(clk, MemRead, MemWrite, alu_out, rfile_rd2, dmem_rdata);
               // output in0 in1
   and BR_AND(PCSrc, Branch, sub_zero);
```

```
mux2 #(5) RFMUX(RegDst, rt, rd, rfile_wn);
                   mux2 #(32) PCMUX(PCSrc, pc_incr, b_tgt, pc_next);
                   mux2 #(32) ALUMUX(ALUSrc, rfile_rd2, extend_immed, alu_b);
                   mux2 #(32) WRMUX(MemtoReg, alu_out, dmem_rdata, rfile_wd);
                   \ensuremath{//} Extend MIPS datapath to handle jump instruction by adding a mux
                   \ensuremath{//} to choose between branch target and jump address.
                   mux2 #(32) JUMPMUX(Jump, pc_next, {pc_incr[31:28], jumpoffset, 2'b00}, pc_final);
                   // BEQ 00010(0) Zero
                   // BNE 00010(1) NOT Zero
                   mux2 #(1) BRMUX(opcode[0], Zero, ~Zero, sub_zero);
                   control_single CTL(.opcode(opcode), .RegDst(RegDst), .ALUSrc(ALUSrc), .MemtoReg(MemtoReg),
                                                                                                                     . \\ \texttt{RegWrite} (\texttt{RegWrite}) \text{, } . \\ \texttt{MemRead} (\texttt{MemRead}) \text{, } . \\ \texttt{MemWrite} (\texttt{MemWrite}) \text{, } . \\ \texttt{Branch} (\texttt{Branch}) \text{, } \\ \texttt{Stanch} (\texttt{Stanch}) \text{, } \\ \texttt{Stanch} (\texttt{S
                                                                                                                     .ALUOp(ALUOp), .Jump(Jump));
                   alu_ctl ALUCTL(ALUOp, funct, Operation);
endmodule
```

Control Signals Code

```
module control_single(opcode, RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, ALUOp, Jump);
   input [5:0] opcode;
   output RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, Jump;
   output [1:0] ALUOp;
   reg RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, Jump;
   reg [1:0] ALUOp;
   parameter R_FORMAT = 6'd0;
   parameter LW = 6'd35;
   parameter SW = 6'd43;
   parameter BEQ = 6'd4;
   parameter JMP = 6'd2;
   parameter ADDI = 6'd8;
   parameter BNE = 6'd5;
   always @(opcode)
   begin
       case (opcode)
         R_FORMAT :
             RegDst=1'b1; ALUSrc=1'b0; MemtoReg=1'b0; RegWrite=1'b1; MemRead=1'b0;
             MemWrite=1'b0; Branch=1'b0; ALUOp = 2'b10; Jump=0;
         end
         LW :
             RegDst=1'b0; ALUSrc=1'b1; MemtoReg=1'b1; RegWrite=1'b1; MemRead=1'b1; MemWrite=1'b0; Branch=1'b0; ALUOp = 2'b00; Jump=0; end
         SW :
         begin
             RegDst=1'bx; ALUSrc=1'b1; MemtoReg=1'bx; RegWrite=1'b0; MemRead=1'b0;
             MemWrite=1'b1; Branch=1'b0; ALUOp = 2'b00; Jump=0;
         end
         BEQ :
         begin
```

```
// RegDst: x, not writing to register
              // ALUSrc: 1, extend_immed
             // Memtoreg: x, not writing to register
              // RegWrite: 0, not writing to register
              // MemRead: 0, not reading from memory
              // MemWrite: 0, not writing to memory
              // Branch: 1,
              // ALUOp: 01, subtract to compare
             RegDst=1'bx; ALUSrc=1'b0; MemtoReg=1'bx; RegWrite=1'b0; MemRead=1'b0;
             MemWrite=1'b0; Branch=1'b1; ALUOp=2'b01; Jump=0;
          \quad \text{end} \quad
          JMP :
          begin
             // RegDst: x, not writing to register
              // ALUSrc: 1, extend_immed
             // Memtoreg: x, not writing to register
             // RegWrite: 0, not writing to register
              // MemRead: 0, not reading from memory
              // Branch: 1, set new PC
              // ALUOp: xx
              // Jump: 1, to jump address, not branch target
             $display("BEGIN JUMP PREP");
              RegDst=1'bx; ALUSrc=1'bx; MemtoReg=1'bx; RegWrite=1'b0; MemRead=1'b0;
              MemWrite=1'b0; Branch=1'b1; ALUOp = 2'bxx; Jump=1'b1;
          end
          ADDT :
          begin
             // RegDst: 0, write to rt
             // ALUSrc: 1, read from second register port
             // Memtoreg: 0, ALU result to register
              // RegWrite: 1, writing to register
              // MemRead: 0, not reading from memory
              // Branch: 0, not branching
              // ALUOp: 00, add two registers
             RegDst=1'b0; ALUSrc=1'b1; MemtoReg=1'b0; RegWrite=1'b1; MemRead=1'b0;
             MemWrite=1'b0; Branch=1'b0; ALUOp=2'b00; Jump=0;
          end
         BNE :
          begin
             // RegDst: x, not writing to register
             // ALUSrc: 1, extend_immed
             // Memtoreg: x, not writing to register
             // RegWrite: 0, not writing to register
              // MemRead: 0, not reading from memory
              // MemWrite: 0, not writing to memory
              // Branch: 1.
             // ALUOp: 01, subtract to compare
             RegDst=1'bx; ALUSrc=1'b0; MemtoReg=1'bx; RegWrite=1'b0; MemRead=1'b0;
             MemWrite=1'b0; Branch=1'b1; ALUOp=2'b01; Jump=0;
          end
          default
         begin
             $display("control_single unimplemented opcode %d", opcode);
             RegDst=1'bx; ALUSrc=1'bx; MemtoReg=1'bx; RegWrite=1'bx; MemRead=1'bx;
             MemWrite=1'bx; Branch=1'bx; ALUOp = 3'bxxx; Jump=0;
          end
   end
endmodule
```