

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

Register	Value
/instr	00001000000000000000000000000000000010
/opcode	000010
/funct	000010
/rs	00000
/rt	00000
/rd	00000
/shamt	00000
/immed	000000000000000010
/extend_immed	000000000000000000000000000000000010
/b_offset	000000000000000000000000000000001000
/jumpoffset	00000000000000000000000000000010
/rfile_wn	00000
/rfile_rd1	000000000000000000000000000000000000
/rfile_rd2	000000000000000000000000000000000000
/rfile_wd	xx
/alu_b	00000000000000000000000000000000x0
/alu_out	xx
/b_tgt	0000000000000000000000000000000001100
/pc_next	0000000000000000000000000000000000100
/pc	000000000000000000000000000000000000
/pc_incr	000000000000000000000000000000000100
/br_add_out	zz
/dmem_rdata	xx
/pc_final	00000000000000000000000000000000001000

2. Waveform diagram of the ADDI instruction.

Immediate value is input to ALU

IT/instr	0001000000110010000000000000000010	100
----------	------------------------------------	-----

Jump to 32'hfffffff  
to show it's  
branching.

1. Watermark diagram of the DIVE instruction:

T/instr	00010100011001000000000000000010	0010000...	0001010001100100...	1111111111111111...	
T/opcode	000101	001000	000101	111111	x
T/funct	000010	000110	000010	111111	x
T/rs	00011	00011		11111	x
T/rt	00100	00011	00100	11111	x
T/rd	00000	00000		11111	x
T/shamt	00000	00000		11111	x
T/immed	000000000000000010	0000000...	0000000000000010	1111111111111111...	x
T/extend_immed	00000000000000000000000000000010	0000000...	0000000000000000...	1111111111111111...	
T/b_offset	0000000000000000000000000000001000	0000000...	0000000000000000...	1111111111111111...	
T/jumpoffset	000110010000000000000000000010	0001100...	0001100000000000...	1111111111111111...	x
T/rfile_wn	00x00	00011	00x00	11111	x
T/rfile_rd1	0000000000000000000000000000001000	0000000...	0000000000000000...		
T/rfile_rd2	000000000000000000000000000000010	0000000...	0000000000000000...		
T/rfile_wd	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	0000000...			
T/b_tgt	00000000000000000000000000000011000	0000000...	0000000000000000...	0000000000000000...	
T/pc_next	00000000000000000000000000000011000	0000000...	0000000000000000...	0000000000000000...	
T/pc	0000000000000000000000000000001100	0000000...	0000000000000000...	0000000000000000...	
T/pc_incr	00000000000000000000000000000010000	0000000...	0000000000000000...	0000000000000000...	
T/pc_final	00000000000000000000000000000011000	0000000...	0000000000000000...	0000000000000000...	
T/ALU/ctl	110	010	110	xxx	
T/ALU/a	0000000000000000000000000000001000	0000000...	0000000000000000...		
T/ALU/b	0000000000000000000000000000000010	0000000...	0000000000000000...	We jump to 32'hffffff	
T/ALU/result	0000000000000000000000000000000110	0000000...	0000000000000000...	to prove it works	
T/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.

0001000001100100...	0010000001100011...	0000100000000000...
000100 <b>BEQ</b>	001000 <b>ADDI</b>	000010 <b>JMP</b>
000010	000001	000011
00011		00000
	00011	00000
0000000000000010	0000000000000001	0000000000000011
0000000000000000...	0000000000000000...	0000000000000000...
0000000000000000...	0000000000000000...	0000000000000000...
0001100100000000...	0001100011000000...	0000000000000000...
<b>00x00</b>	00011	00000
000000000000000000000000000010		0000000000000000...
0000000000000000...	0000000000000000...	0000000000000000...
	0000000000000000...	
0000000000000000...	0000000000000000...	0000000000000000...
1111111111111111...	0000000000000000...	
0000000000000000...	0000000000000000...	0000000000000000...
0000000000000000...	0000000000000000...	0000000000000000...
0000000000000000...	0000000000000000...	0000000000000000...
0000000000000000...	0000000000000000...	0000000000000000...

00010000	1100100...	00100000	1100011...	00001000	0000000...	00010000	1100100...	00100000	1100011...	00001000	0000000...	00010000	1100100...	0
000100	BEQ	001000	ADDI	000010	JMP	000100	BEQ	001000	ADDI	000010	JMP	000100	BEQ	0
000010		000001		000011		000010		000001		000011		000010		0
00011				00000		00011				00000		00011		0
00100		00011		00000		00100		00011		00000		00100		0
00000														0
00000														0
00000000	00000010	00000000	00000001	00000000	00000011	00000000	00000010	00000000	00000001	00000000	00000011	00000000	00000010	0
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	0
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	0
000110010	00000000...	000110001	1000000...	00000000	00000000...	000110010	00000000...	000110001	1000000...	00000000	00000000...	000110010	00000000...	0
00x00		00011		00000		00x00		00011		00000		00x00		0
00000000	0000000000	0000000000	00011	00000000	00000000...	00000000	0000000000	0000000000	0000000000	00000000	00000000...	00000000	0000000000	0
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	0
		00000000	00000000...					00000000	00000000...					0
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	0
11111111	1111111...	00000000	00000000...			11111111	1111111...	00000000	00000000...			11111111	1111111...	0
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	0
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	0
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	0
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	0
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	0

Branch is  
taken

00100000	110011...	00001000	0000000...	00010000	1100100...	00100000	1100011...	00010100	1100100...	11111111	1111111...		
001000	ADDI	000010	JMP	000100	BEQ	001000	ADDI	000101	BNE	111111		xxxxxx	
000001		000011		000010		000001				111111		xxxxxx	
111		00000		00011	Branch taken.					11111		xxxxxx	
00011		00000		00100	Jumps to ADDI	00011		00100		11111		xxxxxx	
100									Branch taken.	11111		xxxxxx	
100									Jumps to 32'hffffff	11111		xxxxxx	
00000000	00000001	00000000	00000011	00000000	00000010	00000000	00000001			11111111	1111111...	xxxxxxx	
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	11111111	1111111...		
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	11111111	1111111...		
000110001	1000000...	00000000	00000000...	000110010	00000000...	000110001	1000000...	000110010	00000000...	11111111	1111111...	xxxxxxx	
00011		00000		00x00		00011		00x00		11111		xxxxxx	
00000000	00000000...	00000000	00000000...	00000000	0000000000	00000000	00000000...	00000000	00000000...				
00000000	00000000...	00000000	00000000...	00000000	0000000000	00000000	00000000...	00000000	00000000...				
00000000	00000000...					00000000	00000000...						
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...				
00000000	00000000...			00000000	00000000...	00000000	00000000...	00000000	00000000...				
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...		
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	
00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	00000000...	00000000	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)
begin
    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
            */

            /*
            // ADDI
            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'd1 : data_out = { 6'd35, 5'd0, 5'd4, 16'd8 }; // lw $4, 20($0)  r4=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
            5'd4 : data_out = { 6'd8, 5'd3, 5'd3, 16'd1 }; // addi $3, $3, 1
            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
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;

    // 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);

// Extend MIPS datapath to handle jump instruction by adding a mux
// 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),
                  .RegWrite(RegWrite), .MemRead(MemRead), .MemWrite(MemWrite), .Branch(Branch),
                  .ALUOp(ALUOp), .Jump(Jump));

alu_ctl1 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 :
                begin
                    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 :
                begin
                    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
            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;
    end
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
ADDI :
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

endcase
end
endmodule

```