Charlie Getzen

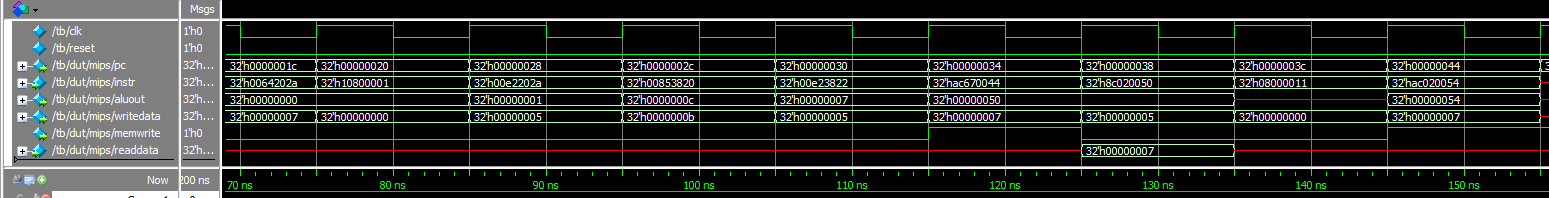
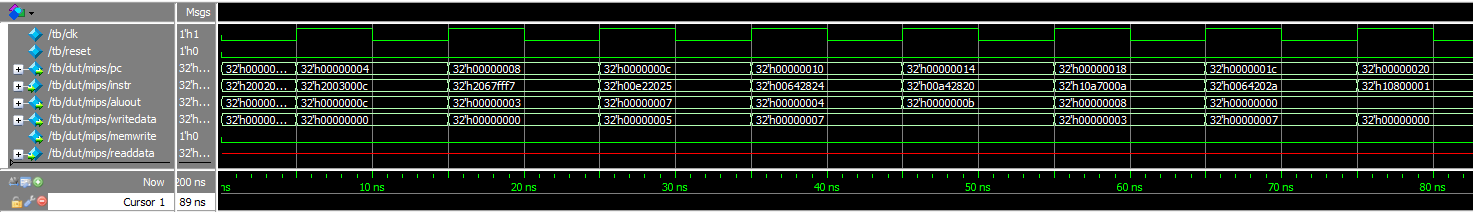
Oliver Thio

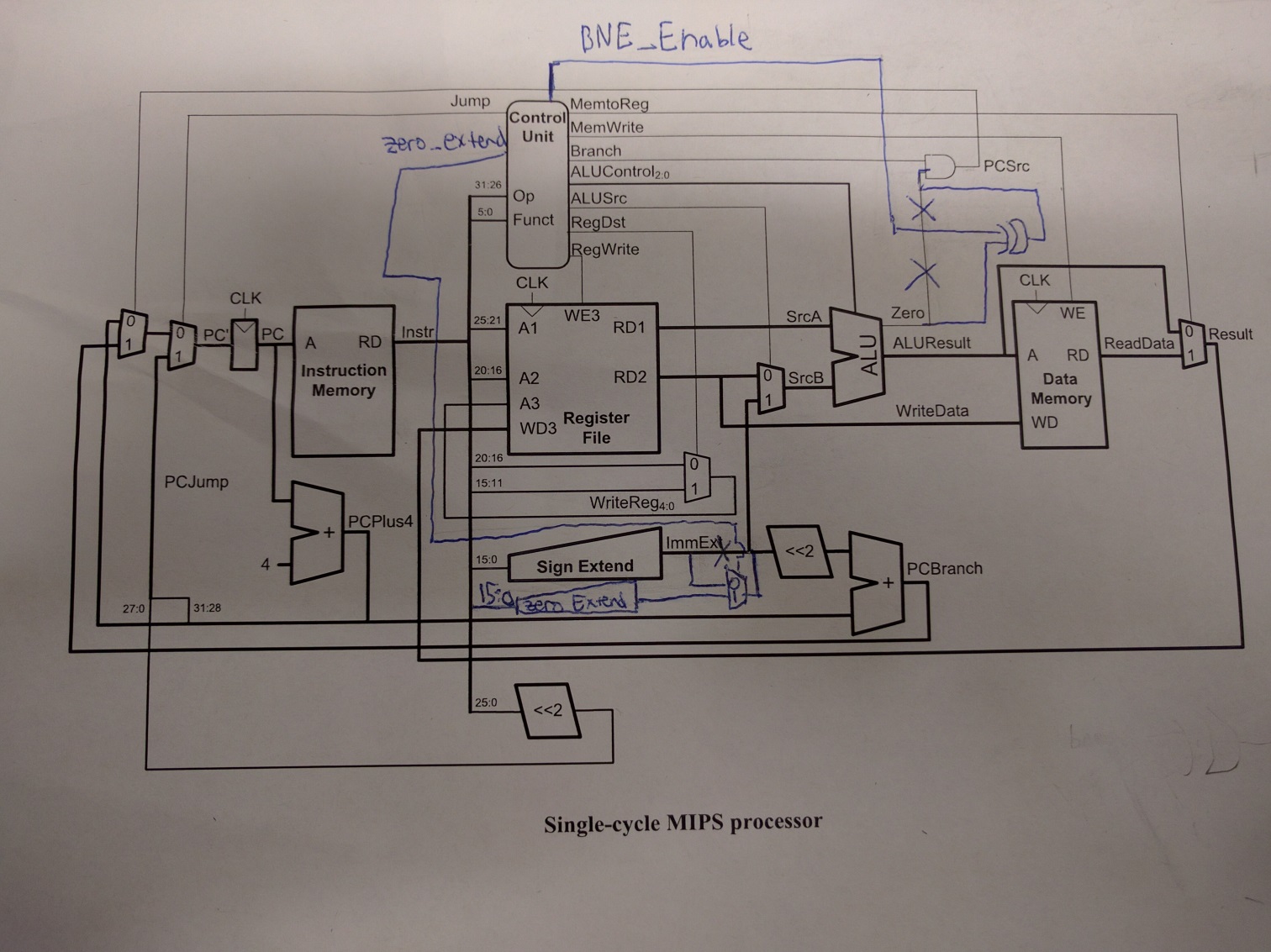
A) time taken: 10 hours

B)

The two values (PC = 14, and PC = 30) were skipped in execution. The values would have been:



C)  These two images are attached as wave1\_0 and wave1\_1.

D) 

File saved as modified-table.jpg.

E)

mips.v

// single-cycle MIPS processor

// instantiates a controller and a datapath module

module mips(input clk, reset,

output [31:0] pc,

input [31:0] instr,

output memwrite,

output [31:0] aluout, writedata,

input [31:0] readdata);

wire memtoreg, branch,

zero,

alusrc, regdst, regwrite, jump, bne\_enable, zero\_ext;

// bne\_enable distinguishes between bne and beq

// zero\_ext distinguishes between combinational logic and other immediates

wire [2:0] alucontrol;

controller c(instr[31:26], instr[5:0], zero,

memtoreg, memwrite,

alusrc, regdst, regwrite, jump,

branch, bne\_enable,

alucontrol);

datapath dp(clk, reset, memtoreg,

alusrc, regdst, regwrite, jump, branch, bne\_enable, zero\_ext,

// Necessary change because our controller dictates the datapath.

alucontrol,

zero, pc, instr,

aluout, writedata, readdata);

endmodule

module controller(input [5:0] op, funct,

input zero,

output reg memtoreg, memwrite,

output reg alusrc,

output reg regdst, regwrite,

output reg jump, branch, bne\_enable, zero\_ext ,

// Needed to communicate to datapath

output reg [2:0] alucontrol);

always @ \* begin

case (op) // Based on op-code, set controller values.

6'b000000 : begin // R-TYPE

branch <= 0;

memtoreg <= 0;

memwrite <= 0;

alusrc <= 0;

regdst <= 1;

regwrite <= 1;

jump <= 0;

case (funct) // Set the ALU to do the correct function

6'b100000: alucontrol <= 3'b010;

6'b100010: alucontrol <= 3'b110;

6'b100100: alucontrol <= 3'b000;

6'b100101: alucontrol <= 3'b001;

6'b101010: alucontrol <= 3'b111;

default: alucontrol <= 3'bxxx;

endcase

end

6'b100011 : begin // lw

zero\_ext <= 0; // zero\_ext is only enabled for combinational logic (ori, andi).

branch <= 0;

memtoreg <= 1;

memwrite <= 0;

alusrc <= 1;

regdst <= 0;

regwrite <= 1;

jump <= 0;

alucontrol <= 3'b010;

end

6'b101011 : begin // sw

zero\_ext <= 0;

branch <= 0;

memtoreg <= 1'bx;

memwrite <= 1;

alusrc <= 1;

regdst <= 1'bx;

regwrite <= 0;

jump <= 0;

alucontrol <= 3'b010;

end

6'b000100 : begin // beq

zero\_ext <= 0;

bne\_enable <= 0; // This distinguishes between beq and bne for our datapath

branch <= 1;

memtoreg <= 1'bx;

memwrite <= 0;

alusrc <= 0;

regdst <= 1'bx;

regwrite <= 0;

jump <= 0;

alucontrol <= 3'b110;

end

6'b001000 : begin // addi

zero\_ext <= 0;

branch <= 0;

memtoreg <= 0;

memwrite <= 0;

alusrc <= 1;

regdst <= 0;

regwrite <= 1;

jump <= 0;

alucontrol <= 3'b010;

end

6'b000010 : begin // j

branch <= 0;

memtoreg <= 1'bx;

memwrite <= 0;

alusrc <= 1'bx;

regdst <= 1'bx;

regwrite <= 0;

jump <= 1;

alucontrol <= 3'b010;

end

6'b000101 : begin // bne

zero\_ext <= 0;

bne\_enable <= 1;

branch <= 1;

memtoreg <= 1'bx;

memwrite <= 0;

alusrc <= 0;

regdst <= 1'bx;

regwrite <= 0;

jump <= 0;

alucontrol <= 3'b110;

end

6'b001101 : begin // ori

zero\_ext <= 1;

branch <= 0;

memtoreg <= 0;

memwrite <= 0;

alusrc <= 1;

regdst <= 0;

regwrite <= 1;

jump <= 0;

alucontrol <= 3'b001;

end

endcase

end

endmodule

module datapath(input clk, reset,

input memtoreg,

input alusrc, regdst,

input regwrite, jump, branch, bne\_enable, // Necessary to distinguish

// between bne and beq. Branch is no longer enough.

input zero\_ext, // A new control signal for ori logic

input [2:0] alucontrol,

output zero,

output reg [31:0] pc,

input [31:0] instr,

output [31:0] aluout, writedata,

input [31:0] readdata);

// \*\*PUT YOUR CODE HERE\*\*

reg [4:0] writereg;

reg [31:0] pcnext, pcnextbr, pcplus4, pcbranch;

reg [31:0] signimm, signimmsh;

reg [31:0] srca, srcb;

reg [31:0] result;

reg pcsrc, xor\_output; // pcsrc is no longer an input, because it does not come out of the

// control unit. Instead, we use combination logic to find the value of pcsrc.

reg [31:0] sign\_ext\_output, zero\_ext\_output; // Necessary for ori logic

initial pc = 32'b0;

//PC LOGIC

flopr pc\_reg(clk, reset, pcnext, pc);

adder pc\_add1(pc, 32'b100, pcplus4);

sl2 immsh(signimm, signimmsh);

adder pc\_add2(pcplus4, signimmsh, pcbranch);

xor xor\_bne(xor\_output, bne\_enable, zero); // Combinational logic is implemented to find pcsrc

and and\_pcsrc(pcsrc, branch, xor\_output);

mux2 pcbrmux(pcplus4, pcbranch, pcsrc, pcnextbr);

mux2 pcmux(pcnextbr, {pcplus4[31:28], instr[25:0], 2'b00}, jump, pcnext);

// Register file logic

regfile rf(clk, regwrite, instr[25:21], instr[20:16], writereg, result, srca, writedata);

fivebitmux2 wrmux(instr[20:16], instr[15:11], regdst, writereg);

mux2 resmux(aluout, readdata,memtoreg, result);

// Add logic for ori – see modified diagram

sign\_ext se(instr[15:0], sign\_ext\_output);

zero\_ext ze(instr[15:0], zero\_ext\_output);

mux2 extend\_mux(sign\_ext\_output, zero\_ext\_output, zero\_ext, signimm);

// ALU Logic

mux2 srcbmux(writedata, signimm, alusrc, srcb);

ALU alu(srca, srcb, alucontrol, aluout, zero);

endmodule

mipsmem.v

// External memories used by MIPS single-cycle processor

module dmem(input clk, we,

input [31:0] a, wd,

output [31:0] rd);

reg [31:0] RAM[63:0]; // stores our 64 words.

assign rd = RAM[a[31:2]]; // reads data that is word aligned

always@ (posedge clk) begin

if(we) RAM[a[31:2]] <= wd; // writes data to memory if enabled

end

endmodule

// Instruction memory (already implemented)

module imem(input [5:0] a,

output [31:0] rd);

reg [31:0] RAM[63:0];

initial

begin

$readmemh("memfile2.dat",RAM); // initialize memory with test program. Change this with memfile2.dat for the modified code

end

assign rd = RAM[a]; // word aligned

endmodule

datapath\_modules.v

// Modules created to help design the datapath

module regfile(input clk, we3,

input [4:0] ra1, ra2, wa3,

input [31:0] wd3,

output reg [31:0] rd1, rd2);

reg [31:0] RAM [31:0];

reg [31:0] index;

initial begin // initialize registers to 0

for (index=0; index < 32; index = index + 1)

RAM[index] = 32'b0;

end

always @ (posedge clk)

if (we3) RAM[wa3] <= wd3; // if enabled, write to register

assign rd1 = (ra1 != 0) ? RAM[ra1] : 0; // Assign read values

assign rd2 = (ra2 != 0) ? RAM[ra2] : 0;

endmodule

// D flip-flop

module flopr(input clk, reset,

input [31:0] d,

output reg [31:0] q);

always @ (posedge clk) begin

if (reset) q <= 0;

else q <= d;

end

endmodule

// Sign extend

module sign\_ext(input [15:0] a,

output [31:0] y);

assign y = {{16{a[15]}}, a };

endmodule

// Zero extend necessary for ori combinational logic

module zero\_ext(input [15:0] a,

output [31:0] y);

assign y = {16'b0, a};

endmodule

// Shift left 2

module sl2(input [31:0] a,

output [31:0] y);

assign y = {a[29:0], 2'b00};

endmodule

// 32-bit 2-input mux

module mux2(input [31:0] d0, d1,

input s,

output [31:0] y);

assign y = s ? d1 : d0;

endmodule

// 5-bit 2-input mux

module fivebitmux2(input [4:0] d0, d1,

input s,

output [4:0] y);

assign y = s ? d1 : d0;

endmodule

// 32-bit adder

module adder(input [31:0] a, b,

output [31:0] y);

assign y = a + b;

endmodule

testbench.v

module tb();

reg clk;

reg reset;

reg [31:0] writedata, dataadr;

reg memwrite;

// instantiate device to be tested

top dut (clk, reset);

// initialize test

initial reset <= 0;

// generate clock to sequence tests

always begin

clk <= 0;

# 5;

clk <= 1;

# 5;

end

endmodule

F)





Our circuit did not change the functionality of alu OP

G)

#memfile2.dat

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | | |
| **Machine Language (hex)** | | |  | **MIPS** | |
| 34088000 | | | # | main: ori $t0, $0, 0x8000 | |
| 20098000 | | | # | addi $t1, $0, -32768 | |
| 350a8001 | | | # | ori $t2, $t0, 0x8001 | |
| 11090005 | | | # | beq $t0, $t1, there | |
| 0128582a | | | # | slt $t3, $t1, $t0 | |
| 15600001 | | | # | bne $t3, $0, here | |
| 08000009 | | | # | j there | |
| 01485022 | | | # | here: sub $t2, $t2, $t0 | |
| 350800ff | | | # | ori $t0, $t0, 0xFF | |
| 016a5820 | | | # | there: add $t3, $t3, $t2 | |
| 01484022 | | | # | sub $t0, $t2, $t0 | |
| ad680052 | | | # | sw $t0, 82($t3) | |
|  |  |  | | |

H)

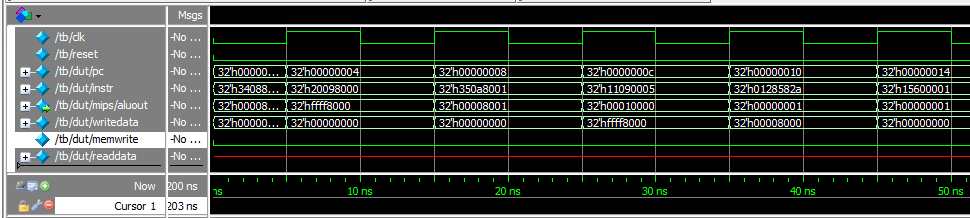
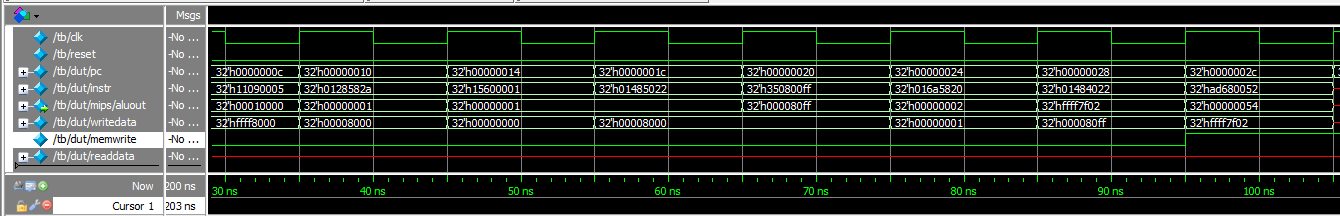
 

Image is also attached as wave2\_0 and wave2\_1.

The function stores the value ffff7f02 into the address at 0x 54