**Department of Electrical and Computer Engineering**

Author: Zainab Hussein

Title: MIPS Assembly Programming

Date: 4-9-2017

1. Time Spent: 7 Hours
2. Table 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Cycle** | **reset** | **pc** | **instr** | **branch** | **srca** | **srcb** | **aluout** | **zero** | **pcsrc** | **writedata** | **memwrite** | **read data** |
| 1 | 1 | 00 | addi $2,$0,5  (0x20020005) | 0 | 0 | 5 | 5 | 0 | 0 | 0 | 0 | x |
| 2 | 0 | 04 | addi $3,$0,12  (0x2003000c) | 0 | 0 | c | c | 0 | 0 | 0 | 0 | x |
| 3 | 0 | 08 | addi $7,$3,-9  (0x2067fff7) | 0 | c | -9 | 3 | 0 | 0 | 0 | 0 | x |
| 4 | 0 | 0C | Or $4,$7, $2  (0x00e22025) | 0 | 3 | 5 | 7 | 0 | 0 | 0 | 0 | X |
| 5 | 0 | 10 | And $5, $3, $4  (0x00642824) | 0 | C | 7 | 4 | 0 | 0 | 0 | 0 | X |
| 6 | 0 | 14 | Add $5, $3, $4  (0x00a42820) | 0 | 4 | 7 | b | 0 | 0 | 0 | 0 | X |
| 7 | 0 | 18 | Beq $5, $7, end  (0x10a7000a) | 1 | B | 3 | 8 | 0 | 0 | 0 | 0 | X |
| 8 | 0 | 1C | Slt $4, $3, $4  (0x0064202a) | 0 | C | 7 | 0 | 1 | 0 | 0 | 0 | X |
| 9 | 0 | 20 | Beq $4, $0, loop  (0x10800001) | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | X |
| 10 | 0 | 28 | Slt $4, $7, $2  (0x00e2202a) | 0 | 3 | 5 | 1 | 0 | 0 | 0 | 0 | X |
| 11 | 0 | 2C | Add $7, $4, $5  (0x00853820) | 0 | 1 | 8 | 9 | 0 | 0 | 0 | 0 | X |
| 12 | 0 | 30 | Sub $7, $7, $2  (0x00e23822) | 0 | 9 | 5 | 4 | 0 | 0 | 0 | 0 | X |
| 13 | 0 | 34 | Sw $7, 68($3)  (0xac670044) | 0 | C | 68 | 80 | 0 | 0 | 4 | 1 | X |
| 14 | 0 | 38 | Lw $2, 80($0)  (0x8c020050) | 0 | 0 | 80 | 80 | 0 | 0 | 0 | 0 | X |
| 15 | 0 | 3C | J end  (0x08000011) | 0 | X | X | X | X | 0 | 0 | 0 | X |
| 16 | 0 | 44 | Sw $2, 84($0)  (0xac020054) | 0 | 0 | 84 | 84 | 0 | 0 | -33023 | 1 | X |

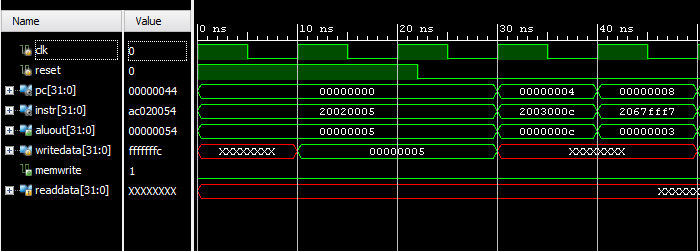
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Op5:0** | **RegWrite** | **RegDst** | **AluSrc** | **Branch** | **MemWrite** | **MemtoReg** | **ALUOp1:0** | **Jump** | **Bne** |
| R-type | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 10 | 0 | 0 |
| lw | 100011 | 1 | 0 | 1 | 0 | 0 | 1 | 00 | 0 | 0 |
| sw | 101011 | 0 | X | 1 | 0 | 1 | X | 00 | 0 | 0 |
| beq | 000100 | 0 | X | 0 | 1 | 0 | X | 01 | 0 | 0 |
| addi | 001000 | 1 | 0 | 1 | 0 | 0 | 0 | 00 | 0 | 0 |
| j | 000010 | 0 | X | X | X | 0 | X | XX | 1 | 0 |
| ori | 001101 | 1 | 0 | 10 | 0 | 0 | 0 | 11 | 0 | 0 |
| bne | 000101 | 0 | x | 00 | 1 | 0 | x | 01 | 0 | 1 |

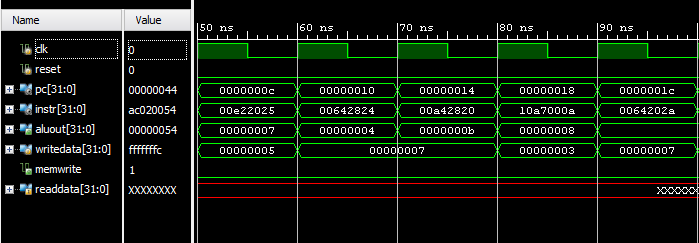
Extended functionality. Main Decoder:

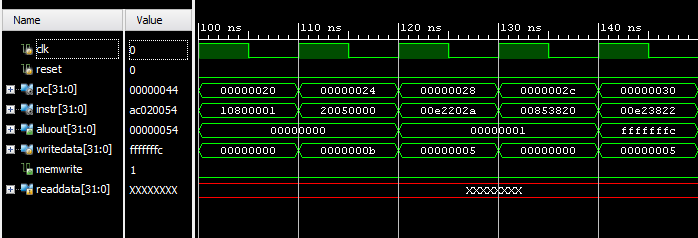
Extended functionality. ALU Decoder:

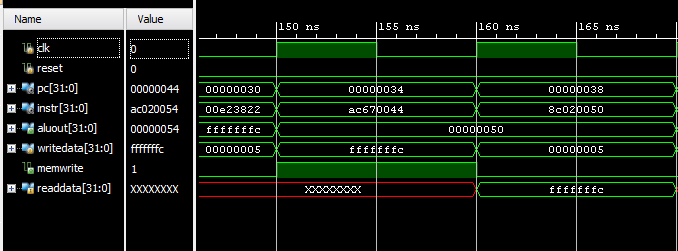
|  |  |
| --- | --- |
| **ALUOp1:0** | **Meaning** |
| 00 | Add |
| 01 | Subtract |
| 10 | Look at funct field |
| 11 | ori |

1. Testbench of the processor

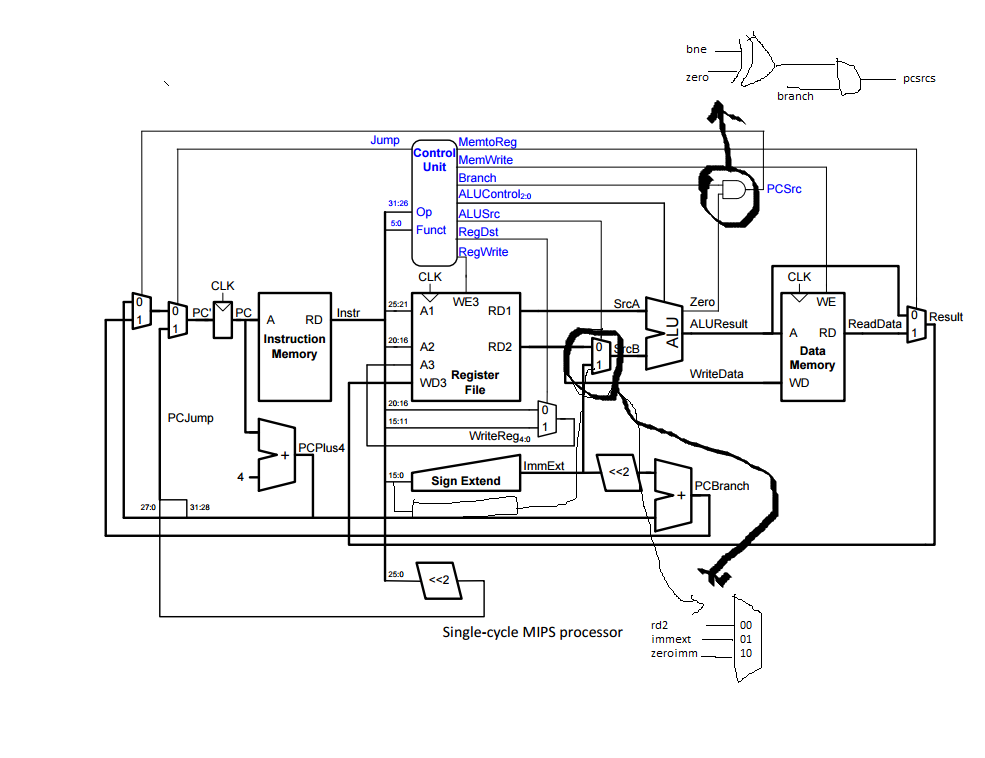








1. Added datapath schematic adding ori and bne functionality



1. Commented code with ori and bne functionality highlighted

//--------------------------------------------------------------

// mips.sv

// David\_Harris@hmc.edu and Sarah\_Harris@hmc.edu 23 October 2005

// Updated to SystemVerilog dmh 12 November 2010

// Single-cycle MIPS processor

//--------------------------------------------------------------

// files needed for simulation:

// mipsttest.v

// mipstop.v

// mipsmem.v

// mips.v

// mipsparts.v

// single-cycle MIPS processor

module mips(input logic clk, reset,

output logic [31:0] pc,

input logic [31:0] instr,

output logic memwrite,

output logic [31:0] aluout, writedata,

input logic [31:0] readdata);

logic memtoreg, branch,

pcsrc, zero,

regdst, regwrite, jump;

logic [2:0] alucontrol;

logic [1:0] alusrc; // CONVERTED FROM 1 to 2 bits TO EXPAND MUX

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

memtoreg, memwrite, pcsrc,

alusrc [1:0], regdst, regwrite, jump,

alucontrol);

datapath dp(clk, reset, memtoreg, pcsrc,

alusrc [1:0], regdst, regwrite, jump,

alucontrol,

zero, pc, instr,

aluout, writedata, readdata);

endmodule

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

input logic zero,

output logic memtoreg, memwrite,

output logic pcsrc,

output logic [1:0] alusrc,

output logic regdst, regwrite,

output logic jump,

output logic [2:0] alucontrol);

logic [1:0] aluop;

logic branch;

logic bne; //ADDED BNE

maindec md(op, memtoreg, memwrite, branch,

alusrc, regdst, regwrite, jump,

bne, aluop);

aludec ad(funct, aluop, alucontrol);

assign pcsrc = branch & ( zero ^ bne ); //XOR zero and bne

endmodule

module maindec(input logic [5:0] op,

output logic memtoreg, memwrite,

output logic branch,

output logic [1:0] alusrc,

output logic regdst, regwrite,

output logic jump, bne,

output logic [1:0] aluop);

logic [10:0] controls; //EXPANDED FROM 9 to 11 BITS

assign {regwrite, regdst, alusrc,

branch, memwrite,

memtoreg, jump, aluop, bne} = controls; //ADDED BNE CONTROL

always\_comb

case(op)

6'b000000: controls <= 11'b11000000100; //Rtype

6'b001101: controls <= 11'b10100000110; //ORI ADDED

6'b100011: controls <= 11'b10010010000; //LW

6'b101011: controls <= 11'b00010100000; //SW

6'b000100: controls <= 11'b00001000010; //BEQ

6'b001000: controls <= 11'b10010000000; //ADDI

6'b000010: controls <= 11'b00000001000; //J

6'b000101: controls <= 11'b00001000011; //BNE ADDED

default: controls <= 11'bxxxxxxxxxxx; //???

endcase

endmodule

module aludec(input logic [5:0] funct,

input logic [1:0] aluop,

output logic [2:0] alucontrol);

always\_comb

case(aluop)

2'b00: alucontrol <= 3'b010; // add

2'b01: alucontrol <= 3'b110; // sub

2'b11: alucontrol <= 3'b001; // ori //ADDED ORI CONTROL SIGNAL

default: case(funct) // RTYPE

6'b100000: alucontrol <= 3'b010; // ADD

6'b100010: alucontrol <= 3'b110; // SUB

6'b100100: alucontrol <= 3'b000; // AND

6'b100101: alucontrol <= 3'b001; // OR

6'b101010: alucontrol <= 3'b111; // SLT

default: alucontrol <= 3'bxxx; // ???

endcase

endcase

endmodule

module datapath(input logic clk, reset,

input logic memtoreg, pcsrc,

input logic [1:0] alusrc,

input logic regdst,

input logic regwrite, jump,

input logic [2:0] alucontrol,

output logic zero,

output logic [31:0] pc,

input logic [31:0] instr,

output logic [31:0] aluout, writedata,

input logic [31:0] readdata);

logic [4:0] writereg;

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

logic [31:0] signimm, signimmsh, zeroimm; //ADDED ZEROIMM

logic [31:0] srca, srcb;

logic [31:0] result;

// next PC logic

flopr #(32) pcreg(clk, reset, pcnext, pc);

adder pcadd1(pc, 32'b100, pcplus4);

sl2 immsh(signimm, signimmsh);

adder pcadd2(pcplus4, signimmsh, pcbranch);

mux2 #(32) pcbrmux(pcplus4, pcbranch, pcsrc,

pcnextbr);

mux2 #(32) 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);

mux2 #(5) wrmux(instr[20:16], instr[15:11],

regdst, writereg);

mux2 #(32) resmux(aluout, readdata,

memtoreg, result);

signext se(instr[15:0], signimm);

zeroext ze(instr[15:0], zeroimm); //ZERO EXTENDER ADDED

// ALU logic

mux3 #(32) srcbmux(writedata, signimm, zeroimm, alusrc, //EXPANDED MUX WITH ADDITIONAL ZEROIMM input

srcb);

alu alu(.a(srca), .b(srcb), .f(alucontrol),

.y(aluout), .zero(zero));

Endmodule

module mux3 #(parameter WIDTH = 8)

(input logic [WIDTH-1:0] d0, d1, k,

input logic [1:0] s,

output logic [WIDTH-1:0] y);

always\_comb

case(s)

2'b00: y <= d0;

2'b01: y <= d1;

2'b10: y <= k;

default: y <= 8'hxx;

endcase

endmodule

//zero extending added

module zeroext(input logic [15:0] a,

output logic [31:0] y);

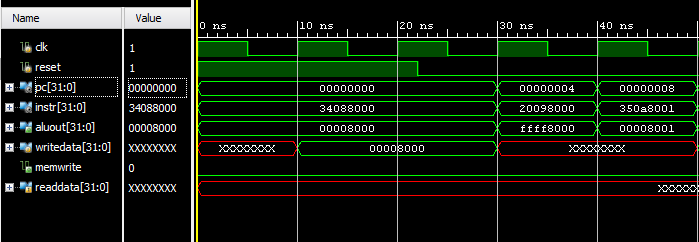
assign y = {16'h0000, a};

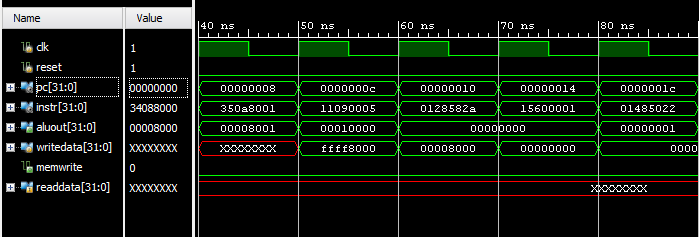
endmodule

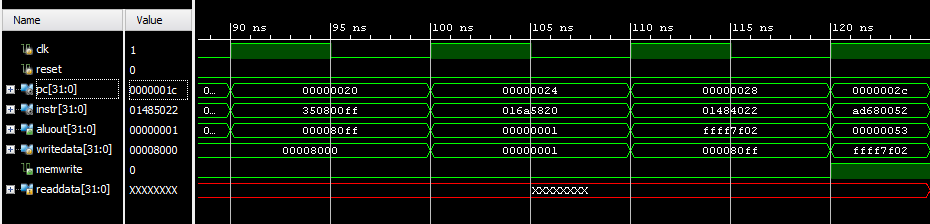
1. Memfile2 content

|  |
| --- |
| 34088000 |
| 20098000 |
| 350a8001 |
| 11090005 |
| 0128582a |
| 15600001 |
| 8000009 |
| 1485022 |
| 350800ff |
| 016a5820 |
| 1484022 |
| ad680052 |

1. Testbench of the modified processor







Sw instruction writes fffffffc to memory which is 0d-33023 to address 84