CS224

LAB004

SECTION 03

FURKAN MERT AKSAKAL

22003191

b-) Machine Codes to Assembly Language

|  |  |  |
| --- | --- | --- |
| **Address (hex)** | **Instruction (hex)** | **Assembly Language** |
| 0x00 | 0x20020005 | addi $v0, $zero, 0x0005 |
| 0x04 | 0x2003000c | addi $v1, $zero, 0x000C |
| 0x08 | 0x2067fff7 | addi $a3, $v1, 0xFFF7 |
| 0x0c | 0x00e22025 | or $a0, $a3, $v0 |
| 0x10 | 0x00642824 | and $a1, $v1, $a0 |
| 0x14 | 0x00a42820 | add $a1, $a1, $a0 |
| 0x18 | 0x10a7000a | beq $a1, $a3, 0x000A |
| 0x1c | 0x0064202a | slt $a0, $v1, $a0 |
| 0x20 | 0x10800001 | beq $a0, $zero, 0x0001 |
| 0x24 | 0x20050000 | addi $a1, $zero, 0x0000 |
| 0x28 | 0x00e2202a | slt $a0, $a3, $v0 |
| 0x2c | 0x00853820 | add $a3, $a0, $a1 |
| 0x30 | 0x00e23822 | sub $a3, $a3, $v0 |
| 0x34 | 0xac670044 | sw $a3, 0x0044($v1) |
| 0x38 | 0x8c020050 | lw $v0, 0x0050($zero) |
| 0x3c | 0x08000011 | j 0x0000011 |
| 0x40 | 0x20020001 | addi $v0, $zero, 0x0001 |
| 0x44 | 0xac020054 | sw $v0, 0x0054($zero) |
| 0x48 | 0x08000012 | j 0x0000012 |

c-) Register Transfer Level Expressions

bncon:   
 IM[PC]

if ((RF[rt ]!= RF[rs] + 4) || (RF[rt] % 4 != 0))

PC <= BTA

else

PC <= PC + 4

spc:

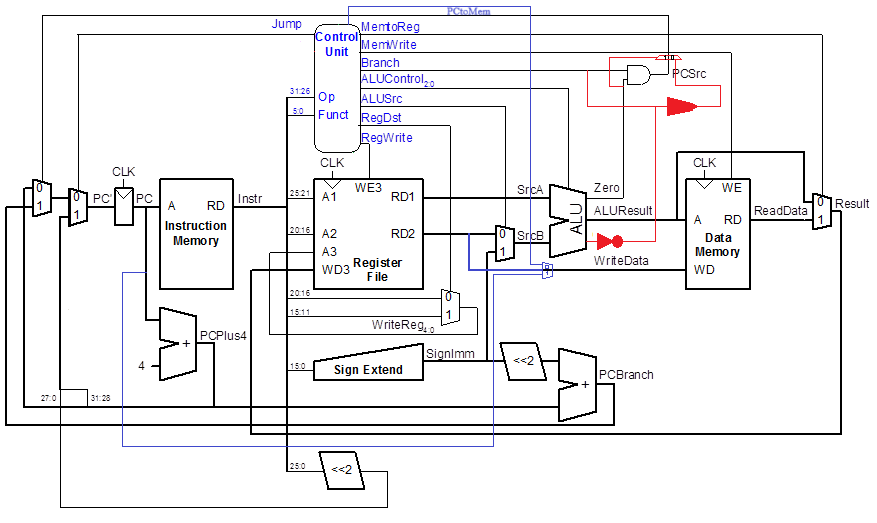
IM[PC]

MEM[RF[rs] + SignExt(Imm)] <= PC

PC <= PC +4

d-) Datapath

bncon: Red

spc: Blue

e-) Main Decoder

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Opcode** | **RegWrite** | **RegDst** | **ALUSrc** | **Branch** | **MemWrite** | **MemtoReg** | **ALUOp** | **Jump** | **PCtpMem** |
| **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 |
| **bncon** | 000111 | 0 | X | 0 | 1 | 0 | X | 01 | 0 | 0 |
| **spc** | 001001 | 0 | X | 1 | 0 | 1 | X | 00 | 0 | 1 |

No changes needed for the ALU decoder.

f-) MIPS Test Program

.data

message: .asciiz "Test completed\n"

.text

.globl main

main:

li $t0, 8 # $t0 = 8

li $t1, 12 # $t1 = 12

li $t2, 16 # $t2 = 16

li $t3, 4 # $t3 = 4

bncon\_test:

bncon $t0, $t1, branch\_target

addi $t4, $zero, 1

branch\_target:

addi $t4, $zero, 2

spc\_test:

li $t5, 20

spc $t0, $t5

add $t6, $t1, $t2

sw $t6, 0($t0)

lw $t7, 0($t0)

li $v0, 4

la $a0, message

syscall

li $v0, 10

syscall

g-) SystemVerilog Modules

**IMEM Module**

module imem ( input logic [7:0] addr, output logic [31:0] instr);

// imem is modeled as a lookup table, a stored-program byte-addressable ROM

always\_comb

case (addr) // word-aligned fetch

// address instruction

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

8'h00: instr = 32'h20020005; // disassemble, by hand

8'h04: instr = 32'h2003000c; // or with a program,

8'h08: instr = 32'h2067fff7; // to find out what

8'h0c: instr = 32'h00e22025; // this program does!

8'h10: instr = 32'h00642824;

8'h14: instr = 32'h00a42820;

8'h18: instr = 32'h10a7000a;

8'h1c: instr = 32'h0064202a;

8'h20: instr = 32'h10800001;

8'h24: instr = 32'h20050000;

8'h28: instr = 32'h00e2202a;

8'h2c: instr = 32'h00853820;

8'h30: instr = 32'h00e23822;

8'h34: instr = 32'hac670044;

8'h38: instr = 32'h8c020050;

8'h3c: instr = 32'h08000011;

8'h40: instr = 32'h20020001;

8'h44: instr = 32'hac020054;

8'h48: instr = 32'h08000012; // j 48, so it will loop here

8’h4c: instr = 32’h1cc5fffe; //bncon

8’h50: instr = 32’h20100040; // spc

default: instr = {32{1'bx}}; // unknown address

endcase

endmodule

**Controller Module**

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

input logic zero, four,

output logic memtoreg, memwrite,

output logic pcsrc, alusrc,

output logic regdst, regwrite,

output logic jump, pctomem,

output logic[2:0] alucontrol);

logic [1:0] aluop;

logic branch;

maindec md (op, memtoreg, memwrite, branch, alusrc, regdst, regwrite,

jump, aluop, pctomem);

aludec ad (funct, aluop, alucontrol);

logic zeroBranch = Branch & zero;

logic fourBranch = Branch & ~four;

mux2 $(1) pcsrcMux (zeroBranch, fourBranch, zero, pcsrc);

endmodule

**MIPS Module**

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, pcsrc, zero, four, alusrc, regdst, regwrite, jump, pctomem;

logic [2:0] alucontrol;

controller c (instr[31:26], instr[5:0], zero, four, memtoreg, memwrite, pcsrc,

alusrc, regdst, regwrite, jump, alucontrol);

datapath dp (clk, reset, memtoreg, pcsrc, alusrc, regdst, regwrite, jump,

alucontrol, pctomem, zero, four, pc, instr, aluout, writedata, readdata);

endmodule

**Maindec Module**

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

output logic memtoreg, memwrite, branch,

output logic alusrc, regdst, regwrite, jump,

output logic[1:0] aluop,

output logic pctomem);

logic [8:0] controls;

assign {regwrite, regdst, alusrc, branch, memwrite,

memtoreg, aluop, jump} = controls;

always\_comb

case(op)

6'b000000: controls <= 9'b110000100; // R-type

6'b100011: controls <= 9'b101001000; // LW

6'b101011: controls <= 9'b001010000; // SW

6'b000100: controls <= 9'b000100010; // BEQ

6'b001000: controls <= 9'b101000000; // ADDI

6'b000010: controls <= 9'b000000001; // J

6'b000010: controls <= 10’b0001000100 // bncon

6'b000010: controls <= 10’b0010100001 // spc

default: controls <= 9'bxxxxxxxxx; // illegal op

endcase

endmodule

**Datapath Module**

module datapath (input logic clk, reset, memtoreg, pcsrc, alusrc, regdst,

input logic pctomem,

input logic regwrite, jump,

input logic[2:0] alucontrol,

output logic zero, four,

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, srca, srcb, 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);

// ALU logic

mux2 #(32) srcbmux (writedata, signimm, alusrc, srcb);

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

assign writedata = pctomem ? pc : result;

endmodule

P.S. For this part all the changes are highlighted. Changes required by spc instruction is also underlined.