CS224 Computer Organization

Preliminary Report

Lab 04

Section 2

Burak Ozturk

21901841

# Part 1-B:

|  |  |  |
| --- | --- | --- |
| Address | Hexadecimal | Instruction |
| 0x00000000 | 0x20020005 | addi $v0, $zero, 0x0005 |
| 0x00000004 | 0x2003000c | addi $v1, $zero, 0x000C |
| 0x00000008 | 0x2067fff7 | addi $a3, $v1, 0xFFF7 |
| 0x0000000c | 0x00e22025 | or $a3, $a3, $v0 |
| 0x00000010 | 0x00642824 | and $a1, $v1, $a0 |
| 0x00000014 | 0x00a42820 | add $a1, $a1, $a0 |
| 0x00000018 | 0x10a7000a | beq $a1, $a3, 0x000A |
| 0x0000001c | 0x0064202a | slt $a0, $v1, $a0 |
| 0x00000020 | 0x10800001 | beq $a0, $zero, 0x0001 |
| 0x00000024 | 0x20050000 | addi $a1, $zero, 0x0000 |
| 0x00000028 | 0x00e2202a | slt $a0, $a3, $v0 |
| 0x0000002c | 0x00853820 | add $a3, $a0, $a1 |
| 0x00000030 | 0x00e23822 | sub $a3, $a3, $v0 |
| 0x00000034 | 0xac670044 | sw $a3, 0x0044($v1) |
| 0x00000038 | 0x8c020050 | lw $v0, 0x0050($zero) |
| 0x0000003c | 0x08000011 | j 0x0000011 |
| 0x00000040 | 0x20020001 | addi $v0, $zero, 0x0001 |
| 0x00000044 | 0xac020054 | sw $v0, 0x0054($zero) |
| 0x00000048 | 0x08000012 | j 0x0000012 |

# Part 1-C:

## subi:

Usage: subi $rt, $rs, imm

IM[PC]

RF[rt] <- RF[rs] - SignExt(imm)

PC <- PC + 4

## sw+:

Usage: sw+ $rt, imm($rs)

IM(PC)

DM[RF[rs]+SignExt(imm)] <- RF[rt]

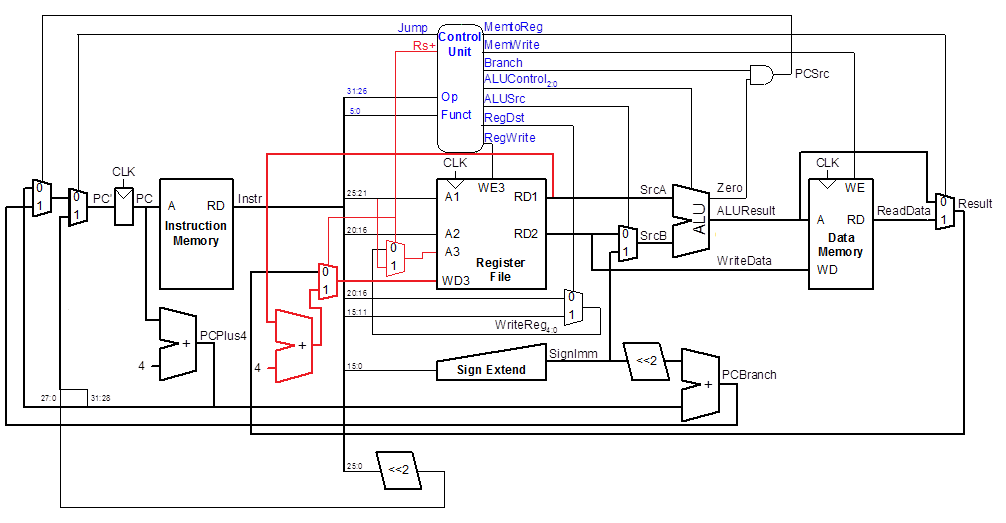
RF[rs] <- RF[rs] + 4

PC <- PC + 4

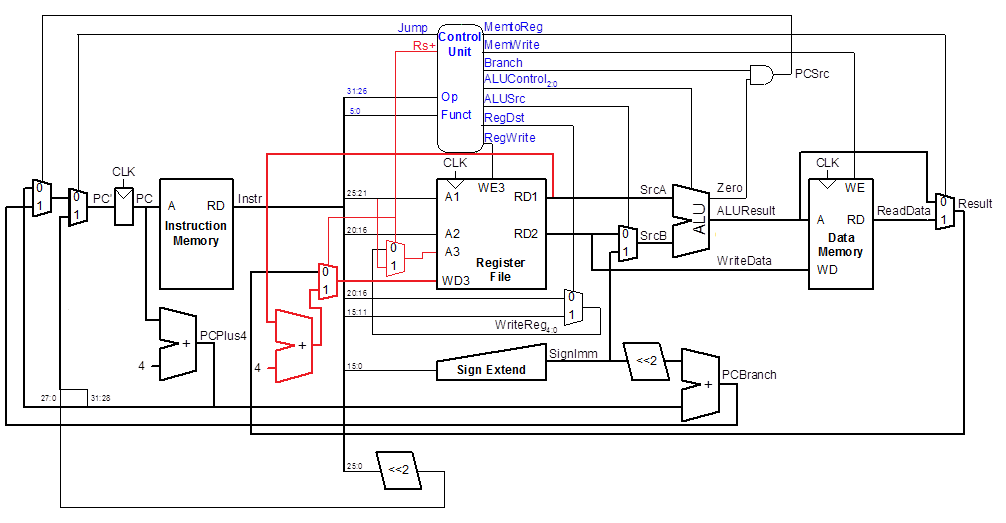
# Part 1-D:

subi instruction does not need any hardware changes. It will be possible by controlling AluControl.

Datapath changed to support sw+ instruction is as follows:

Two new multiplexers, a new adder and a new control signal Rs+ added and shown in red.

There is a bigger version of the datapath in the next page.

Bigger version of updated datapath:

# Part 1-E:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr. | Op5:0 | RegWrite | RegDst | AluSrc | Branch | MemWrite | MemtoReg | ALUOP1:0 | Jump | Rs+ |
| Rtype | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 10 | 0 | 0 |
| lw | 100011 | 1 | 0 | 1 | 0 | 0 | 1 | 00 | 0 | 0 |
| sw | 101011 | 0 | 0 | 1 | 0 | 1 | 0 | 00 | 0 | 0 |
| sw+ | 101100 | 1 | 0 | 1 | 0 | 1 | 0 | 00 | 0 | 1 |
| beq | 000100 | 0 | 0 | 0 | 1 | 0 | 0 | 01 | 0 | 0 |
| addi | 001000 | 1 | 0 | 1 | 0 | 0 | 0 | 00 | 0 | 0 |
| subi | 001001 | 1 | 0 | 1 | 0 | 0 | 0 | 01 | 0 | 0 |
| j | 000010 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 1 | 0 |

# Part 1-F:

addi $s0, $0, 16 # s0 should be 16

subi $s1, $s0, 6 # s1 should be 10

addi $v0, $0, 1

add $a0, $0, $s1

syscall

addi $s0, $0, 2

addi $s1, $0, 5

or $s2, $s0, $s1 # s2 should be 7

and $s3, $s0, $s1 # s3 should be 0

slt $at, $s3, $s2 # at should be 1

subi $s0, $0, -10 # s0 should be 10

sw+ $s2, 0($s0)

sw+ $s3, 0($s0)

sw+ $s4, 0($s0)

addi $s1, $0, 3

add $s0, $0, $0

L: beq $s0, $s1, ex

subi $s0, $s0, 4

lw $a0, 0($s0)

syscall

j L

ex: addi $v0, $0, 10

syscall

# Part 1-G:

## alu module:

module alu(input logic [31:0] a, b,

input logic [2:0] alucont,

output logic [31:0] result,

output logic zero);

always\_comb

case (alucont)

3'b000: result = a & b;

3'b001: result = a | b;

3'b010: result = a + b;

3'b110: result = a - b;

3'b111: result = (a < b) ? 1 : 0;

default: result = 32'bx;

endcase

assign zero = (result == 0);

endmodule

## datapath module:

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

input logic regwrite, jump, rsp,

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, writereg2;

logic [31:0] pcnext, pcnextbr, pcplus4, pcbranch, srca\_p4;

logic [31:0] signimm, signimmsh, srca, srcb, result, result2;

// 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], writereg2,

result2, srca, writedata);

adder rd1p4 (srca, 32’b100, srca\_p4);

mux2 #(32) wd3mux (result, srca\_p4, rsp, result2);

mux2 #(5) a3mux (writereg, instr[25:21], rsp, writereg2);

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);

endmodule

## maindec module:

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

output logic memtoreg, memwrite, branch,

output logic alusrc, regdst, regwrite, jump, rsp,

output logic[1:0] aluop );

logic [8:0] controls;

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

memtoreg, aluop, jump, rsp} = controls;

always\_comb

case(op)

6'b000000: controls <= 10'b1100001000; // R-type

6'b100011: controls <= 10'b1010010000; // LW

6'b101011: controls <= 10'b0010100000; // SW

6’b101100: controls <= 10’b1010100001; // SW+

6'b000100: controls <= 10'b0001000100; // BEQ

6'b001000: controls <= 10'b1010000000; // ADDI

6'b001001: controls <= 10'b1010000100; // SUBI

6'b000010: controls <= 10'b0000000010; // J

default: controls <= 10'bxxxxxxxxxx; // illegal op

endcase

endmodule

## controller module:

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

input logic zero,

output logic memtoreg, memwrite,

output logic pcsrc, alusrc,

output logic regdst, regwrite,

output logic jump, rsp,

output logic[2:0] alucontrol);

logic [1:0] aluop;

logic branch;

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

jump, rsp, aluop);

aludec ad (funct, aluop, alucontrol);

assign pcsrc = branch & zero;

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, alusrc, regdst, regwrite, jump, rsp;

logic [2:0] alucontrol;

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

alusrc, regdst, regwrite, jump, rsp, alucontrol);

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

rsp, alucontrol, zero, pc, instr, aluout, writedata, readdata);

endmodule

## imem module:

module imem ( input logic [5:0] addr, output logic [31:0] instr);

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

always\_comb

case ({addr,2'b00}) // word-aligned fetch

// address instruction

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

// My test program from Part F

8'h00: instr = 32'h20100010;

8'h04: instr = 32'h26110006;

8'h08: instr = 32'h20020001;

8'h0c: instr = 32'h00112020;

8'h10: instr = 32'h0000000c;

8'h14: instr = 32'h20100002;

8'h18: instr = 32'h20110005;

8'h1c: instr = 32'h02119025;

8'h20: instr = 32'h02119824;

8'h24: instr = 32'h0272082a;

8'h28: instr = 32'h2410fff6;

8'h2c: instr = 32'hb2120000;

8'h30: instr = 32'hb2130000;

8'h34: instr = 32'hb2140000;

8'h38: instr = 32'h20110003;

8'h3c: instr = 32'h00008020;

8'h40: instr = 32'h12110004;

8'h44: instr = 32'h26100004;

8'h48: instr = 32'h8e040000;

8’h4c: instr = 32’h0000000c;

8'h50: instr = 32'h08000040;

8'h54: instr = 32'h2002000a;

8'h58: instr = 32'h0000000c;

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

endcase

endmodule