**PRELIMINARY REPORT**

**Lab 04**

MUHAMMAD ARHAM KHAN

21701848 - CS

SECTION 06

SPRING 2019

**Dated: 31 March, 2019**

**Part b:**

|  |  |  |
| --- | --- | --- |
| Address | Machine Instruction (hex) | MIPS equivalent |
| 8'h00 | 32'h20020005 | ADDI $v0 $zero 0x0005 |
| 8'h04 | 32'h2003000c | ADDI $v1 $zero 0x000C |
| 8'h08 | 32'h2067fff7 | ADDI $a3 $v1 0xFFF7 |
| 8'h0c | 32'h00e22025 | OR $a0 $a3 $v0 |
| 8'h10 | 32'h00642824 | AND $a1 $v1 $a0 |
| 8'h14 | 32'h00a42820 | ADD $a1 $a1 $a0 |
| 8'h18 | 32'h10a7000a | BEQ $a1 $a3 0x000A |
| 8'h1c | 32'h0064202a | SLT $a0 $v1 $a0 |
| 8'h20 | 32'h10800001 | BEQ $a0 $zero 0x0001 |
| 8'h24 | 32'h20050000 | ADDI $a1 $zero 0x0000 |
| 8'h28 | 32'h00e2202a | SLT $a0 $a3 $v0 |
| 8'h2c | 32'h00853820 | ADD $a3 $a0 $a1 |
| 8'h30 | 32'h00e23822 | SUB $a3 $a3 $v0 |
| 8'h34 | 32'hac670044 | SW $a3 0x0044 $v1 |
| 8'h38 | 32'h8c020050 | LW $v0 0x0050 $zero |
| 8'h3c | 32'h08000011 | J 0x0000011 |
| 8'h40 | 32'h20020001 | ADDI $v0 $zero 0x0001 |
| 8'h44 | 32'hac020054 | SW $v0 0x0054 $zero |
| 8'h48 | 32'h08000012 | J 0x0000012 |

**Part c:**

**jalr $rs, $rd:**

IM[PC]

PC <- RF[rs]

RF[rd] <- PC + 4

**Bgt $rs, $rt, label:**

IM[PC]

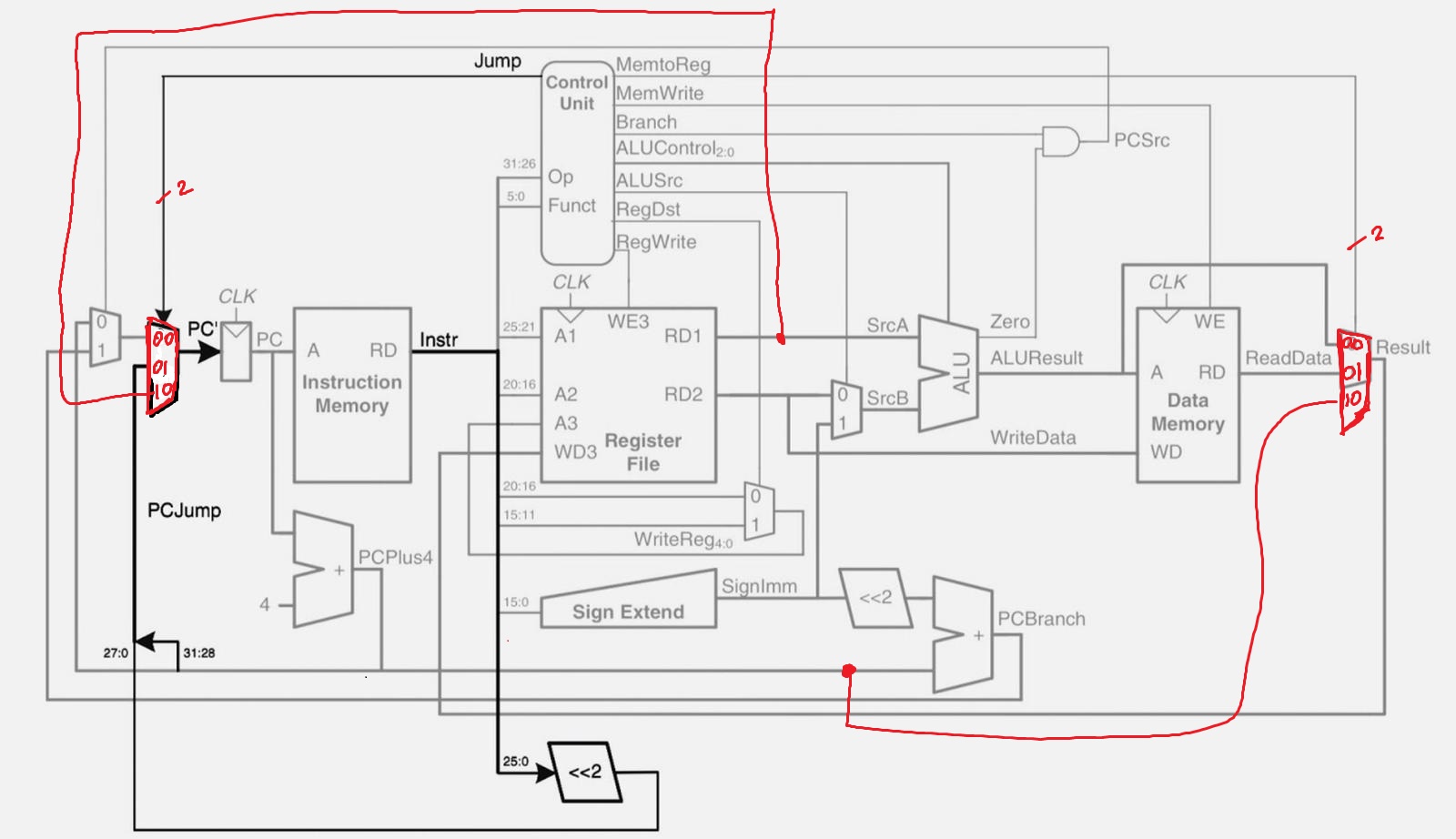
If (RF[rs] > RF[rt])

PC <- PC + 4 + SignExt(imm)

Else

PC <- PC + 4

**Part d:**



**Part e:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction | | Opcode | Reg Write | RegDst | ALUSrc | Branch | MemWrite | MemtoReg | ALUOp | Jump |
| R-type | | 000000 | 1 | 1 | 0 | 0 | 0 | 00 | 1x | 00 |
| lw | | 100010 | 1 | 0 | 1 | 0 | 0 | 01 | 00 | 00 |
| sw | | 101011 | 0 | x | 1 | 0 | 1 | xx | 00 | 00 |
| beq | | 000100 | 0 | x | 0 | 1 | 0 | xx | 01 | 00 |
| addi | | 001000 | 1 | 0 | 1 | 0 | 0 | 00 | 00 | 00 |
| j | | 000010 | 0 | x | x | x | 0 | XX | xx | 01 |
| jalr | | 000000 | 1 | X | X | X | 0 | 10 | XX | 10 |
| bgt | slt | 101010 | 1 | 1 | 0 | 0 | 0 | 00 | 1X | 00 |
| bne | 000101 | 0 | X | 0 | 1 | 0 | XX | 01 | 00 |

**Part f:**

.text

.globl \_start

\_start:

la $s2, 11 #Loading a value into s2

la $s3, 10 #Loading another value into s3

#Beginning execution of bgt with $rs = $s2, $rt = $s3

#If s3 is less than s2, put 0 in t2

slt $t1, $s3, $s2

#bne if t1 is not equal to 0, jumpto bgtcheck

bne $t1, $zero, bgtcheck

#if bgt failed, go to run jalr

j jalrcheck

bgtcheck: #BGT checker

#printing a prompt of success

la $a0, bgtworks

li $v0, 4

syscall

jalrcheck:

la $s1, end #loading address of next label

jalr $s0, $s1 #loading PC+4 in s0 and jumping to s1 instr

li $v0, 10 #System call to end program

syscall

end: #Jalr works

la $a0, jalrworks #loading the address of string

li $v0, 4

syscall

jr $s0 #jumping back to the s0 stating that jalr worked

.data

jalrworks: .asciiz "Jalr’s operation successful"

bgtworks: .asciiz "BGT succeed"

**Part g:**

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

input logic zero,

output logic memwrite,

output logic pcsrc, alusrc,

output logic regdst, regwrite,

output logic[1:0] jump, memtoreg,

output logic[2:0] alucontrol);

logic [1:0] aluop;

logic branch;

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

jump, aluop);

aludec ad (funct, aluop, alucontrol);

assign pcsrc = branch & zero;

endmodule

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

output logic, memwrite, branch,

output logic alusrc, regdst, regwrite,

output logic[1:0] aluop, memtoreg, jump );

logic [10:0] controls;

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

memtoreg, aluop, jump} = controls;

always\_comb

case(op)

6'b000000: controls <= 11'b11000001000; // R-type

6'b100011: controls <= 11'b10100010000; // LW

6'b101011: controls <= 11'b00101000000; // SW

6'b000100: controls <= 11'b00010000100; // BEQ

6'b001000: controls <= 11'b10100000000; // ADDI

6'b000010: controls <= 11'b00000000001; // J

6'b000000: controls <= 11'b10000100010; // JALR

6'b101010: controls <= 11'b11000001000; // SLT

6'b000101: controls <= 11'b00010000100; // BNE

default: controls <= 11'bxxxxxxxxxxx; // illegal op

endcase

endmodule

**module datapath** (input logic clk, reset, pcsrc, alusrc, regdst, regwrite

input logic[1:0] jump, memtoreg,

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, 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);

mux3 #(32) pcmux(pcnextbr, {pcplus4[31:28],

instr[25:0], 2'b00}, srca, 32’b0, 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);

mux3 #(32) resmux (aluout, readdata, pcplus4, 32’b0, 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

// paramaterized 2-to-1 MUX

**module mux2** #(parameter WIDTH = 11)

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

input logic s,

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

assign y = s ? d1 : d0;

endmodule

// paramaterized 4-to-1 MUX

**module mux3** #(parameter WIDTH = 11) // adding a 4 – to – 1 mux

(input logic[WIDTH-1:0] d0, d1, d2, d3, // to accommodate jalr

input logic[1:0] s,

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

assign y = s[1] ? (s[0] ? d3 : d2) : (s[0] ? d1 : d0);

endmodule