CS224

Lab 4

Section 2

Utku Kurtulmuş

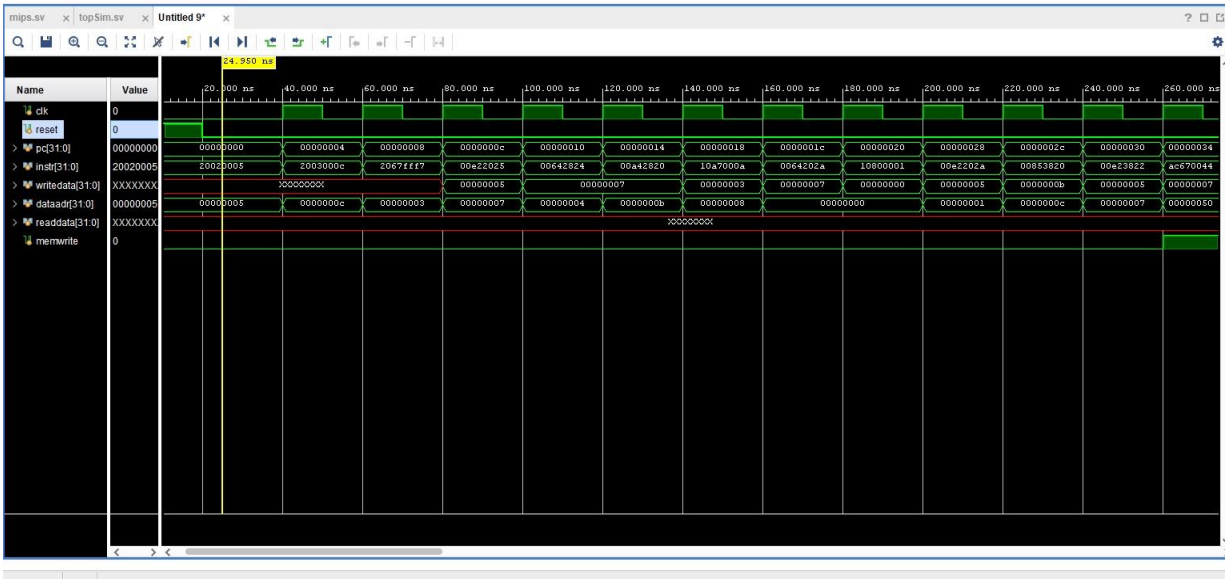
21903025

PART 1

A)

```
8'h00: instr = 32'h20020005;      //addi $v0, $0, 0x0005
8'h04: instr = 32'h2003000c;      //addi $v1, $0, 0x000c
8'h08: instr = 32'h2067fff7;      //addi $a3, $v1,0xfff7 (-7) (a3 = 5)
8'h0c: instr = 32'h00e22025;      //or $a0, $a3, $v0
8'h10: instr = 32'h00642824;     //and $a1, $v1, $a0
8'h14: instr = 32'h00a42820;    //add $a1, $a1, $a0
8'h18: instr = 32'h10a7000a;    //beq $a1, $a3, 0xA
8'h1c: instr = 32'h0064202a;    //slt $a0, $v1, $a0
8'h20: instr = 32'h10800001;    //beq $a0, $0, 0x0001
8'h24: instr = 32'h20050000;    //addi $a1, $0, 0x0000
8'h28: instr = 32'h00e2202a;    //slt $a0, $a3, $v0
8'h2c: instr = 32'h00853820;    //add $a3, $a0, $a1
8'h30: instr = 32'h00e23822;    //sub $a3, $a3, $v0
8'h34: instr = 32'hac670044;    //sw $a3, 0x0044, $v1
8'h38: instr = 32'h8c020050;    //lw $v0, 0x0050, $0
8'h3c: instr = 32'h08000011;    //j 0x00000011
8'h40: instr = 32'h20020001;    //addi $v0, $zero, 0x1
8'h44: instr = 32'hac020054;    //sw $v0, 0x0054, $0
8'h48: instr = 32'h08000012;      //  j 0x12
```

D)



(Zoom in to see clearly)

E)

i) In an R type instruction write data corresponds to the value of $rt register, in the picture RD2.
ii) Because we started the program with I type instructions. And we don't use rt register in I type instructions.
iii) Because most of the time we don't need to access the values in data memory. If we use something like lw, we get the value from memory (hence read data is defined) and store it in register file.
iv) Alu output.
v) SW. Because we write to the data memory when we use sw.

F)

The 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'b010: result = a + b;
        3'b011: result = a << b; // The change
        3'b110: result = a - b;
        3'b000: result = a & b;
        3'b001: result = a | b;
        3'b111: result = (a < b) ? 1 : 0;
        default: result = {32{1'bx}};
      endcase

    assign zero = (result == 0) ? 1'b1 : 1'b0;
Endmodule
```

PART 2

A)RTL

SRACC

IM[pc]

RF[RD] = RF[RD] + (RF[RS] << RF[RT])

PC = PC +4;

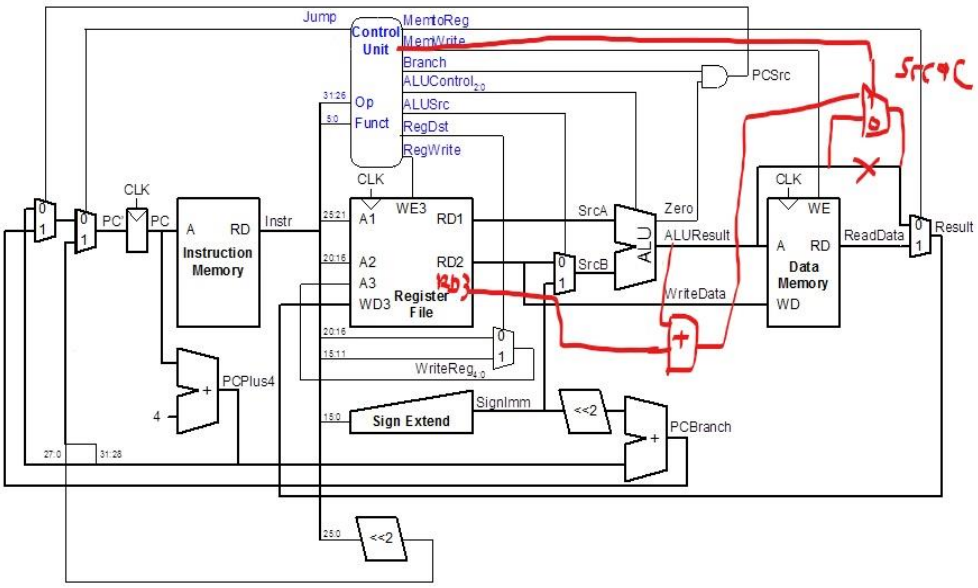
SUBI

IM[pc]

RF[RS] = RF[RT] - immediate

PC = PC + 4

B)



No change for subi, just giving the correct control signals is enough.

C)

```
module maindec (input logic[5:0] op,
                output logic memtoreg, memwrite, branch,
                output logic alusrc, regdst, regwrite, jump, srrac,
                output logic[1:0] aluop );
  logic [8:0] controls;

  assign {regwrite, regdst, alusrc, branch, memwrite,
                memtoreg,  aluop, jump, srrac} = 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'b000100: controls <= 10'b0001000100; // BEQ
     6'b001000: controls <= 10'b1010000000; // ADDI
```

```
    6'b000010: controls <= 10'b0000000010; // J
    6'b111111: controls <= 10'b1100001101; //srrac
    6'b111110: controls <= 10'b1010000100; //subi
    default:   controls <= 10'bxxxxxxxxxx; // illegal op
  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  (for lw/sw/addi)
    2'b01: alucontrol  = 3'b110;  // sub   (for beq)
    2'b11: alucontrol =  3'b011;  //srrac (for srrac)
    default: case(funct)         // R-TYPE instructions
       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
```