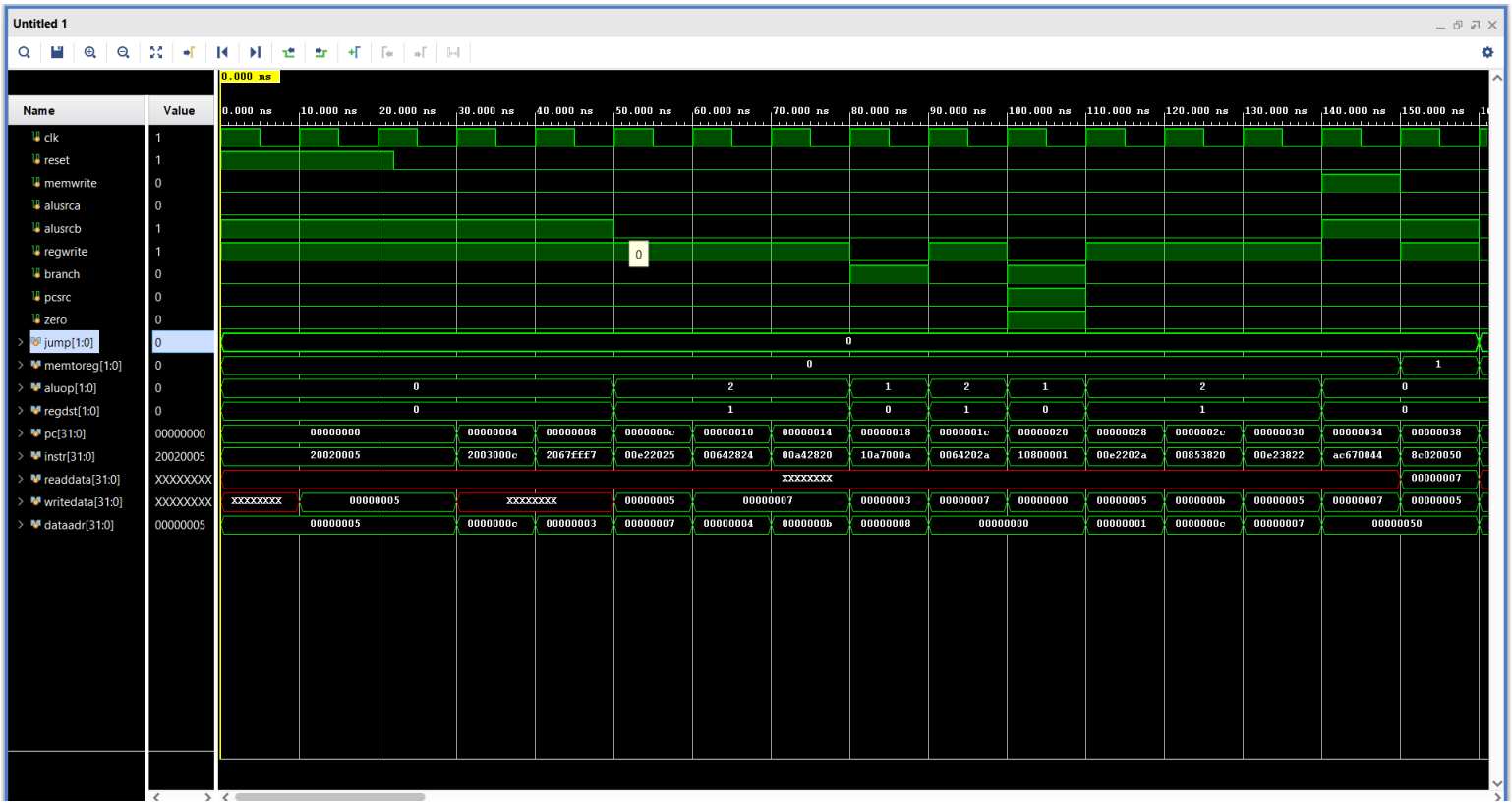
CS224
LAB04
Zülal Nur Hıdıroğlu
21903125

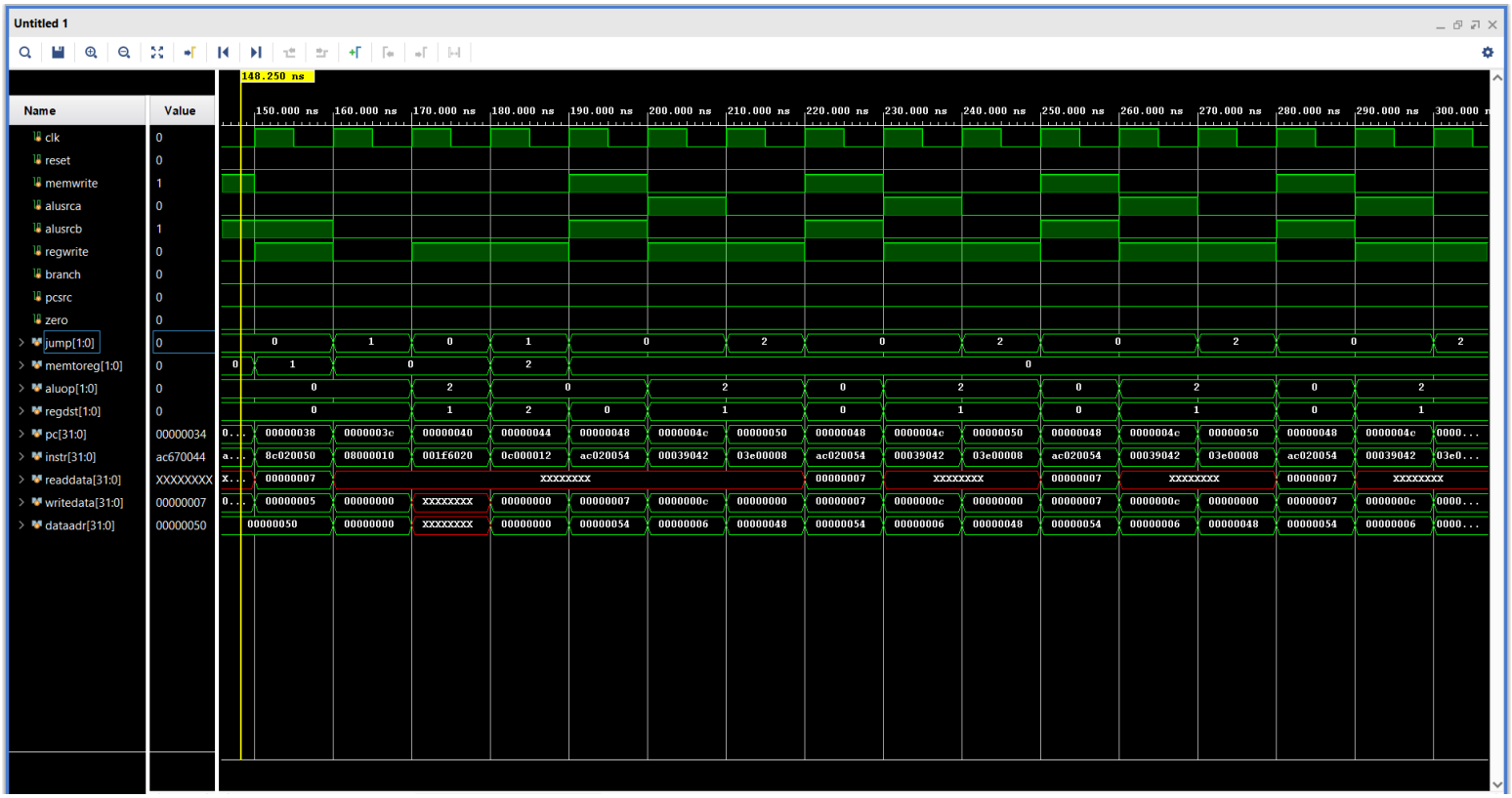
Part 1

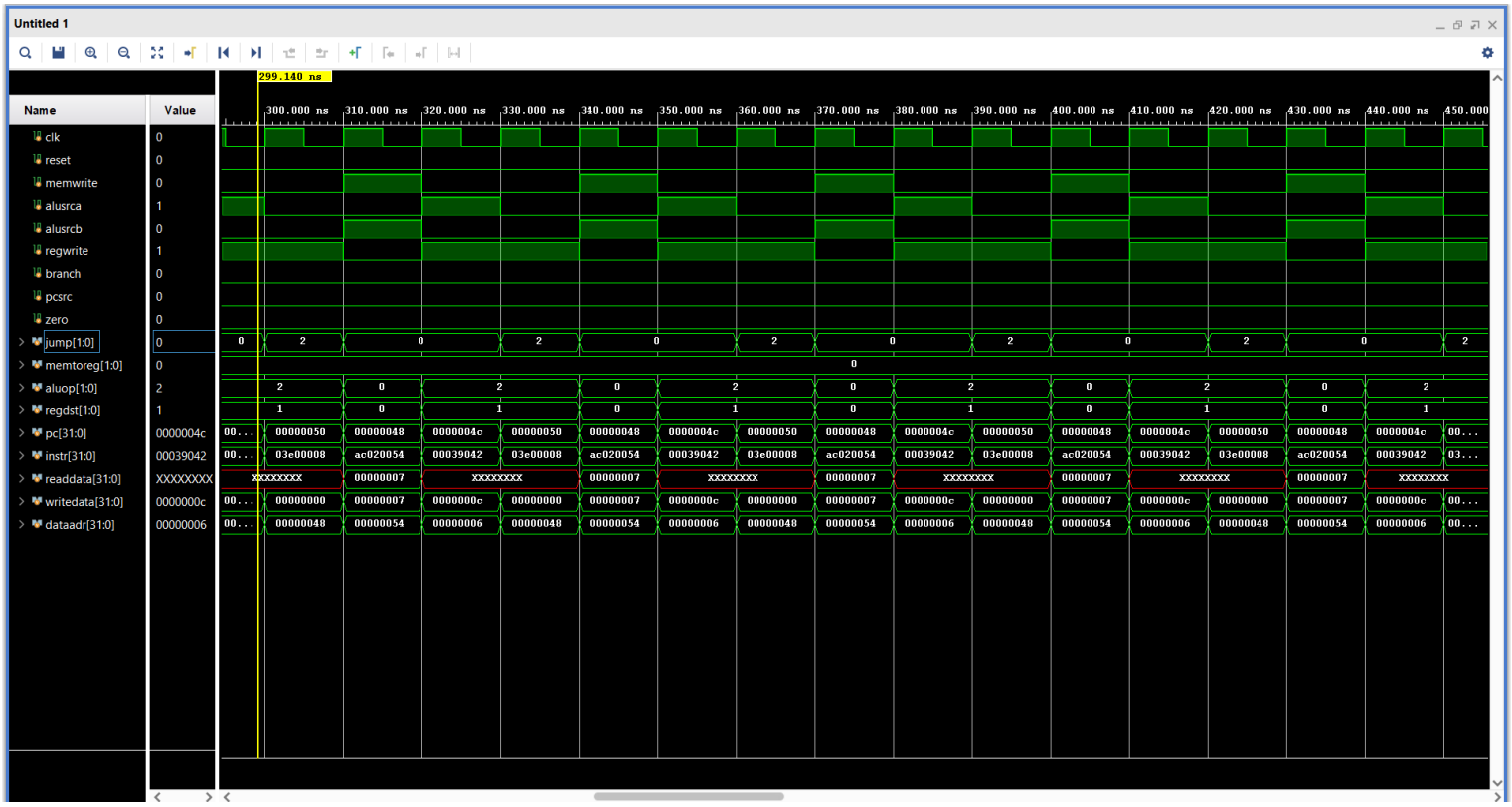
a)

Address (hex)	Machine Instruction	Assembly Language
00	0x20020005	addi $v0, $zero, 5
04	0x2003000c	addi $v1, $zero, 12
08	0x2067fff7	addi $a3, $v1, -9
0c	0x00e22025	or $a0, $a3, $v0
10	0x00642824	and $a1, $v1, $a0
14	0x00a42820	add $a1, $a1, $a0
18	0x10a7000a	beq $a1, $a3, 10
1c	0x0064202a	slt $a0, $v1, $a0
20	0x10800001	beq $a0, $zero,1
24	0x20050000	addi $a0, $zero, 0
28	0x00e2202a	slt $a0, $a3, $v0
2c	0x00853820	add $a3, $a0, $a1
30	0x00e23822	sub $a3, $a3, $v0
34	0xac670044	sw $a3, 68($v1)
38	0x8c020050	lw $v0, 80($zero)
3c	0x08000010	j 0x00000010
40	0x001f6020	add $t4, $zero, $ra
44	0x0c000012	jal 0x000012
48	0xac020054	sw $v0, 84($zero)
4c	0x00039042	srl $s2, $v1, 1
50	0x03e00008	jr $ra

Part1
e)


Untitled 1
Name
Value
clk
reset
memwrite
alusrca
alusrcb
regwrite
branch
pcsrc
zero
jump[1:0]
memtoreg[1:0]
aluop[1:0]
regdst[1:0]
pc[31:0]
instr[31:0]
readdata[31:0]
writedata[31:0]
dataadr[31:0]


Untitled 1
Name
Value
clk
reset
memwrite
alusrca
alusrcb
regwrite
branch
pcsrc
zero
jump[1:0]
memtoreg[1:0]
aluop[1:0]
regdst[1:0]
pc[31:0]
instr[31:0]
readdata[31:0]
writedata[31:0]
dataadr[31:0]



Part1

f)

i)In an R-type instruction writedata corresponds to readdata2(RD2) of the register file.

ii) In I-type instruction srcb takes sign immediate instead of rd2. Moreover in the case where we don't write anything to data memory, WD port is empty like lw instruction. In our instruction memory module there is less r-type instruction.

iii) The reason why the readdata is mostly undefined is in the instruction memory there is just one lw instruction which loads the data from data memory to register file.

iv) In an R-type instruction dataadr corresponds to AluResult which goes to WD3 port of the register file through memtoreg mux.

iv)

Part1

g)

i) The difference between sll and srlv is in srlv, rt is shifted to the left by the numbed contained in the register rs. In that case, asrca should be zero because shamt will not be used. However, sll can be done by just adding new information to alu unit. Therefore there is no need for adding new block.

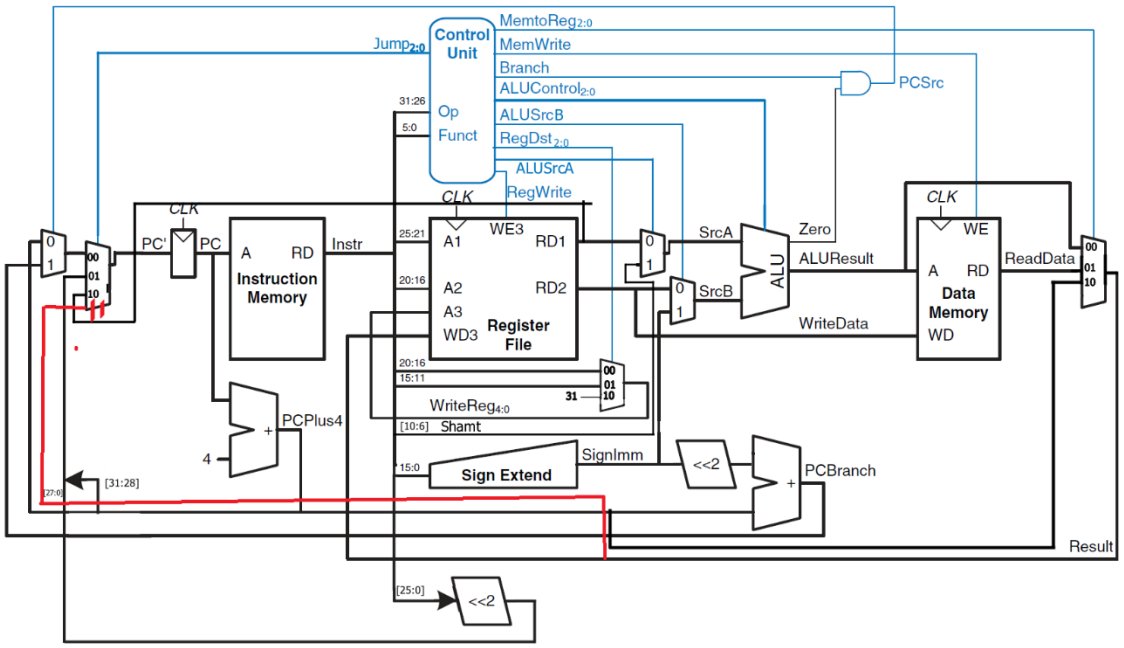
ii) To support the sll instruction of the system, one more AluControl should be added to alu unit which 111 will correspond sll.

Part2

a) IM[PC]

PC <- DM[RF[rs] + SignExt(immed16)]

b)



c)

Instruction	Opcode	RegWrite	RegDst	ALUSrcA	ALUSrcB	Branch	MemWrite	MemToReg	AluOp	Jump
R-type	000000	1	01	0	0	0	0	00	10	00
srl	000000	1	01	1	0	0	0	00	10	00
lw	100011	1	00	0	1	0	0	01	00	00
sw	101011	0	X	0	1	0	1	XX	00	00
beq	000100	0	X	0	0	1	0	01	01	00
addi	001000	1	00	0	1	0	0	00	00	00
j	000010	0	X	X	X	X	0	XX	XX	01
jal	000011	1	10	X	X	X	0	10	XX	01
jr	000000	1	01	0	0	0	0	00	10	10
jm	111111	0	xx	0	1	x	0	00	00	11

Implementing jr instruction

```
module maindec (input logic[5:0] op,funct,
                output logic[1:0] memtoreg,
                output logic memwrite, branch,
                output logic alusrcb, regwrite,
                output logic [1:0] jump,
                output logic[1:0] aluop, regdst);
  logic [10:0] controls;

  assign {regwrite, regdst, alusrcb, branch, memwrite,
          memtoreg, aluop} = controls;
```

```
  always_comb
    case(op)
       6'b000000: begin controls <= 10'b1_01_0_0_0_00_10;
   if(funct == 6'b001000) //check for jr instruction
    jump <= 2'b10;
    else
   jump <= 2'b00;
   end // R-type
       6'b100011: begin controls <= 10'b1_00_1_0_0_01_00; jump <= 2'b00; end //
LW
       6'b101011: begin controls <= 10'b0_00_1_0_1_00_00; jump <= 2'b00; end //
SW
       6'b000100: begin controls <= 10'b0_00_0_1_0_00_01; jump <= 2'b00; end //
BEQ
       6'b001000: begin controls <= 10'b1_00_1_0_0_00_00; jump <= 2'b00; end //
ADDI
       6'b000010: begin controls <= 10'b0_00_0_0_0_00_00; jump <= 2'b01; end //
J
       6'b000011: begin controls <= 10'b1_10_0_0_0_10_00; jump <= 2'b01; end //
JAL
       6'b111111: begin controls <= 10'b0_xx_1_x_0_01_00; jump <= 2'b11;
end//jm
       default:   begin controls <= 10'bx_x_x_x_x_xx_xx; jump <= 2'bx; end //
illegal op
    endcase
endmodule

module datapath (input  logic clk, reset, pcsrc, alusrca, alusrcb,
                 input  logic regwrite,
                 input logic [1:0] jump,
                 input  logic[1:0] memtoreg, regdst,
                        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, rd1, srca, srcb, result;


   // register file logic
   regfile     rf (clk, regwrite, instr[25:21], instr[20:16], writereg,
                   result, rd1, writedata);
```

```
  mux4 #(5)    wrmux (instr[20:16], instr[15:11], 5'h0001F, 0, regdst, writereg);
   signext         se (instr[15:0], signimm);

   mux4 #(32)  resmux (aluout, readdata, pcplus4, 32'b0, memtoreg, result);

     // next PC logic
    mux4 #(32)  pcmux(pcnextbr, {pcplus4[31:28],
                              instr[25:0], 2'b00},rd1, result, jump, pcnext);
  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);

   // ALU logic
   mux2 #(32)  srcamux (rd1, {27'b0, instr[10:6]}, alusrca, srca);
   mux2 #(32)  srcbmux (writedata, signimm, alusrcb, srcb);
   alu         alu (srca, srcb, alucontrol, aluout, zero);

endmodule
```