CS224

Lab 04

Section: 04

Mehmet Hasat Serinkan

21901649

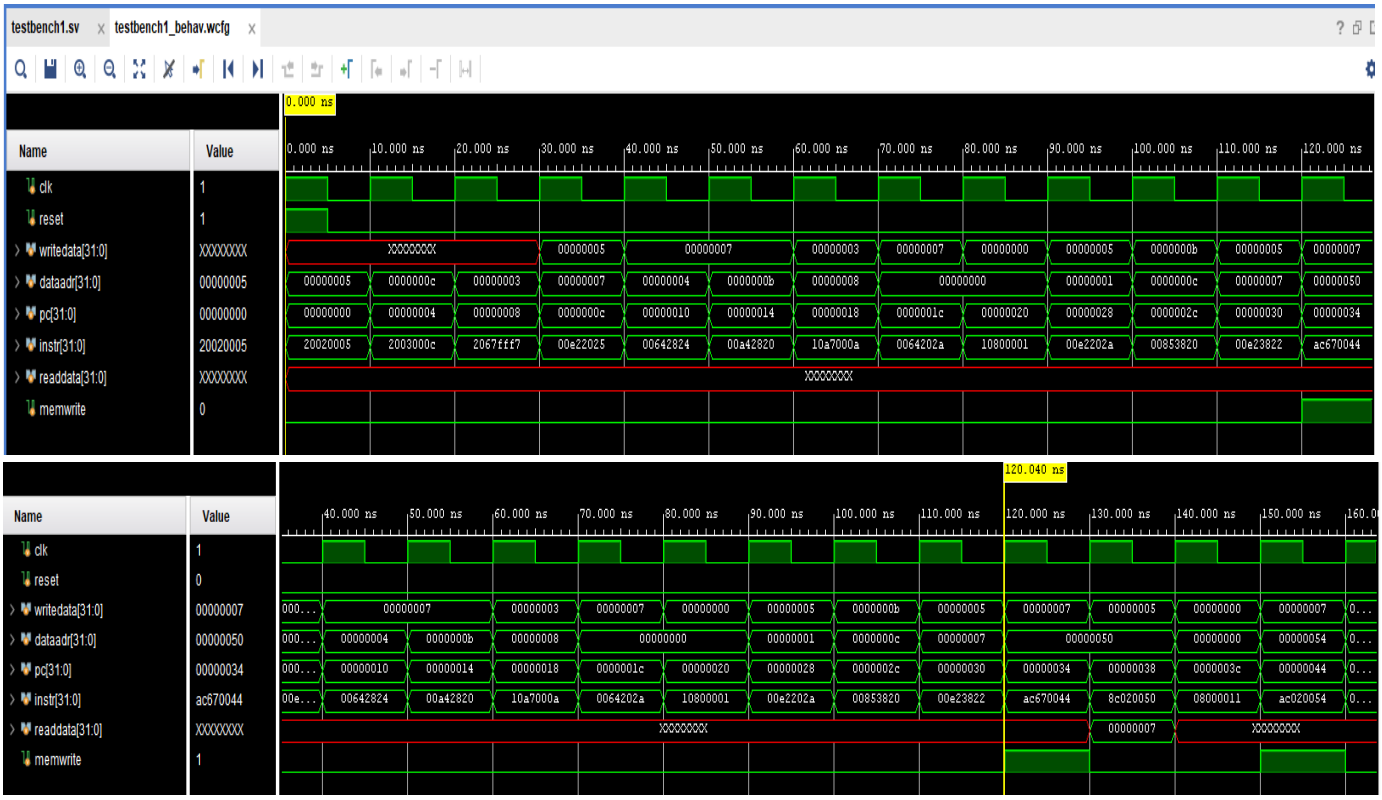
14.11.2022

PART 1

1.a)

Address(hex)	Machine Instruction(hex)	Assembly Language Equivalent
00	20020005	addi $v0, $zero, 5
04	2003000c	addi $v1, $zero, 12
08	2067fff7	addi $a3, $v1, -9
0c	00e22025	or $a0, $a3, $v0
10	00642824	and $a1, $v1, $a0
14	00a42820	add $a1, $a1, $a0
18	10a7000a	beq $a1, $a3, 0x00000044
1c	0064202a	slt $a0, $v1, $a0
20	10800001	beq $a0, $zero, 0x00000028
24	20050000	addi $a1, $zero, 0
28	00e2202a	slt $a0, $a3, $v0
2c	00853820	add $a3, $a0, $a1
30	00e23822	sub $a3, $a3, $v0
34	ac670044	sw $a3, 68($v1)
38	8c020050	lw $v0, 80($zero)
3c	08000011	j 0x00000044
40	20020001	add $t4, $zero, $ra
44	ac020054	sw $v0, 84($zero)
48	08000012	j 0x00000048

1.d)



1.e)

i- In an R-type instruction what does writedata correspond to?

Writedata corresponds to data which comes from register files. They are written into data memory.

ii- Why is writedata undefined for some of the early instructions in the program?

It is undefined for early instructions because the first three instructions are I-type instruction. There is no data written into data memory such as R-type instructions. Therefore, the writedata is undefined.

iii- Why is readdata most of the time undefined?

Readdata is the data that loaded from data memory. It becomes defined when there is a lw instruction. Since there is only one lw instruction, the readdata is undefined most of the time.

iv- In an R-type instruction what does dataadr correspond to?

Dataadr corresponds to the result which occurs in ALU. They are written into the register files.

v- In which instructions memwrite becomes 1?

Memwrite becomes 1 when there is a sw instruction. It causes data to written into data memory.

1.f)

```
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)
      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
          6'b000000: alucontrol  = 3'b011; // SLL
          default:   alucontrol  = 3'bxxx; // ???
        endcase
    endcase
endmodule

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'b110: result = a - b;
            3'b000: result = a & b;
            3'b001: result = a | b;
            3'b111: result = (a < b) ? 1 : 0;
```

```
            3'b011: result = a << b;
            default: result = {32{1'bx}};
        endcase

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

PART 2

2.a)

jm:

IM[PC]

PC ← DM[RF[rs] + SignExtImm]

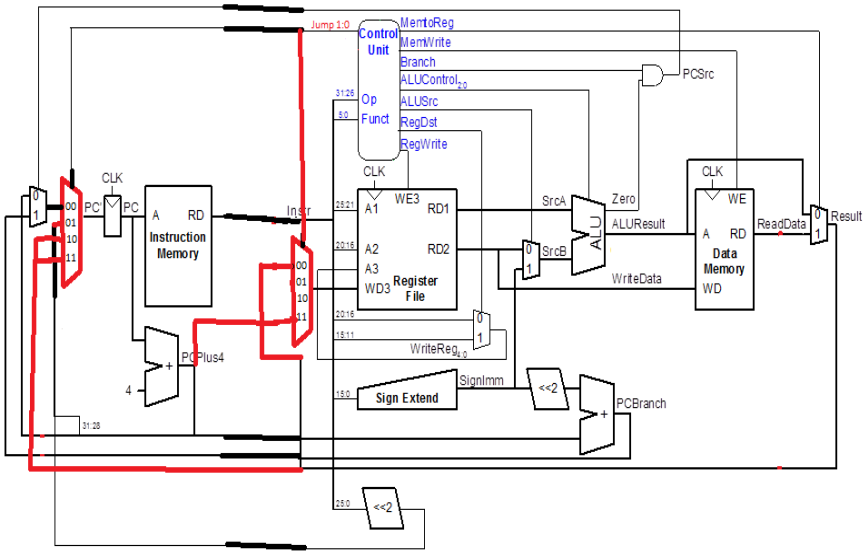
jalm:

IM[PC]

RF[rt] ← PC + 4

PC ← DM[RF[rs] + SignExtImm]

2.b)



2.c)

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp	Jump
R-type	000000	1	1	0	0	0	0	10	00
lw	100011	1	0	1	0	0	1	00	00
sw	101011	0	X	1	0	1	X	00	00
beq	000100	0	X	0	1	0	X	01	00
addi	001000	1	0	1	0	0	0	00	00
j	000010	0	X	X	X	0	X	XX	01
jm	001111	0	X	1	X	0	1	00	10
jalm	001110	1	0	1	X	0	1	00	11