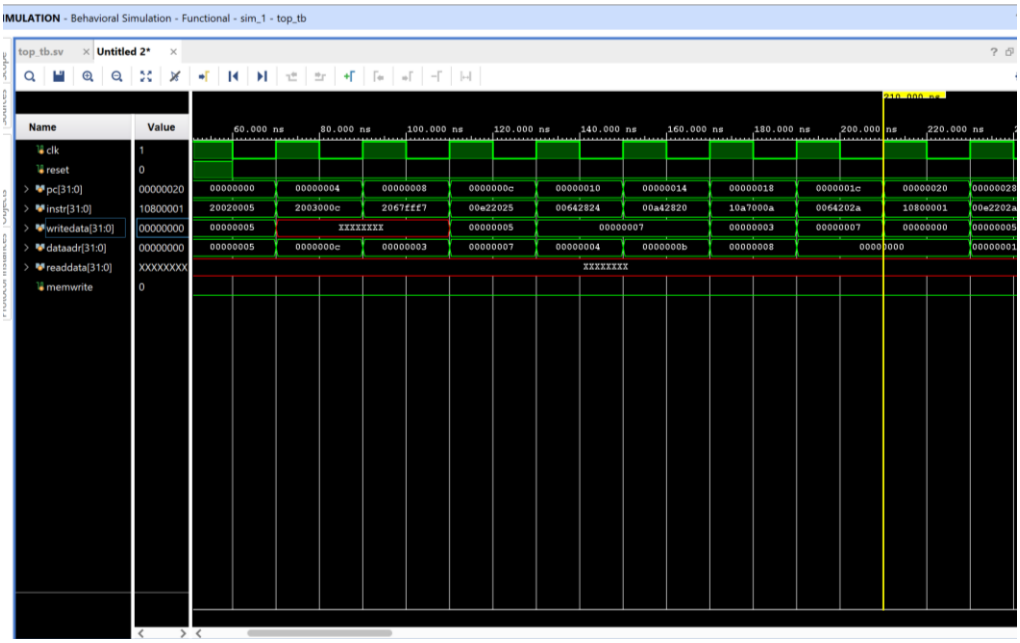
CS224
Lab 4
Section 3
İdil Atmaca
22002491
15/11/2022

Part 1.

a)

INSTRUCTION (HEX)	assembly language equivalent
20020005	addi $v0, $zero, 5
2003000c	addi $v1, $zero, 0xc
2067fff7	addi $a3, $v1, -9
00e22025	or $a0, $a3, $v0
00642824	and $a1, $v1, $a0
00a42820	add $a1, $a1, $a0
10a7000a	beq $a1, $a3, 0x44
0064202a	slt $a0, $v1, $a0
10800001	beqz $a0, 0x28
20050000	addi $a1, $zero, 0
00e2202a	slt $a0, $a3, $v0
00853820	add $a3, $a0, $a1
00e23822	sub $a3, $a3, $v0
ac670044	sw $a3, 0x44($v1)
8c020050	lw $v0, 0x50($zero)
08000011	j 0x44
20020001	addi $v0, $zero, 1
ac020054	sw $v0, 0x54($zero)
08000012	j 0x48

d)



e)

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

It will be the value that is hold by the rt register.

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

Because both 2003000c and 2067fff7 instructions are J type instructions. In a J type instruction, we do not write any data. RegWrite = 0 and MemWrite = 0 for a j function. That is why the writedata will be a don't care variable.

iii) *Why is readdata most of the time undefined?*

Because we mostly use ALUResult to put into a target register.If we have used lw more, we would see more defined values for readdata. Because lw uses the read value from the data memory to put into target register.

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

ALU result that will be written to rd. writedata = (RF[rs] op RF[rt]) where op is the corresponding ALU operation.

v) *In which instructions memwrite becomes 1?*

sw

f) Modified parts in the code

```
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; // SHIFT LEFT
          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'b011: 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)

jm:

IM[PC]

DM[RF[rs] + signImm] → PC

subi:

IM[PC]

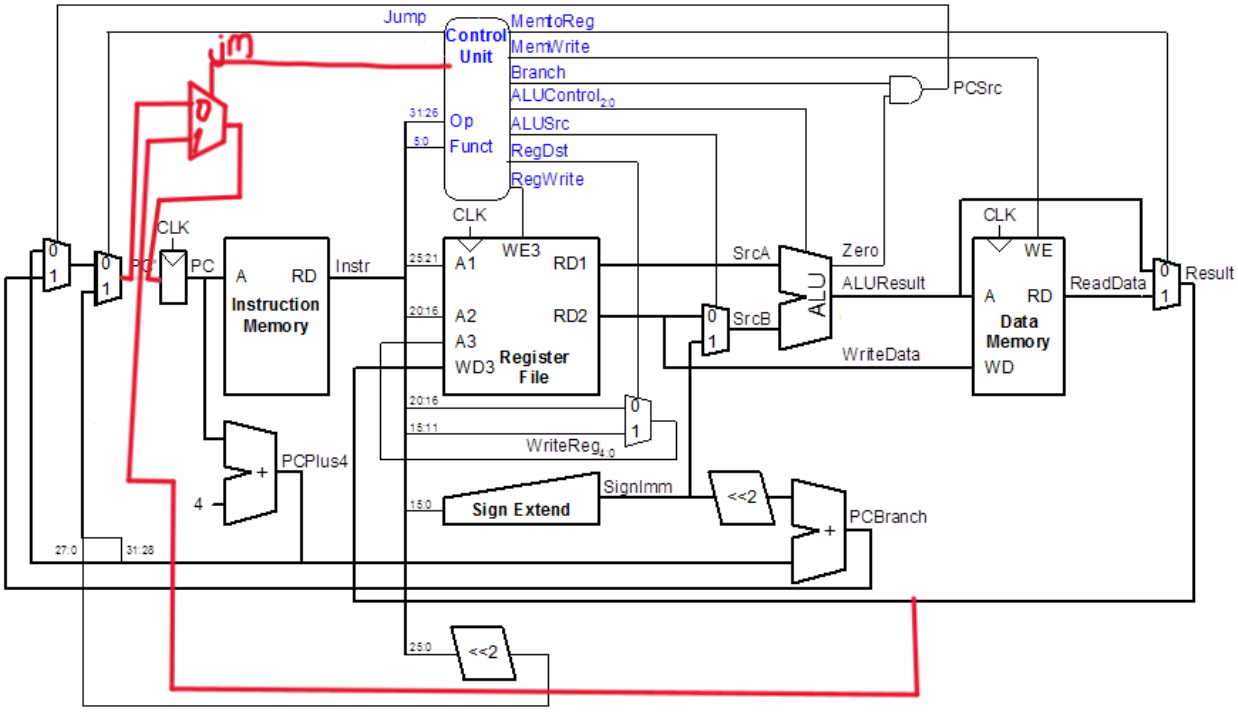
RF[rs] - SignImm → RF[rt]

PC → PC + 4

b)

jm: Hardware changes in the datapath are marked with red lines. We get imm[rs] value just like in lw instruction. Then, the address stored in imm[rs] will be read and put into a mux. This mux will be controlled by jm control signal.

subi: No need for addition or change in the hardware for the datapath, AluControl = 110 for subi function to be executed



c)

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp	Jump	JM
jm	110000	0	X	1	0	0	1	010	0	1
subi	110001	1	0	1	0	0	0	110	0	0

There is no need for a change in ALU decoder table since new instructions will use add (ALUOp= 010) and sub (ALUOp = 110) functions which are already available in the ALU.