CS224
Lab No: 4
Section No: 06
Barış Tan Ünal
22003617

PART 1

Part 1.a

Location	Machine Instruction	Assembly Language
0x80001000	0x20020005	addi $2, $0, 5
0x80001004	0x2003000c	addi $3, $0, 12
0x80001008	0x2067fff7	addi $7, $3, -9
0x8000100c	0x00e22025	or $4, $7, $2
0x80001010	0x00642824	and $5, $3, $4
0x80001014	0x00a42820	add $5, $5, $4
0x80001018	0x10a7000a	beq $5, $7, 0x80001044
0x8000101c	0x0064202a	slt $4, $3, $4
0x80001020	0x10800001	beq $4, $0, 0x80001028
0x80001024	0x20050000	addi $5, $0, 0
0x80001028	0x00e2202a	slt $4, $7, $2
0x8000102c	0x00853820	add $7, $4, $5
0x80001030	0x00e23822	sub $7, $7, $2
0x80001034	0xac670044	sw $7, 68($3)
0x80001038	0x8c020050	lw $2, 80($0)
0x8000103c	0x08000011	j 0x80000044
0x80001040	0x20020001	addi $2, $0, 1
0x80001044	0xac020054	sw $2, 84($0)
0x80001048	0x08000012	j 0x80000048

Part 1.d

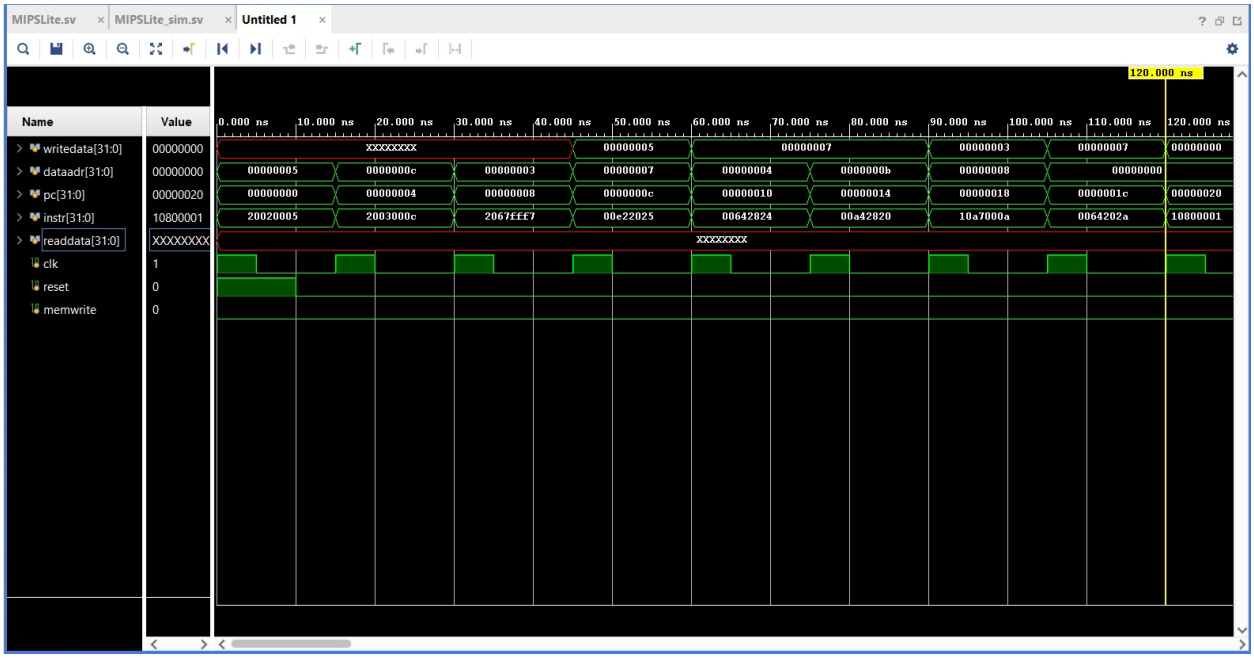

Figure 1. Interval of 0 - 120ns of the waveform.

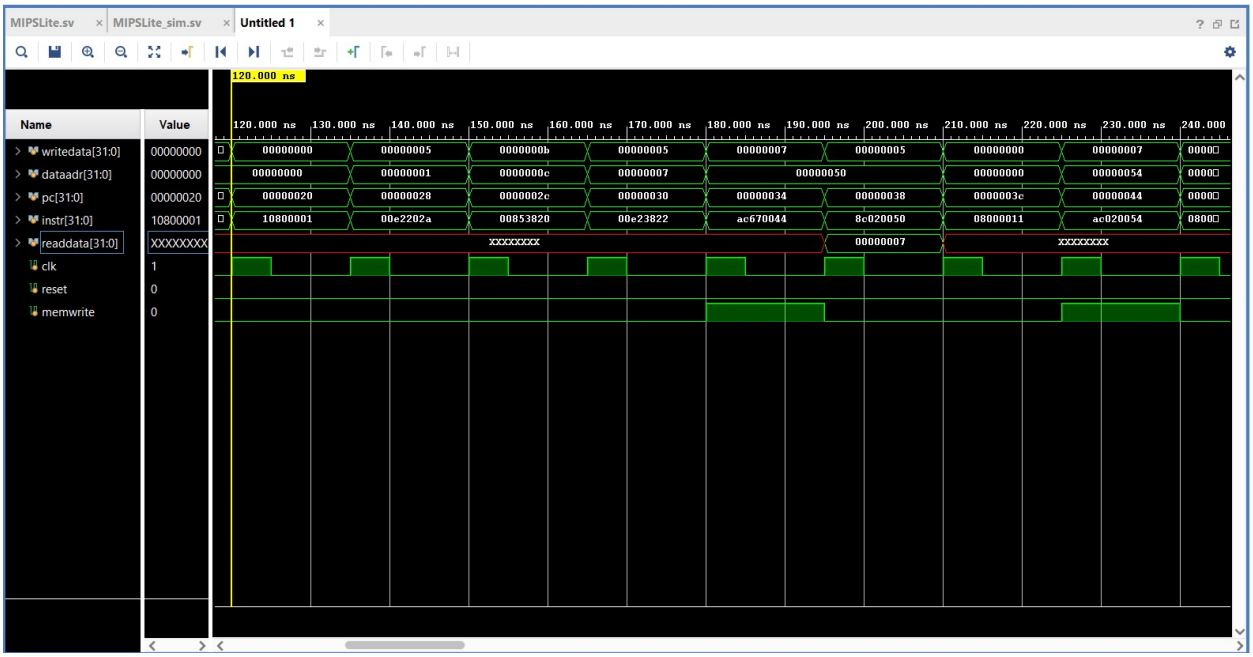

Figure 2. Interval of 120 - 240ns of the waveform.

Part 1.e

i) In an R-type instruction, *writedata* represents the *rd* register, which is *instr[15:11]*.

ii) *writedata* is undefined at the beginning of the program, because the first three instructions are I-type instructions (*addi*). I-type instructions do not use rd register, instead they use immediate (16-bits). The first R-type instruction, where *writedata* was defined for the first time, is the *or* instruction where PC = 0000_000c.

iii) readdata is undefined for most of the time because mips processor reads data from memory only in *lw* instruction for original10 instruction set.

iv) In an R-type instruction, *dataadr* corresponds to the result (32-bit output) of the ALU.

v) *memWrite* becomes 1 in the instrcutions where something has to be written to the memory. The only example in original10 instrcution set is *sw*.

Part 1.f

```
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

Part 2.a

RTL Design of sracc:

IM [PC]
RF [rd] ← RF [rd] + (RF [rs] << RF [rt])
PC ← PC + 4

RTL Design of jm:

IM [PC]
PC ← RF [rs] + (SignExt(imm) * 4)

Part 2.b

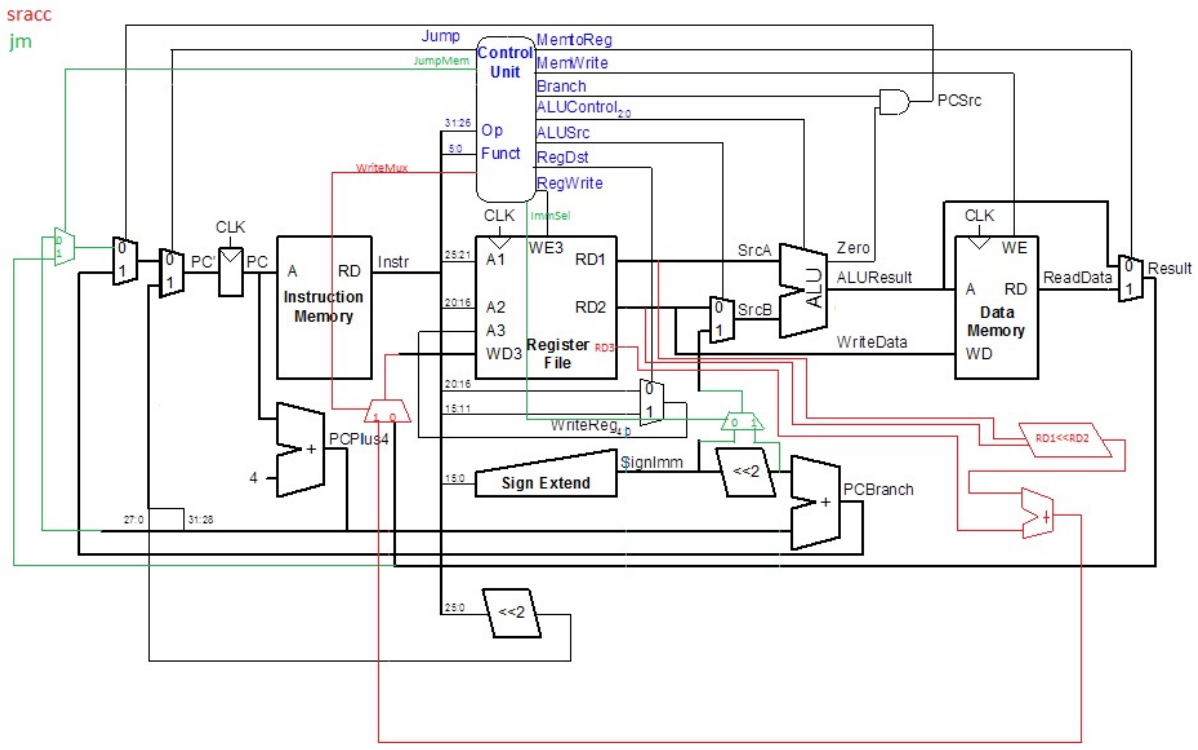

Figure 3. Updated datapath with sracc & jm instructions

Part 2.c

Inst	Op Code	Reg Write	Reg Dst	ALU Src	Branch	Mem Write	Mem toReg	ALU Op	Jump	Jump Mem	Imm Sel	Write Mux
R-type	000000	1	1	0	0	0	0	10	0	0	X	0
lw	100011	1	0	1	0	0	1	00	0	0	0	0
sw	101011	0	X	1	0	1	X	00	0	0	0	X
beq	000100	0	X	0	1	0	X	01	0	0	X	X
addi	001000	1	0	1	0	0	0	00	0	0	0	0
j	000010	0	X	X	X	0	X	XX	1	0	X	X
sracc	000111	1	1	X	0	0	X	XX	0	0	X	1
jm	000011	0	X	1	0	0	0	00	0	1	1	X