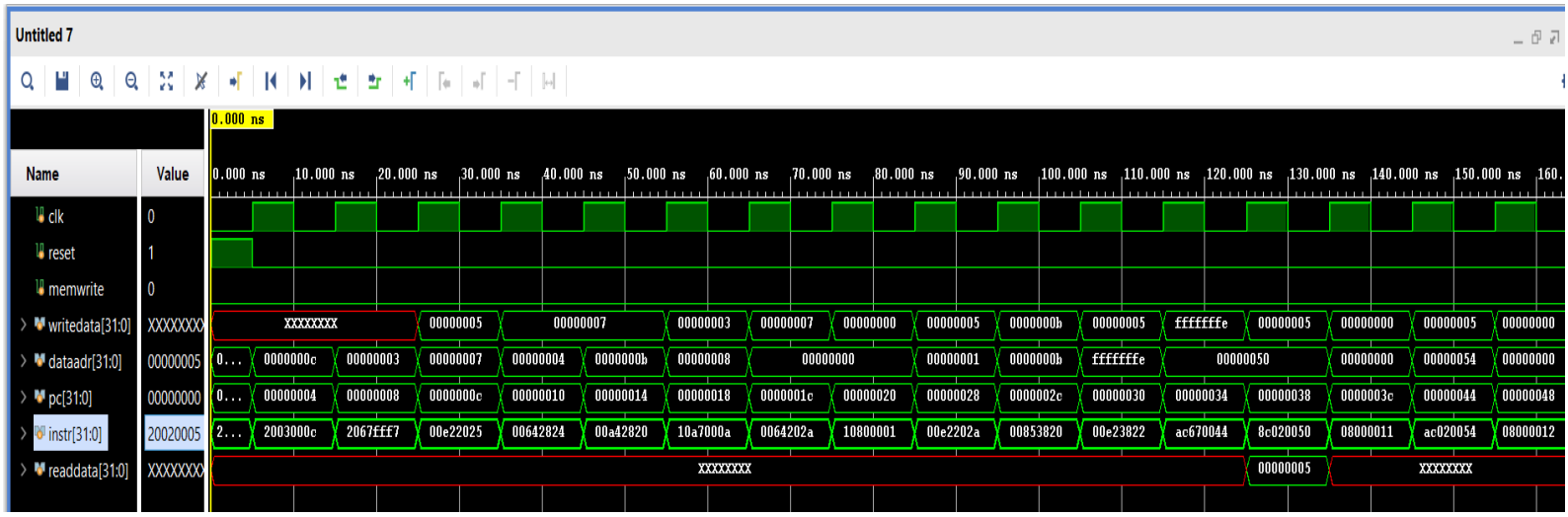
CS224
Lab 4.
Section 2.
Barkın Saday
21902967

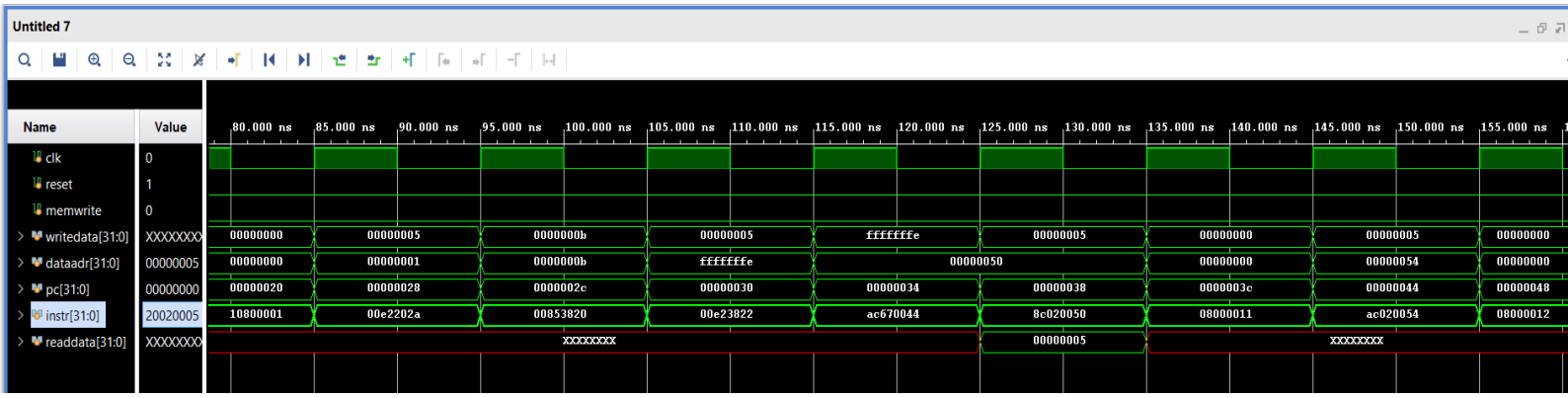
Part 1

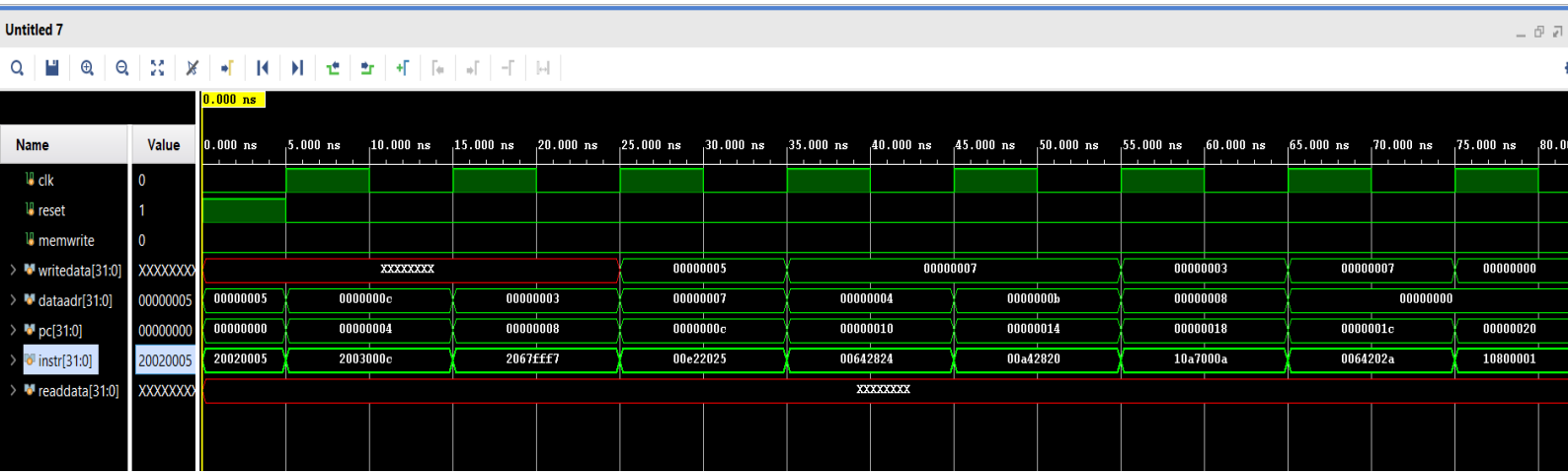
1.a)

Machine Instruction	MIPS Equivalent
0x20020005	addi $v0, $zero, 5
0x2003000c	addi $v1, $zero, 12
0x2067fff7	addi $a3, $v1, 0xfff7
0x00e22025	or $a0, $a3, $v0
0x00642824	and $a1, $v1, $a0
0x00a42820	add $a1, $a1, $v0
0x10a7000a	beq $a1, $a3, 0x000a
0x0064202a	slt $a0, $v1, $a0
0x10800001	beq $a0, $zero, 0x0001
0x20050000	addi $a1, $zero, 0
0x00e2202a	slt $a0, $a3, $v0
0x00853820	add $a3, $a0, $a1
0xh00e23822	sub $a3, $a3, $v0
0xac670044	sw $a3, 68($v1)
0x8c020050	lw $v0, 80($zero)
0x08000011	j 0x0000011
0x20020001	addi $v0, $zero, 1
0xac020054	sw $v0, 84($zero)
0x08000012	j 0x0000012

1.d)







-All three pictures are from the same simulation but different perspectives (different scales or different time periods)

1.e)

i) In an R-Type instruction “writedata” corresponds to register source register (t register)

ii) The first instructions are all I-Type instructions which causes “writedata” to be undefined for a time.

iii) We use “readdata” when we are reading data. For example a MIPS instruction equivalent is lw instruction where we read the data from register. Since we do not use lw (we do not read the data) most of times it is undefined most of the time.

iv) “dataadr” in an I-Type instruction corresponds to immediate value

v) “memwrite” becomes 1 when we are writing to memory. Enables storing instructions.

1.f)

-Modifed line is shown with comment

```
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’b001: result = a<<b; // modifed …...
           default: result = {32{1'bx}};
       endcase

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

Part 2

2.a)

RTL Expression for **SRACC**:

IM[PC]

RF[rs] ← RF[rs] << RF[rt]

RF[rd] ← RF[rd] + RF[rs]

PC ← PC + 4

RTL Expression for **subi**:

IM[PC]

RF[rt] ← RF[rs] - SignExt(immed)

PC ← PC + 4

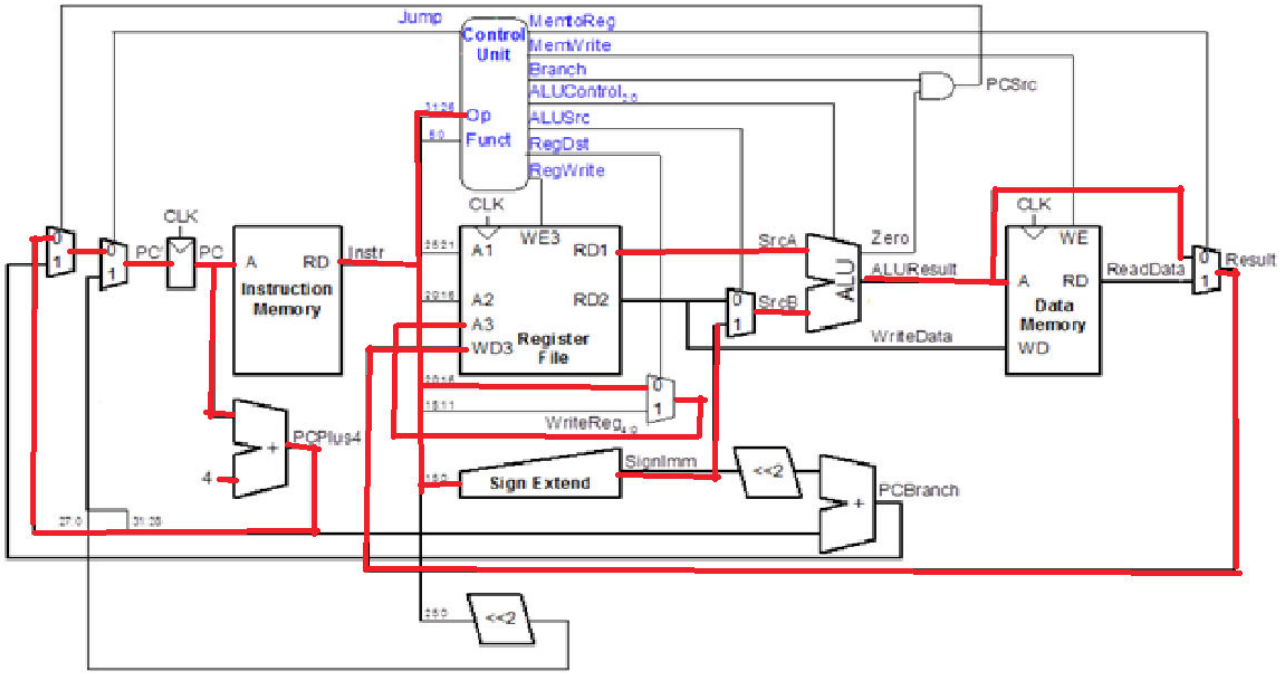
2.b)

sracc:

-There are many ways to implement this new instruction. One is by calling already existing instructions one after the other so we will not need aditional hardware changes. As shown in the RTL expression if we we first use a shift instruction and than an add instruction we can implement the sracc instruction. However, for the shit instruction we get the amount to be shifted from the rt register instead of shamt. So we need to use the data in rt register and instead of just looking to the shamt from the machine code.

subi:

-No additional hardware changes needed to the datapath since everything will be as performing addi instruction except instead of addition we will apply a subtraction. The change in the Control Unit is ALUControl is set to 110 to perform a subtraction. So the only changes we made is in the path of addi.



2.c)

Instruction	$OP_{5:0}$	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	$ALUOp_{1:0}$	Jump
sracc	000000	1	1	1	X	1	1	00	0
subi	001110	1	0	0	0	0	0	01	0