CS 224
LAB 4

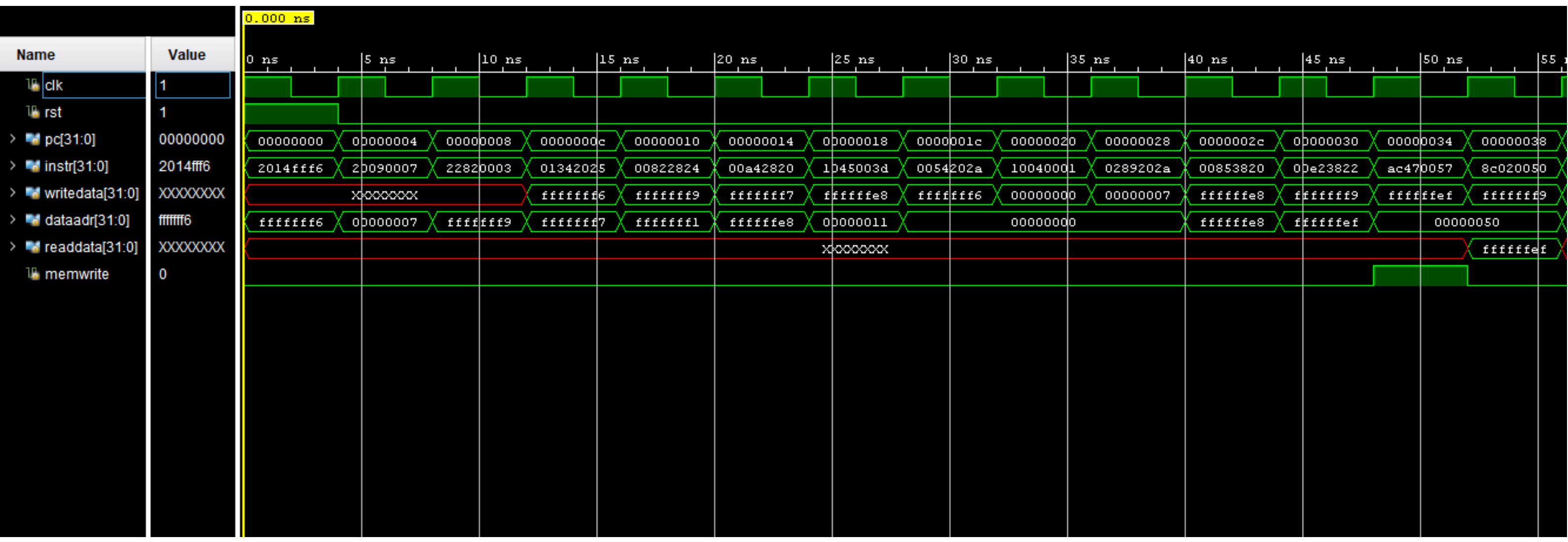
SECTION 6

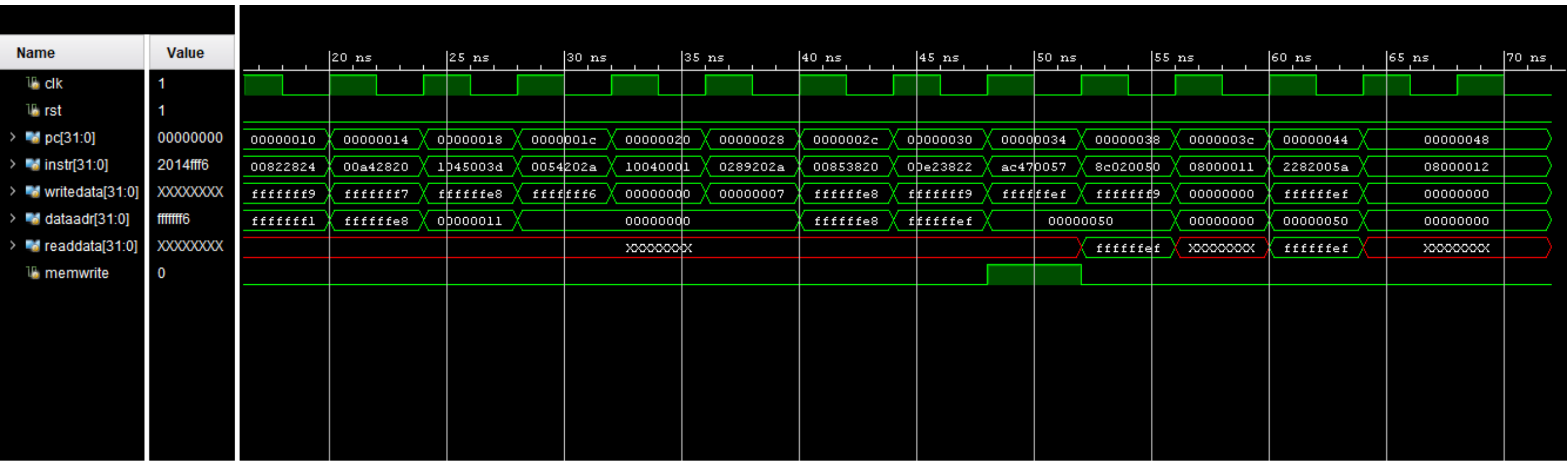
YASEMİN AKIN
22101782

1. a

Location (Hex)	Machine Instruction (Hex)	Assembly Language Equivalent
00	2014fff6	addi $s4, $zero, 0xfff6
04	20090007	addi $t1, $zero, 0x0007
08	22820003	addi $v0, $s4, 0x0003
0C	01342025	or $a0, $t1, $s4
10	00822824	and $a1, $a0, $v0
14	00a42820	add $a1, $a1, $a0
18	1045003d	beq $v0, $a1, 0x003d
1C	0054202a	slt $a0, $v0, $s4
20	10040001	beq $zero, $a0, 0x0001
24	00002820	add $a1, $zero, $zero
28	0289202a	slt $a0, $s4, $t1
2C	00853820	add $a3, $a0, $a1
30	00e23822	sub $a3, $a3, $v0
34	ac470057	sw $a3, 0x0057($v0)
38	8c020050	lw $v0, 80($zero)
3C	08000011	j 0x0011 ≅ j 44
40	20020001	addi $v0, $zero, 1
44	2282005a	addi $v0, $s4, 0x005a
48	08000012	j 0x0012 ≅ j 48

1. d


0.000 ns
Name
Value
clk 1
rst 1
pc[31:0] 00000000
instr[31:0] 2014fff6
writedata[31:0] XXXXXXXX
dataadr[31:0] fffffff6
readdata[31:0] XXXXXXXX
memwrite 0
0 ns
5 ns
10 ns
15 ns
20 ns
25 ns
30 ns
35 ns
40 ns
45 ns
50 ns
00000000 00000004 00000008 0000000c 00000010 00000014 00000018 0000001c 00000020 00000028 0000002c 00000030 00000034 00000038
2014fff6 20090007 22820003 01342025 00822824 00a42820 1045003d 0054202a 10040001 0289202a 00853820 00e23822 ac470057 8c020050
XXXXXXXX fffffff6 fffffff9 fffffff7 ffffffe8 fffffff6 00000000 00000007 ffffffe8 fffffff9 ffffffef fffffff9
fffffff6 00000007 fffffff9 fffffff7 fffffff1 ffffffe8 00000011 00000000 ffffffe8 ffffffef 00000050
XXXXXXXX ffffffef


Name
Value
clk 1
rst 1
pc[31:0] 00000000
instr[31:0] 2014fff6
writedata[31:0] XXXXXXXX
dataadr[31:0] fffffff6
readdata[31:0] XXXXXXXX
memwrite 0
20 ns
25 ns
30 ns
35 ns
40 ns
45 ns
50 ns
55 ns
60 ns
65 ns
70 ns
00000010 00000014 00000018 0000001c 00000020 00000028 0000002c 00000030 00000034 00000038 0000003c 00000044 00000048
00822824 00a42820 1045003d 0054202a 10040001 0289202a 00853820 00e23822 ac470057 8c020050 08000011 2282005a 08000012
fffffff9 fffffff7 ffffffe8 fffffff6 00000000 00000007 ffffffe8 fffffff9 ffffffef fffffff9 00000000 ffffffef 00000000
fffffff1 ffffffe8 00000011 00000000 ffffffe8 ffffffef 00000050 00000000 00000050 00000000
XXXXXXXX ffffffef XXXXXXXX ffffffef XXXXXXXX

1. e

i. The second operand, which is the rt register in assembly language format, is writedata.

ii. Because the first 3 instructions, where writedata is undefined, are not R-type instructions. Instead they are I-type instructions.

iii. For sw instructions, memwrite changes to 1.

iv. It's likely that the ALU is conducting the addition operation using two's complement arithmetic, which is frequently employed for signed integers in computer systems if the result arriving from the ALU in a single cycle MIPS processor is -24 instead of 24. The result could be calculated as -24 by treating the decimal numbers 9 and 15 as signed integers. The ALU should conduct the addition using unsigned arithmetic or interpret the operands as unsigned integers in order to yield the desired result of 24. To accomplish this, use the addu instruction instead of the add instruction, which conducts unsigned addition.

v. When we read from the memory (RAM) by giving specific address values to instructions or by using the lw instruction which is an instruction that loads a word from the memory directly, the output of read data becomes defined because we need to read the data for these instructions to take place properly.

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’b011: 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;
          default: result = {32{1'bx}};
       endcase

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

2. a

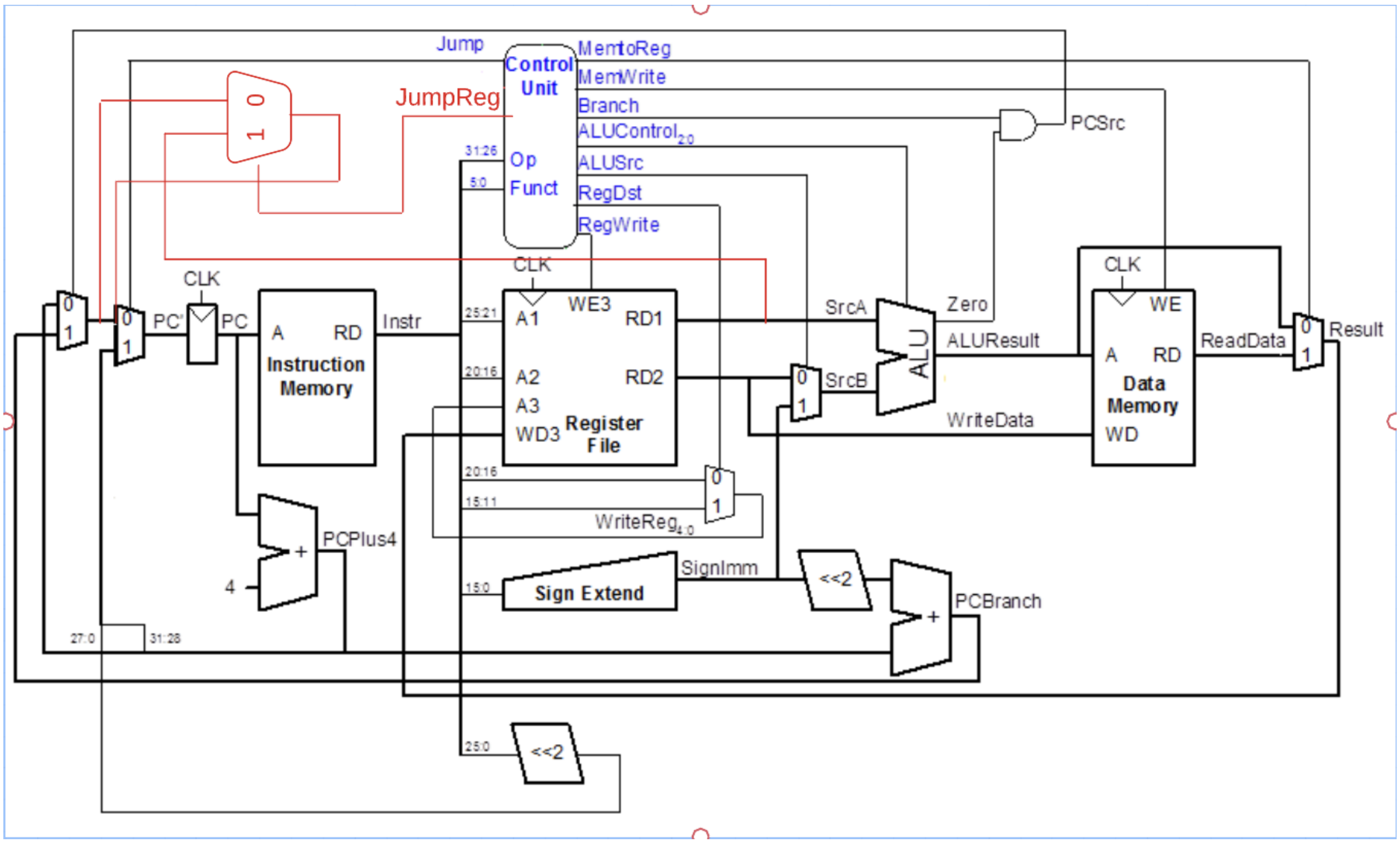
jr

IM[PC]
PC ← RF[rs]

xori

IM[PC]
ALUResult ← RF[rs] XOR SignExt(immed)
RF[rs] ← ALUResult
PC ← PC + 4

2. b



2. c

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp	Jump	JumpReg
R-type	000000	1	1	0	0	0	0	10	0	0
lw	100011	1	0	1	0	0	1	00	0	0
sw	101011	0	X	1	0	1	X	00	0	0
beq	000100	0	X	0	1	0	X	01	0	0
addi	001000	1	0	1	0	0	0	00	0	0
j	000010	0	X	X	X	0	X	XX	1	0
jr	000011	X	0	X	0	X	X	XX	0	1
xori	001110	1	0	1	0	0	0	11	0	0

ALUOp	Funct	ALUControl
00	X	010 (add)
01	X	110 (subtract)
10	100000 (add)	010 (add)
10	100010 (sub)	110 (subtract)
10	100100 (and)	000 (and)
10	100101 (or)	001 (or)
10	101010 (slt)	111 (set less than)
11	X	011 (xor)