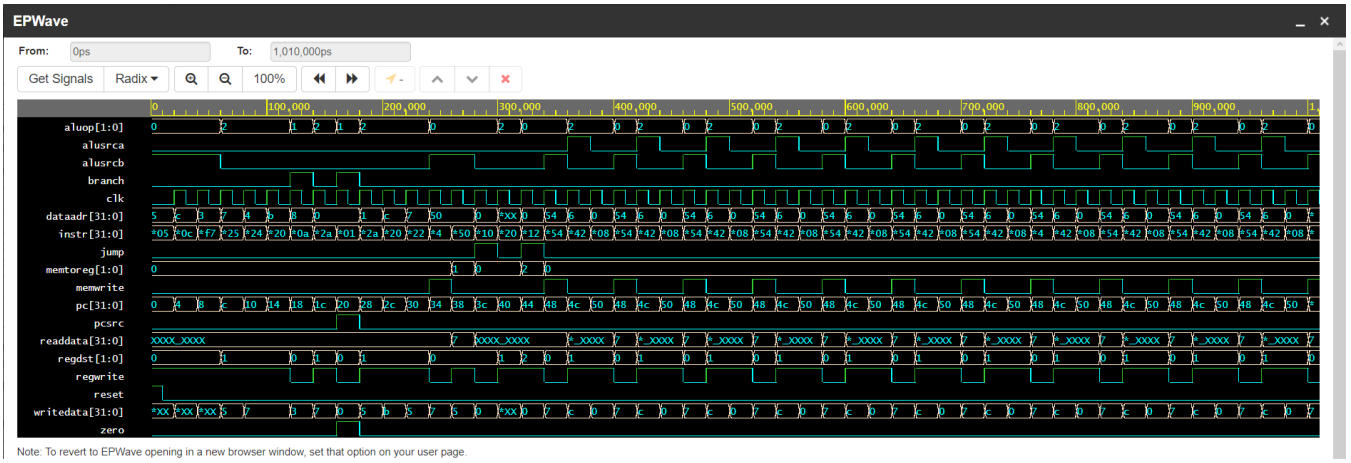
CS224
Lab 04
Section 01
Arda Önal
21903350

Part 1:
a)

Address (hex)	Machine Language (hex)	Assembly language
00	20020005	addi $v0, $0, 5
04	2003000c	addi $v1, $0, 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, 0x44
1c	0064202a	slt $a0, $v1, $a0
20	10800001	beq $a0, $0, 0x28
24	20050000	addi $a1, $0, 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($0)
3c	08000010	j 0x40
40	001f6020	add $t4, $0, $ra
44	0c000012	jal 0x48
48	ac020054	sw $v0, 84($0)
4c	00039042	srl $s2, $v1, 1
50	03E00008	jr $ra

e)



f)

i) In an R-type instruction, writedata is the output of RD2 of register file which corresponds to Rf[rt]. The address of it is in instruction[20:16] bits.

ii) Writedata is undefined for some of the early instructions in the program because the values in register are not initialized and when we try to read Rf[rt], it is undefined. For example, the first three instructions have the Rf[rt] as $v0, $v1, $a3 and the data of these registers are not initialized which corresponds to undefined writedata.

iii) Readdata is undefined most of the time because memory is not initialized in the implementation of the data memory. The only times where readdata has a value is when we do sw instruction and store some data into data memory.

iv) In an R-type instruction, dataadr corresponds to the cable that comes out of the alu which is named ALUResult in the datapath which is the result of operation that is executed on Rf[rs] and Rf[rt]. It holds the value that is going to be written in Rf[rd] in R-type instructions.

v) Dataadr becomes undefined in the instruction 001f6020 which corresponds to add $t4, $0, $ra. Data address holds the alu result which in this case is the addition of the values inside $0 and $ra. Since $ra and $0 is not initialized, both of them are undefined and the result of their addition which is dataadr is undefined. In other instructions, we use $0 with other registers that are initialized which results in defined values.

g)

i) We have to make changes to the alu decoder in the controller to support srlv instruction. srlv is a R-Type instruction so, we must add a case to aludec where the function is 6'b000110. Then, we need to modify the alu so that it will do the srlv operation with a

specified opcode in the aludec. Rf[rs] and Rf[rt] will be selected as the data coming into the alu because in our implementation, R-type instructions select SrcB as Rf[rt] which is already implemented in the solution. Lastly, the alu result will be written into Rf[rd] because in our implementation, the alu result is written into Rf[rd] in R-type instructions. RegDst will be 01 which will select rd of the instruction and MemToReg will be 00 so that alu result is selected and written into Rf[rd].

ii) We would modify aludec and alu modules to do the sll instruction. We have to specify a new alucode according to the function of the sll then, we have to add a new instruction to alu with our new opcode which will do the sll operation. Since we already have srl operation implemented, we do not have to do other changes to any modules.

Part 2:

a) Full RTL Expression for sw+:

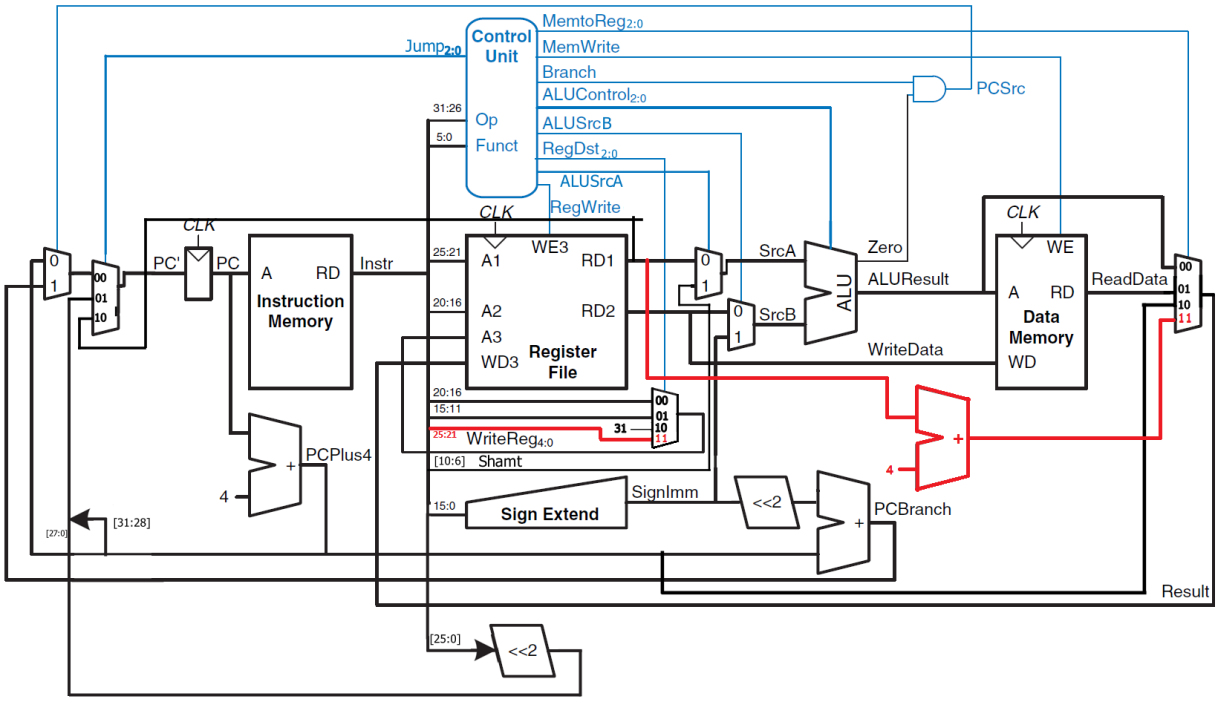
IM[PC]

DM[RF[rs] + SignExt(immed)] ← RF[rt]

RF[rs] ← RF[rs] + 4

PC ← PC + 4

b)



c)

Instruction	Opcode	RegWrite	RegDst	ALUSrcA	ALUSrcB	Branch	MemWrite	MemToReg	ALUOp	Jump
R-type	000000	1	01	0	0	0	0	00	10	00
srl	000000	1	01	1	0	0	0	00	10	00
lw	100011	1	00	0	1	0	0	01	00	00
sw	101011	0	X	0	1	0	1	XX	00	00
beq	000100	0	X	0	0	1	0	01	01	00
addi	001000	1	00	0	1	0	0	00	00	00
j	000010	0	X	X	X	X	0	XX	XX	01
jal	000011	1	10	X	X	X	0	10	XX	01
jr	000000	1	01	0	0	0	0	00	10	10
sw+	101100	1	11	0	1	0	1	11	00	00