CS224
Lab No.4
Section No.3
NURETTİN ONUR VURAL
21902330

a)

Location (hex)	machine instruction (hex)	assembly language equivalent
8'h00	32'h20020005	addi $v0, $zero, 0x5
8'h04	32'h2003000c	addi $v1, $zero, 0xc
8'h08	32'h2067fff7	addi $a3, $v1, 0xfff7
8'h0c	32'h00e22025	or $a0, $a3, $v0
8'h10	32'h00642824	and $a1, $v1, $a0
8'h14	32'h00a42820	add $a1, $a1, $a0
8'h18	32'h10a7000a	beq $a1, $a3, 0x000a
8'h1c	32'h0064202a	slt $a0, $v1, $a0
8'h20	32'h10800001	beq $a0, $zero, 0x0001
8'h24	32'h20050000	addi $a1, $zero, 0x0000
8'h28	32'h00e2202a	slt $a0, $a3, $v0
8'h2c	32'h00853820	add $a3, $a0, $a1
8'h30	32'h00e23822	sub $a3, $a3, $v0
8'h34	32'hac670044	sw $a3, 0x44($v1)
8'h38	32'h8c020050	lw $v0, 0x50($zero)
8'h3c	32'h08000010	j 0x10
8'h40	32'h001f6020	add $t4, $zero, $ra
8'h44	32'h0c000012	jal 0x12
8'h48	32'hac020054	sw $v0, 0x54($zero)
8'h4c	32'h00039042	srl $s2, $v1, 0x0001
8'h50	32'h03E00008	jr $ra

Table 1: hardcoded MIPS instructions

e)

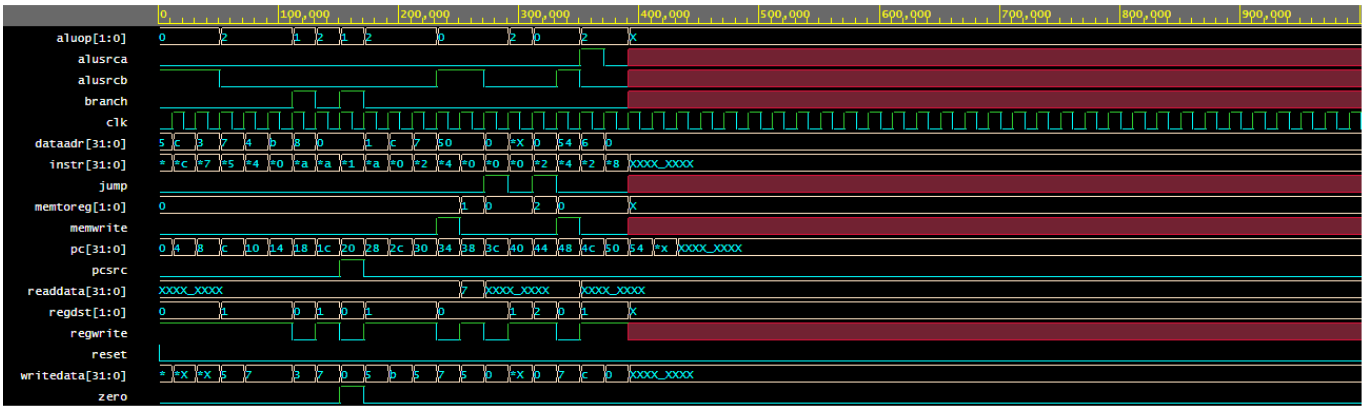

Figure 1: Waveform before jr instruction implementation

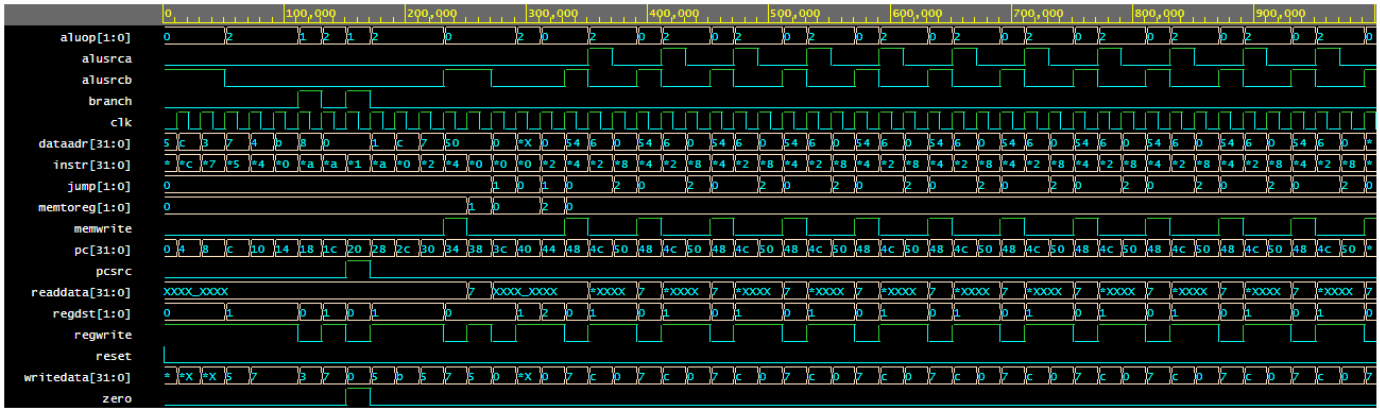

Figure 2: Waveform after jr instruction implementation

f)

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

- The writedata is the data corresponding to the desired address of rt, in other words the rd2 of RF. In R type, the fetched instruction is used to determine readport addresses from RF and as a following step, it selects the data which is represented by rt address which can be seen in the waveform as the writedata.

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

- writedata is undefined as in the beginning we fetch three addi instructions and they are I type instructions(addi) so that we never call the address and read data on readPort2. We only use readAdress2 (rs), therefore as the address two read data from second port of RF is never given, the data value that must be read from readPort2, **which is writedata**, of RF is undefined.

iii) Why is readdata most of the time undefined?

- readdata comes out to be the output of Data Memory and accordingly it gets the address of this data from ALU operation as it is connected to the address (a) as an input. The ALU operation must be “meaningful”, in other words it must be arranged to give the address of the desired DM location. This happens in the case when the address is calculated by getting RD1 as SourceA and the sign extended value of immediate field and adding them.

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

It comes out to be the result of the ALU operation with RD1 selected as SrcA, RD2 selected as SrcB

v) Why does dataaddress become undefined at some point?

It becomes undefined at the point of instruction 0x20 as there is happening an addition we do not want to get a wrong value from the data memory that can be obtained from the ALU result, as WE for data memory is not active, the addresses is shown as don’t care.

g)

i) Do you need to make any changes to support srlv instruction?

-We can implement srlv instruction by taking RD1 as SrcA, and RD2 as SrcB and using the ALU as it has the option to perform srl and therefore the instruction can function with the present form.

ii) Which module would you modify to support sll instruction? How would you modify it?

-In order to make sll function properly the ALU still can be used but it needs changes in such a way that it makes it possible to get IM[10:6] as an input to ALU so that it understands the shift amount. And accordingly new ALUop and control signals are required for the operation in the controller.

PART 2 – IMPLEMENTING JALM

PC = DM[RF[rs] + SignExtImm], RF[rt] = PC + 4

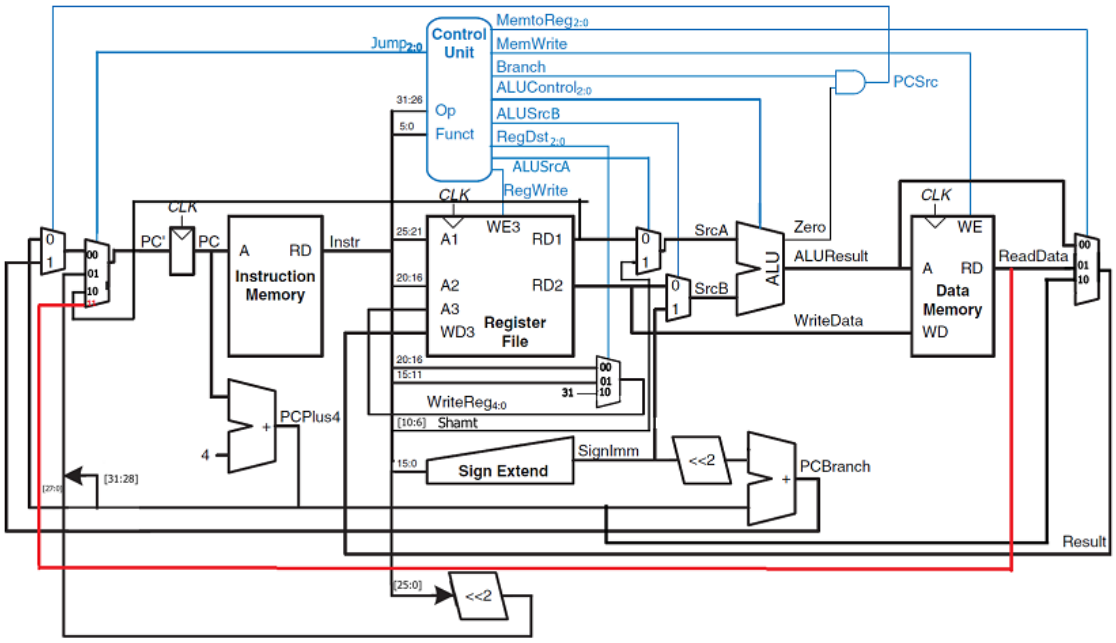

Figure 3: The upgraded datapath (changes marked in red)

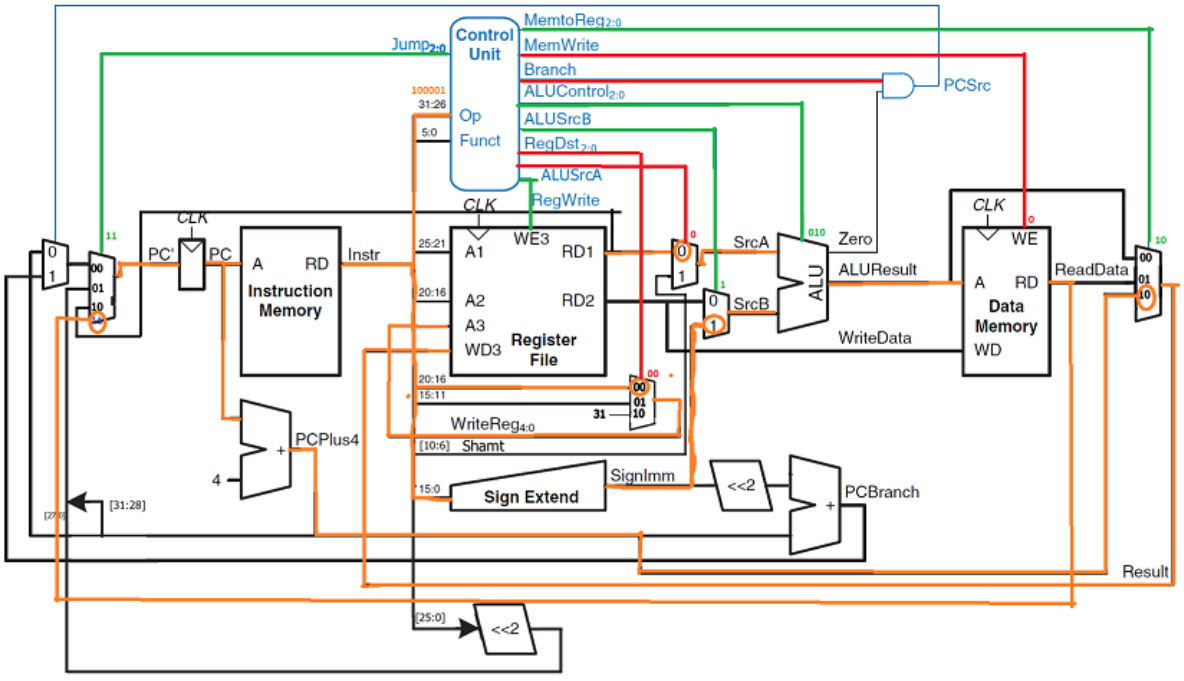

Figure 4: The detailed diagram of upgraded datapath showing how jalm functions

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
jalm	100001	1	00	0	1	0	0	10	00	11

Table 2: Controller Changes (marked in red)