CS224

Lab No: 4

Section No: 5

Full Name: Esad İsmail Tök

Bilkent ID: 21801679

Part 1 - a

Location Address (hex)	Machine Instruction (hex)	MIPS Assembly Equivalent
00	0x20020005	addi $v0, $zero, 5
04	0x2003000c	addi $v1, $zero, 12
08	0x2067fff7	addi $a3, $v1, -9
0c	0x00e22025	or $a0, $a3, $v0
10	0x00642824	and $a1, $v1, $a0
14	0x00a42820	add $a1, $a1, $a0
18	0x10a7000a	beq $a1, $a3, 10
1c	0x0064202a	slt $a0, $v1, $a0
20	0x10800001	beq $a0, $zero, 1
24	0x20050000	addi $a1, $zero, 0
28	0x 00e2202a	slt $a0, $a3, $v0
2c	0x00853820	add $a3, $a0, $a1
30	0x00e23822	sub $a3, $a3, $v0
34	0xac670044	sw $a3, 68($v1)
38	0x8c020050	lw $v0, 80($zero)
3c	0x08000010	J 16
40	0x001f6020	add $t4, $zero, $ra
44	0x0c000012	jal 18
48	0xac020054	sw $v0, 84($zero)
4c	0x00039042	srl $s2, $v1, 1
50	0x03e00008	jr $ra

Table 1: Instruction Table

Part 1 - e

- *Figure 1* below is the screenshot of the waveform that is the simulation result of the single-cycle MIPS Processor.

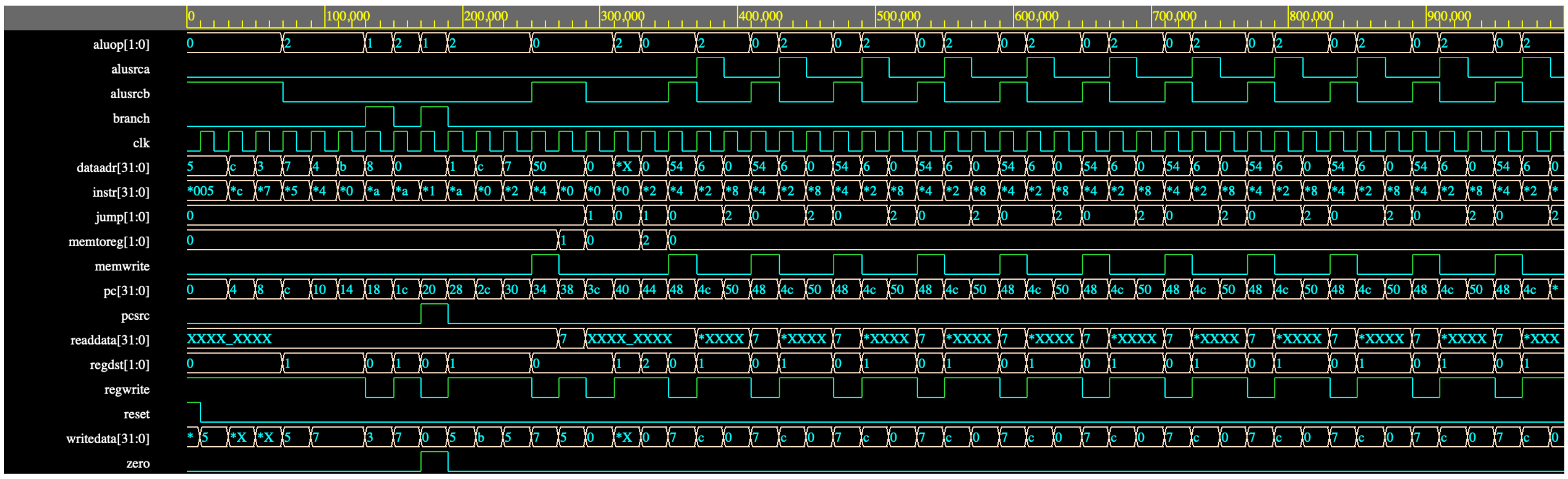

Figure 1: Screenshot of the waveform

Part 1 - f

i) In R-type instructions writedata corresponds to RF[rt], namely the second read data output of the register file, which is RD2. Writedata is also connected to the first input of the mux that selects the SrcB of the ALU. Moreover writedata is the data input of the write port of the data memory.

ii) Before program counter (PC) gets the value of 0x0c, the instructions are all I-type “addi” instructions, and the only register that is to be read is the “rs” register. Since “rt” register is the destination register but not a read register, the signal writedata, namely RF[rt], is undefined for some of the early instructions until the processor finds an R-type instruction that also reads from the “rt” register

iii) Readdata is the data output of the data memory and this signal can only be set when a data is read from the memory, namely readdata signal should be undefined if the “lw” instruction is not used. Therefore, since “lw” instruction is only used one time, the readdata signal is undefined most of the times except the clock cycle in which “lw” instruction is executed.

iv) In R-type instructions dataadr corresponds to the result of ALU, namely the ALUResult signal. Dataadr signal is also the input address of the data memory and this signal also connected to the multiplexer that selects the correct signal for the write data, WD3, of the register file.

v) When the program counter (PC) hold the value 0x40, the instruction in that address is “add $t4, $zero, $ra”. However until this instruction the register “$ra” has not been initialized because the “jal” instruction has not been used before. Therefore the second input of the ALU is undefined. And the addition instruction with an undefined value is also undefined. Thus, the result of the ALU, namely dataaddress is also undefined in that point.

Part 1 - g

i) “srlv” instruction is a similar instruction with “srl” and the only difference is the shift amount is read from a register. Therefore there is no need to make any change in the datapath. Since it is an R-type instruction there is also no need to change main decoder. Bun in the aludec module the code snipped “6'b000110: alucontrol = 3'b011; “ needs to be added to represent the function code of the “srlv” instruction which is 000110. The alucontrol signal do not need to be changed since we again want our ALU to do a shift right logic operation and since this time the code snipped “assign alusrca = (funct == 6’b000010);” in the controller module make the “alusrca” select signal to 0 and therefore the mux that selects the first input of the ALU chooses the value RF[rs] instead of the shift amount and thus we can perform the “srlv”. Therefore there is no need to change the datapath but we only need to add the function code of the “srlv” into the ALU Decoder and assert the alucontrol signal for srl as well.

ii) For the “sll” instruction first of all we need to add its function code to the ALU Decoder, namely, aludec module, and we need to decide a new ALUControl signal to represent the code for the “sll” instruction. We do not need to change main decoder module since “sll” is an R-type instruction and cen be fully represented with the signals of R-type instructions. Then we need to adjust the alu module by adding a new case statement to the switch-case block. This case makes the shift left logic by the code “result = b << a;” and also this case makes the zero signal 0 by the code “zero = 0;”. Those are the only necessary changes that is made in the processor to support “sll” instruction.

Part 2 - a

- The RTL expression of the “ble” instruction:

```
IM[PC]
if (RF[rs] < RF[rt])
        PC ← BTA
else if (RF[rs] == RF[rt])
        PC ← BTA
else
        PC ← PC + 4
```

Part 2 - b

- The image below is the updated datapath in order to support the “ble” instruction.

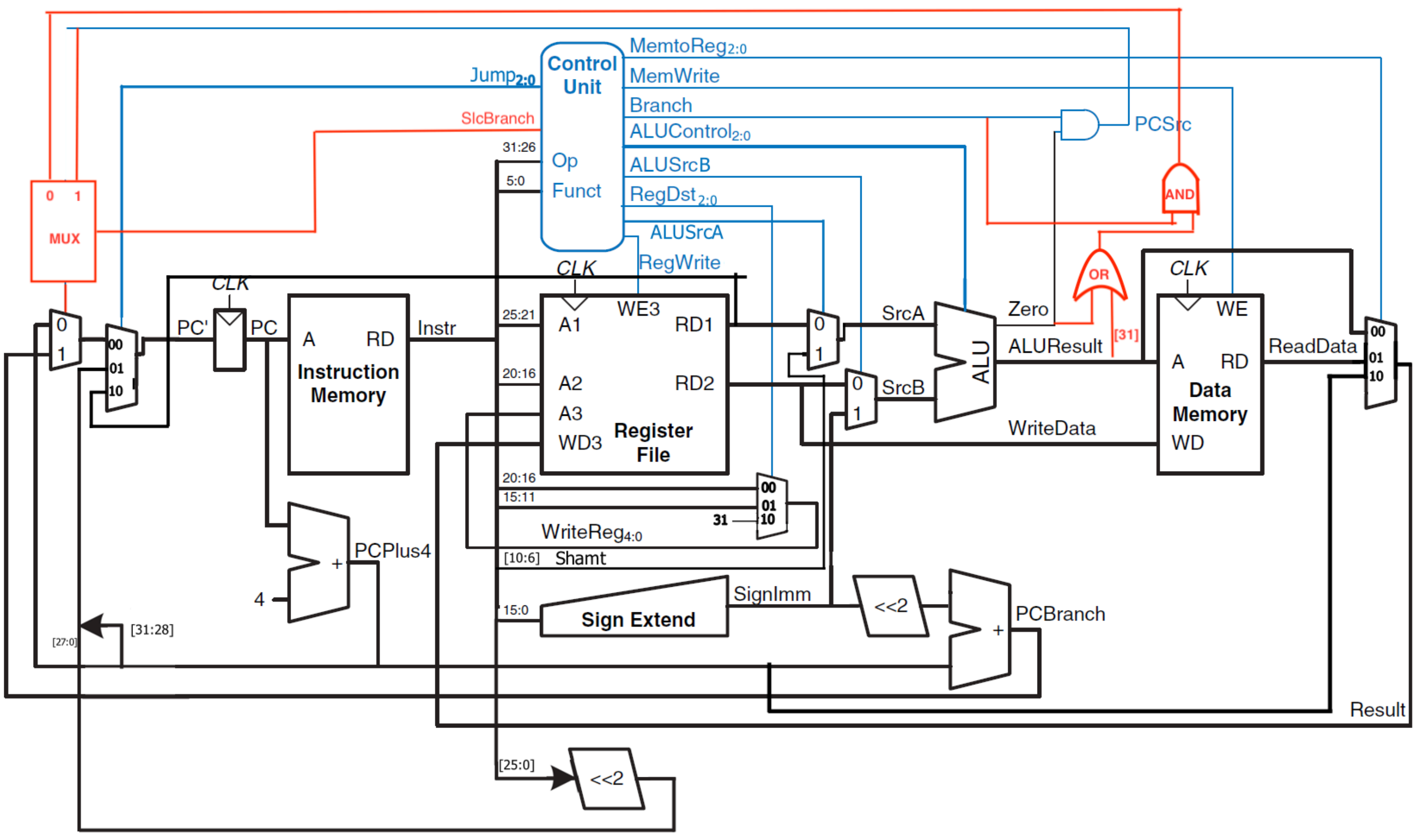

Figure 2: Updater version of the datapath

Part 2 - c

- The table below is the updated main decoder's table after the addition of the "jr" and "ble" instructions.

- There is no need to update the ALU Decoder since the new instructions do not require the ALU Decoder to be changed.

Instruction	Opcode	RegWrite	RegDst	ALUSrcA	ALUSrcB	Branch	MemWrite	MemToReg	ALUOp	Jump	SlcBranch
R-type	000000	1	01	0	0	0	0	00	10	00	X
srl	000000	1	01	1	0	0	0	00	10	00	X
lw	100011	1	00	0	1	0	0	01	00	00	X
sw	101011	0	XX	0	1	0	1	XX	00	00	X
beq	000100	0	XX	0	0	1	0	01	01	00	1
addi	001000	1	00	0	1	0	0	00	00	00	X
J	000010	0	XX	X	X	X	0	XX	XX	01	X
jal	000011	1	10	X	X	X	0	10	XX	01	X
jr	000000	1	01	0	0	0	0	00	10	10	X
ble	000001	0	XX	0	0	1	0	01	01	00	0

Table 2: Updated version of the main decoder