Arda İçöz

21901443

CS 224 – Lab04

Section 03

Preliminary Report

18/03/2021

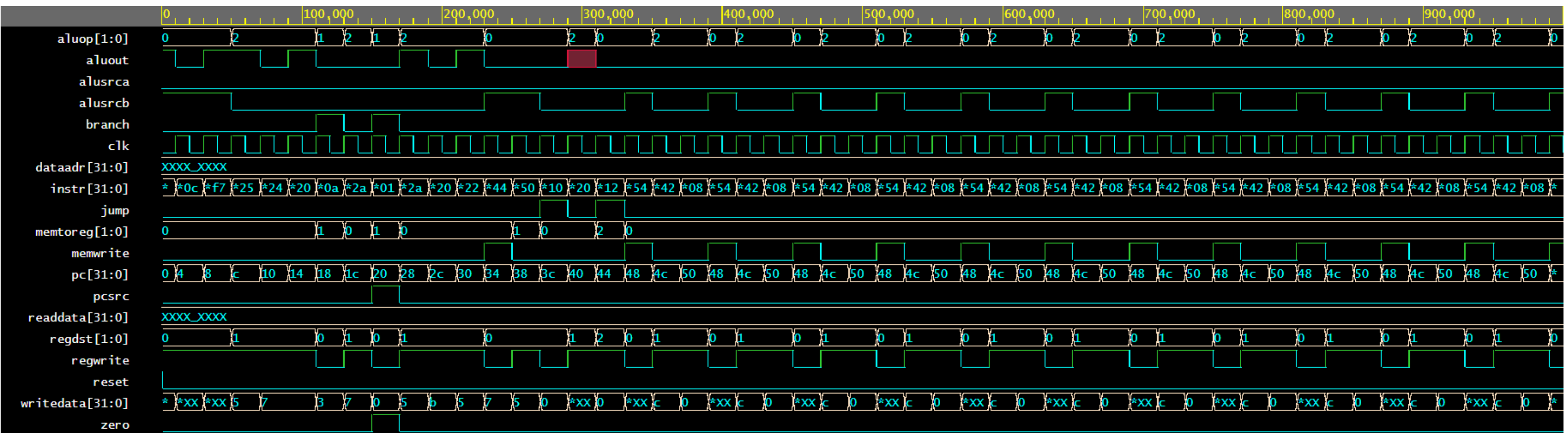
Part 1:

a) Instructions in IMEM module

Address	Machine Instruction	MIPS Assembly Equivalent
0x00000000	0x20020005	addi $v0, $zero, 5
0x00000004	0x2003000c	addi $v1, $zero, 12
0x00000008	0x2067fff7	addi $a3, $v1, -9
0x0000000c	0x00e22025	or $a0, $a3, $v0
0x00000010	0x00642824	and $a1, $v1, $a0
0x00000014	0x00a42820	add $a1, $a1, $a0
0x00000018	0x10a7000a	beq $a1, $a3, 0x0000000a
0x0000001c	0x0064202a	slt $a0, $v1, $a0
0x00000020	0x10800001	beq $a0, $zero, 0x00000001
0x00000024	0x20050000	addi $a1, $zero, 0
0x00000028	0x00e2202a	slt $a0, $a3, $v0
0x0000002c	0x00853820	add $a3, $a0, $a1
0x00000030	0x00e23822	sub $a3, $a3, $v0
0x00000034	0xac670044	sw $a3, 68($v1)
0x00000038	0x8c020050	lw $v0, 80($zero)
0x0000003c	0x08000010	j 0x00000010
0x00000040	0x001f6020	add $t4, $zero ,$ra
0x00000044	0x0c000012	jal 0x00000012
0x00000048	0xac020054	sw $v0, 84(0)
0x0000004c	0x00039042	srl $s2, $v1, 1

0x00000050	0x03E00008	jr $ra

e) Waveform



f) Questions about waveform

a) “writedata” represents the register file. But, because WE (writeEnable) is zero, it cannot perform writing data.

b) For example, first three instructions are “addi” and because “addi” is I-type it takes the value as immediate. Thus, it does not need an additional variable to write the data.

c) Because “memtoreg” is equal to zero (at those specific times) it cannot read, thus “readdata” is undefined at those.

d) “dataadr” represents the address of a value after the calculations in ALU, since ALU can also produce a value as an address (these values can then go into data memory to point to the address of another value).

e) This is the instruction lw $v0, 80($zero). Register $zero, as its name represents, has the only value of zero and it cannot be changed, making it the constant value. So, we cannot increment the address by 80 since $zero does not hold or point to a address. Thus it creates an undefined value.

g) Questions about the architecture

1. No, there is no need for change. Because ALU can already perform shift operation. By using this shifted value, it is also possible to shift another (second) value. Thus, especially ALU doesn't need to be changed.
2. ALU has to be modified. ALU has already a shift right operation but it needs to shift left for this. So, we can add a feature that shifts the values left in ALU. But a new 3-bit code has to be reserved in ALUControl for this operation as well.

Part 2:

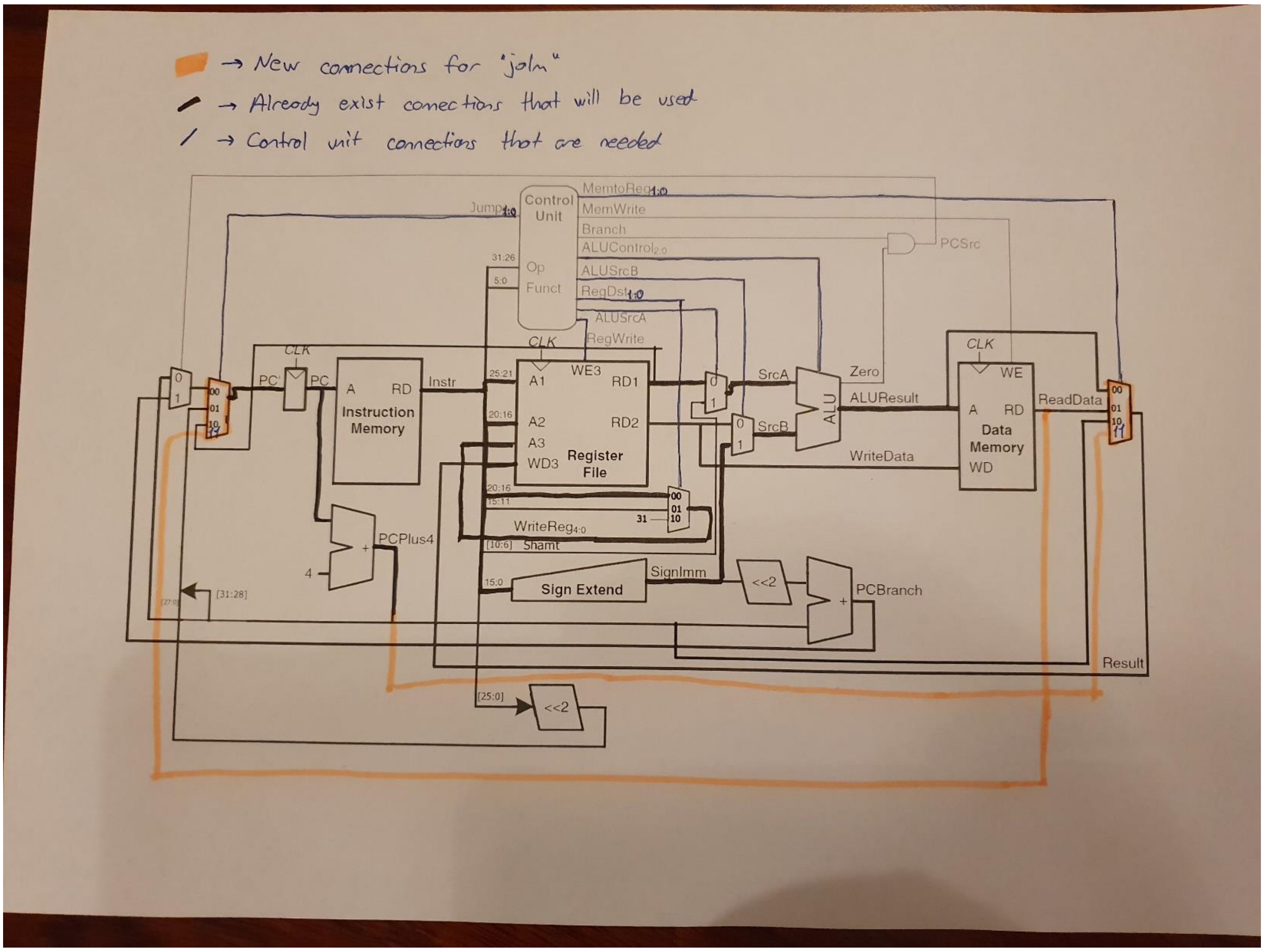
a) Register Transfer Level-Language for "jalm" instruction

IM[PC]

RF[rt] 🡐 PC + 4 (storing the next instruction's address into target reg)

PC 🡐 DM[RF[rs] + (SignExtImm)] (we put the address of the instruction we are going to jump to the PC, also we get the address from data memory)

b) New Datapath



c) Main Decoder Table

Instruction	Opcode	RegWrite	RegDst	ALUSrcA	ALUSrcB	Branch	MemWrite	MemToReg	ALU Op	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	011111	1	00	0	1	X	0	11	00	11