Bilkent University

Computer Engineering

Computer Organization – CS224

Design Report

Lab # 4

Section # 4

Mannan Abdul

21801066

14/04/2020

Part B

Location (In Hex)	Machine Instruction (In Hex)	Assembly Code
0x00	0x20020005	addi $v0, $zero, 5
0x04	0x2003000c	addi $v1, $zero, 12
0x08	0x2067fff7	addi $a3, $v1, -9
0x0c	0x00e22025	or $a0, $a3, $v0
0x10	0x00642824	and $a1, $v1, $a0
0x14	0x00a42820	add $a1, $a1, $a0
0x18	0x10a7000a	beq $a1, $a3, 10
0x1c	0x0064202a	slt $a0, $v1, $a0
0x20	0x10800001	beq $a0, $zero, 1
0x24	0x20050000	addi $a1, $zero, 0
0x28	0x00e2202a	slt $a0, $a3, $v0
0x2c	0x00853820	add $a3, $a0, $a1
0x30	0x00e23822	sub $a3, $a3, $a1
0x34	0xac670044	sw $a3, 68($v1)
0x38	0x8c020050	lw $v0, 80($zero)
0x3c	0x08000011	j 0x0000011
0x40	0x20020001	addi $v0, $zero, 1
0x44	0xac020054	sw $v0, 84($zero)
0x48	0x08000012	j 0x0000012

Part C

RTL expression for “jm”:

IM[PC];

PC ← RF[rs] + SignExt(immed);

RTL expression for “bgt”:

IM[PC];

If(RF[rs] > RF[rt])

PC ← PC + 4 + SignExt(immed) * 4;

else

PC ← PC + 4;

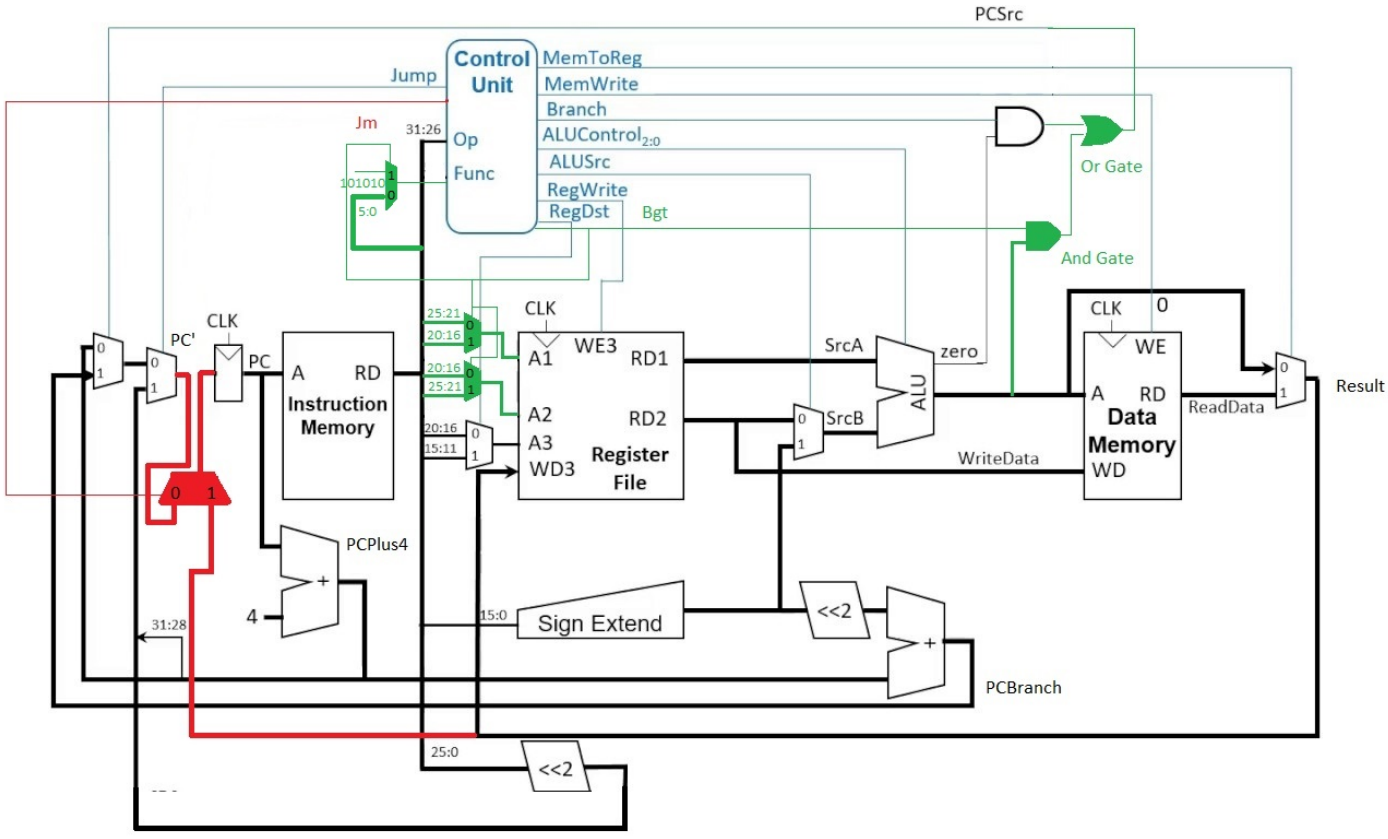
Part D

Jm (red lines in datapath):

We add a new signal named “jm” from the control unit for this instruction. The instruction needs only one new piece of hardware to load the result into the PC. This multiplexer is shown in red in the figure along with its control signal. The PC’ needs to be loaded with the sum of the address in the register and the sign extended offset which we get in the result, the necessary control signals will be shown in the table in the next part.

Bgt (green lines in datapath):

A bgt instruction can be broken down into 2 steps, first we use the slt instruction to check 2 values, then we check the result in the ALU and act on it. As such, for this instruction, we have added 2 multiplexers that will exchange the places of rs and rt when bgt is 1. We will set the ALU to set less than and it will check if rt is less then rs, if it is it will output 1 and we will take that result and apply and AND gate to it and the bgt signal to check if the result 1 is consistent with the signal, we also put an OR gate so that we can use the already existing hardware for this purpose to get the next instruction into PC’. Also, because bgt is a I-type instruction, it has no function code, so we add a multiplexer there that manually gives the function code for slt to the control unit.



Part E

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp	Jump	Jm	Bgt
R-type	000000	1	1	0	0	0	0	10	0	0	0
lw	100011	1	0	1	0	0	1	00	0	0	0
sw	101011	0	X	1	0	1	X	00	0	0	0
beq	000100	0	X	0	1	0	X	01	0	0	0
addi	001000	1	0	1	0	0	0	00	0	0	0
j	000010	0	X	X	X	0	X	XX	1	0	0
jm	000001	0	X	1	0	0	0	00	0	1	0
bgt	000011	0	X	0	0	0	0	10	0	0	1

*for bgt, ALUOp is 1X, Funct is 101010 (slt) and ALUControl is 111

Part F

To test our new instructions in conjunction with the old ones, we will write a test program that will have R-type, lw, sw, beq, addi, j instructions.

If even one R-type instruction works, we know that the rest will also work because their datapath is the same.

```
addi $t7, $zero, 4    // checks an addi instruction

addi $t0, $zero, 4

addi $t1, $zero, 10

add $t2, $t0, $t1  // checks an R-type instruction

lw $t3, 0($t2)  // checks lw instruction

sw $t3, 0($t6)  // checks sw instruction

beq $t7, $t0, 1  // checks beq instruction

sub $t7, $t7, $t0

j 0x00000028   // checks j instruction

sub $t7, $t7, $t0

bgt $t1, $t0, 1  // checks bgt instruction
```

sub $t7, $t7, $t0

jm 0($t0) // checks jm instruction

This test code will only do the addition on an infinite loop because all the other testers are linked in such a way that they point directly to the next jump/branch instruction and so on. The last instruction points to the first one and therefore will enable an infinite loop.