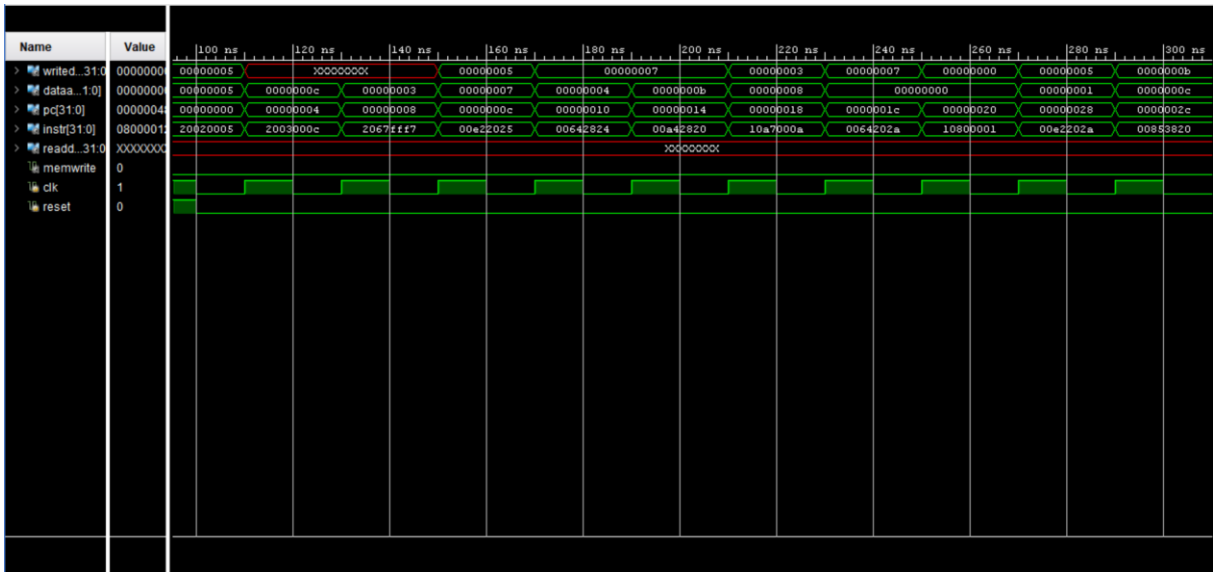
CS224
Lab No. 4
Section No. 1
Your Full Name Mert Gençtürk
Bilkent ID 2003506
Date 14.11.2022

Location	Hex Machine	Assembly eq.
0x80000000	20020005	addi $2,$0,5
0x80000004	2003000c	addi $3,$0,12
0x80000008	2067fff7	addi $7,$3,-9
0x8000000c	00e22025	or $4,$7,$2
0x80000010	00642824	and $5,$3,$4
0x80000014	00a42820	add $5,$5,$4
0x80000018	10a7000a	beq $5,$7, 0x80000044
0x8000001c	0064202a	slt $4,$3,$4
0x80000020	10800001	beq $4,$0, 0x80000028
0x80000024	20050000	addi $5,$0,0
0x80000028	00e2202a	slt $4,$7,$2
0x8000002c	00853820	add $7,$4,$5
0x80000030	00e23822	sub $7,$7,$2
0x80000034	ac670044	sw $7,68($3)
0x80000038	8c020050	lw $2,80($0)
0x8000003c	08000011	j 0x80000044
0x80000040	20020001	addi $2,$0,1
0x80000044	ac020054	sw $2,84($0)
0x80000048	08000012	j 0x80000048



1-e)

1. In R type instructions, writedata corresponds to data in rt register.
2. Since instructions in early stages are not R type and corresponding rt values are not used, writedata comes as x from mux.
3. Readdata is the data value that is readed from memory and there is not much lw instruciton in the program; therefore, mostly it is not used.
4. In R type operations dataadr is the result of the alu operation.
5. In sw instruction (ac670044 && ac020054 in imem) memwrite becomes 1.

1-f)

```
 module alu(input  logic [31:0] a, b,
input  logic [2:0]  alucont,
output logic [31:0] result,
output logic zero);

 always_comb
 case(alucont)
3'b010: result = a + b;
3'b110: result = a - b;
3'b000: result = a & b;
3'b001: result = a | b;
3'b011: result = a<<b;
3'b111: result = (a < b) ? 1 : 0;
default: result = {32{1'bx}};
endcase

 assign zero = (result == 0) ? 1'b1 : 1'b0;
endmodule
```

Part 2:

a)

jm:

IM[PC]

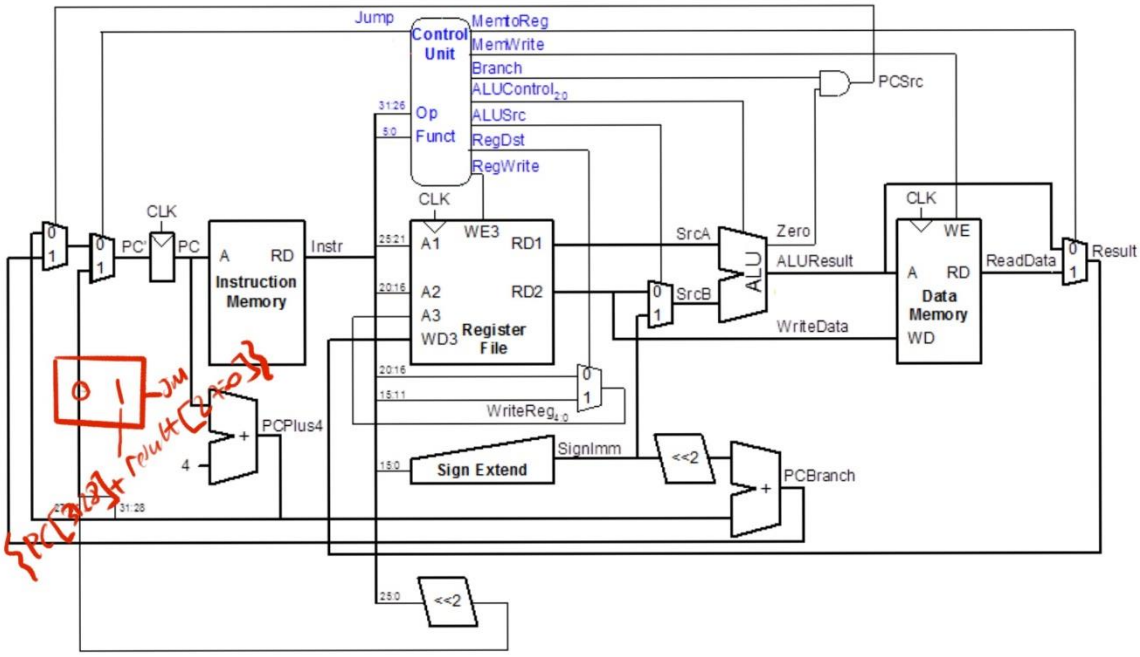
PC = DM[RF[rs]+imm]

bgt:

IM[PC]

PC = PC + 4 + (RF[rs]>RF[rt] ? imm<<2 : 0)

b)



For jm instruction only need is to add a multiplexer for selecting jump address and jm address. Jm address comes from leftmost six bits of pc and other bits of result. To get result only need is perform an addition operation in alu for rs and immediate value. So, instruction is actually similiar to:

lw $at, Imm($rs)

jr $at

but since jm needs to be done in 1 clock cycle we do not use $at. Instead, we get the jm address from memory and select in with a control variable jm before jump mux.

For bgt instruction no extra changes are needed. Just changing alu module and alu decoder module is enough to implement bgt operation. Adding set greather than operation to ALU and setting its output to Zero wire will be enough when branch is set as 1.

c)

Instruction	**Opcode**	**RegWrite**	**RegDst**	**ALUSrc**	**Branch**	**MemWrite**	**MemToReg**	**ALUOp**	**Jump**	**Jm**
Jm	000011	0	0	1	0	0	1	00	1	1
bgt	000111	0	0	0	1	0	0	10	0	0
R-type	000000	1	1	0	0	0	0	10	0	0
lw	100011	1	0	1	0	0	1	00	0	0
sw	101011	0	X	1	0	1	X	00	0	0
beq	000100	0	X	0	1	0	X	01	0	0
addi	001000	1	0	1	0	0	0	00	0	0
j	000010	0	X	X	X	0	X	XX	1	0

ALUOp	**Funct**	**ALUControl**
00	X	010 (add)
01	X	110 (subtract)
1X	100000 (add)	010 (add)
1X	100010 (sub)	110 (subtract)
1X	100100 (and)	000 (and)
1X	100101 (or)	001 (or)
1X	101010 (slt)	111 (set less than)
1X	000111(sgt)	100(set greater than)