CA3 – MULTI-CYCLE PROCCESSOR

Zahra Hojati – 810199403 Fatemeh Mohammadi – 810199489

31 26	25 21	20 16	15 11	10 6	5 0
000000(RT)	Rs	Rt	Rd	shift	func

31 26	25 21	20 16	15 0
001001(addi)	Rs	Rt	data

31 26	25 21	20 16	15 0
001010(slti)	Rs	Rt	data

31 26	25 21	20 16	15 0
100011(lw)	Rs	Rt	address

31 26	25 21	20 16	15 0
101011(sw)	Rs	Rt	address

31 26	25 21	20 16	15 0
000100(beq)	Rs	Rt	address

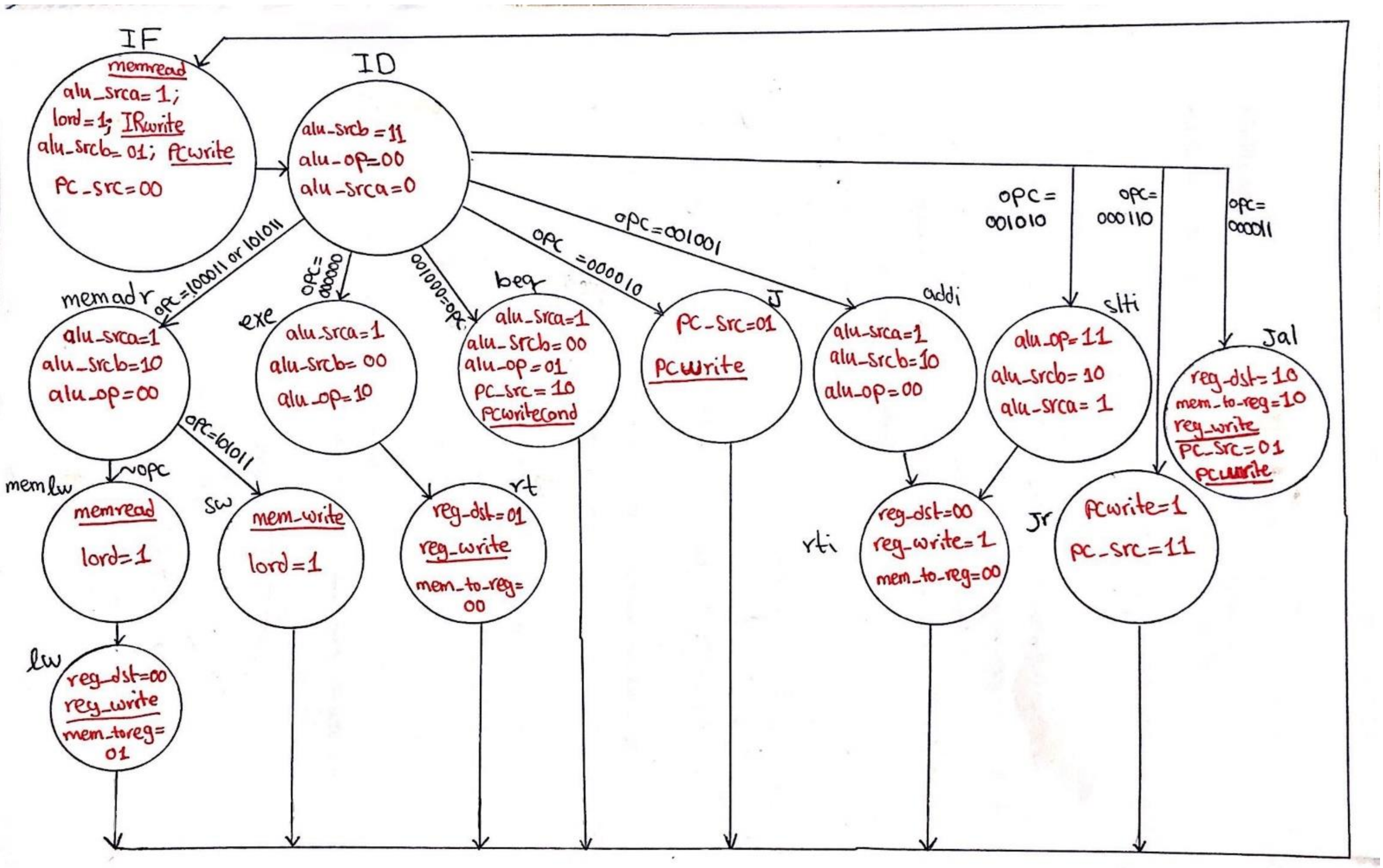
31 26	25 21	20 16	15 11	10 6	5 0
000010(J)	address				

31 26	25 21	20 16	15 11	10 6	5 0
000110(Jr)	Rs	Unimportant			

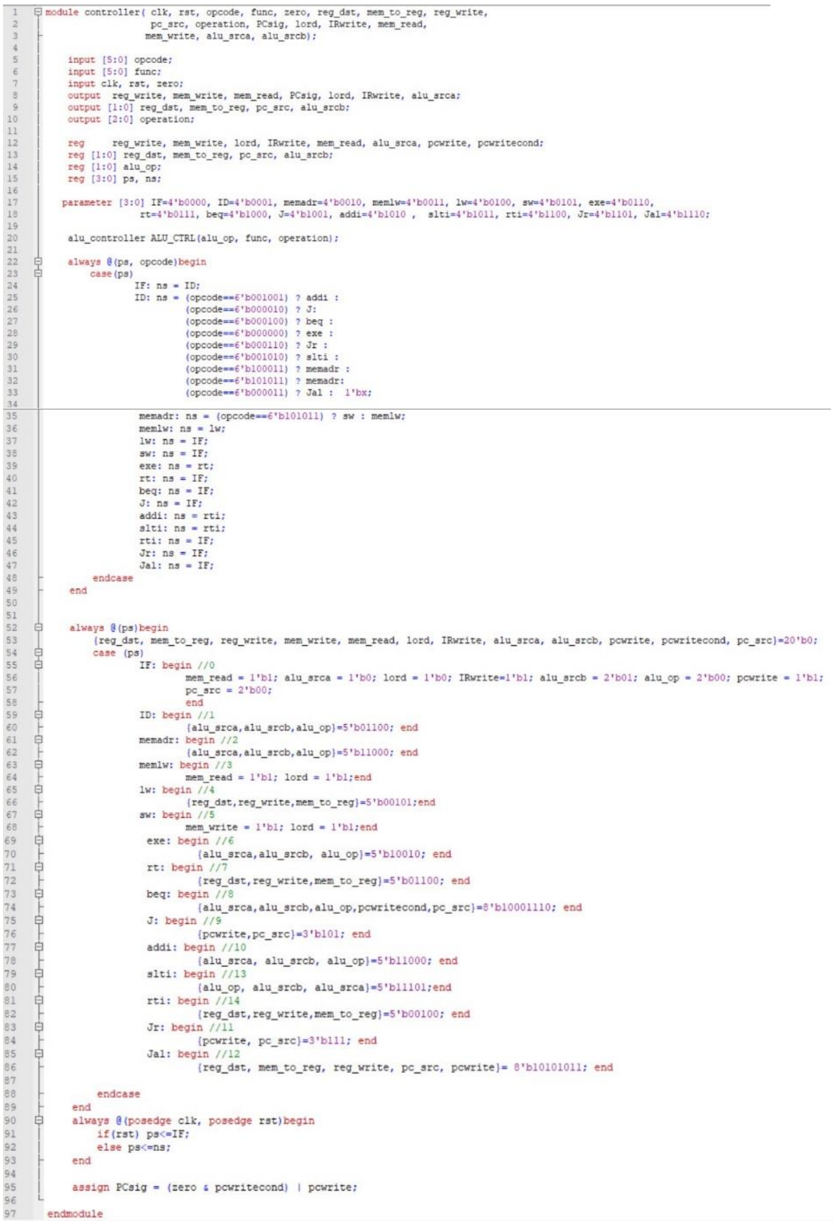
31 26	25 21	20 16	15 11	10 6	5 0
000011(Jal)	address				

Controller's State Machine:

Inst.	Clock Cycles
RT	4
addi	4
slti	4
lw	5
sw	4
beq	3
J	3
Jr	3
Jal	3

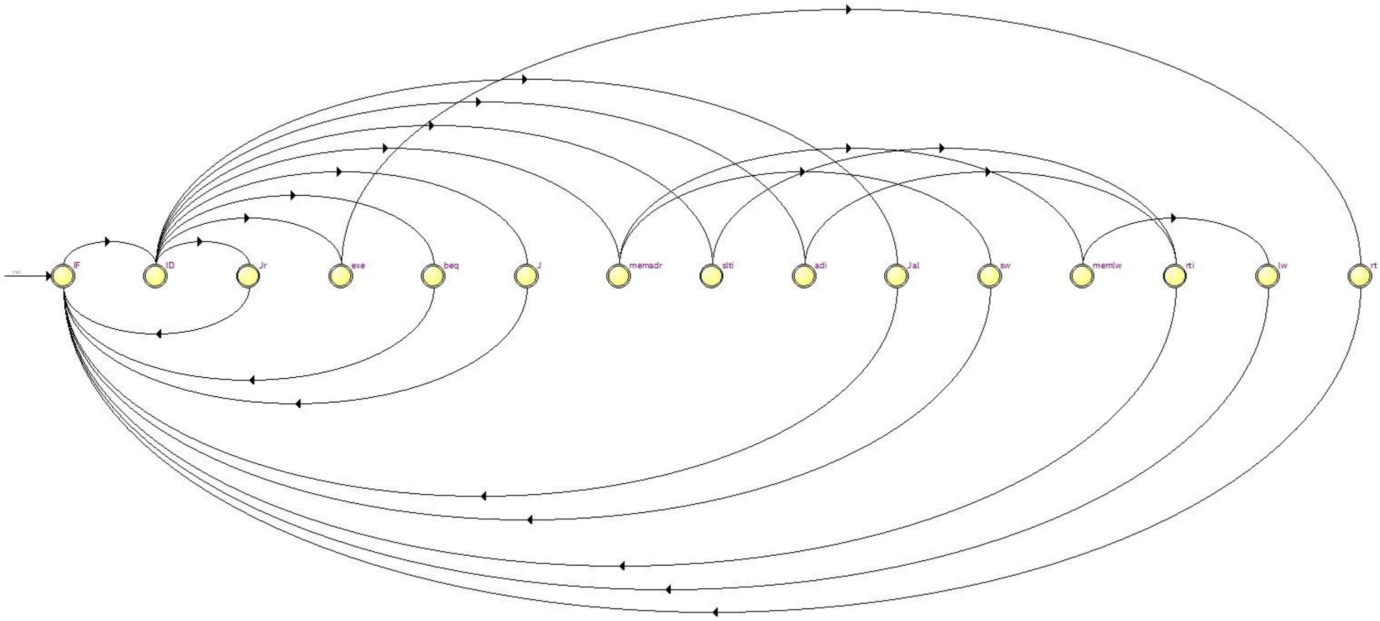


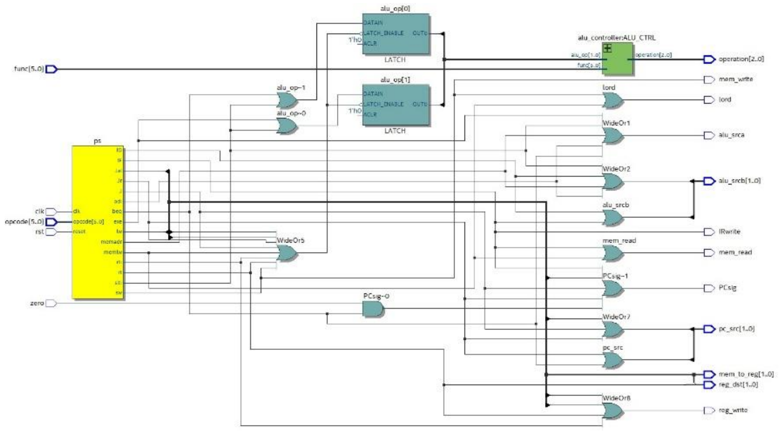
Controller Verilog Code:



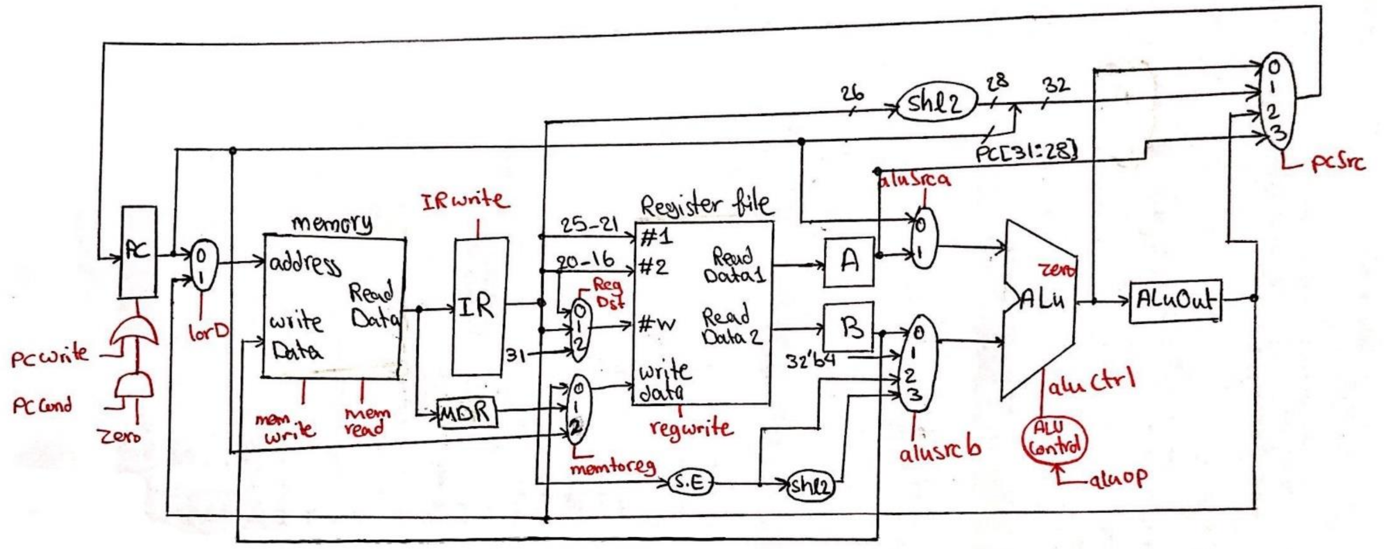
```verilog
module controller( clk, rst, opcode, func, zero, reg_dst, mem_to_reg, reg_write,
                   pc_src, operation, PCsig, lord, IRwrite, mem_read,
                  mem_write, alu_srca, alu_srcb);

    input [5:0] opcode;
    input [5:0] func;
    input clk, rst, zero;
    output  reg_write, mem_write, mem_read, PCsig, lord, IRwrite, alu_srca;
    output [1:0] reg_dst, mem_to_reg, pc_src, alu_srcb;
    output [2:0] operation;

    reg     reg_write, mem_write, lord, IRwrite, mem_read, alu_srca, pcwrite, pcwritecond;
    reg [1:0] reg_dst, mem_to_reg, pc_src, alu_srcb;
    reg [1:0] alu_op;
    reg [3:0] ps, ns;

   parameter [3:0] IF=4'b0000, ID=4'b0001, memadr=4'b0010, memlw=4'b0011, lw=4'b0100, sw=4'b0101, exe=4'b0110,
                 rt=4'b0111, beq=4'b1000, J=4'b1001, addi=4'b1010 ,  slti=4'b1011, rti=4'b1100, Jr=4'b1101, Jal=4'b1110;

    alu_controller ALU_CTRL(alu_op, func, operation);

    always @(ps, opcode)begin
        case(ps)
                IF: ns = ID;
                ID: ns = (opcode==6'b001001) ? addi :
                         (opcode==6'b000010) ? J:
                         (opcode==6'b000100) ? beq :
                         (opcode==6'b000000) ? exe :
                         (opcode==6'b000110) ? Jr :
                         (opcode==6'b001010) ? slti :
                         (opcode==6'b100011) ? memadr :
                         (opcode==6'b101011) ? memadr:
                         (opcode==6'b000011) ? Jal :  1'bx;

                memadr: ns = (opcode==6'b101011) ? sw : memlw;
                memlw: ns = lw;
                lw: ns = IF;
                sw: ns = IF;
                exe: ns = rt;
                rt: ns = IF;
                beq: ns = IF;
                J: ns = IF;
                addi: ns = rti;
                slti: ns = rti;
                rti: ns = IF;
                Jr: ns = IF;
                Jal: ns = IF;
        endcase
    end


    always @(ps)begin
        {reg_dst, mem_to_reg, reg_write, mem_write, mem_read, lord, IRwrite, alu_srca, alu_srcb, pcwrite, pcwritecond, pc_src}=20'b0;
        case (ps)
                IF: begin //0
                        mem_read = 1'b1; alu_srca = 1'b0; lord = 1'b0; IRwrite=1'b1; alu_srcb = 2'b01; alu_op = 2'b00; pcwrite = 1'b1;
                        pc_src = 2'b00;
                        end
                ID: begin //1
                        {alu_srca,alu_srcb,alu_op}=5'b01100; end
                memadr: begin //2
                        {alu_srca,alu_srcb,alu_op}=5'b11000; end
                memlw: begin //3
                        mem_read = 1'b1; lord = 1'b1;end
                lw: begin //4
                        {reg_dst,reg_write,mem_to_reg}=5'b00101;end
                sw: begin //5
                        mem_write = 1'b1; lord = 1'b1;end
                 exe: begin //6
                          {alu_srca,alu_srcb, alu_op}=5'b10010; end
                 rt: begin //7
                          {reg_dst,reg_write,mem_to_reg}=5'b01100; end
                 beq: begin //8
                          {alu_srca,alu_srcb,alu_op,pcwritecond,pc_src}=8'b10001110; end
                 J: begin //9
                          {pcwrite,pc_src}=3'b101; end
                 addi: begin //10
                          {alu_srca, alu_srcb, alu_op}=5'b11000; end
                 slti: begin //13
                          {alu_op, alu_srcb, alu_srca}=5'b11101;end
                 rti: begin //14
                          {reg_dst,reg_write,mem_to_reg}=5'b00100; end
                 Jr: begin //11
                          {pcwrite, pc_src}=3'b111; end
                 Jal: begin //12
                          {reg_dst, mem_to_reg, reg_write, pc_src, pcwrite}= 8'b10101011; end

        endcase
    end
    always @(posedge clk, posedge rst)begin
        if(rst) ps<=IF;
        else ps<=ns;
    end

    assign PCsig = (zero & pcwritecond) | pcwrite;

endmodule
```

Controller State machine and RTL overview in Quartus:

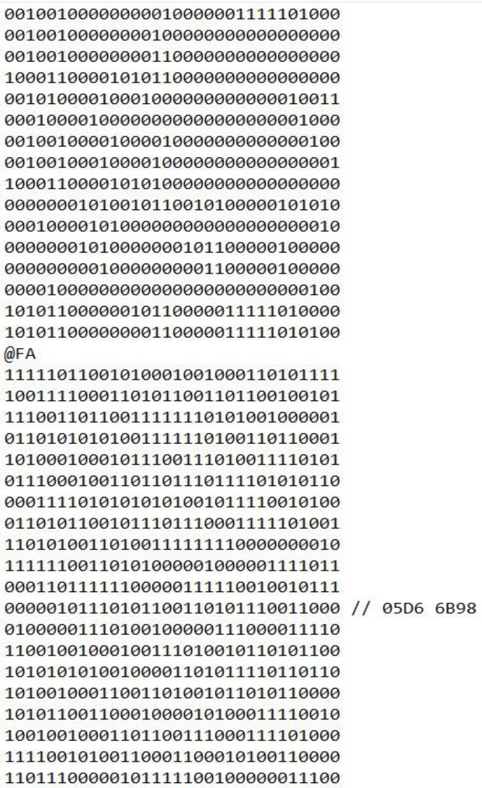




Datapath:



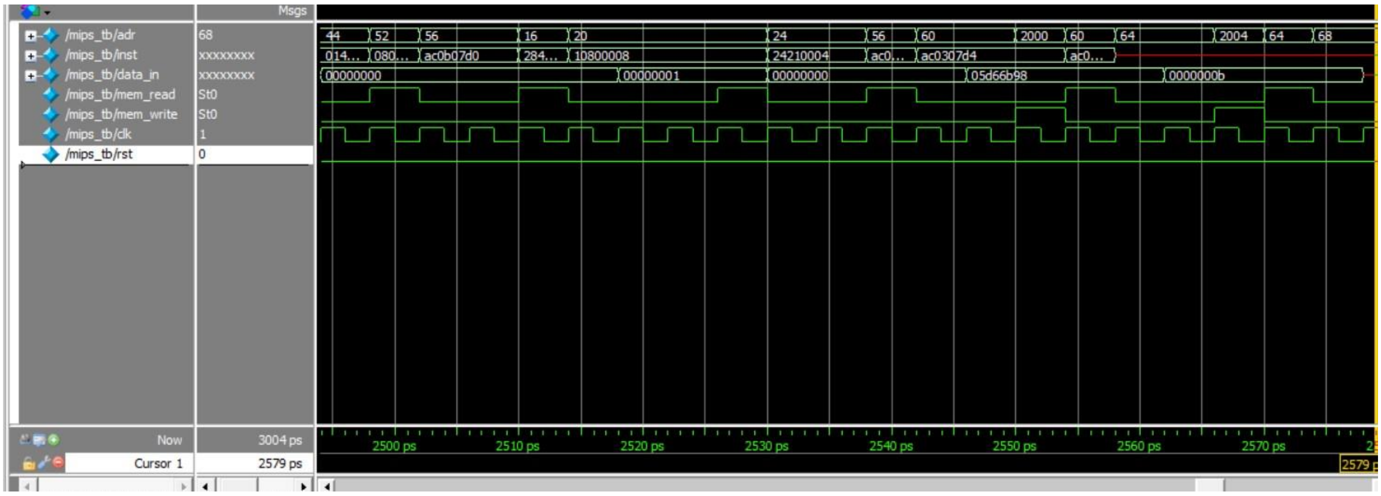
Memory: Since the Datapath contains only one memory, we allocate instructions in the first half of the memory and the data in the second half. The smallest data is shown below with its hex equivalent, its index is mem[11].



```
00100100000000010000001111101000
00100100000000100000000000000000
00100100000000110000000000000000
10001100001010110000000000000000
00101000010001000000000000010011
00010000100000000000000000001000
00100100001000010000000000000100
00100100010000100000000000000001
10001100001010100000000000000000
00000001010010110010100000101010
00010000101000000000000000000010
00000001010000000101100000100000
00000000010000000001100000100000
00001000000000000000000000000100
10101100000010110000001111010000
10101100000000110000001111010100
@FA
11111011001010001001000110101111
10011110001101011001101100100101
11100110110011111110101001000001
01101010101001111110100110110001
10100010001011100111010011110101
01110001001101101110111101010110
00011110101010101001011110010100
01101011001011101110001111101001
11010100110100111111110000000010
11111100110101000001000001111011
000110111111100000111110010010111
```

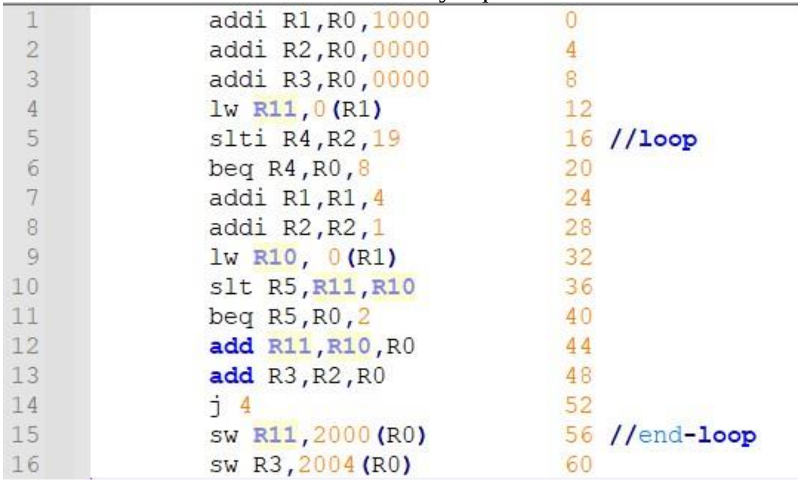
Output Waveform:

As shown in the waveform, the smallest data (05d66b98) is stored in the 2000th index of memory and its index (b) is stored in 2004th.



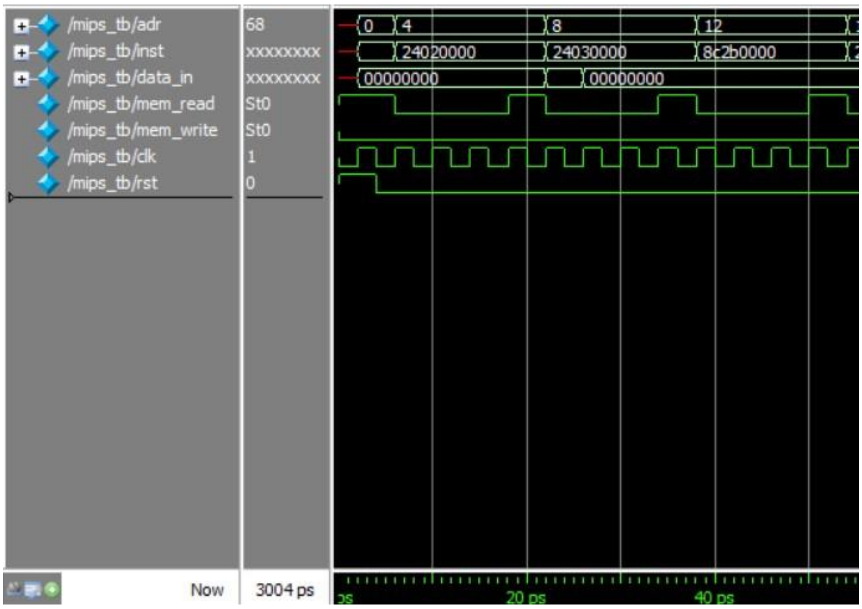
Based on the controller, the instructions can take different clock cycles to complete, each instruction that was added to the processor, is performed based on the instruction given to the memory and is shown in the images below:

Instruction assembly equivalent:

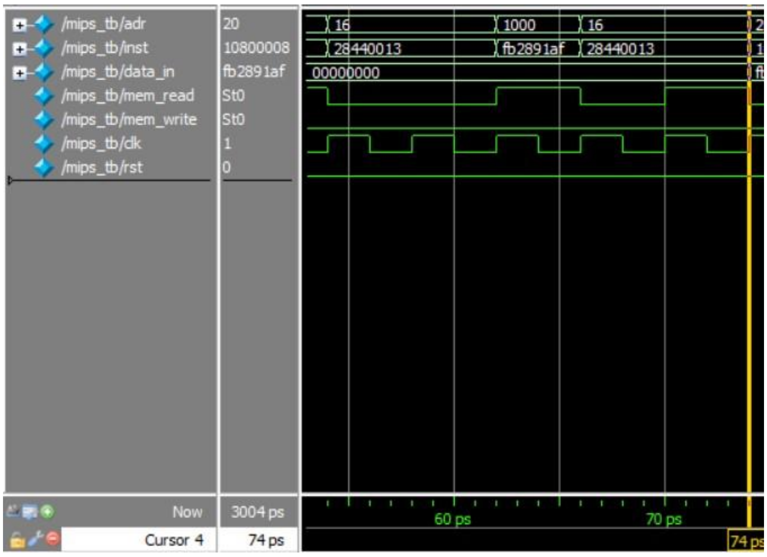


```
addi R1,R0,1000       0
addi R2,R0,0000       4
addi R3,R0,0000       8
lw R11,0(R1)          12
slti R4,R2,19         16 //loop
beq R4,R0,8           20
addi R1,R1,4          24
addi R2,R2,1          28
lw R10, 0(R1)         32
slt R5,R11,R10        36
beq R5,R0,2           40
add R11,R10,R0        44
add R3,R2,R0          48
j 4                   52
sw R11,2000(R0)       56 //end-loop
sw R3,2004(R0)        60
```

addi: the first 3 instruction in the code are addi, it takes 4 c.c to be done. It works based on the instruction, for example the first addi is supposed to add 0 and R0 and store the result in R1.

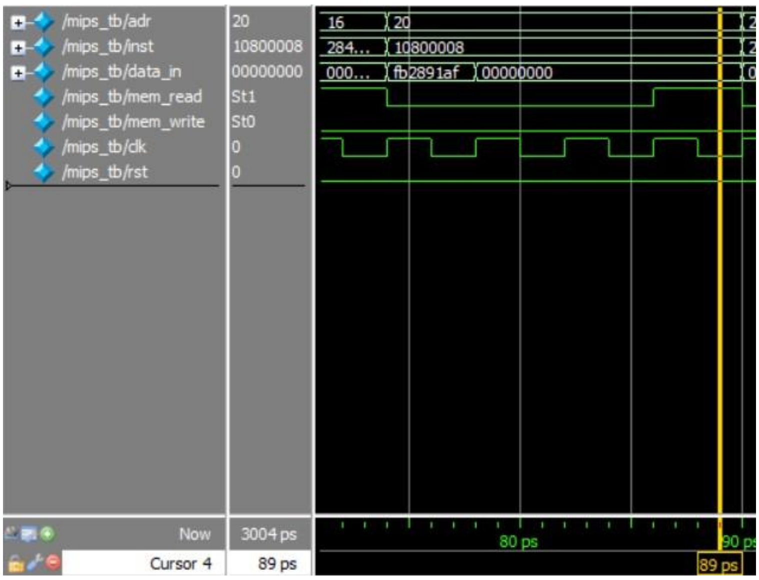


lw: the fourth instruction is lw, and it takes 5c.c to complete the task. In the code, it writes the result of 0 and R1's addition, the data stored in R1, into R11, which in the first loop iteration is 1000, once the inst in turned to the result, memory read signal is issued.



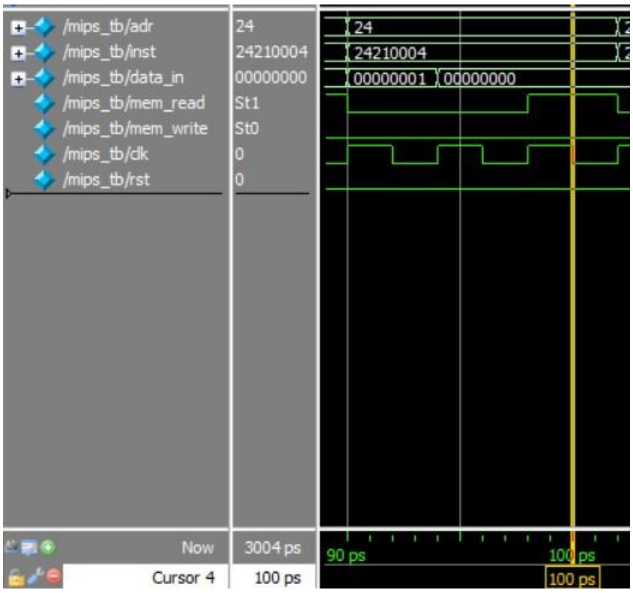
/mips_tb/adr	20
/mips_tb/inst	10800008
/mips_tb/data_in	fb2891af
/mips_tb/mem_read	St0
/mips_tb/mem_write	St0
/mips_tb/clk	1
/mips_tb/rst	0
Now	3004 ps
Cursor 4	74 ps

slti: it's the fifth instruction and takes 4c.c. if the data stored in R2 is less than 19 (number of the loop iterations since the array's size is 20), R4 turns to 1.



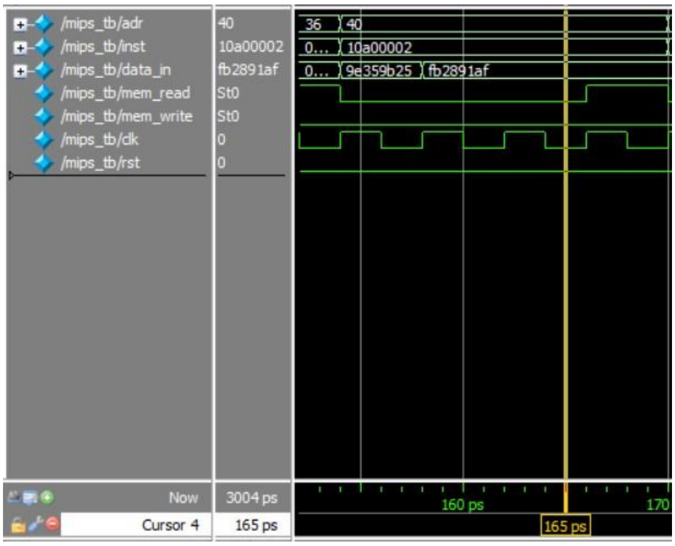
/mips_tb/adr	20
/mips_tb/inst	10800008
/mips_tb/data_in	00000000
/mips_tb/mem_read	St1
/mips_tb/mem_write	St0
/mips_tb/clk	0
/mips_tb/rst	0
Now	3004 ps
Cursor 4	89 ps

beq: it's the 6th instruction that takes 3c.c. if R4 (condition of slti) is zero (false), it skips 8 instructions.



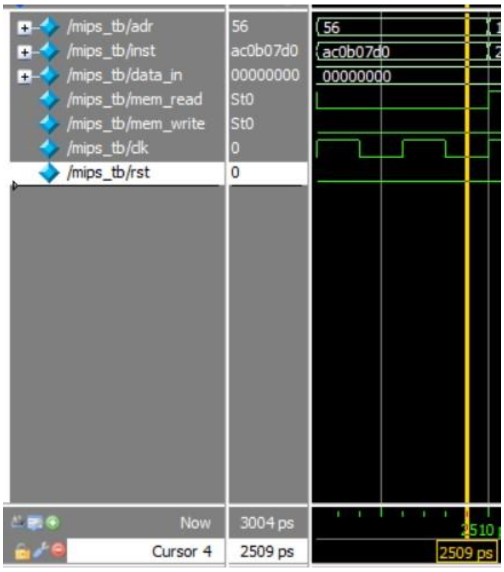
/mips_tb/adr	24
/mips_tb/inst	24210004
/mips_tb/data_in	00000000
/mips_tb/mem_read	St1
/mips_tb/mem_write	St0
/mips_tb/clk	0
/mips_tb/rst	0
Now	3004 ps
Cursor 4	100 ps

slt: is R-Type instruction and takes 4c.c. to complete. It functions the same as slti, but it compares two datas stores in two registers rather than immediate values.



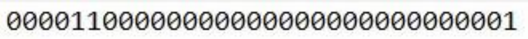
```
/mips_tb/adr        40          36 | 40
/mips_tb/inst       10a00002    0... | 10a00002
/mips_tb/data_in    fb2891af    0... | 9e359b25 | fb2891af
/mips_tb/mem_read   St0
/mips_tb/mem_write  St0
/mips_tb/clk        0
/mips_tb/rst        0
Now       3004 ps
Cursor 4  165 ps
```

J: takes 3c.c. and based on the instruction, it jumps to the 4th instruction which is

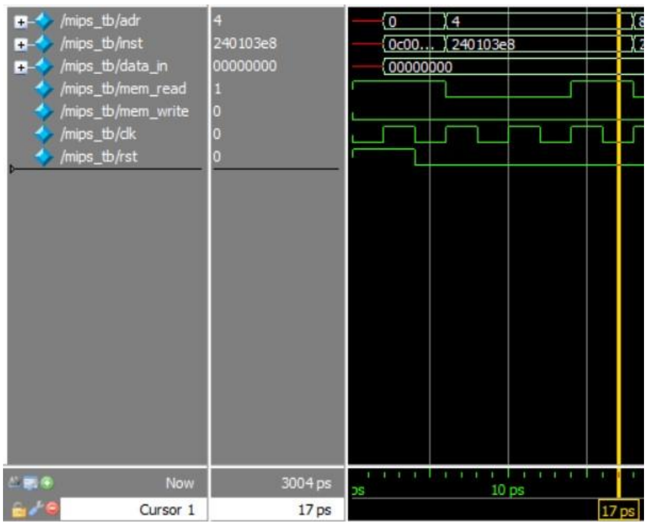


```
/mips_tb/adr        56          56
/mips_tb/inst       ac0b07d0    ac0b07d0
/mips_tb/data_in    00000000    00000000
/mips_tb/mem_read   St0
/mips_tb/mem_write  St0
/mips_tb/clk        0
/mips_tb/rst        0
Now       3004 ps
Cursor 4  2509 ps
```

Jal: according to the table of instruction and clock cycles it takes 3c.c to complete, in the instruction shown below, Jump to the first address and reads the address and stores it in R29.

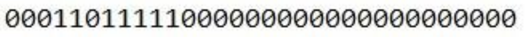


```
00001100000000000000000000000001
```

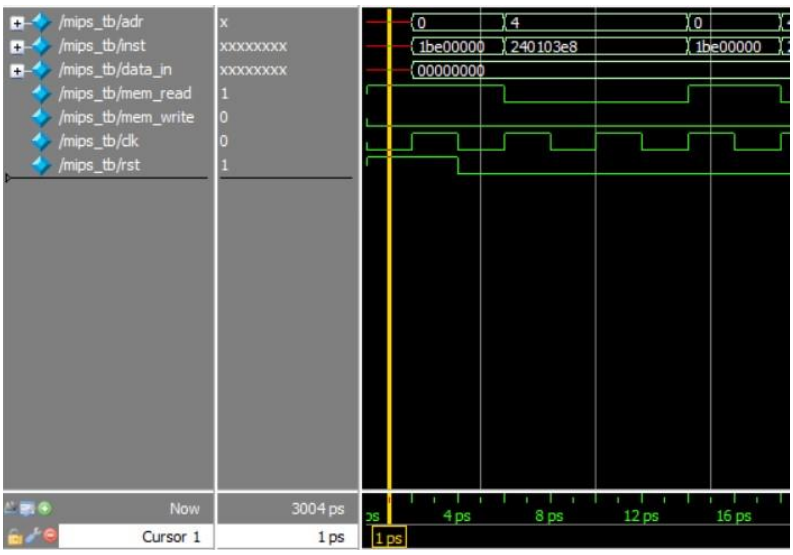


```
/mips_tb/adr        4           0 | 4
/mips_tb/inst       240103e8    0c00... | 240103e8
/mips_tb/data_in    00000000    00000000
/mips_tb/mem_read   1
/mips_tb/mem_write  0
/mips_tb/clk        0
/mips_tb/rst        0
Now       3004 ps
Cursor 1  17 ps
```

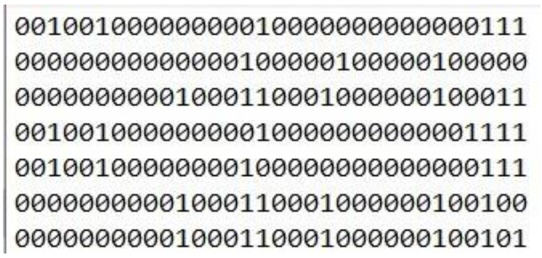
Jr: takes 3c.c to complete and in the instruction below, it jumps to the according Register.



```
00011011111000000000000000000000
```

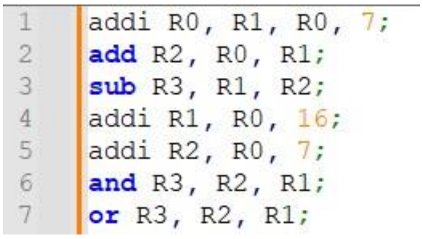


Instructions in the image below is to the test all the R-type instructions:

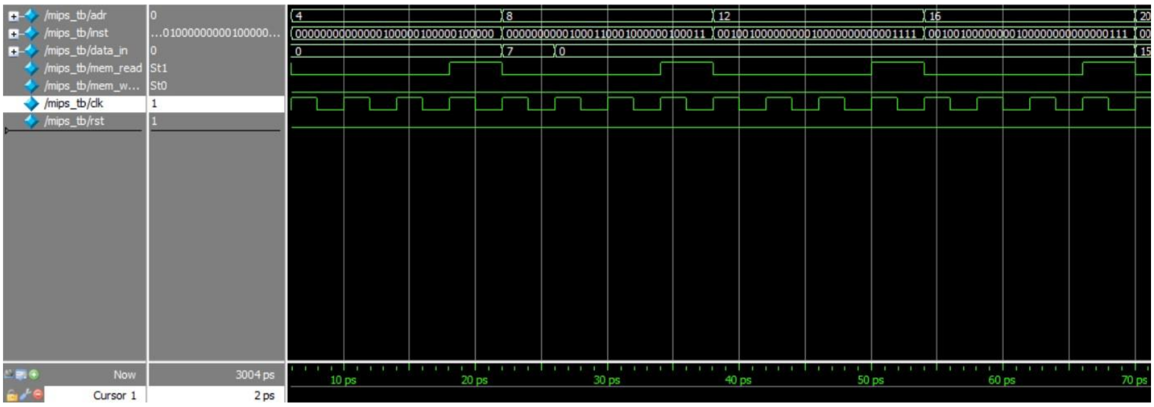


```
00100100000000010000000000000111
00000000000000010000010000100000
00000000001000110001000000100011
00100100000000010000000000001111
00100100000000010000000000000111
00000000001000110001000000100100
00000000001000110001000000100101
```

Their equivalent assembly code is:



```
addi R0, R1, R0, 7;
add R2, R0, R1;
sub R3, R1, R2;
addi R1, R0, 16;
addi R2, R0, 7;
and R3, R2, R1;
or R3, R2, R1;
```


/mips_tb/adr
/mips_tb/inst
/mips_tb/data_in
/mips_tb/mem_read
/mips_tb/mem_w...
/mips_tb/clk
/mips_tb/rst
0
...01000000000100000...
0
St1
St0
1
1
4
8
12
16
20
00000000000000100000100000100000
00000000001000110001000000100011
00100100000000010000000000001111
00100100000000010000000000000111
0
7
0
15
Now
3004 ps
Cursor 1
2 ps
10 ps
20 ps
30 ps
40 ps
50 ps
60 ps
70 ps