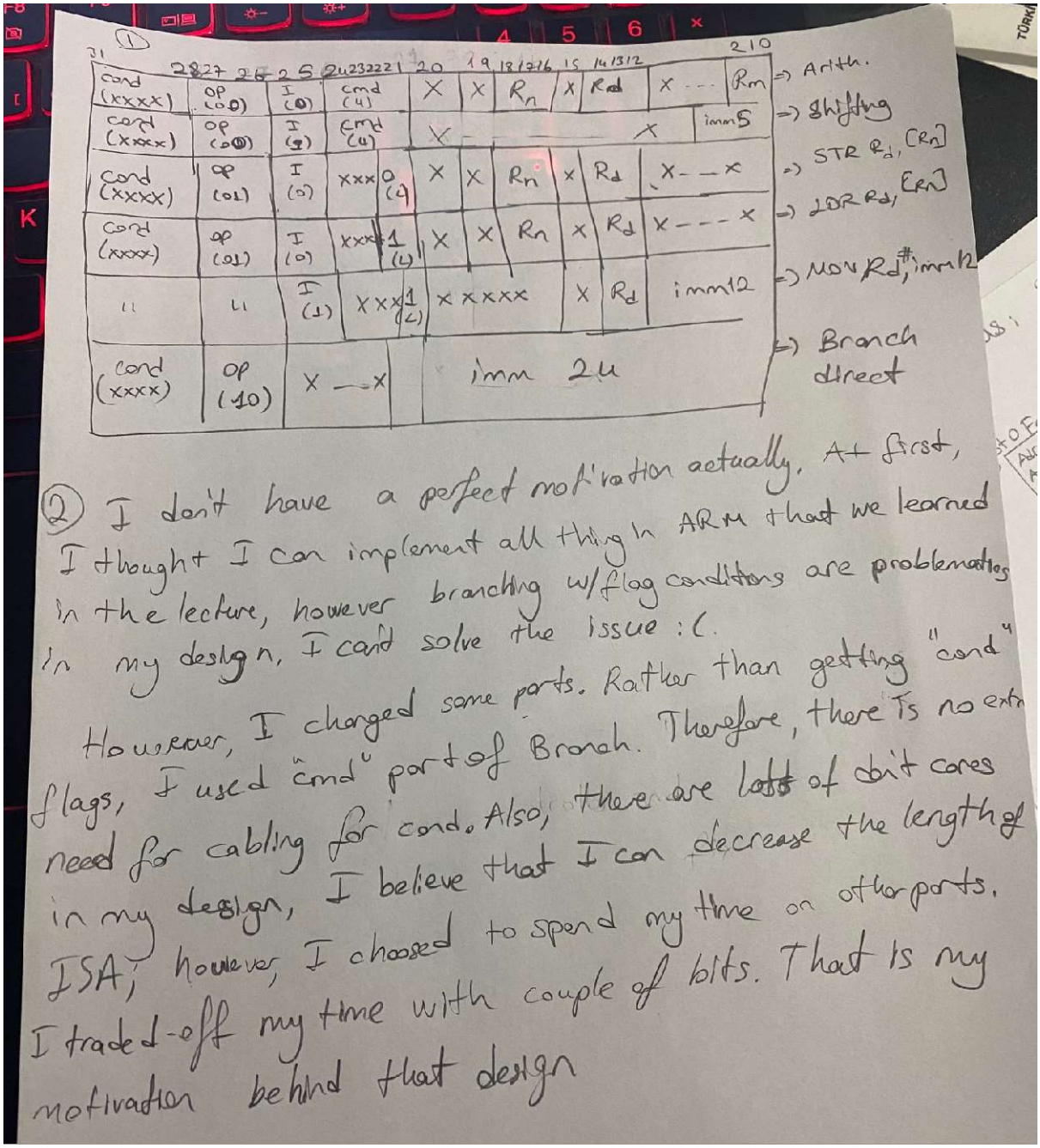
Berkay İPEK

EE446 -2304814

Laboratory Work 5 - Control Unit Design for Multi-Cycle CPU

Contents

1.2 ISA Configuration: From Mnemonics to Machine Code (10% Credits)



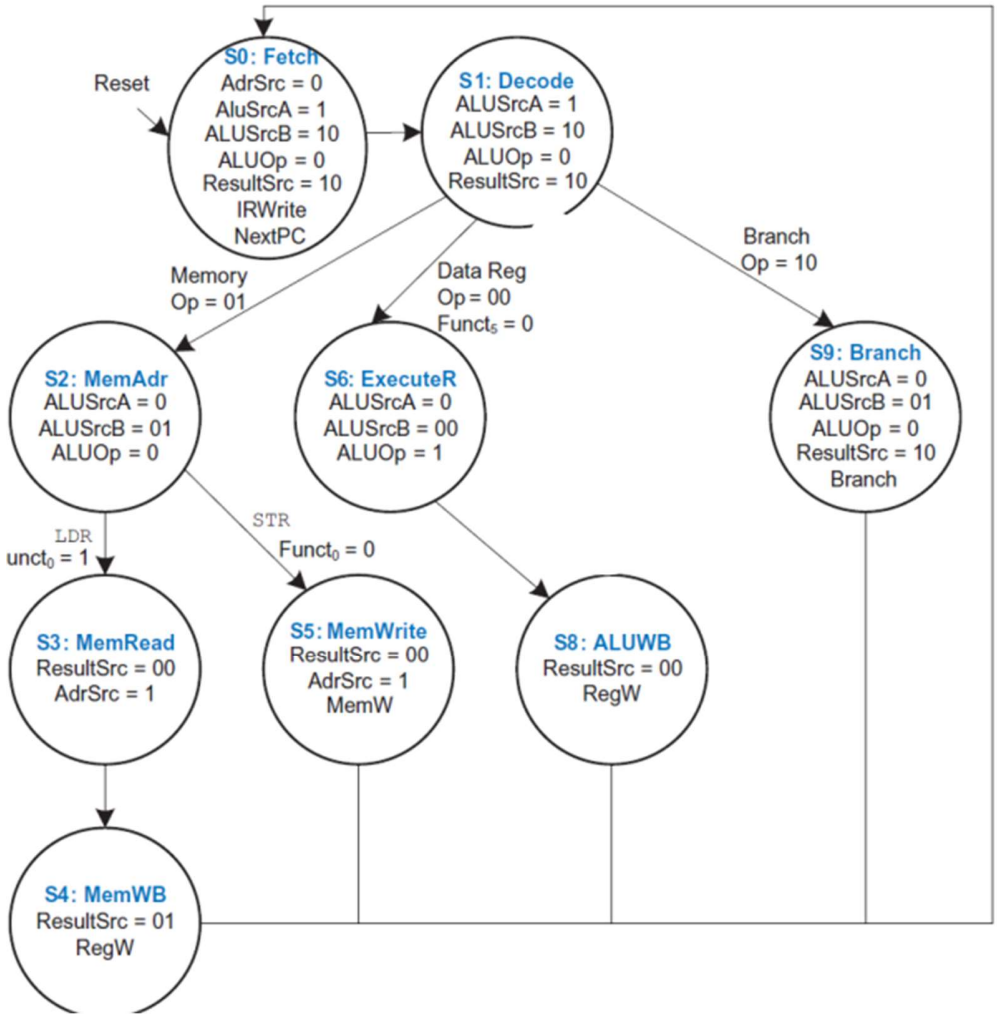
(2) I don't have a perfect motivation actually. At first, I thought I can implement all things in ARM that we learned in the lecture, however branching w/flag conditions are problematic in my design, I can't solve the issue :(.

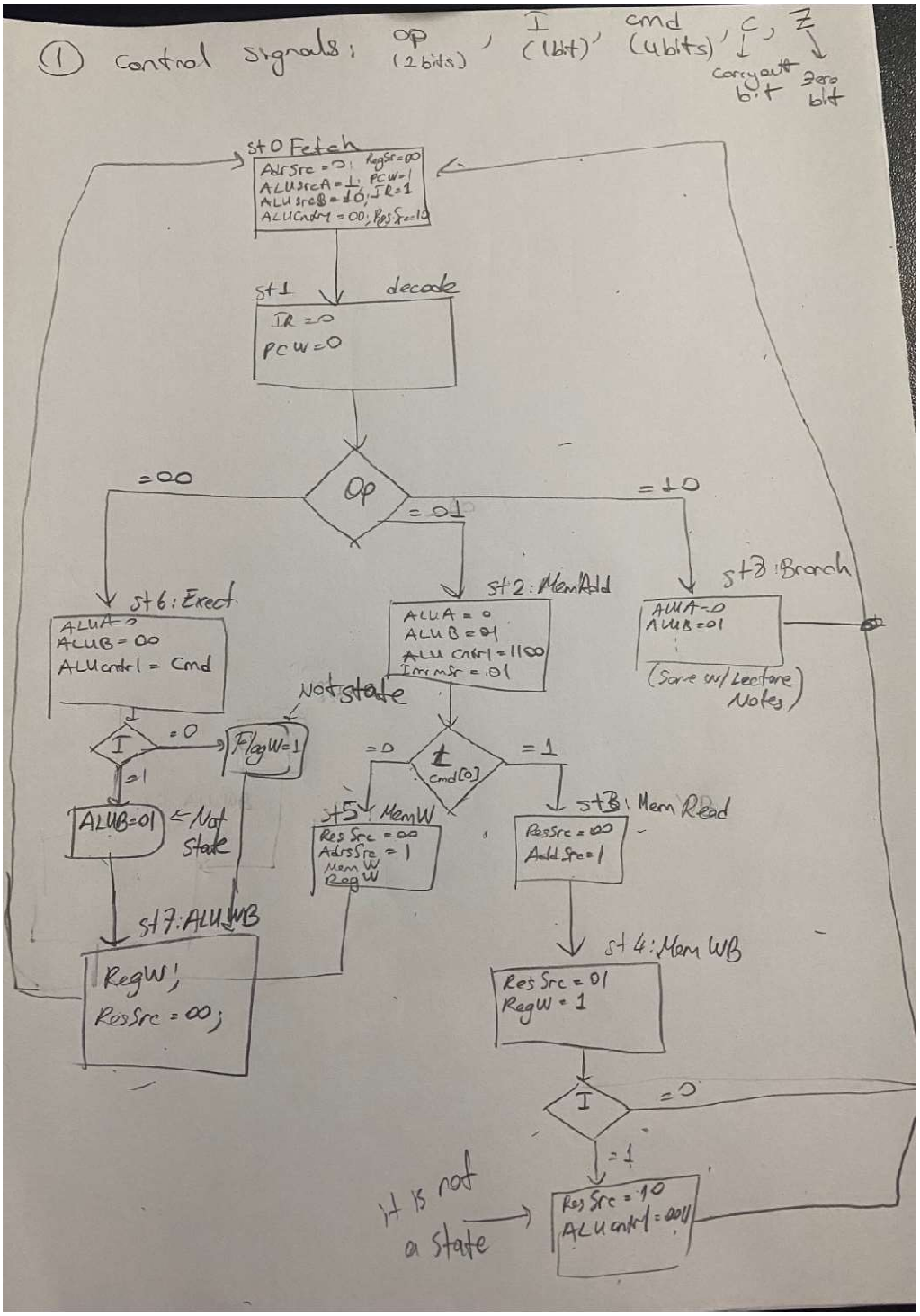
However, I changed some parts. Rather than getting "cond" flags, I used "cmd" part of Branch. Therefore, there is no extra need for cabling for cond. Also, there are lots of don't cares in my design, I believe that I can decrease the length of ISA; however, I choosed to spend my time on other parts. I traded-off my time with couple of bits. That is my motivation behind that design

1.2.6 Multi-cycle CPU Controller Unit Design (60% Credits)

1. Design an ASM chart for the operation of the FSM as described above. You are not expected to include each instruction separately, but to illustrate the overall functioning of the designed circuit. Assert whether extra states are required

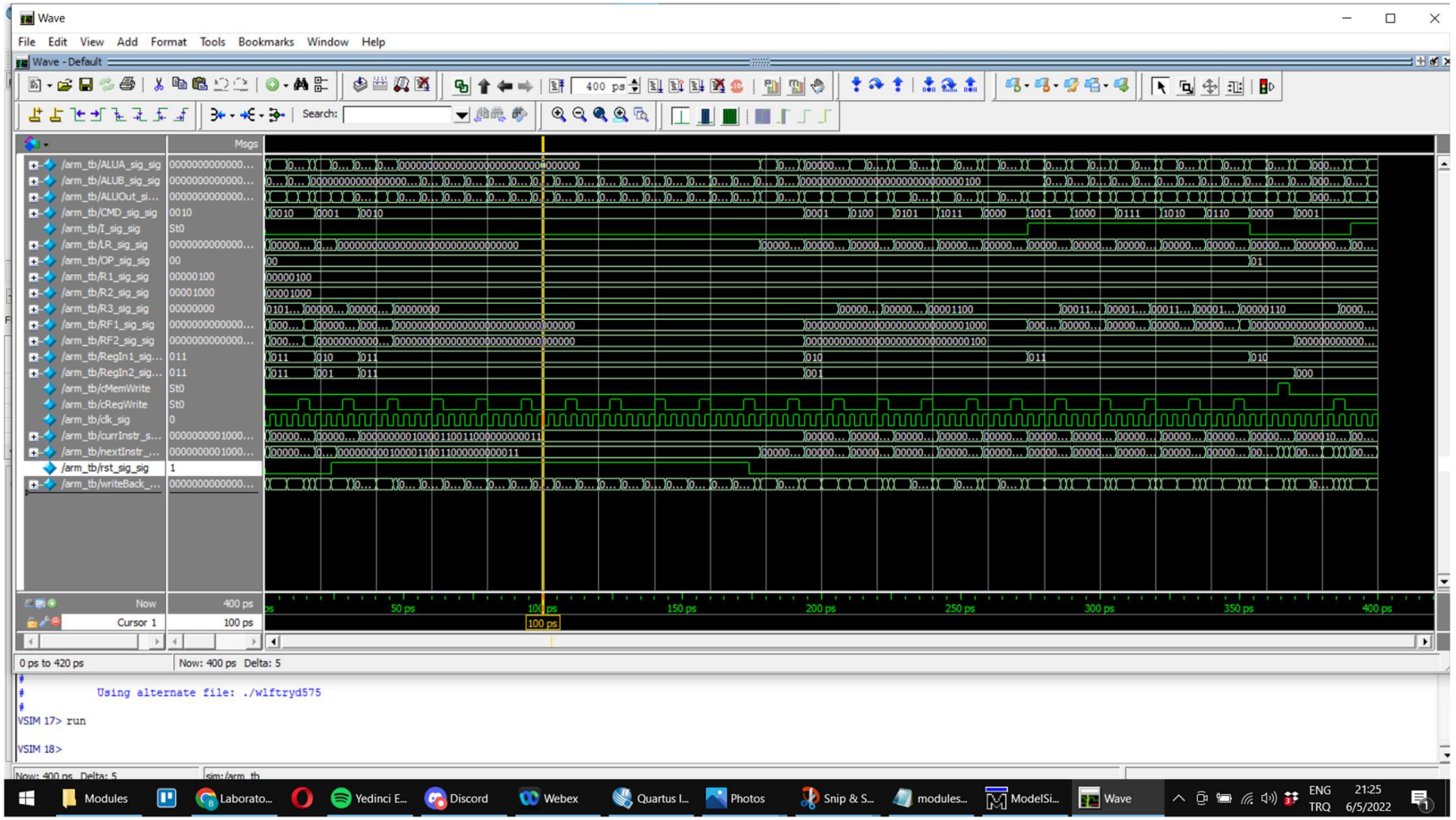
My motivation was ARM structure that is discussed in the lecture.



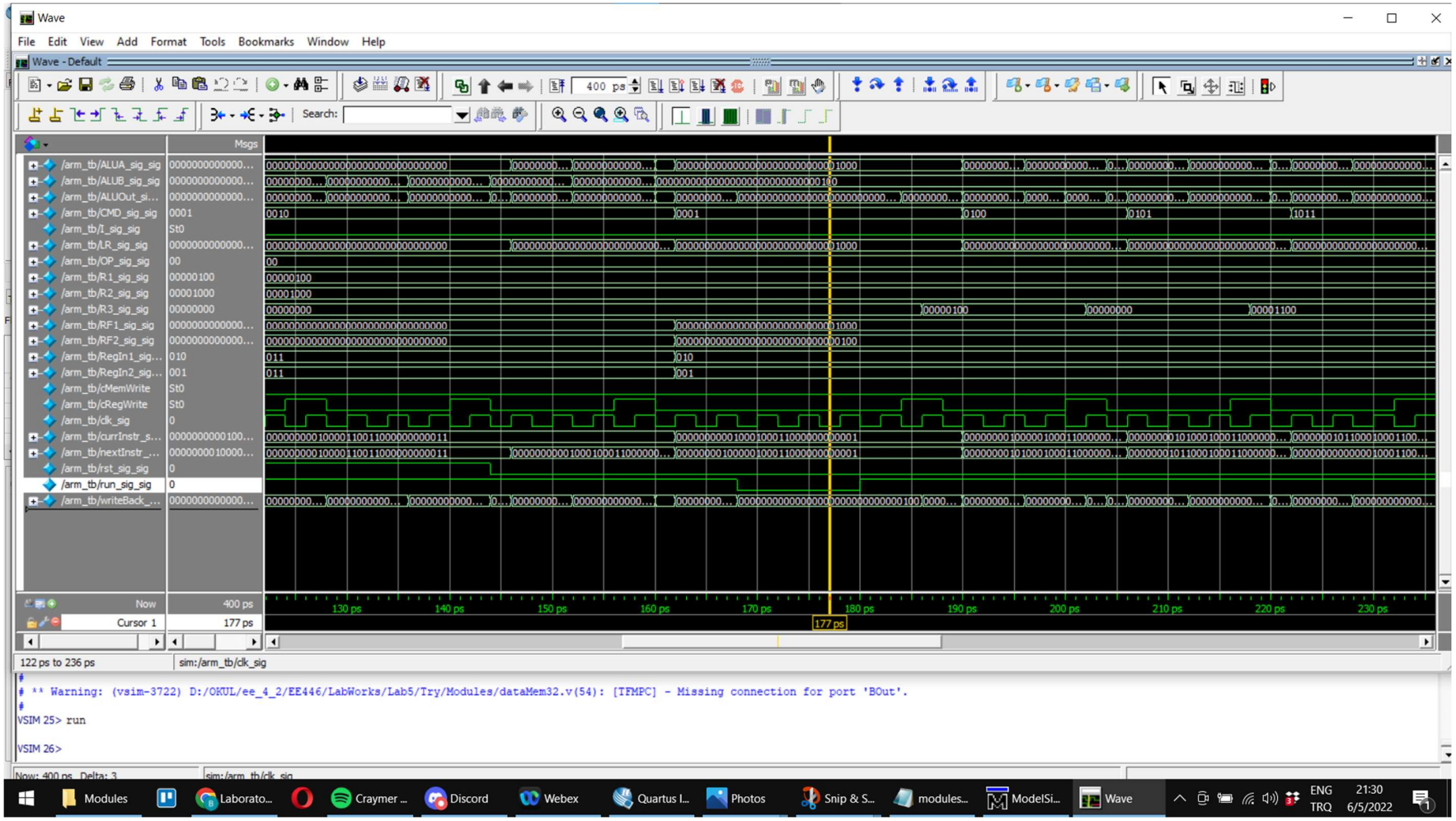

① Control signals: Op (2 bits), I (1 bit), cmd (4 bits), C, Z
carry out bit
zero bit
st0 Fetch
AdrSrc = 0; RegSr=00
ALUSrcA = 1; PCW=1
ALUSrcB = 10; IR=1
ALUCntrl = 00; ResSrc=10
st1 decode
IR = 0
PCW = 0
Op
=00
=01
=10
st6: Exect
ALUA = 0
ALUB = 00
ALUcntrl = Cmd
st2: MemAdd
ALUA = 0
ALUB = 01
ALU Cntrl = 1100
ImmSr = 01
st3: Branch
ALUA = 0
ALUB = 01
(Same w/ Lecture Notes)
I
=0
=1
FlagW=1
Not state
ALUB=01
Not State
t
cmd[0]
=0
=1
st5: MemW
ResSrc = 00
AdrSrc = 1
MemW
RegW
st3: Mem Read
ResSrc = 00
AdrSrc = 1
st7: ALUWB
RegW;
ResSrc = 00;
st4: Mem WB
ResSrc = 01
RegW = 1
I
=0
=1
it is not a state
ResSrc = 10
ALUcntrl = 0011

3. Implement the design of the control unit, by providing a state register to demonstrate the current state and setting up the chunks of circuitry required for each control signal.

Check the rst_sig_sig signal. It is clearing PC non arch design so that whenever rst_sig_sig = 1, PC is going to be 0.



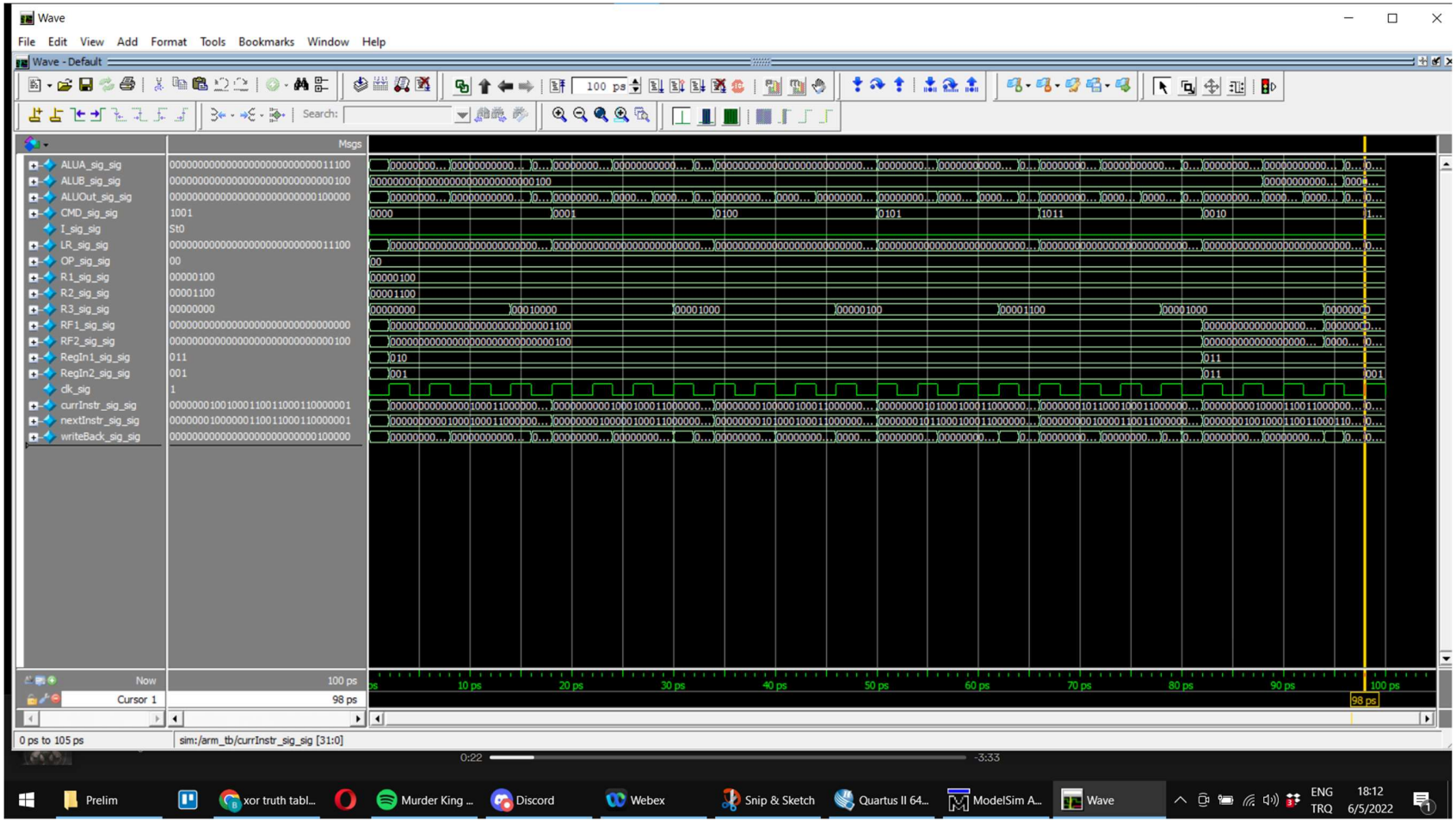
Check the run_sig_sig signal. It is stalling the clock in the whole design so that whenever run_sig_sig = 0, state of that instruction will be waited.



4. Write a test bench to simulate the operation of the design and verify its operation through simulations

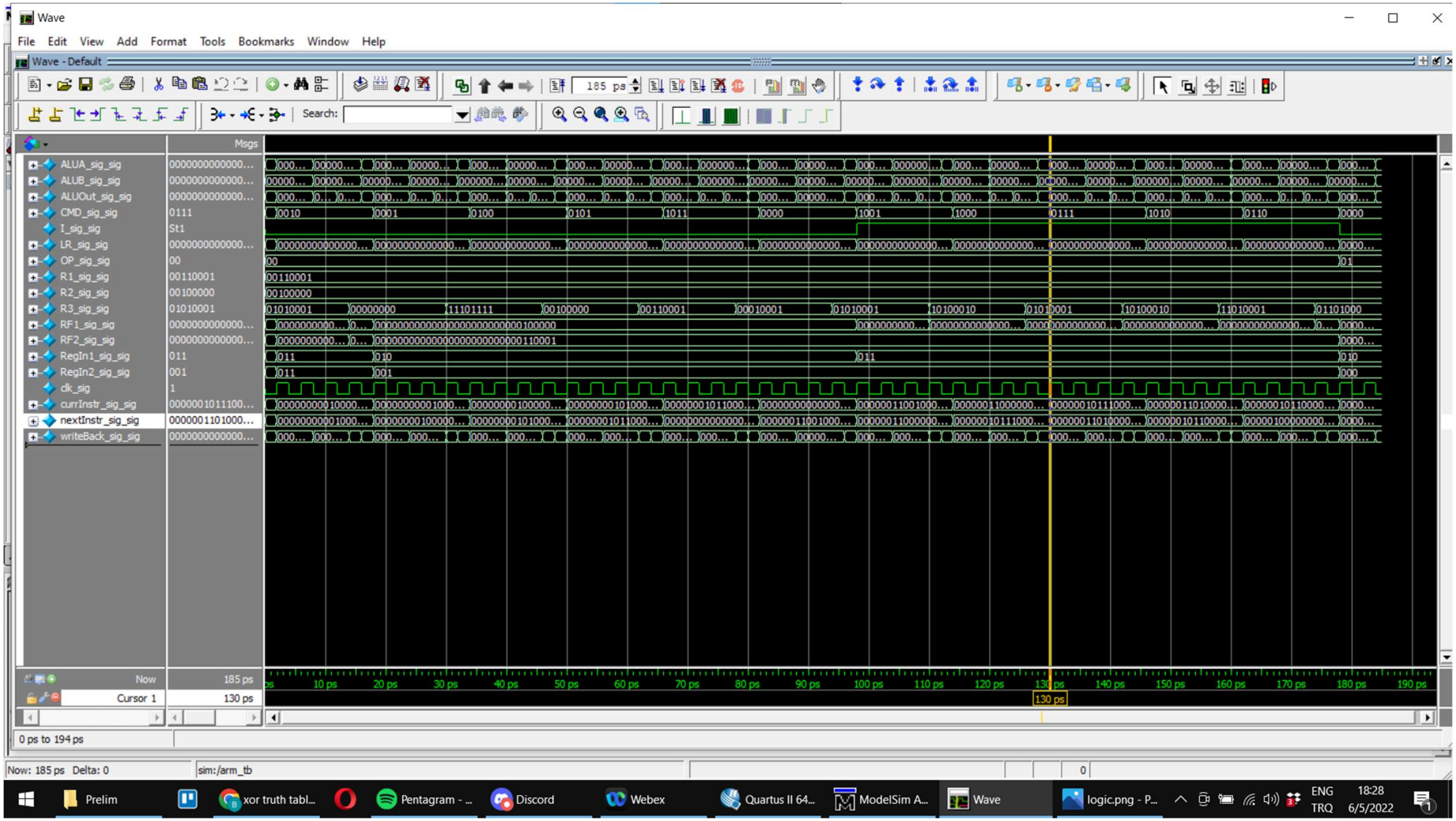
For Data Processing & Logic instructions:

```
integer i;
initial begin
// cond(4)_op(2'b00)_I(1)_cmd(4)_s(1)_rn(4)_rd(4)_smth5_sh_0_Rm(4)
   mem[0] <= 32'b0000_00_0_0000_0_0010_0011_000000000001;//ADD R3,R2,R1
   //mem[0] <= 32'b0000_10_0_0_00000000000000000001000;//B  ??
   mem[1] <= 32'b0000_00_0_0001_0_0010_0011_000000000001;//SUB R3,R2,R1
   mem[2] <= 32'b0000_00_0_0100_0_0010_0011_000000000001;//AND R3,R2,R1
   //mem[2] <= 32'b1;
   mem[3] <= 32'b0000_00_0_0101_0_0010_0011_000000000001;//OR R3,R2,R1
   mem[4] <= 32'b0000_00_0_1011_0_0010_0011_000000000001;//XOR R3,R2,R1
   mem[5] <= 32'b0000_00_0_0010_0_0011_0011_000000000011;//CLEAR R3,R2,R1
   //mem[0] <= 32'b0000_00_0_0000_0_0010_0011_000000000001;//ADD R3,R2,R1
   mem[6] <= 32'b0000_00_0_1001_0_0011_0011_00011_00_0_0001;//RL XX,R3,XX(SHMT5=1)
   mem[7] <= 32'b0000_00_0_1000_0_0011_0011_00011_00_0_0001;//RR XX,R3,XX(SHMT5=1)
   mem[8] <= 32'b0000_00_0_0111_0_0011_0011_00011_00_0_0001;//SL XX,R3,XX(SHMT5=1)
   mem[9] <= 32'b0000_00_0_1010_0_0011_0011_00011_00_0_0001;//ASR XX,R3,XX(SHMT5=1)
   mem[10] <= 32'b0000_00_0_0110_0_0011_0011_00011_00_0_0001;//LSR XX,R3,XX(SHMT5=1)
   //cond(4) op(2) I(1) PUBW(4) L(1) Rn(4) Rd(4) src(12)
```


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
ALUA_sig_sig
ALUB_sig_sig
ALUOut_sig_sig
CMD_sig_sig
I_sig_sig
LR_sig_sig
OP_sig_sig
R1_sig_sig
R2_sig_sig
R3_sig_sig
RF1_sig_sig
RF2_sig_sig
RegIn1_sig_sig
RegIn2_sig_sig
clk_sig
currInstr_sig_sig
nextInstr_sig_sig
writeBack_sig_sig
Now
100 ps
Cursor 1
98 ps
10 ps
20 ps
30 ps
40 ps
50 ps
60 ps
70 ps
80 ps
90 ps
0 ps to 105 ps
sim:/arm_tb/currInstr_sig_sig [31:0]
0:22
-3:33
Prelim
xor truth tabl...
Murder King ...
Discord
Webex
Snap & Sketch
Quartus II 64...
ModelSim A...
Wave
ENG
TRQ
18:12
6/5/2022

For Shifting Instructions

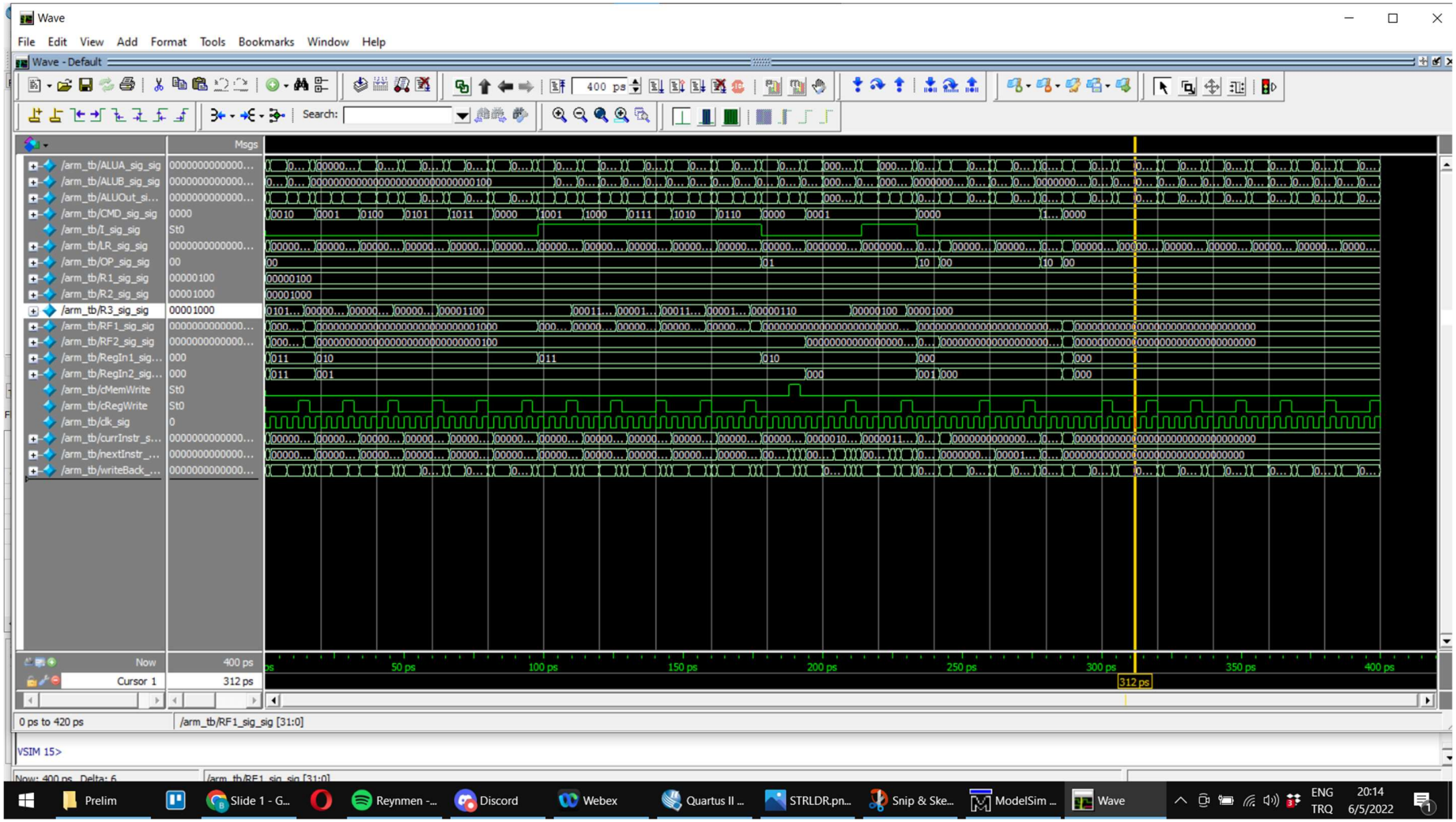
```
7   );
8   // 512x32 bits memory
9   reg [W-1:0] mem [0:63];
10
11  integer i;
12  initial begin
13  // cond(4)_op(2'b00)_I(1)_cmd(4)_s(1)_rn(4)_rd(4)_smth5_sh_0_Rm(4)
14     //mem[0] <= 32'b0000_00_0_0000_0_0010_0011_000000000001;//ADD R3,R2,R1
15     mem[0] <= 32'b0000_00_0_0010_0_0011_0011_000000000011;//CLEAR R3,R2,R1
16     //mem[0] <= 32'b0000_10_0_0_00000000000000000001000;//B  ??
17     mem[1] <= 32'b0000_00_0_0001_0_0010_0011_000000000001;//SUB R3,R2,R1
18     mem[2] <= 32'b0000_00_0_0100_0_0010_0011_000000000001;//AND R3,R2,R1
19     //mem[2] <= 32'b1;
20     mem[3] <= 32'b0000_00_0_0101_0_0010_0011_000000000001;//OR R3,R2,R1
21     mem[4] <= 32'b0000_00_0_1011_0_0010_0011_000000000001;//XOR R3,R2,R1
22     //mem[5] <= 32'b0000_00_0_0010_0_0011_0011_000000000011;//CLEAR R3,R2,R1
23     mem[5] <= 32'b0000_00_0_0000_0_0010_0011_000000000001;//ADD R3,R2,R1
24     //mem[0] <= 32'b0000_00_0_0000_0_0010_0011_000000000001;//ADD R3,R2,R1
25     mem[6] <= 32'b0000_00_1_1001_0_0011_0011_00011_00_0_0001;//RL XX,R3,XX(SHMT5=1)
26     mem[7] <= 32'b0000_00_1_1000_0_0011_0011_00011_00_0_0001;//RR XX,R3,XX(SHMT5=1)
27     mem[8] <= 32'b0000_00_1_0111_0_0011_0011_00011_00_0_0001;//SL XX,R3,XX(SHMT5=1)
28     mem[9] <= 32'b0000_00_1_1010_0_0011_0011_00011_00_0_0001;//ASR XX,R3,XX(SHMT5=1)
29     mem[10] <= 32'b0000_00_1_0110_0_0011_0011_00011_00_0_0001;//LSR XX,R3,XX(SHMT5=1)
30     //cond(4)_op(2)_I(1)_PUBW(4)_L(1)_Rn(4)_Rd(4)_src(12)
31     mem[11] <= 32'b0000_01_0_0000_0_0010_0001_000000000000;//STR
32     mem[12] <= 32'b0000_01_0_0001_0_0010_0011_000000000000;//LDR1
33     mem[13] <= 32'b0000_01_1_0001_0_0010_0011_000000001000;//LDR2
```



Note: At the beginning, it is doing Arithmetic operations in the previous part**.**

```
mem[11] <= 32'b0000_01_0_0000_0_0010_0001_000000000001;//STR R1,[R2]
mem[12] <= 32'b0000_01_0_0001_0_0010_0011_000000000000;//LDR R3,[R2]
mem[13] <= 32'b0000_01_1_0001_0_0010_0011_000000001000;//MOV R3,#8
```

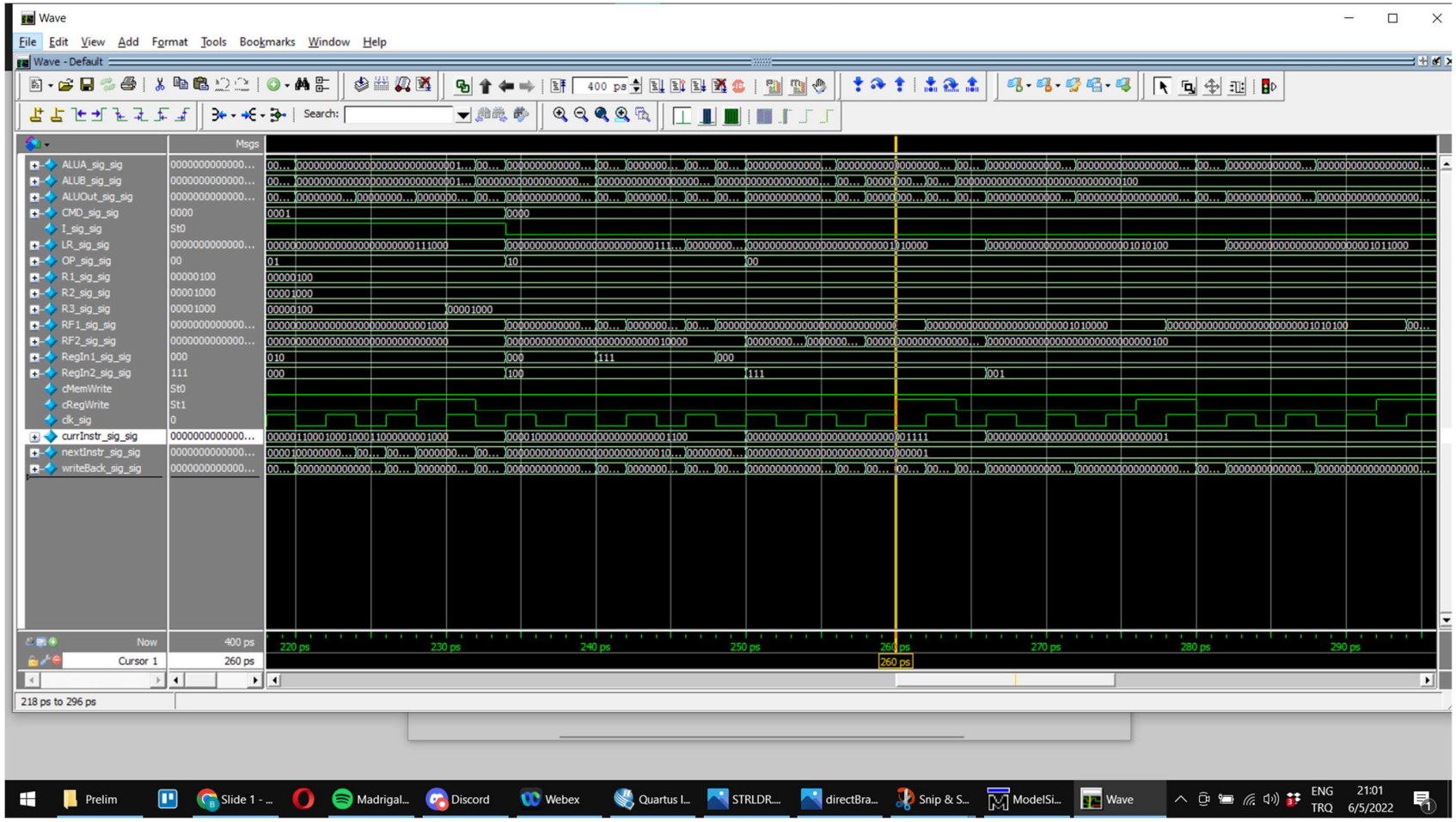
Initial of these instructions (at about 175 ps , you can check where OP_sig_sig=01), R1=0100 ; R2=1000; and R4 has 0110 in binary format. At first R1 is written to [R2] then it is taken into R3. After that, R3 is assigned to 8 in decimal.



```
//cond(4)_op(2)_x(1)_L(1)_src(24)
mem[14] <= 32'b0000_10_0_0_00000000000000000001100;//B  19
mem[15] <= 32'b0000_00_0_0000_0_0000_0000_000000010000;//Location that will be used in LDR/STR
mem[16] <= 32'b0;//EMPTY LOCATION
mem[17] <= 32'b0;//EMPTY LOCATION
mem[18] <= 32'b0000_00_0_0_00000000000000000000110;//BL  49
mem[19] <= 32'b0000_00_0_0_00000000000000000000111;//ADD R3,R2,R1
for(i=20;i<48;i=i+1)
```

Branch instruction at the location 14 will move PC to 19. Check the figure below. At about 235 ps, where OP_sig_sig == 10, PC is showing 14. (check currInstr_sig at the bottom). Then after executing this branch instruction (at 250ps), it went to instruction of 00..00111 , which is the mem[19].

Also, I was not be able to implement branch with conditions ☹.

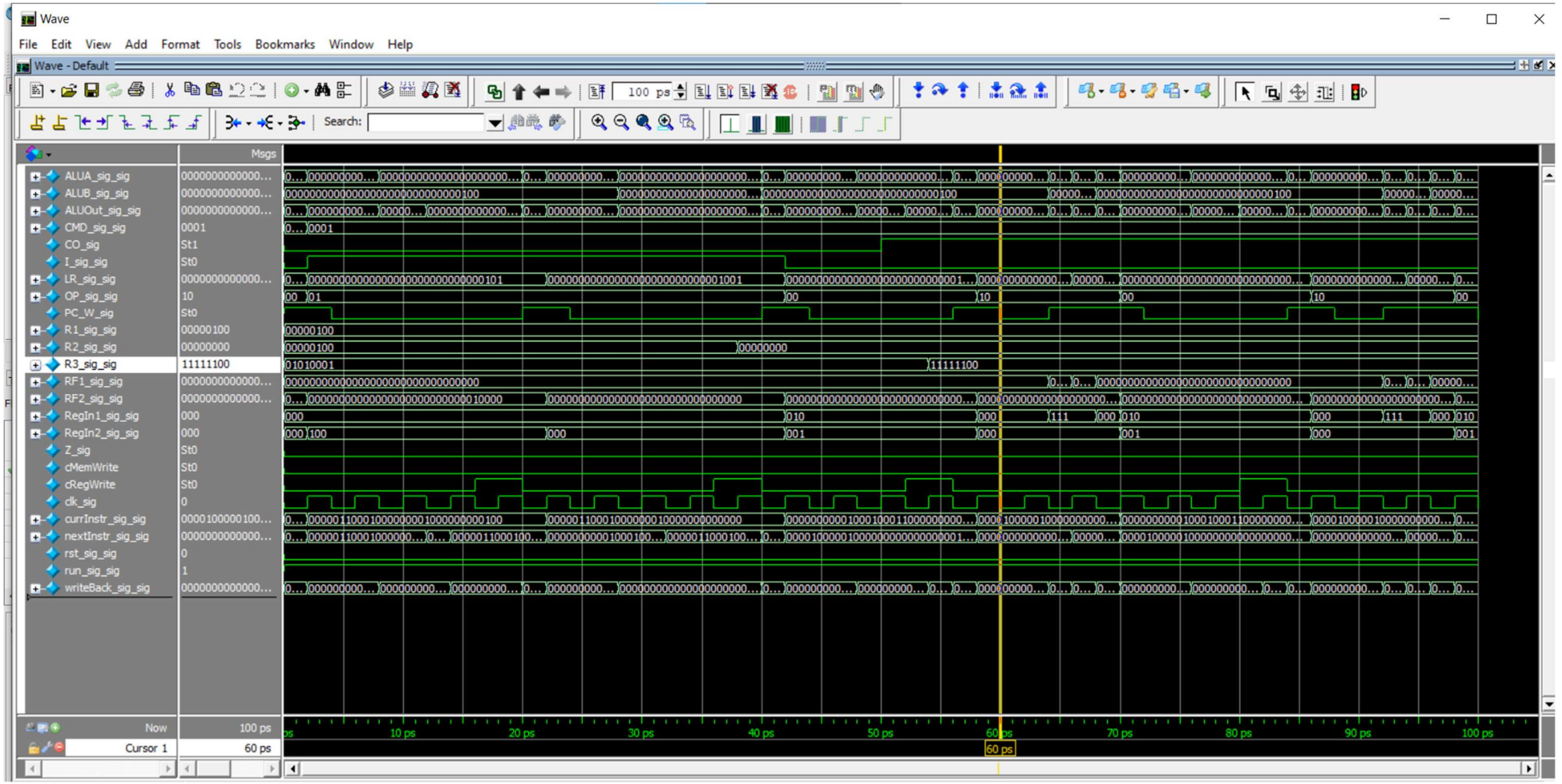

Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
ALUA_sig_sig
ALUB_sig_sig
ALUOut_sig_sig
CMD_sig_sig
I_sig_sig
LR_sig_sig
OP_sig_sig
R1_sig_sig
R2_sig_sig
R3_sig_sig
RF1_sig_sig
RF2_sig_sig
RegIn1_sig_sig
RegIn2_sig_sig
cMemWrite
cRegWrite
clk_sig
currInstr_sig_sig
nextInstr_sig_sig
writeBack_sig_sig
Now
400 ps
Cursor 1
260 ps
220 ps
230 ps
240 ps
250 ps
260 ps
270 ps
280 ps
290 ps
218 ps to 296 ps
Prelim
Slide 1 - ...
Madrigal...
Discord
Webex
Quartus I...
STRLDR...
directBra...
Snip & S...
ModelSi...
Wave
ENG
TRQ
21:01
6/5/2022

1.2.2 Validation of Operation via Microprogrammed Control (30% Credits)

1. Write a subroutine that get an 8-bit number and computes it 2’s complement.

```
MOV R1, #INPUT
MOV R2, #0
SUB R3,R2,R1
//Result is in R3
```

EX: (INPUT IS 0100)



2. Write a subroutine that computes the sum of an array of numbers and stores it in a memory location. The length of the array is constant and can be taken as 5.

Since I couldn't implement Branch with condition flags, I will be solving this problem with a long method ☹

```
MOV R1, #inputLocation
MOV R3,#4
MOV R2,#0
LDR R4,[R1]     //Get the number from array
ADD R1,R1,R3    //Increment Location
ADD R2,R2,R4    //Update counter
LDR R4,[R1]     //Get the number from array
ADD R1,R1,R3    //Increment Location
ADD R2,R2,R4    //Update counter
LDR R4,[R1]     //Get the number from array
ADD R1,R1,R3    //Increment Location
ADD R2,R2,R4    //Update counter
LDR R4,[R1]     //Get the number from array
ADD R1,R1,R3    //Increment Location
ADD R2,R2,R4    //Update counter
LDR R4,[R1]     //Get the number from array
ADD R1,R1,R3    //Increment Location
```

ADD R2,R2,R4 //Update counter

STR R2,[R1]

//Result is at the end of array and in the R2

Ex:

In the example loop is:

LDR R3,[R1]

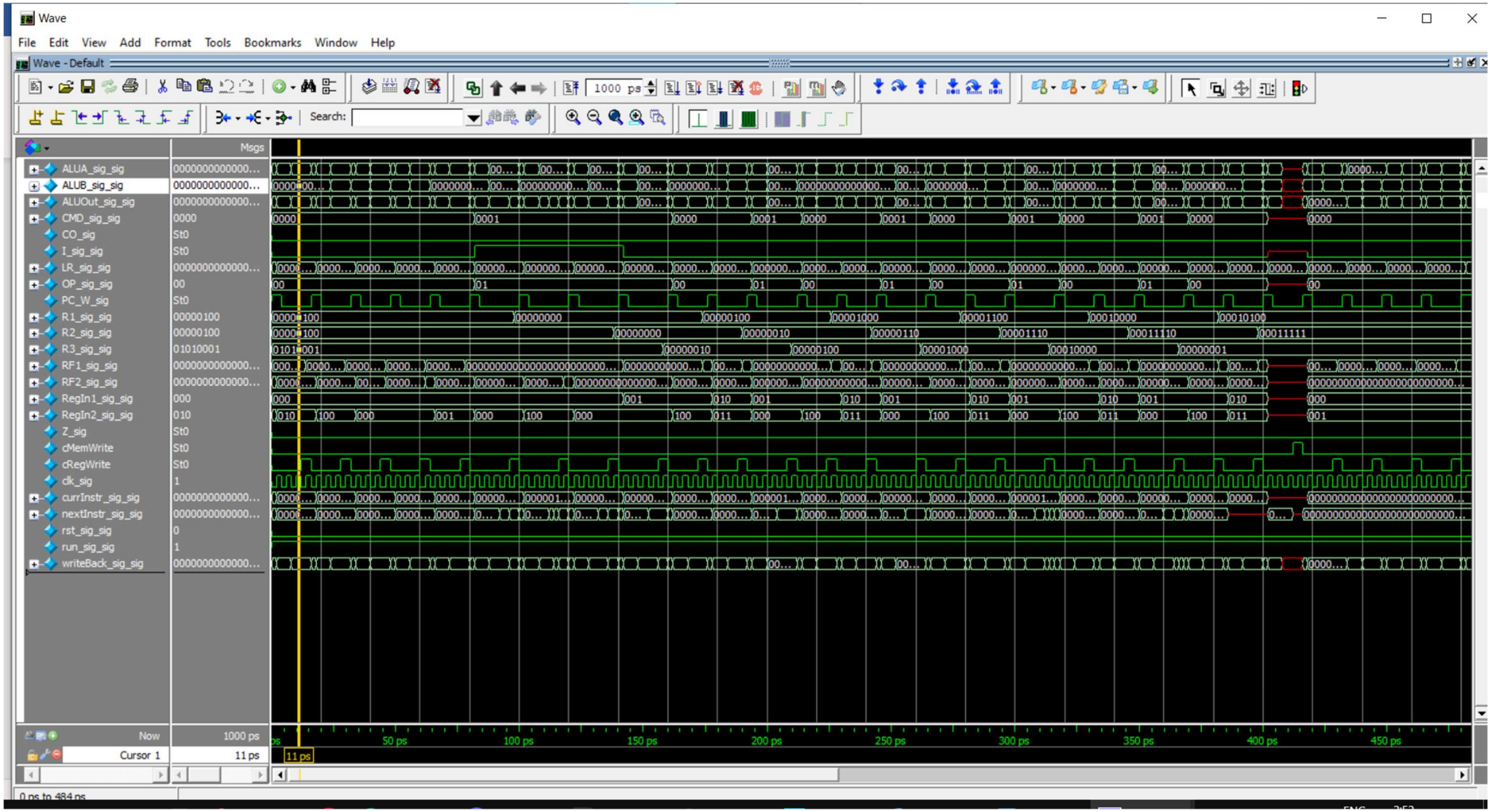
ADD R1,R1,R4

ADD R2,R2,R3

And input is 0, therefore array is:

```
mem[0] <= 32'b0000_00_0_0000_0_0000_0000_000000000010;//array of first element
mem[1] <= 32'b0000_00_0_0000_0_0000_0000_000000000100;//XX
mem[2] <= 32'b0000_00_0_0000_0_0000_0000_000000001000;//XX
mem[3] <= 32'b0000_00_0_0000_0_0000_0000_000000010000;//XX
mem[4] <= 32'b1;//XXX
```

And output is: (See the R2 value as counter, at the end it gives red line, which means in board implementation it will be crushed at the final element ☹)


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
ALUA_sig_sig
ALUB_sig_sig
ALUOut_sig_sig
CMD_sig_sig
CO_sig
I_sig_sig
LR_sig_sig
OP_sig_sig
PC_W_sig
R1_sig_sig
R2_sig_sig
R3_sig_sig
RF1_sig_sig
RF2_sig_sig
RegIn1_sig_sig
RegIn2_sig_sig
Z_sig
cMemWrite
cRegWrite
clk_sig
currInstr_sig_sig
nextInstr_sig_sig
rst_sig_sig
run_sig_sig
writeBack_sig_sig
Now
1000 ps
Cursor 1
11 ps
50 ps
100 ps
150 ps
200 ps
250 ps
300 ps
350 ps
400 ps
450 ps
0 ps to 484 ps

3. Write a subroutine that determines the evenness/oddity of a 8-bit number, n7n6n5n4n3n2n1n0. If odd, the expected output is n4n3n2n10n7n6n5, otherwise, 0n4n3n2n1000.

```
        MOV R1,#input
        MOV R2,#0
        STR R1,[R2]     //Store the input
        LSR R1,#1       //Taking last bit -1
        LSL R1,#1       //Taking last bit -2
        LDR R3,[R2]     //Take the original one
        SUBS R1,R1,R3   //Taking last bit -3 (if Z)
        BEQ Even        //Branch if it is zero
        LSR R3,#1       //Odd Case 1 (get rid of n0)
        LSL R3,#1       //Odd Case 2 (get rid of n0)
        ROL R3,#3       //Odd Case 3 (Rotate left 3 times)
        B Exit          //Exit
Even    LSR R3,#1       //Even Case 1
        LSL R3,#4       //Even Case 2
        LSR R3,#1       //Even Case 3
Exit    B Exit
```

However, I do not have branch with condition. ☹ Therefore, I will be assuming all inputs are odd.

EX:

Let’s say we have 10101010 (and assuming it is odd as I mentioned). The result is shown below (10100010, check R1_sig_sig)