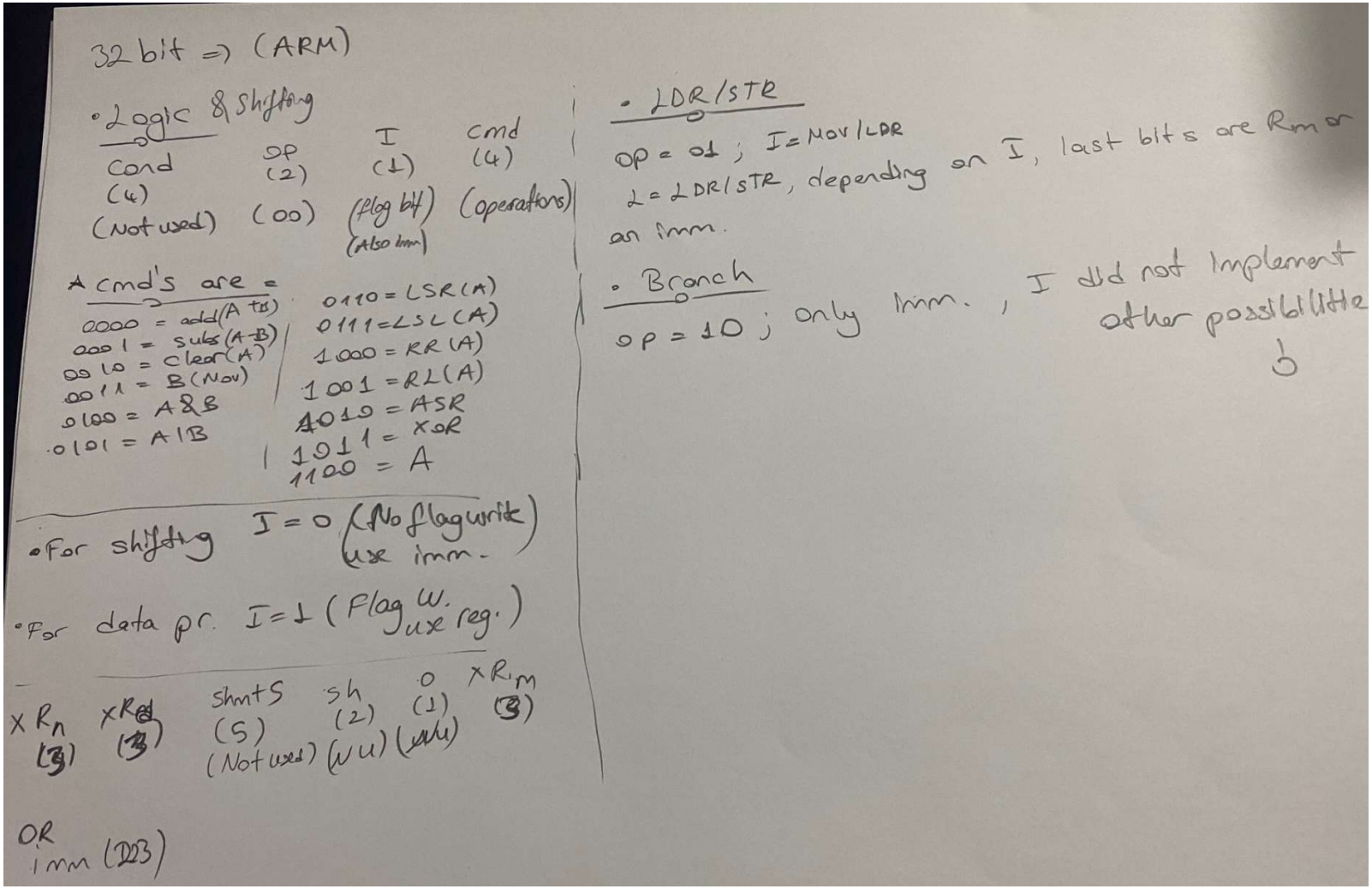
Berkay İPEK

EE446 -2304814

Laboratory Work 4 - ISA & Datapath Design for Multi-Cycle CPU

Contents

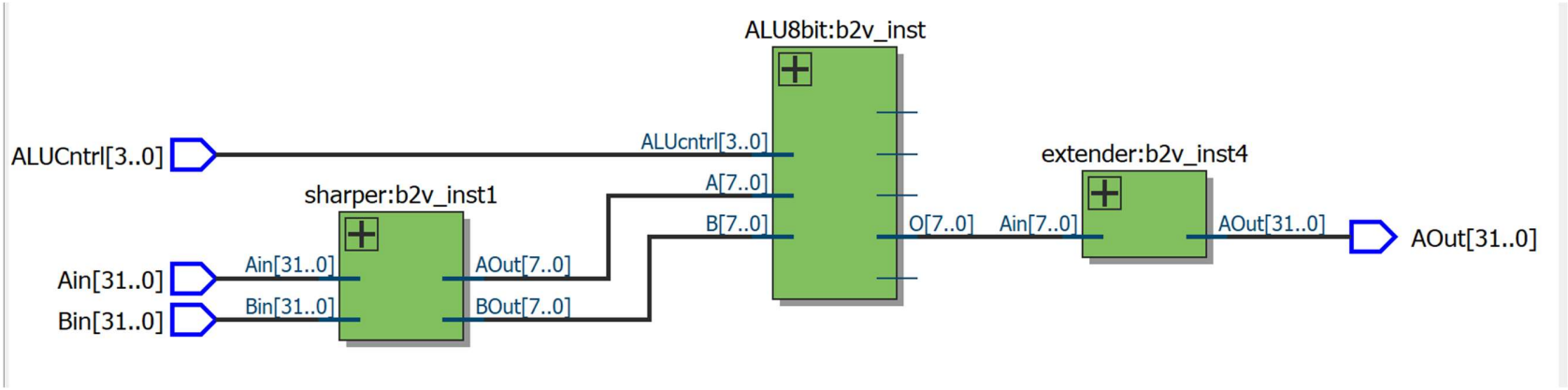
1.2.5 ISA Configuration (30% Credits)



1.2.6 Multi-cycle CPU Datapath Design (60% Credits)

1. Modify or rewrite the required modules for the register file, ALU, shift register and other functional components in Verilog HDL.

- Since we are going to do operations on 8-bit register, I did two different modules, which are extender and smaller. One of them is extending bits while other one decreases the input register's bit size. Therefore, although ALU inputs are 32 bits, I am downing its bit size to 8 bits, and doing the necessary operations and convert it to 32 bits again. (puts zeros)

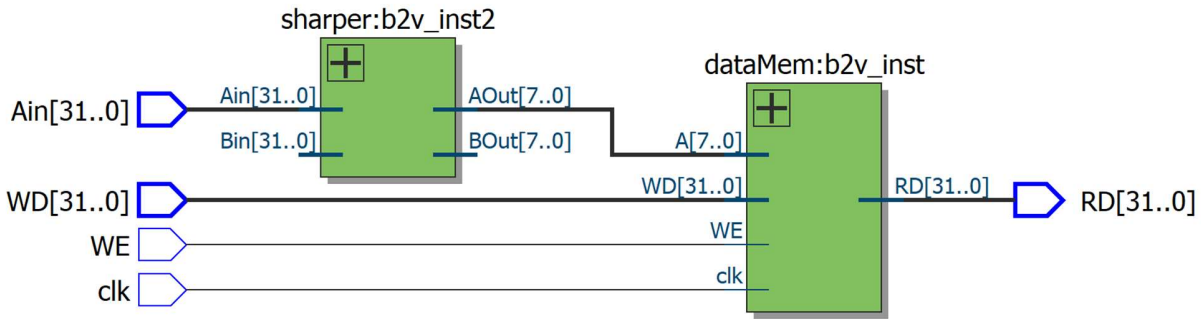


- Also, I added some necessary cases into the ALU. New methods are:
  - =A+B
  - =A-B
  - Clear A (=A & 8b'0)
  - Mov (=B)
  - A&B
  - A|B
  - LSR
  - LSL
  - RR
  - RL

    - ASR
    - XOR
    - (=A)
- I changed input of register size, since it is needed (8 Registers)
- I built non arch design unit as in the multi cycle processors. (with clock only)

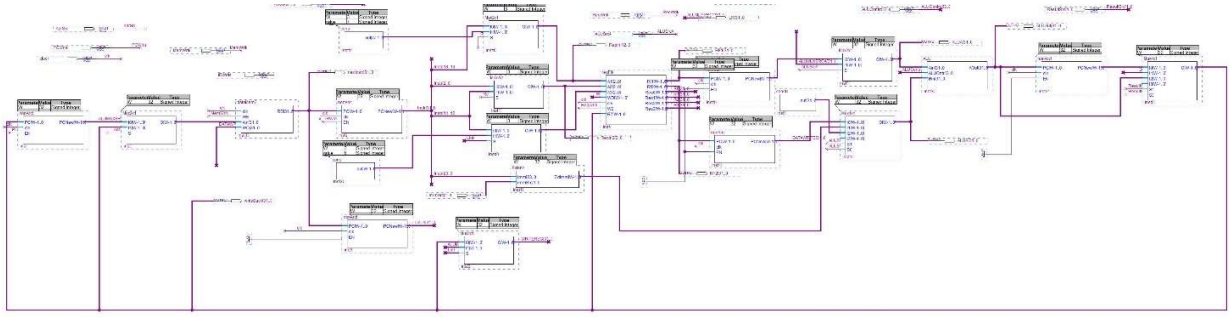
2. Design and implement the instruction/data memory module with respect to the required design constraints.

As I said in the previous question, I did 32 bit – 8 bit manipulations.



See the code in the zip file for dataMem struct. (It is almost same with the previous one except for data size!)

3. Design and implement the entire datapath using the schematics extracted for the functional modules and provide the wiring and bus connections for proper operation

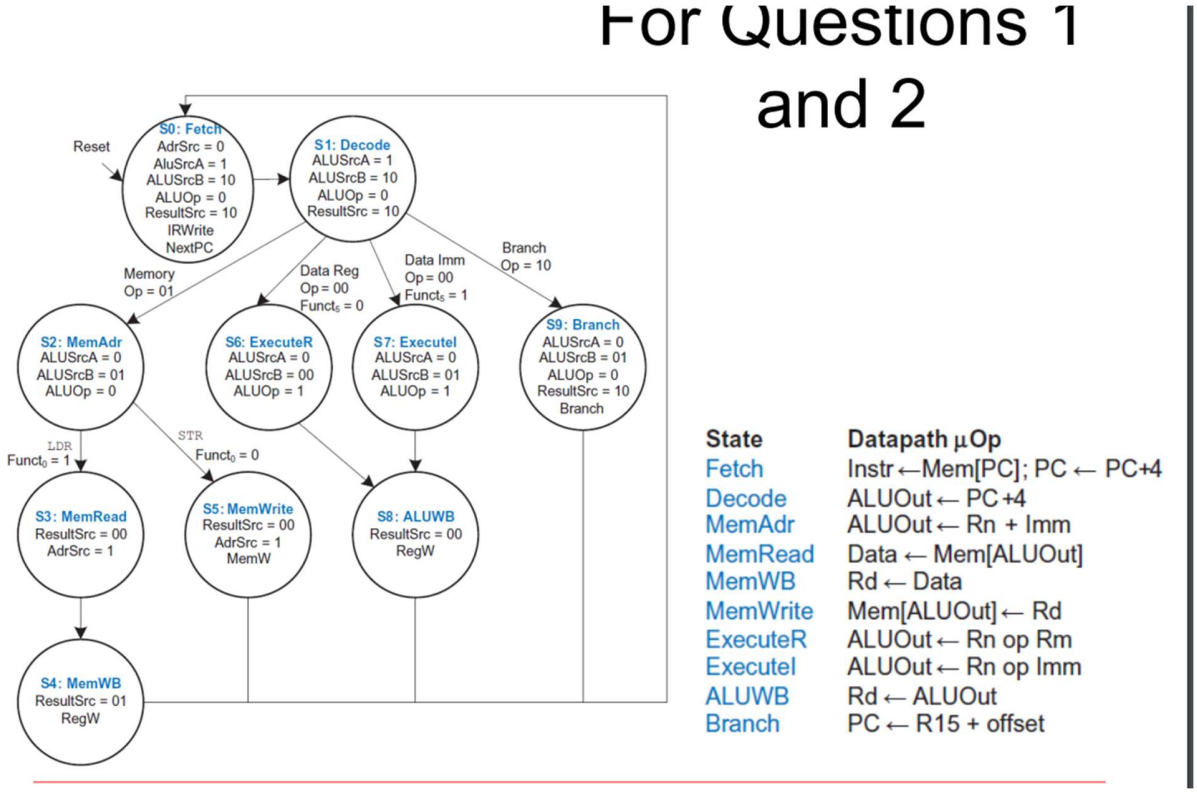


It is not possible to discuss all details of this design. It is same with the ARM instruction as you can guess. See module.bdf in the zip file.

Note: I think it does not make any sense to write test benches for all these operations since I am going to test all of my design in the next part.

1.2.7 Validation of Operation via TestBench (30% Credits)

1. Explain the required data flow within each cycle and state the control signals required to maintain the validity of operation.



Directly taken from lecture notes since it is ARM set. However, Data IMM part is not included in this report.

2. Write a testbench module including the control signals that you have stated in the previous step

See the “add_tb.v” to check addition & subtraction & logic operations , “branch_direct_tb.v” to check branch operations (no flag cases), “shift_tb.v” to check shifting operations , for the memory/register transfer operations go to “mem_tb.v” file.

3. Verify that your implementation is correct by providing the external signals with the proper timing that would obey the multi-cycle operation.

```
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[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[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
```

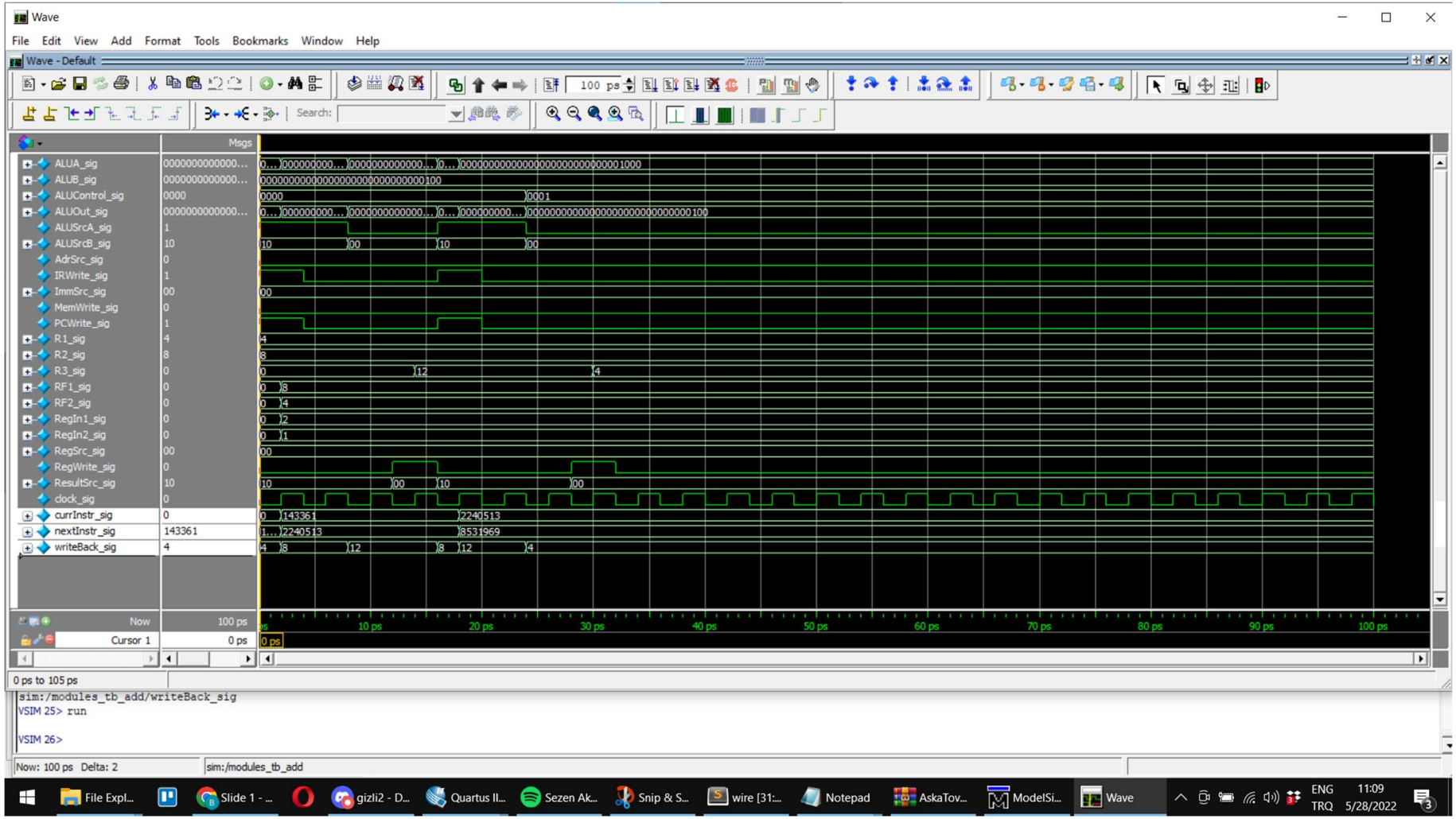
Instructions are given above. Also, initial value of registers are :

R1=4

R2=8

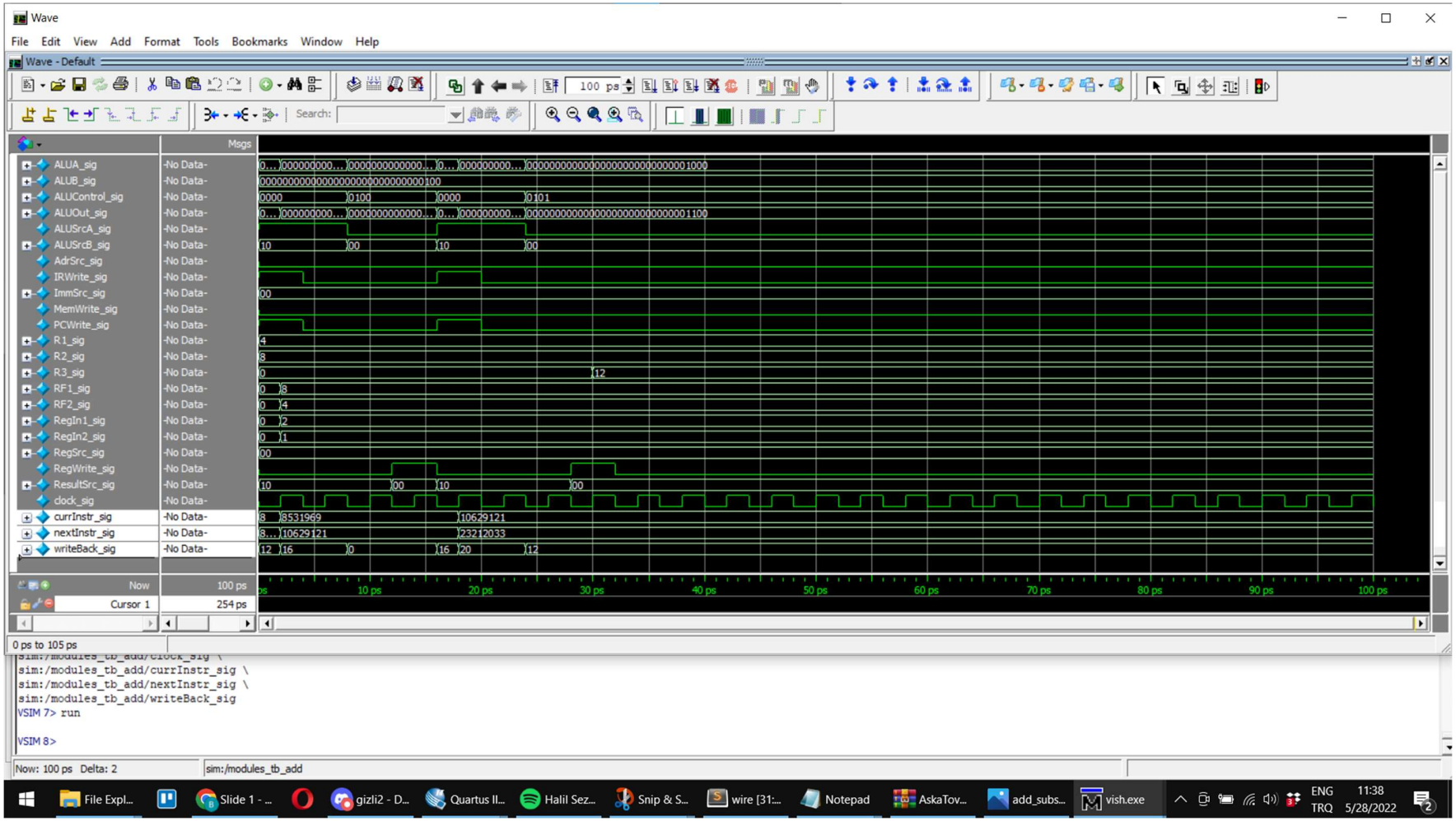
R3=0

- Addition & Subtraction (R1_sig , R2_sig, R3_sig)



As you can see figure above, in the operation it is added (8+4=12) and subtracted (8-4=4)

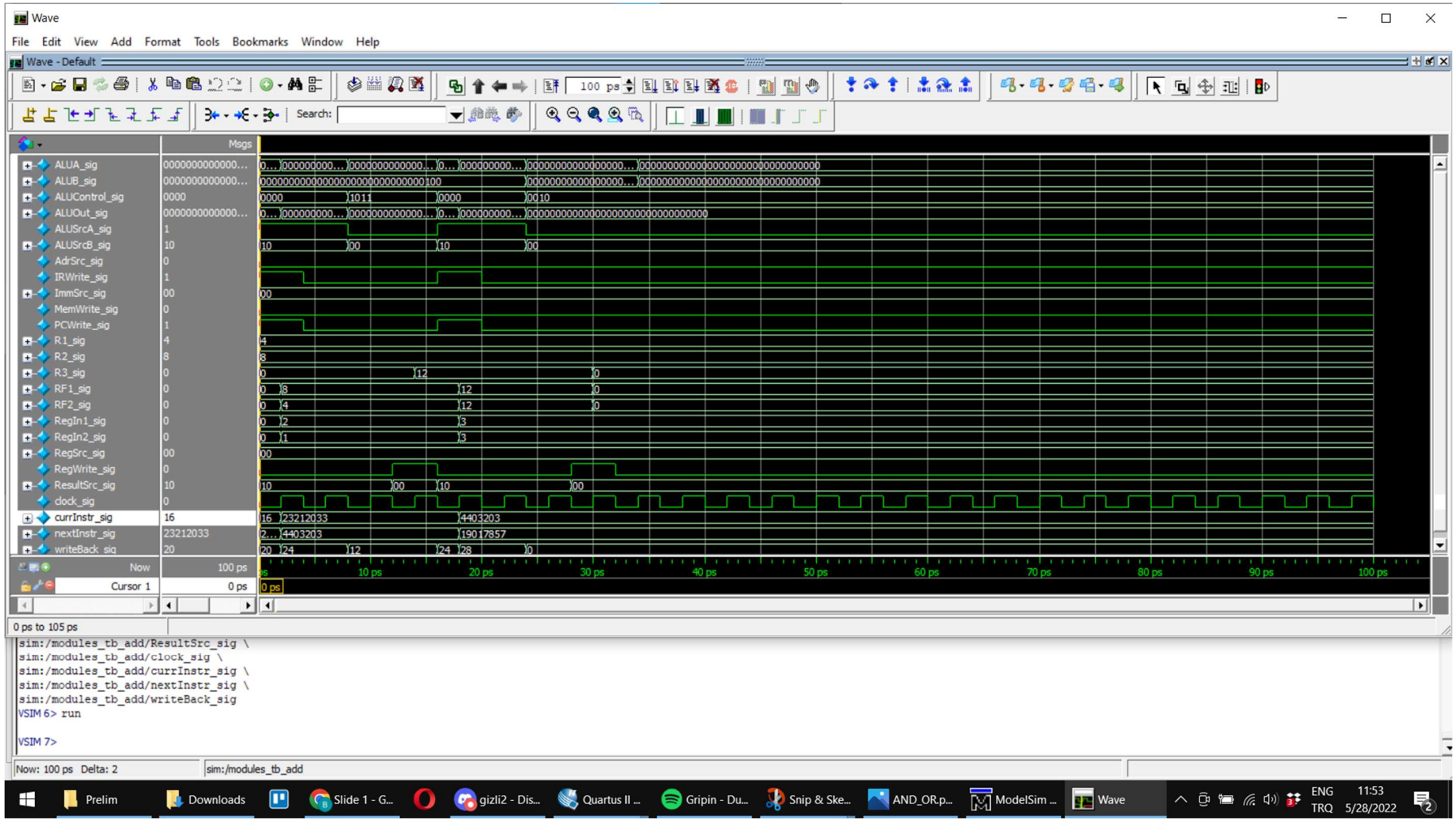
- **AND & OR (R1_sig, R2_sig, R3_sig)**



8 & 4 = 1000 & 0100 = 0000 = 0 (decimal)

8 | 4 = 1000 | 0100 = 1100 = 12 (decimal)

- **XOR & Clear (R1_sig, R2_sig, R3_sig)**

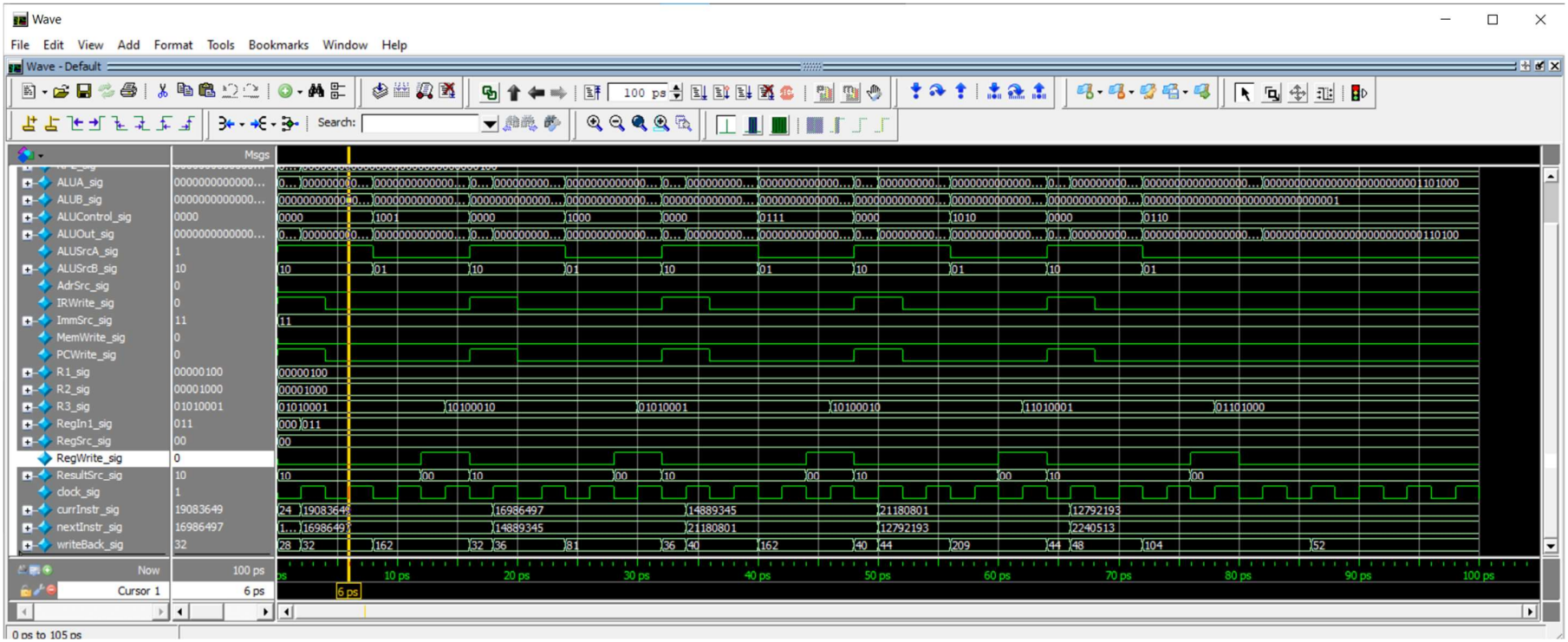


8 xor 4 = 1000 xor 0100 = 1100 = 12 (decimal)

Clear(R3) = 0

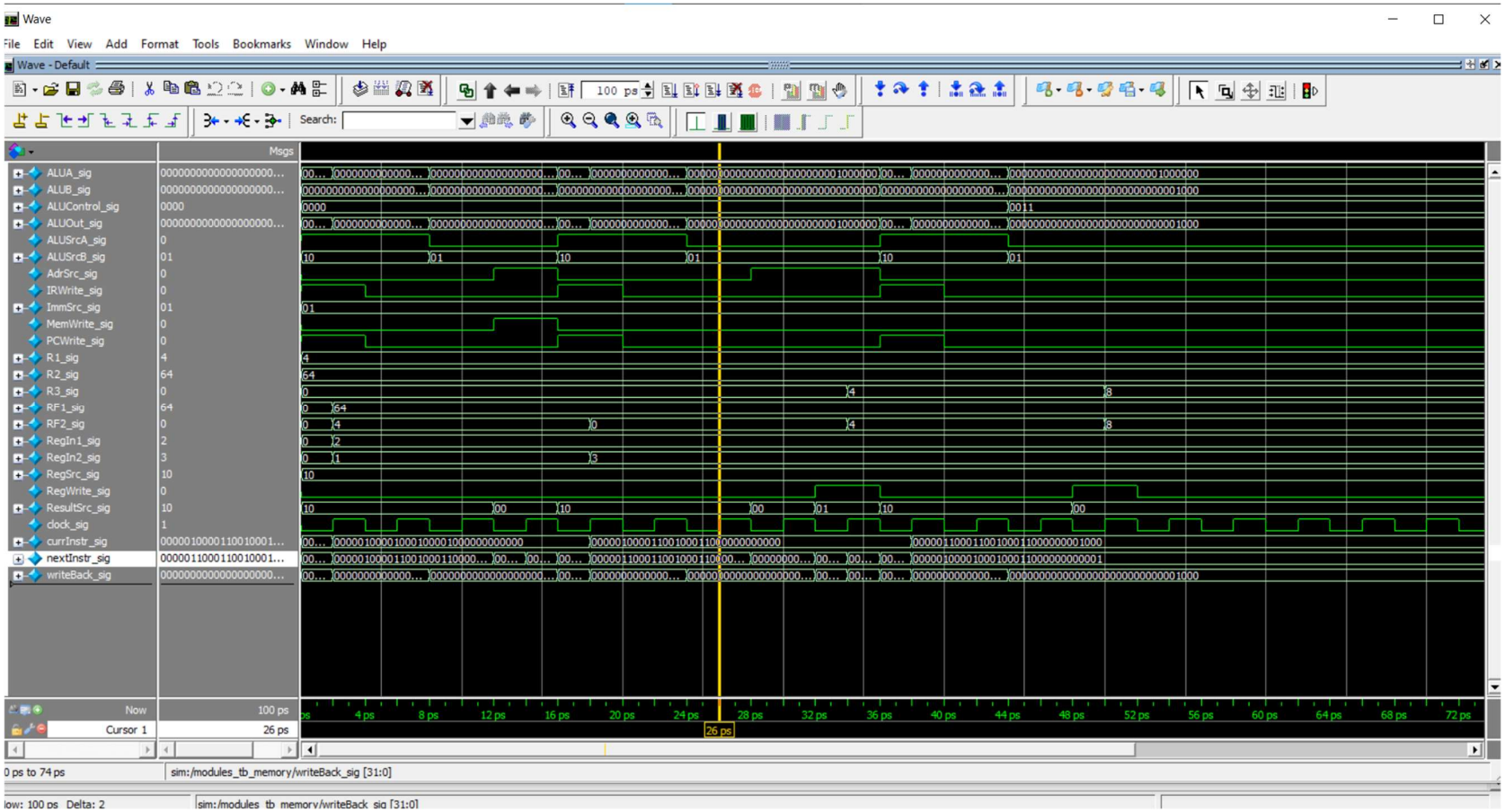
For logic operations: (Initial R3 value is 01010001 in binary, See **R3_sig**)

```
21
22
23      mem[6] <= 32'b0000_00_0_1001_0_0011_0011_00011_00_0_0001;//RL XX,R3,XX(SHMT5=1)
24      mem[7] <= 32'b0000_00_0_1000_0_0011_0011_00011_00_0_0001;//RR XX,R3,XX(SHMT5=1)
25      mem[8] <= 32'b0000_00_0_0111_0_0011_0011_00011_00_0_0001;//SL XX,R3,XX(SHMT5=1)
26      mem[9] <= 32'b0000_00_0_1010_0_0011_0011_00011_00_0_0001;//ASR XX,R3,XX(SHMT5=1)
27      mem[10] <= 32'b0000_00_0_0110_0_0011_0011_00011_00_0_0001;//LSR XX,R3,XX(SHMT5=1)
28
29
```



For data operations: (Initial R1 value is 4 in decimal, See **R3_sig**)

```
//cond(4)_op(2)_I(1)_PUBW(4)_L(1)_Rn(4)_Rd(4)_src(12)
mem[11] <= 32'b0000_01_0_0001_0_0010_0001_000000000000;//STR R1,[R2]
mem[12] <= 32'b0000_01_0_0001_1_0010_0011_000000000000;//LDR R3,[R2]
mem[13] <= 32'b0000_01_1_0001_1_0010_0011_000000001000;//LDR R3,#8
```



```
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_00000000001;//ADD R3,R2,R1
   mem[0] <= 32'b0000_10_0_0_00000000000000000001000;//B  ??
   mem[1] <= 32'b0000_00_0_0001_0_0010_0011_00000000001;//SUB R3,R2,R1
   //mem[2] <= 32'b0000_00_0_0100_0_0010_0011_00000000001;//AND R3,R2,R1
   mem[2] <= 32'b1;
   mem[3] <= 32'b0000_00_0_0101_0_0010_0011_00000000001;//OR R3,R2,R1
   mem[4] <= 32'b0000_00_0_1011_0_0010_0011_00000000001;//XOR R3,R2,R1
   mem[5] <= 32'b0000_00_0_0010_0_0011_0011_00000000011;//CLEAR R3,R2,R1
```

It is branching to mem[2]. (See the currIntrs_sig)