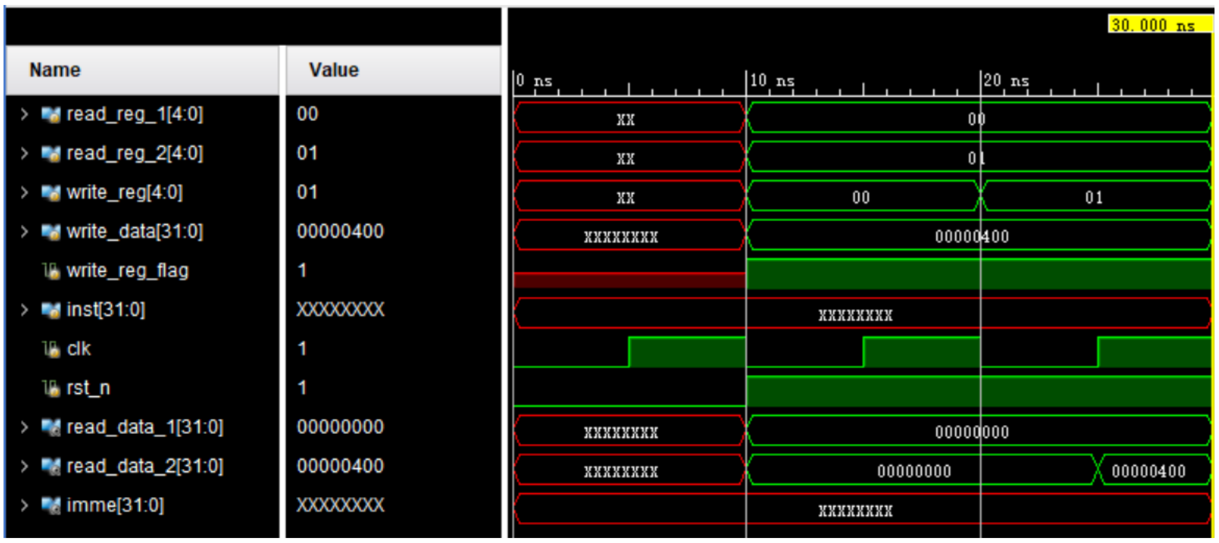
1 Ports of the Decoder module

Port Name	From	To	Bit Width	Description
`clk`	Clock signal generator	Decoder	1	clock signal
`rst_n`	Reset button	Decoder	1	reset all registers (active low)
`inst`	IFetch	Decoder	32	instruction
`read_reg_1`	IFetch	Decoder	5	first register to read
`read_reg_2`	IFetch	Decoder	5	second register to read
`write_reg`	IFetch	Decoder	5	register to write
`write_data`	Memory	Decoder	32	data to write
`write_reg_flag`	Control	Decoder	1	flag to write register
`read_data_1`	Decoder	ALU	32	data in `read_reg_1`
`read_data_2`	Decoder	ALU	32	data in `read_reg_2`
`imme`	Decoder	ALU	32	immediate value from `inst`

Table 1: Ports of the Decoder module

2 Testbench for the Decoder module

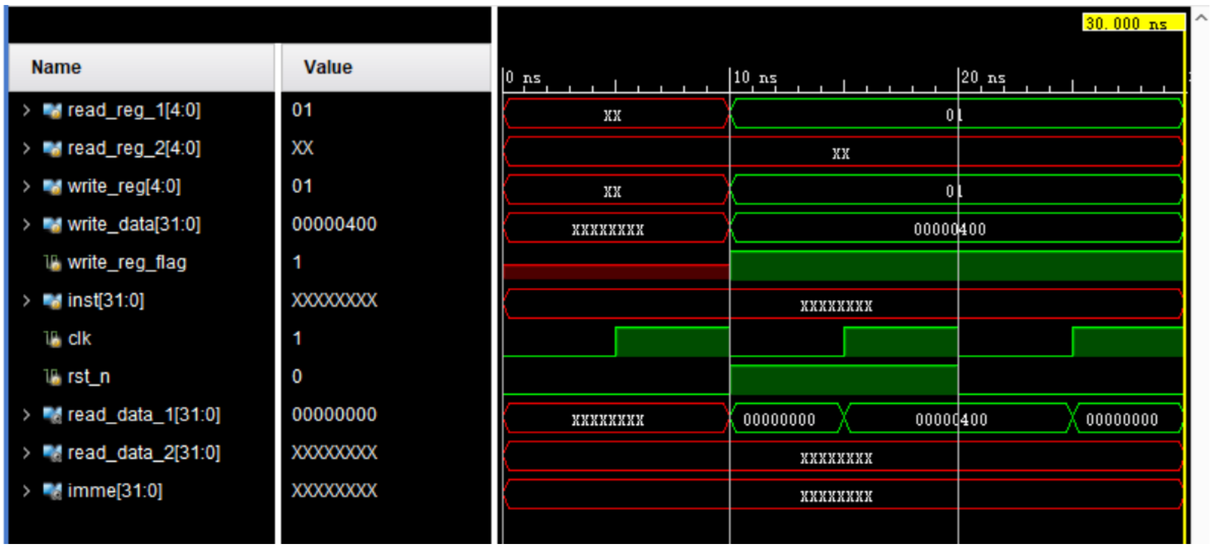


Name	Value
read_reg_1[4:0]	00
read_reg_2[4:0]	01
write_reg[4:0]	01
write_data[31:0]	00000400
write_reg_flag	1
inst[31:0]	XXXXXXXX
clk	1
rst_n	1
read_data_1[31:0]	00000000
read_data_2[31:0]	00000400
imme[31:0]	XXXXXXXX

Figure 1: Testbench 1

This testbench tests the following cases:

1. Register `x0` cannot be written.
2. Register `x1` is correctly written.
3. Register `x0` and `x1` are correctly read.
4. Register write operation only occurs on the positive edge of the clock signal.

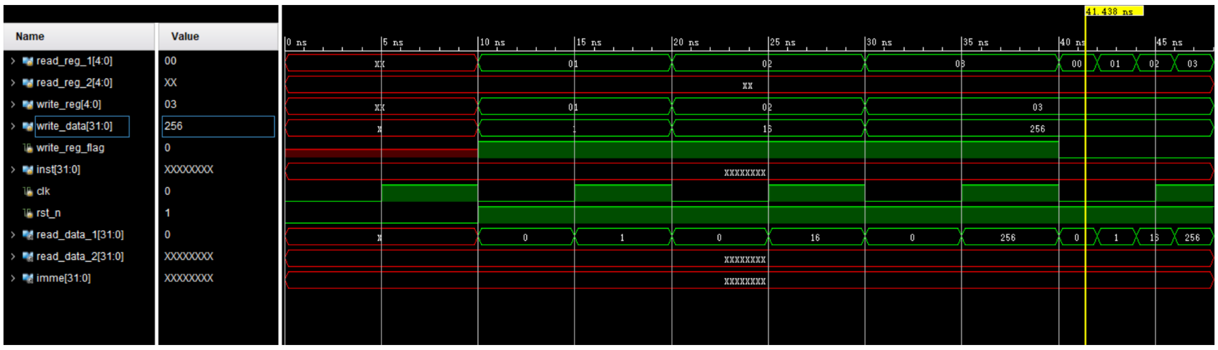


Name	Value
read_reg_1[4:0]	01
read_reg_2[4:0]	XX
write_reg[4:0]	01
write_data[31:0]	00000400
write_reg_flag	1
inst[31:0]	XXXXXXXX
clk	1
rst_n	0
read_data_1[31:0]	00000000
read_data_2[31:0]	XXXXXXXX
imme[31:0]	XXXXXXXX

Figure 2: Testbench 2

This testbench tests the following cases:

1. `rst_n` signal resets all registers (happen on the positive edge of the clock signal).
2. `rst_n` signal overrides the writing operation of the register.



Name	Value
read_reg_1[4:0]	00
read_reg_2[4:0]	XX
write_reg[4:0]	03
write_data[31:0]	256
write_reg_flag	0
inst[31:0]	XXXXXXXX
clk	0
rst_n	1
read_data_1[31:0]	0
read_data_2[31:0]	XXXXXXXX
imme[31:0]	XXXXXXXX

Figure 3: Testbench 3

This testbench tests the following cases:

1. The reading operation of the register can occur at any time (last 8ms of the simulation).

The immediate extraction is tested last week and is not included in this testbench.

3 Code for the Decoder module

```
`timescale 1ns / 1ps

module decoder(
input [4:0] read_reg_1,
input [4:0] read_reg_2,
input [4:0] write_reg,
input [31:0] write_data,
input write_reg_flag,
input [31:0] inst,
input clk,
input rst_n,
output [31:0] read_data_1,
output [31:0] read_data_2,
output [31:0] imme
);

immediate_generator imme_gen(.inst(inst), .imme(imme));

reg [31:0] register [0:31];
assign read_data_1 = register[read_reg_1];
assign read_data_2 = register[read_reg_2];

always @(posedge clk)
begin
    if(!rst_n)
    begin
        register[0] <= 32'b0;
        register[1] <= 32'b0;
        register[2] <= 32'b0;
        register[3] <= 32'b0;
        register[4] <= 32'b0;
        register[5] <= 32'b0;
        register[6] <= 32'b0;
        register[7] <= 32'b0;
        register[8] <= 32'b0;
        register[9] <= 32'b0;
        register[10] <= 32'b0;
        register[11] <= 32'b0;
        register[12] <= 32'b0;
        register[13] <= 32'b0;
        register[14] <= 32'b0;
        register[15] <= 32'b0;
        register[16] <= 32'b0;
        register[17] <= 32'b0;
        register[18] <= 32'b0;
        register[19] <= 32'b0;
        register[20] <= 32'b0;
        register[21] <= 32'b0;
        register[22] <= 32'b0;
        register[23] <= 32'b0;
        register[24] <= 32'b0;
        register[25] <= 32'b0;
        register[26] <= 32'b0;
        register[27] <= 32'b0;
        register[28] <= 32'b0;
        register[29] <= 32'b0;
        register[30] <= 32'b0;
        register[31] <= 32'b0;
    end else begin
        if(write_reg_flag && write_reg != 0)
        begin
            register[write_reg] <= write_data;
        end
    end
end


endmodule
```