ISTANBUL TECHNICAL UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

BLG 222E

Computer Organization
Project Report

Project NO : 1

Due Date : 12.04.2023

GROUP NO : G37

GROUP MEMBERS:

150200075 : Matay AYDIN

150200096 : Mustafa KIRCI

150200916 : Denıs Iurıe DAVIDOGLU

SPRING 2023

Contents

1 Introduction

Computers can be different by their architecture, instruction sets, peripherals and performance. However, the processors lying on their core perform same basic functions: fetching data and instructions from memory, carry logic operations, adding, multiplying, comparing, outputing results to the system bus, and so on. All of these are done using multiplexers, registers and arithmetic logic units. In this project, the aim is to design the most basic ALU system in Verilog HDL. All the required modules are described at the behavioral level, and checked detaily in testbenches. In part one, n-bit register with four functions is presented, which will be used in all succeeding parts. In general, the entire system is just a large network of registers; therefore, it is crucial. In the second part, three different register files are derived from the main register module, which will be used to store instructions, temporary data, as well as registers for executing a program in the correct order. Part three is the brain of the system, the arithmetic logic unit, which performs operations on two values, gives the result of computation, as well as flags for interpreting the data. In the last fourth part, all modules are combined to create a fully-functional computer, that connects to an external memory.

Responsibility distribution between team members is as follows:

- **Part 1:** Denis
- **Part 2a:** Denis
- **Part 2b:** Mustafa
- **Part 2c:** Denis
- **Part 3:** Matay
- **Part 4:** Mustafa
- **Report:** Denis, Mustafa, Matay

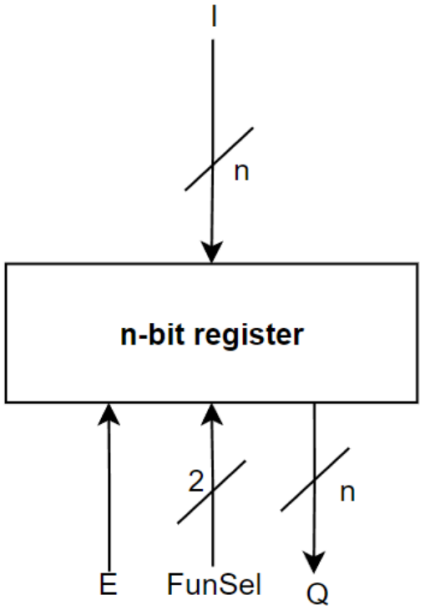
2 Project

2.1 Part 1

To get started with modeling on the register-transfer level, the most basic unit, the register, is needed. Plain register that Verilog offers is not enough, since we want a structure that will be synchronous with system clock, have enable input, two-bit function select input, load input data lines and output data lines. The table below describes the behaviour of our register:

E	FunSel	Q^+
0	ϕ	Q (retain value)
1	00	0 (Clear)
1	01	I (Load)
1	10	$Q - 1$ (Decrement)
1	11	$Q + 1$ (Increment)

Moreover, for this register to be universal, it should have a parametrized number of data lines. The block diagram on the left is equivalent to the Verilog code on the right.



(a) n-bit register diagram

part1.v

```
1  module register #(
2  parameter NBits = 16
3  ) (
4    input               clk,
5    input[1:0]          funsel,
6    input               e,
7    input[NBits-1: 0]   i,
8    output[NBits-1: 0]  q
9  );
```

(b) Module definition in Verilog

Although not drawn in the diagram, the clock signal is also an input, whose rising edge triggers operations, described in the function table given above. These simple rules were translated to Verilog too, by using statements such as **always**, **if** and **case**. Module's testbench creates a 4-bit register instance, which is a reasonable size for testing. Register is simulated for all functions, i.e. clear, load, decrement and increment:

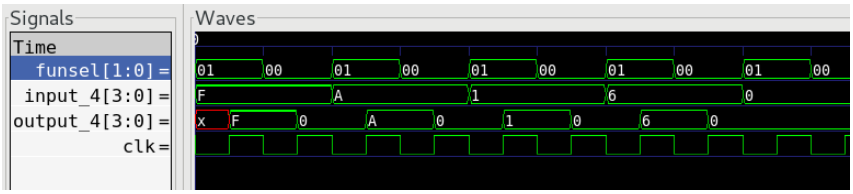


Figure 2: Load and clear test (part 1)

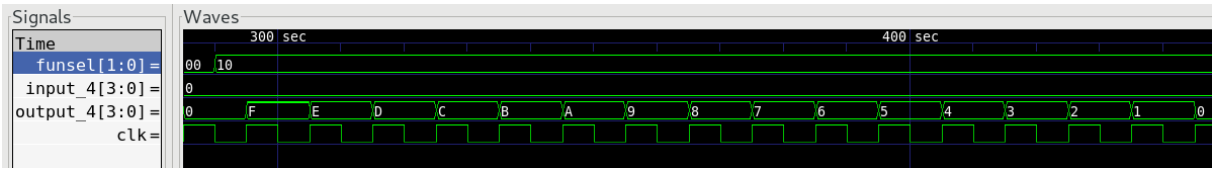


Figure 3: Decrement test (part 1)

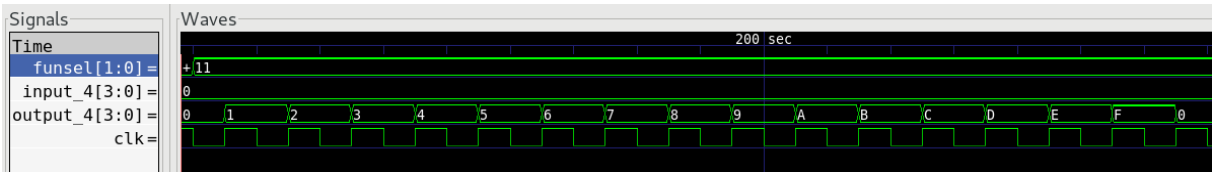


Figure 4: Increment test (part 1)

Note: In the testbench, enable signal was always high and was not tested for low value, since the implementation already guarantees changes in output only for the high value.

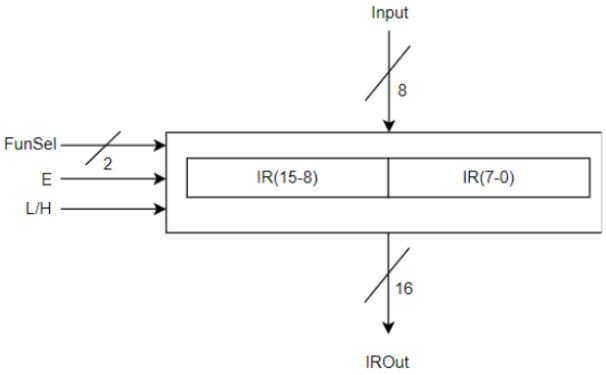
2.2 Part 2

2.2.1 Part 2a

In this part, a 16-bit instruction register design is provided. While the internal structure contains a 16-bit instance of the register from previous part, data is coming in in halves, 8 bit at a time. The selection between bits is done by an additional $\overline{\mathbf{L}}/\mathbf{H}$ input, and the other functionalities remain the same, as described in the table:

$\overline{\mathbf{L}}/\mathbf{H}$	**E**	**FunSel**	$\mathbf{IR}^{+}$
ϕ	0	$\phi\phi$	IR (Retatin Value)
ϕ	1	00	0 (Clear)
0	1	01	IR(7−0) ← I (Load LSB)
1	1	01	IR(15−8) ← I (Load MSB)
ϕ	1	10	IR−1 (Decrement)
ϕ	1	11	IR+1 (Increment)

From the graphical representation on left, Verilog module is derived:



(a) 16-bit instruction register diagram

part2a.v

```
3   module IR_16_bit (
4     input          clk,
5     input[7:0]     i_half,
6     input[1:0]     funsel,
7     input          e,
8     input          l_h,
9     output[15:0]   ir_out
10  );
```

(b) Module definition in Verilog

Clock signal is not connected to the internal n-bit register module directly. Instead, it uses a delayed clock, so that there are no race conditions with the parent module. We want first to process the inputs in the parent module, then after a delay, process them in the internal one. As a result, the output of instruction register changes slightly after the main clock's rising edge:

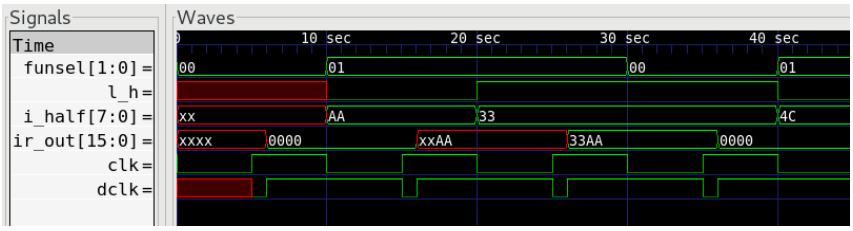


Figure 6: Delayed clock (part 2a)

Testbench includes similar tests as for n-bit register module:

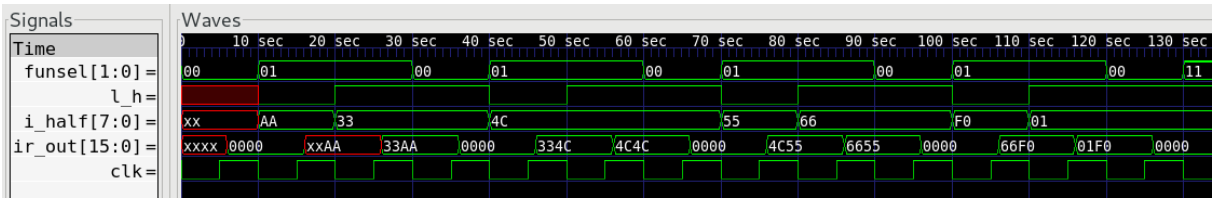


Figure 7: Load and clear test (part 2a)

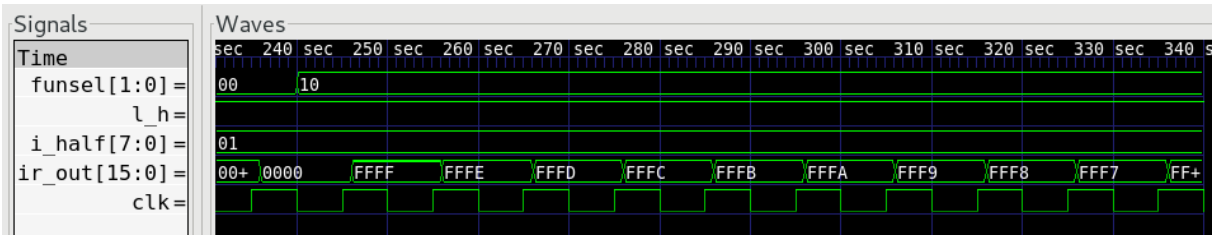


Figure 8: Decrement test (part 2a)

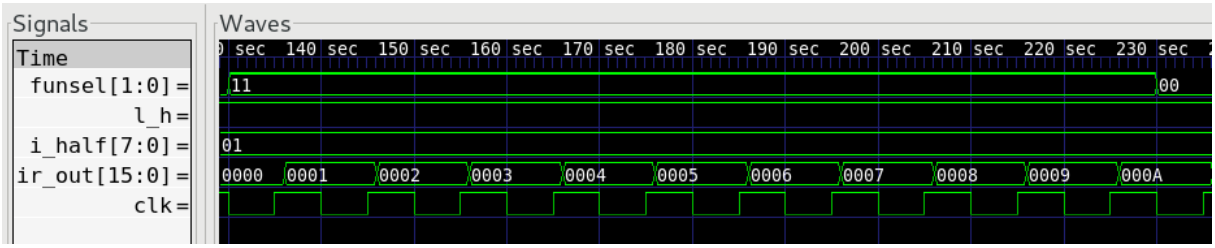


Figure 9: Increment test (part 2a)

2.2.2 Part 2b

The purpose of the register file (RF) module is to control the 4 8-bit general purpose registers and 4 8-bit temporary registers. To achieve this, module has 6 inputs. Those are: i, O1Sel, O2Sel, FunSel, RSel and TSel. These i input wire provides the bits that will be loaded into the active registers if the given FunSel allowes it. O1Sel and O2Sel selects the outputs of registers for the output of the module. FunSel determines which operation the active registers will compute. Lastly, RSel and TSel are used to determine active registers. OSel inputs work as if they represent the index of the register. Register selection inputs works as if each bit represents the enable-disable state of the corresponding register. The details are given in the project describtion file.

Registers inside this module are connected to delayed clock. The reason for this implementation detail is to avoid the race conditions. If same clock is given to registers, undetermined behaviours occurs at each positive clock edge.
The schematics of register file is as below.

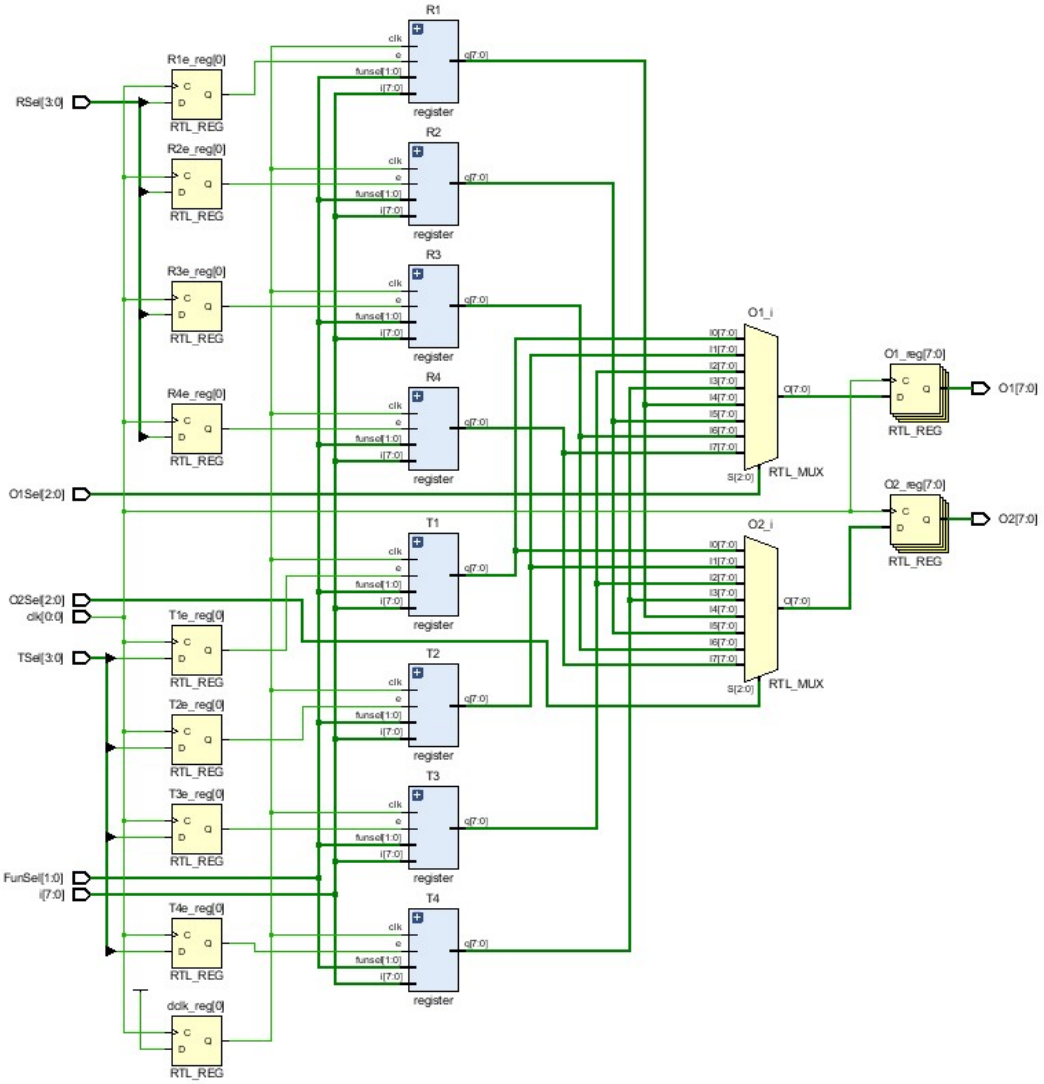


Figure 10: Schematics of Register File

For testing the RF module, FunSel is given as 01 so that module is initialized correctly. To initialize every register inside the RF, ALL registers are enabled.
RSel = 1111, TSel = 1111, FunSel = 01

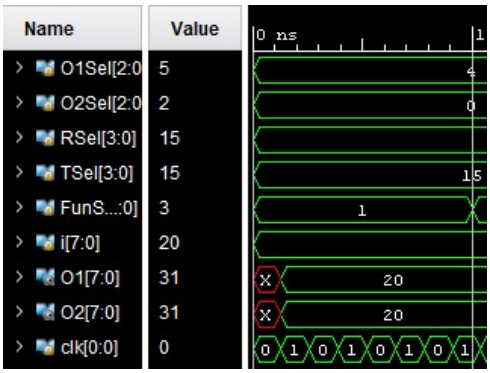


Figure 11: Loading Test (part 2b)

Testing incrementing while all registers are enabled.

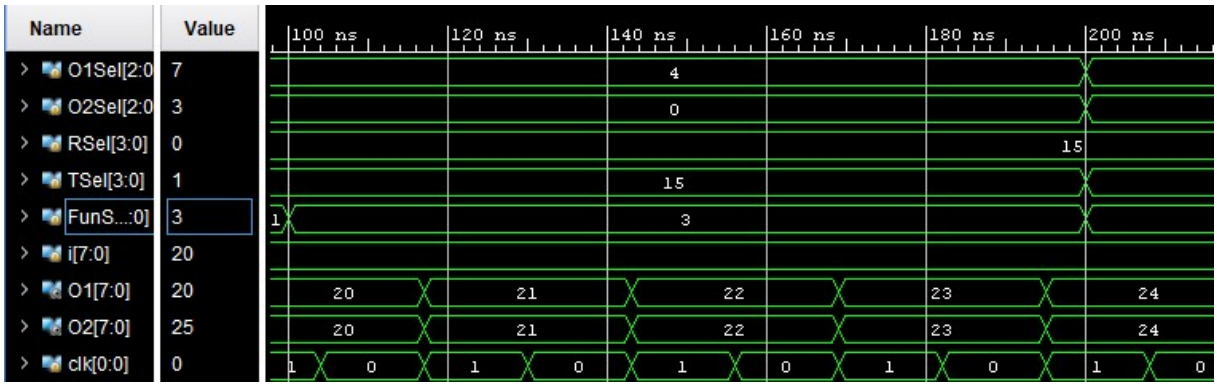


Figure 12: Increment Test (part 2b)

Testing decrementing while all registers are enabled.

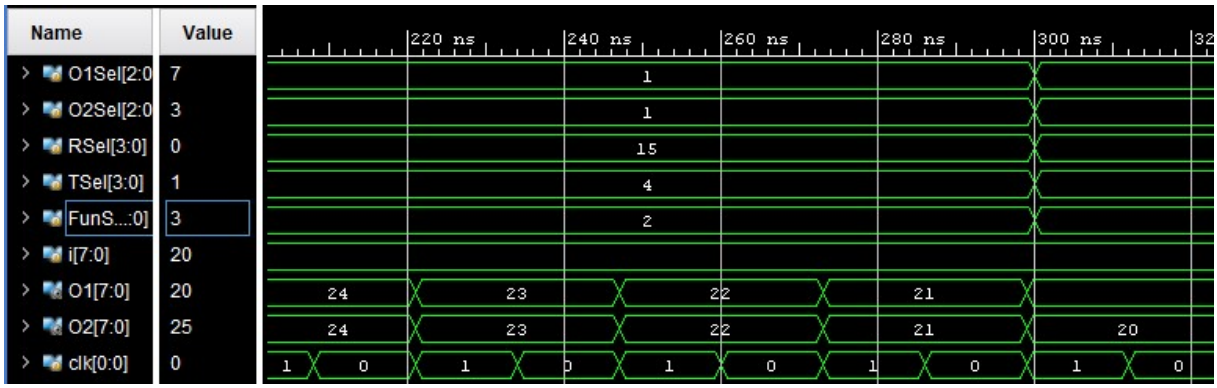


Figure 13: Decrement Test (part 2b)

Disabling every general purpose register and enabling only T4. Selecting R4 for O1 and T4 for O2 outputs. With these conditions, only O2 should change its value at first because it was disabled before decrement. After that clock signal, only O2 should increment.

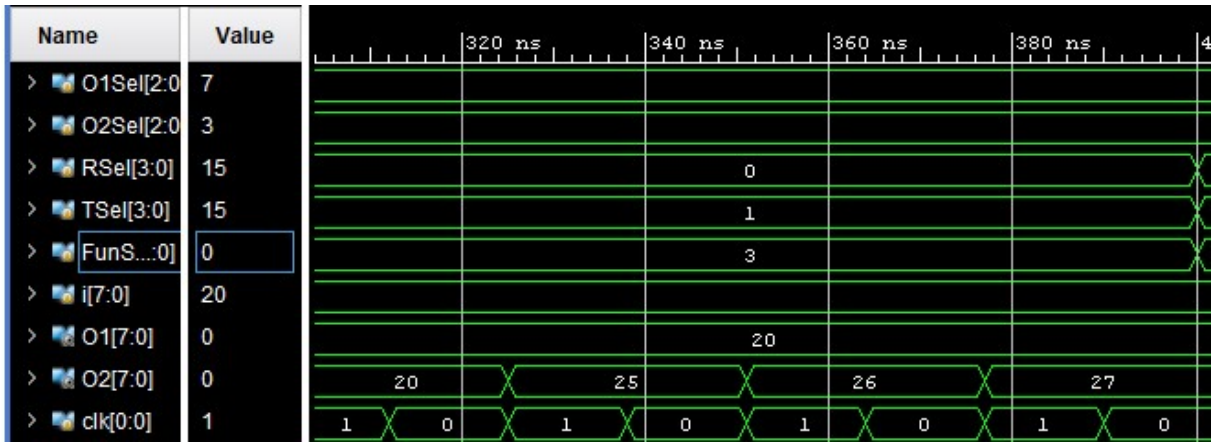


Figure 14: Deactivate Register and Switch Output Test (part 2b)

Testing clearing while all registers are enabled.

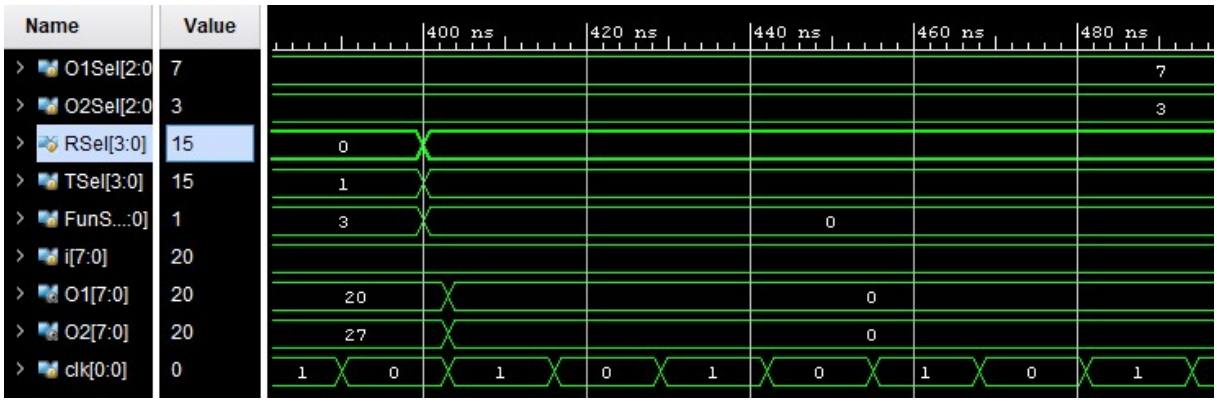


Figure 15: Reset Test (part2b)

All conducted tests are as below.

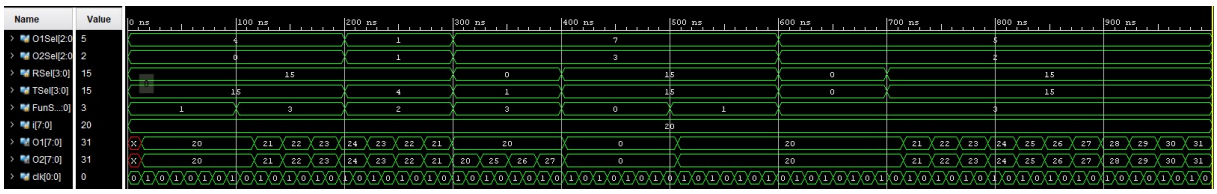
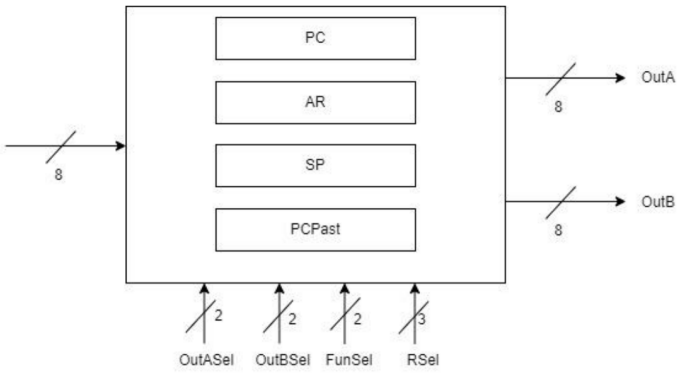


Figure 16: Simulation Diagram As a Whole (part 2b)

2.2.3 Part 2c

Adress register file (ARF) contains four 8-bit registers, and has a single 8-bit input, two 8-bit output ports, two 2-bit selectors for the outputs, 2-bit function select input and 4-bit register enable select input:



(a) ARF diagram

part2c.v

```
3  module ARF(
4    input           clk,
5    input[7:0]      i,
6    input[1:0]      out_b_sel,
7    input[1:0]      out_a_sel,
8    input[1:0]      funsel,
9    input[3:0]      r_sel,
10   output reg[7:0] out_a,
11   output reg[7:0] out_b
12 );
```

(b) Module definition in Verilog

Output selectors choose which register's data is shown on the corresponding port:

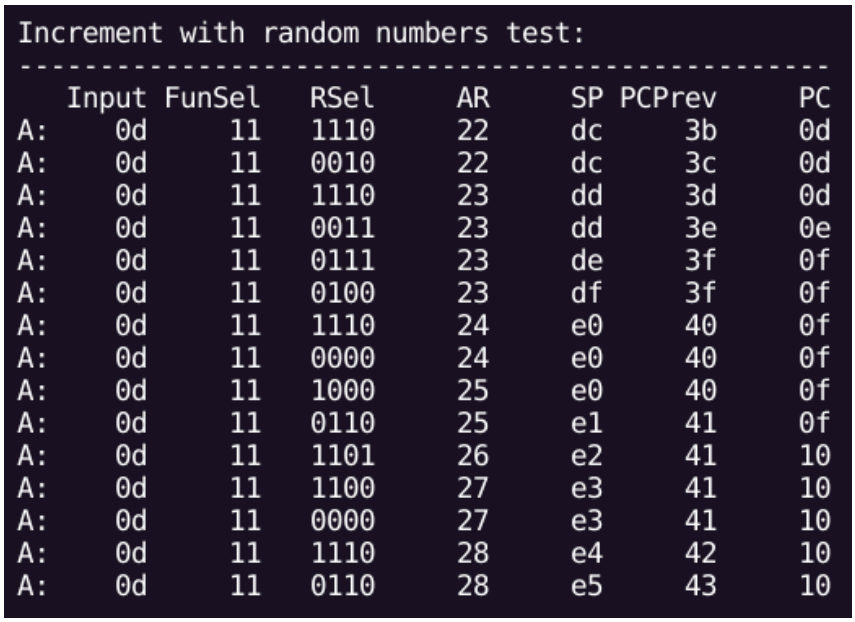
OutASel	OutA
00	AR
01	SP
10	PCPrev
11	PC

OutBSel	OutB
00	AR
01	SP
10	PCPrev
11	PC

Results of the adress register file's simulation have a complex and not easy to follow waveform, since all of the registers cannot be viewed at the same time and are shown one by one, by altering the OutASel and OutBSel. For this reason, only the testbench output generated with *$diplay* and *$write* functions is provided. For proper formatting and code readability, testbench has tasks such as *displayHead* (prints table header containing port names), *showRegistersThroughA* and *showRegistersThroughB* (show the contents of all registers through port A and port B), *displayLine* and *showAll* (calls all task described at once). Also, tests with different parameters can be easily regenerated, because **Input** and **RSel** come as random values.

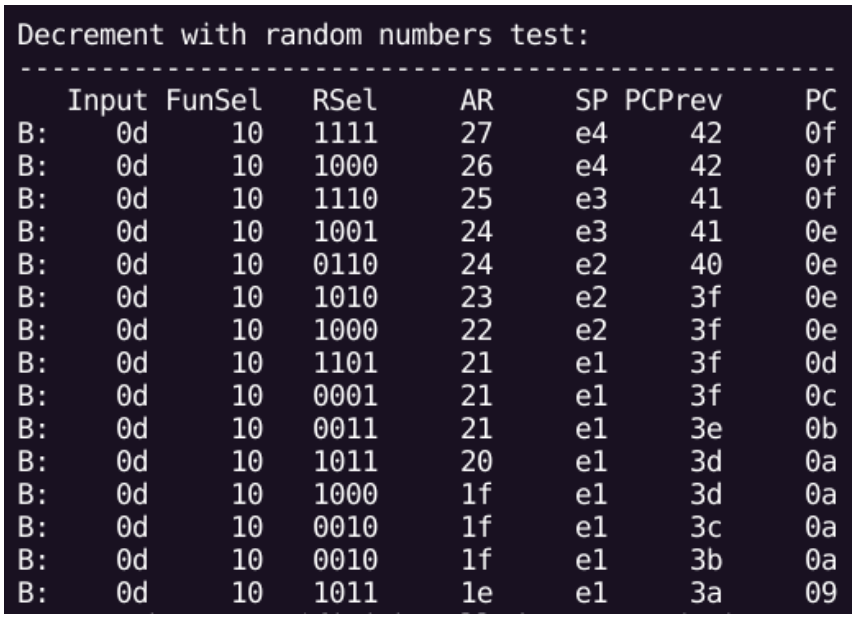
```
Testing adress register file (ARF)

Clear test
-------------------------------------------------
   Input FunSel   RSel     AR     SP PCPrev     PC
A:    xx     00   1111     00     00     00     00
B:    xx     00   0000     00     00     00     00

Load with random numbers test:
-------------------------------------------------
   Input FunSel   RSel     AR     SP PCPrev     PC
A:    21     01   1000     21     00     00     00
B:    21     01   0000     21     00     00     00
-------------------------------------------------
   Input FunSel   RSel     AR     SP PCPrev     PC
A:    db     01   0100     21     db     00     00
B:    db     01   0000     21     db     00     00
-------------------------------------------------
   Input FunSel   RSel     AR     SP PCPrev     PC
A:    3a     01   0010     21     db     3a     00
B:    3a     01   0000     21     db     3a     00
-------------------------------------------------
   Input FunSel   RSel     AR     SP PCPrev     PC
A:    0d     01   0001     21     db     3a     0d
B:    0d     01   0000     21     db     3a     0d
```

Figure 18: Clear and load test (part 2c)



```
Increment with random numbers test:
-----------------------------------------------
   Input FunSel   RSel     AR     SP PCPrev     PC
A:    0d     11   1110     22     dc     3b     0d
A:    0d     11   0010     22     dc     3c     0d
A:    0d     11   1110     23     dd     3d     0d
A:    0d     11   0011     23     dd     3e     0e
A:    0d     11   0111     23     de     3f     0f
A:    0d     11   0100     23     df     3f     0f
A:    0d     11   1110     24     e0     40     0f
A:    0d     11   0000     24     e0     40     0f
A:    0d     11   1000     25     e0     40     0f
A:    0d     11   0110     25     e1     41     0f
A:    0d     11   1101     26     e2     41     10
A:    0d     11   1100     27     e3     41     10
A:    0d     11   0000     27     e3     41     10
A:    0d     11   1110     28     e4     42     10
A:    0d     11   0110     28     e5     43     10
```

Figure 19: Increment test (part 2c)



```
Decrement with random numbers test:
-----------------------------------------------
   Input FunSel   RSel     AR     SP PCPrev     PC
B:    0d     10   1111     27     e4     42     0f
B:    0d     10   1000     26     e4     42     0f
B:    0d     10   1110     25     e3     41     0f
B:    0d     10   1001     24     e3     41     0e
B:    0d     10   0110     24     e2     40     0e
B:    0d     10   1010     23     e2     3f     0e
B:    0d     10   1000     22     e2     3f     0e
B:    0d     10   1101     21     e1     3f     0d
B:    0d     10   0001     21     e1     3f     0c
B:    0d     10   0011     21     e1     3e     0b
B:    0d     10   1011     20     e1     3d     0a
B:    0d     10   1000     1f     e1     3d     0a
B:    0d     10   0010     1f     e1     3c     0a
B:    0d     10   0010     1f     e1     3b     0a
B:    0d     10   1011     1e     e1     3a     09
```

Figure 20: Decrement test (part 2c)

2.3 Part 3

The Arithmetic Logic Unit (ALU) module has six ports. The table below introduces those ports.

Port Type	Port Name	Bit Width	Description
input	A	8	First operand of the ALU
input	B	8	Second operand of the ALU
input	FunSel	4	Function Selector
input	CLK	1	Clock input
output	OutALU	8	The resultant binary after an operation
output	ZCNO	4	Status flags of the ALU

Table 1: List of ports of the "alu" module

The module applies one of the sixteen operations defined according to selector input. The operations can be divided into three main sections. Arithmetic operations, logic operations, and shift operations.

1. **Arithmetic Operations**

Op Code	Function	Description
0100	A + B	Addition
0101	A - B	Subtraction
0110	Compare A, B	Return A if A > B, else 0

Table 2: List of arithmetic operations

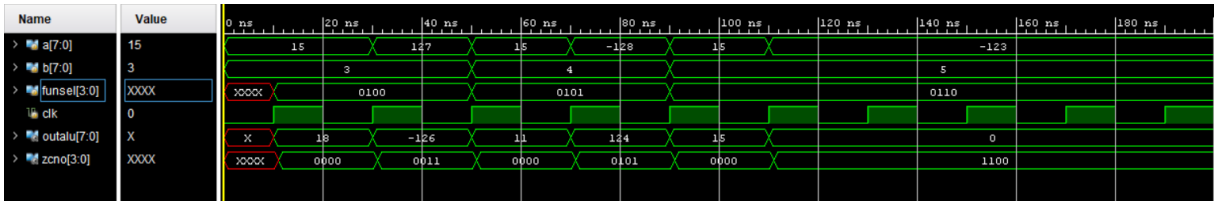


Figure 21: Simulation of arithmetic operations

Note that carry bit is added in addition operation. This allows 16-bit addition applicable. A subtraction operation is used to compare integers, flags are set according to that operation.

2. **Logic Operations**

Op Code	Function	Description
0000	A	Return A
0001	B	Return B
0010	$\overline{A}$	Complement of A
0011	$\overline{B}$	Complement of B
0111	$A \cdot B$	Bitwise AND
1000	$A + B$	Bitwise OR
1001	$\overline{(A \cdot B)}$	Bitwise NAND
1010	$A \oplus B$	Bitwise XOR

Table 3: List of logic operations

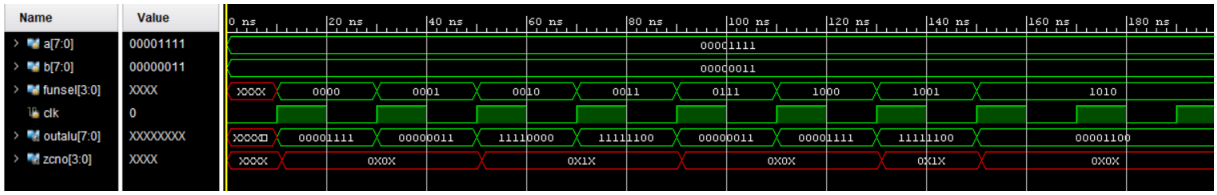


Figure 22: Simulation of logic operations

3. **Shift Operations**

Op Code	Function	Description
1011	LSL A	Logical left shift of the bits of A
1100	LSR B	Logical right shift of the bits of A
1101	ASL A	Arithmetic left shift
1110	ASL A	Arithmetic right shift
1111	CSR A	Circular right shift

Table 4: List of shift operations

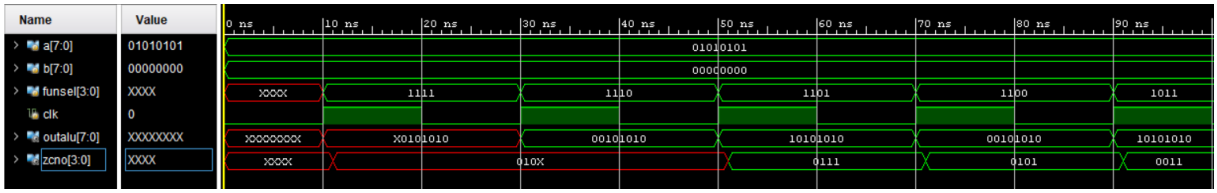


Figure 23: Simulation of shift operations

Note that the content of the carry bit will be visible in the output after a CSR operation.

- The difference between the response times of OutALU and ZCNO is due to the delay of the register module employed during the implementation of the flag register.

2.4 Part 4

This part is simply a gathered up version of previous parts we did. Inputs for this module are as follows:
Multiplexers: MuxASel, MuxBSel, MuxCSel are used as selectors for corresponding multiplexers.

MUXSelA	MUXOutA
00	OutALU
01	Memory Output
10	IROut(7-0)
11	ARF OutA

MUXSelB	MUXOutB
00	OutALU
01	Memory Output
10	IROut(7-0)
11	ARF OutA

MUXSelC	MUXOutC
0	RF_O1
1	ARF_OutA

Register File: RF_OutASel is for O1Sel, RF_OutBSel is for O2Sel, RF_FunSel is for FunSel, RF_TSel is for RSel and RF_RSel is for RSel.

Arithmetic Logic Unit: ALU_FunSel is for FunSel.

Address Register File: ARF_OutASel is for out_a_sel, ARF_OutBSel is for out_b_sel, ARF_FunSel is for funsel, ARF_RSel is for r_sel.

Instruction Register: IR_Funsel is for funsel, IR_Enable is for e, IR_LH is for l_h.

Memory: Mem_WR is for wr, Mem_CS is for cs.

Clock: Same clock signal is connected to every module.

The other wires inside the system are connected as in the project description.

The schematics of the implemented ALU system module is as below.

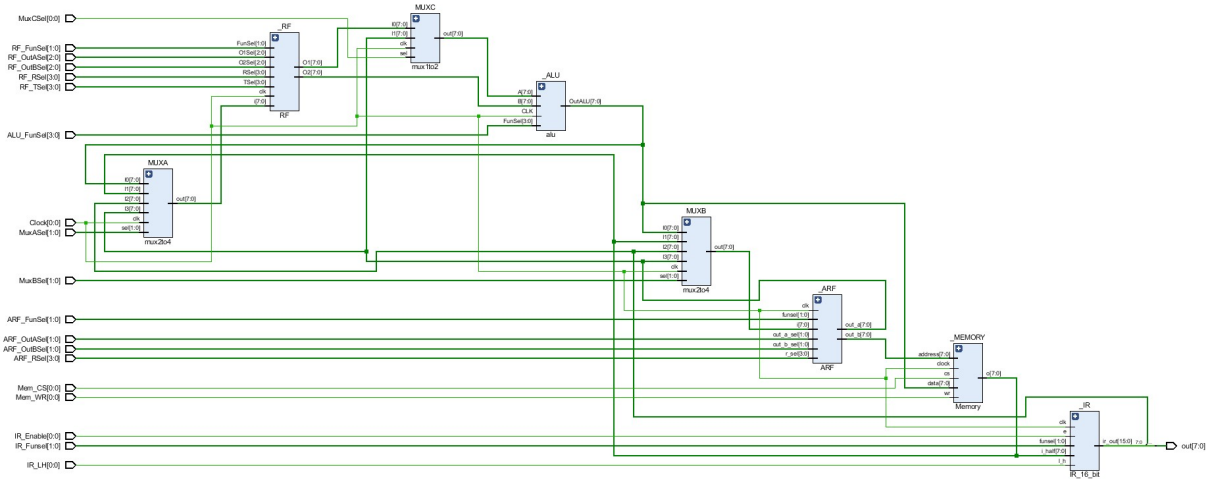


Figure 24: ALU System Schematics

With the given simulation file and the testbench.mem file, inputs and corresponding outputs of our design are as follows.

Input Values:
Operation: 0
Register File: O1Sel: 0, O2Sel: 0, FunSel: 0, RSel: 15, TSel: 15
ALU FunSel: 0
Addres Register File: OutASel: 0, OutBSel: 0, FunSel: 0, Regsel: 15
Instruction Register: LH: 1, Enable: 1, FunSel: 0
Memory: WR: 0, CS: 1
MuxASel: 3, MuxBSel: 3, MuxCSel: 1

Output Values:
Register File: AOut: x, BOut: x
ALUOut: x, ALUOutFlag: 0, ALUOutFlags: Z:0, C:0, N:0, O:0,
Address Register File: AOut: x, BOut (Address): x
Memory Out: z
Instruction Register: IROut: 0
MuxAOut: x, MuxBOut: x, MuxCOut: x

Input Values:
Operation: 1
Register File: O1Sel: 4, O2Sel: 5, FunSel: 1, RSel: 12, TSel: 0
ALU FunSel: 3
Addres Register File: OutASel: 3, OutBSel: 3, FunSel: 3, Regsel: 15
Instruction Register: LH: 0, Enable: 0, FunSel: 0
Memory: WR: 0, CS: 1
MuxASel: 3, MuxBSel: 1, MuxCSel: 1

Output Values:
Register File: AOut: 0, BOut: 0
ALUOut: x, ALUOutFlag: 0, ALUOutFlags: Z:0, C:0, N:0, O:0,
Address Register File: AOut: 0, BOut (Address): 0
Memory Out: z
Instruction Register: IROut: 0
MuxAOut: 0, MuxBOut: z, MuxCOut: 0

Input Values:
Operation: 1
Register File: O1Sel: 4, O2Sel: 6, FunSel: 1, RSel: 10, TSel: 0
ALU FunSel: 12
Addres Register File: OutASel: 3, OutBSel: 3, FunSel: 3, Regsel: 15
Instruction Register: LH: 0, Enable: 0, FunSel: 0
Memory: WR: 0, CS: 1
MuxASel: 3, MuxBSel: 3, MuxCSel: 1

Output Values:
Register File: AOut: 1, BOut: 1
ALUOut: 255, ALUOutFlag: 2, ALUOutFlags: Z:0, C:0, N:1, O:0,
Address Register File: AOut: 1, BOut (Address): 1
Memory Out: z
Instruction Register: IROut: 0
MuxAOut: 1, MuxBOut: 1, MuxCOut: 1
3 tests completed.

3 Conclusion

In conclusion, this project has helped us gain a deeper understanding of basic computer and how it works. By designing each part as a module, we were able to reuse them in other parts, which made the project more efficient. Simulating each module for each combination of input allowed us to test the functionality of each part and ensure that they worked correctly. Overall, this project has been an excellent learning experience that has helped us develop our skills in basic computer organization and Verilog.