Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

Advanced Digital Design - ENCS3310

Repot Project

First Semester 2023/2024

Prepared by: Heba Jamal - 1211315 .

Instructor: Dr. Abdallatif Abuissa.

Section: 2.

Date: 21-1-2024.

Abstract

The aim of this project is to design a simple part of microprocessor, the main blocks in this design are ALU and Register file, first The ALU is one of the most important components in a microprocessor, and is typically the part of the processor that is designed first, it performs arithmetic and logical operations ,second The register file is the component that contains all the general purpose registers of the microprocessor, A register file is typically implemented as an array of registers, where each register is a collection of storage elements. The register file has two major operations: read and write. During a read operation, the contents of a register are transferred to an output port. During a write operation, the contents of an input port are transferred to a register, finally connected them together and running it .

Table Of Content

Contents

Table Of Figure

Theory

ALU

An Arithmetic Logic Unit (ALU) is a combinational logic circuit that can perform different arithmetic and bitwise logical operations on integer binary numbers. ALU is the fundamental building block of many computing circuits including Central Processing Units (CPUs). An ALU usually takes two inputs, called operands, and a code, called OPCODE, which specifies the operation to be performed on the operands. ALU also has a result output which is the result of the operation on the operands. In some designs, usually the values (operands) fed to the ALU and/or the result generated from ALU are read from/stored in registers. A general symbolic representation of an ALU is shown in Figure 1.

Operations Performed by ALU

Although the ALU is a critical component of the CPU, the design and function of the ALU may vary amongst processors. Some ALUs, for example, are designed solely to conduct integer calculations, whereas others are built to perform floating-point computations. Some processors have a single arithmetic logic unit that performs operations, whereas others have many ALUs that conduct calculations. ALU's operations are as follows:

1. Arithmetic Operators**:** It refers to bit subtraction and addition, despite the fact that it does multiplication and division. Multiplication and division processes, on the other hand, are more expensive to do. Addition can be used in place of multiplication, while subtraction can be used in place of division.

2. Bit-Shifting Operators**:** It is responsible for a multiplication operation, which involves shifting the locations of a bit to the right or left by a particular number of places.

3. Logical Operations: These consist of NOR, AND, NOT, NAND, XOR, OR, and more.

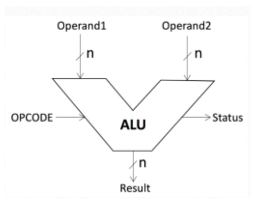


Figure 1 : ALU

Register File

A Register File is an integral part of a microprocessor. Registers are used to store operands for ALU operations, addresses for branch instructions and data in the form of intermediate values during a computation. The number of registers and the width of each register characterize a Register File. An (M x N) Register File has M registers, each of which is N bits wide. Typically, a Register File supports two operations, namely, read and write. A read operation involves placing the data value within a given register onto an output bus, while a write operation involves storing a given value into a specified register.

Properties of register file:

- m n-bit storage words Few words, many ports, dedicated read/write ports, so that read and write operations can be done simultaneously.

- Writing to a register takes just one clock cycle. Read op doesn’t require a clock or additional control inputs
- Logically Static Content
- Synchronous

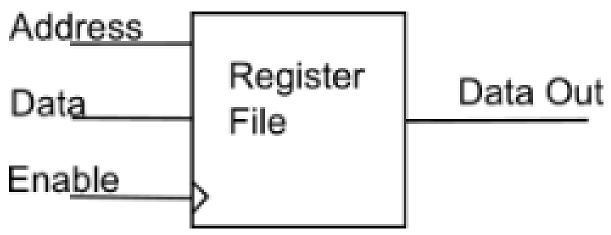


Figure 2 : Register File

Test-Bench

Verilog test benches are used for the verification of the digital hardware design. Verification is required to ensure the design meets the timing and functionality requirements.

Verilog Test benches are used to simulate and analyze designs without the need for any physical hardware or any hardware device. The most significant advantage of this is that you can inspect every signal /variable (reg, wire in Verilog) in the design. This certainly can be a time saver when you start writing the code or loading it onto the FPGA. In reality, only a few signals are taken out to the external pins. Though, you could not get this all for free. Firstly, you need to simulate your design; you must first write the design corresponding test benches. A test bench is, in fact, just another Verilog file, and this Verilog file or code you write as a testbench is not quite the same as the Verilog you write in your designs. Because Verilog design should be synthesizable according to your hardware meaning, but the Verilog testbench you write in a testbench does not need to be synthesizable because you will only simulate it , also a test module typically has no inputs or outputs.

Design and Result

Part One : ALU

To design the ALU, I will start by defining an opcode Enum that represents the different operations that the ALU can perform. I will then use this Enum to implement the arithmetic and logical operation using a case statement, I will define a module named (alu) that takes two 32-bit input signals(A and B), an 6 -bit opcode signal (opcode), and generates an 33-bit output signal (result), In result use 33 bit to store result because in addition of 32 bit with 32 bit the result will be over flow , then to solve this issue set the size to 33 bit to avoid this case. Inside the always block, I will use a case statement to match each operation with the corresponding output value.

Add Operation

As shown in figure (3) .When opcode is (3) that means the add operation is given , the ALU will take the two binary inputs (A and B) and add them together. The output of this operation is the sum of the two given numbers, as shown in the figure a = 0000_0001 , b = 0000_0002 , when performing the addition process the result = 0_0000_0003 , same as shown on waveform so this operation works correctly.

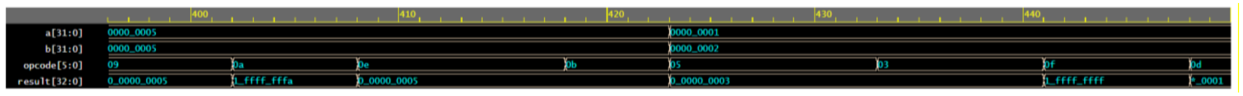


Figure 3 : Simulation snapshot contain add operation

Subtract Operation

As shown in the figure (4).When opcode is (15) that means the subtract operation is given, the ALU will take the two binary inputs (A and B)and subtract the first from the second. The output of this operation is the difference between the two given numbers, as shown in the figure

a = 0000_0003 ,b = 0000_0007, when performing the subtraction process

result = a + 2's complement of b → 0000_0000011 + 1111_1110001 = 1111_1111100 notes that when represent this number to binary will get (1_ffff_fffc) ,this result same as shown in wave form so this operation works correctly.

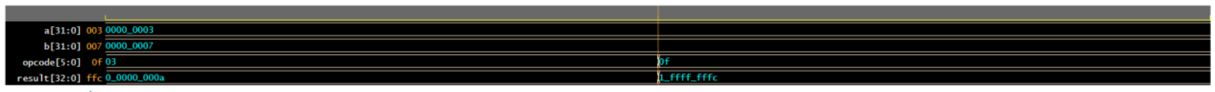


Figure 4 : Simulation snapshot contain subtract operation

Absolute Operation

As shown in the figure (5). When opcode is (13) that means the absolute value operation is given, the ALU will take the binary input (A) if this input is negative I will convert it to positive number by multiplying it by -1 else it remains as it is . The output always is positive number as shown in the figure   a = 0000_0003, take the absolute of this number, the result = 0_0000_0003, this result same as shown in waveform so this operation works correctly.

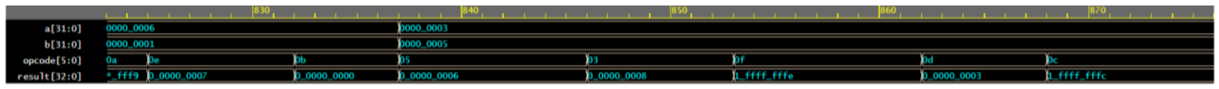


Figure 5 : Simulation snapshot contain absolute operation

Invertor Operation

As shown in the figure (6) .When opcode is (12) that means the inverter operation is given ,the ALU will take the binary input (A) then multiply it by -1.The output maybe positive or negative number depends on the value of input if the input is positive the result will be negative else if the input is negative the result is positive, as shown in the figure a = 0000_0003 , take 1’st complement for input the

result =, 1_ffff_fffc this result same as shown in waveform so this operation works correctly.

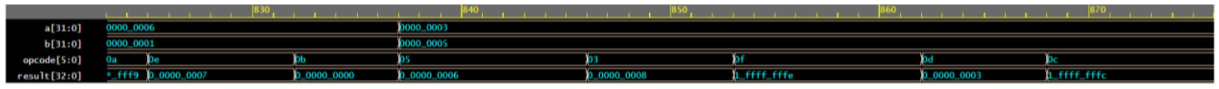


Figure 6 : Simulation snapshot contain inverter operation

Maximum Operation

As shown in the figure (7). When opcode is (7) that means the maximum operation is given ,the ALU will take the binary inputs (A and B) and return the larger of the two numbers .The output maybe a larger number as shown in the figure a = 0000_0003 ,b = 0000_0002 ,the larger of the two number is a so the result = 0_0000_0003 this result same as shown in the waveform so this operation works correctly ,so this operation works correctly.

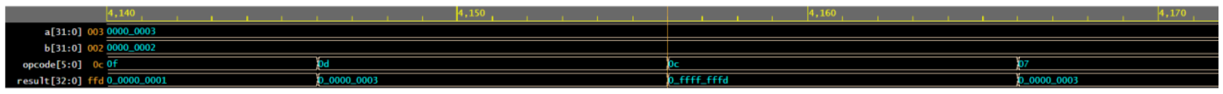


Figure 7 : Simulation snapshot contain maximum operation

Minimum Operation

As shown in the figure (8) . When opcode is (1) that means the minimum operation is given ,the ALU will take the binary inputs (A and B) and return the smaller of the two numbers .The output maybe a small number as shown in the figure a =0000_0004 ,b = 0000_0006 ,the smaller of the two number is a so the

result = 0_0000_0004 this results same as shown in the waveform, so this operation works correctly.

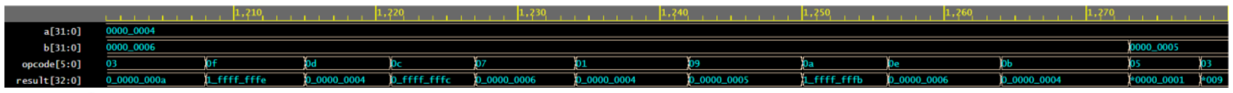

1,210 1,220 1,230 1,240 1,250 1,260 1,270
a[31:0] 0000_0004
b[31:0] 0000_0006 0000_0005
opcode[5:0] 03 0f 0d 0c 07 01 09 0a 0e 0b 05 03
result[32:0] 0_0000_000a 1_ffff_fffe 0_0000_0004 0_ffff_fffc 0_0000_0006 0_0000_0004 0_0000_0005 1_ffff_fffb 0_0000_0006 0_0000_0004 *0000_0001 *009

Figure 8 : Simulation snapshot contain minimum operation

Average Operation

As show in the figure (9) . When opcode is (9) that means the average operation is given, the ALU will take the binary inputs (A and B), add the two numbers then dividing the by two   .The output is average of two numbers as shown in the figure a = 0000_0004 ,b = 0000_0006 the average of two number

result = 0_0000_0005 this results same as shown in the waveform, so this operation works correctly.

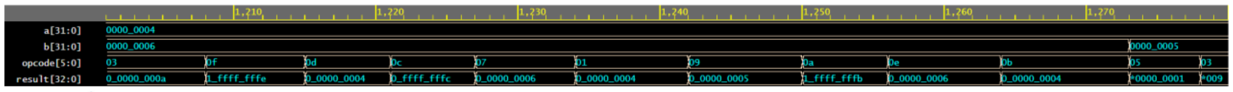

1,210 1,220 1,230 1,240 1,250 1,260 1,270
a[31:0] 0000_0004
b[31:0] 0000_0006 0000_0005
opcode[5:0] 03 0f 0d 0c 07 01 09 0a 0e 0b 05 03
result[32:0] 0_0000_000a 1_ffff_fffe 0_0000_0004 0_ffff_fffc 0_0000_0006 0_0000_0004 0_0000_0005 1_ffff_fffb 0_0000_0006 0_0000_0004 *0000_0001 *009

Figure 9 : Simulation snapshot contain average operation

NOT Gate Operation

As show in the figure (10). When opcode is (10) that means the NOT gate operation is given, the ALU will take the one binary input A, not gate works by converting 0's to 1's and 1's converts to 0's   .The output is inverter of give number as shown in the figure a = 0000_0005 by taken complement for this input the result =1_ffff_fffa same as shown in the waveform, so this operation works correctly.

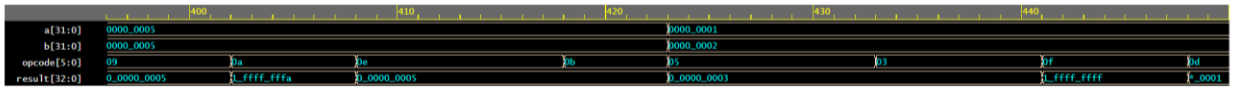

400 410 420 430 440
a[31:0] 0000_0005 0000_0001
b[31:0] 0000_0005 0000_0002
opcode[5:0] 09 0a 0e 0b 05 03 0f 0d
result[32:0] 0_0000_0005 1_ffff_fffa 0_0000_0005 0_0000_0003 1_ffff_ffff *_0001

Figure 10 : Simulation snapshot contain NOT gate operation

OR Gate Operation

As shown in the figure (11) . When opcode is (14) that means the OR gate operation is given ,the ALU will take the binary inputs (A and B) .The output is 1's if one or both inputs are 1,else the output is 0's if two input are 0   as shown in the figure a = 0000_0005 ,b = 0000_0005 using truth table for OR gate will be obtained the result = 0_0000_0005, as same as shown in the wave form ,so this operation works correctly.

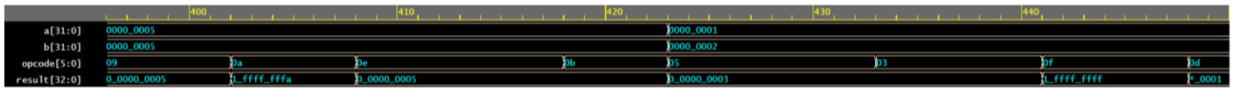


Figure 11 : Simulation snapshot contain OR gate operation

AND Gate Operation

As shown in the figure (12) .When opcode is (11) that means the AND gate operation is given ,the ALU will take the binary inputs (A and B) .The output is 1's if both inputs are 1,else the output is 0's if one or both inputs are 0   as shown in the figure a =0000_0002 ,b = 0000_0002 by using truth table for AND gate will be obtained the result = 0_0000_00002 as same as shown in the waveform, so this operation works correctly.

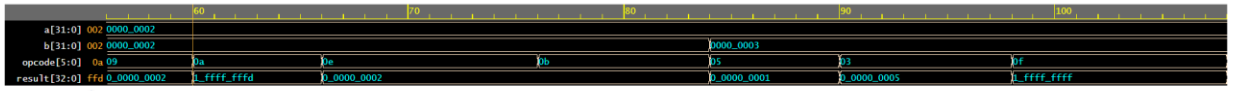


Figure 12 : Simulation snapshot contain AND operation

XOR Gate Operation

As shown in the figure (13). When opcode is (5) that means the XOR gate instruction is given, the ALU will take the binary inputs (A and B). The output is 1's if one and only one of the inputs is 1,else the output is 0's if both inputs are 0 or both inputs are 1 as shown in the figure a = 0000_0001,b = 0000_0002 by truth table for XOR gate will be obtained the result = 0_0000_0003,so this operation works correctly.

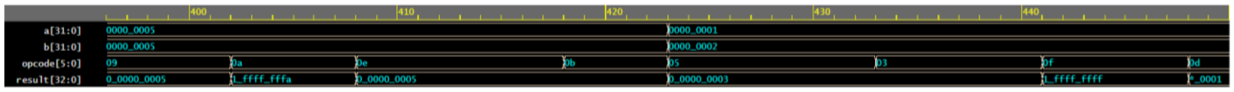


Figure 13 : Simulation snapshot contain XOR operation

Part Two : Register File

Register File is a type of digital memory component that allows data to be read from and written to a single memory location (address) at a time. During a read operation, the data stored in specific address is retrieved. During a write operation, new data is stored at a specific address, replacing the previous data.

In this Register File there is two port for read and one port for write, memory unit stores the initial values, select this by second from last digit of students id. Data input (initial value) lines provide the information to be stored into the memory, Data output lines carry the information out from the memory. The control lines Read and write specifies the direction of transfer of data. there is 32 location permission 5 bits for each address.

Output 1 produces the item within the register file that is address by Address 1. Similarly Output 2 produces the item within the register file that is address by Address 2. In input signal is used to supply a value that is written into the location addressed by Address 3, also this register file works based on value of positive edge of valid opcode ,that means if there is a positive edge of valid opcode the register file will work, otherwise the register file will store previous value of address.

As shown in figure below (14). The address one and address two using for reading data and store it in output one and output two respectively, different from the previous the register file get the value input and put it in location of address 3

For example:

The value in address 1 is (0e), it's read this value and stored it in output1 (0002_008) in hexadecimal.

To make sure this is correct, referring to the initial values stored in register file, in location (14) the value of register file is (0002_008) in hexadecimal so this part is correct.

The value in address 2 is (07), it's read this value and stored it in output2 (0000_172c).

Also, to make sure of this part is correct, like what was done before part referring to the initial

Values stored in register file, in location (7) the value of register file is (172c) in hexadecimal, this part is also true.

The register file gets the value input (0000_0014) and put it in location of address 3 (16).

From all results will obtained previously,which confirm the validity of the design register file.

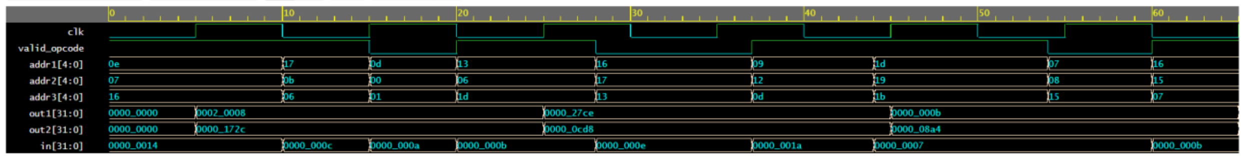


Figure 14 : Simulation snapshot Register File

Part Three :Machine Instructions

Machine instructions are supplied to this arrangement in the form of 32-bit numbers. they were divided as follows :first 6 bits (from bit zero to bit 5) means the value of opcode , the next 5 bits(from bit 6 to bit 10) means the value of first register, the next 5 bits (from bit 11 to bit 15) means the value of second register, the next 5 bits (from bit 16 to bit 20) means the value of third register.

The previous idea will be clarified in the following example if it is take the instruction (c09f_f677) convert it to binary number I get (11000000100111111111011001110111).

→ The first six bits (11000000100111111111011001110111), identify the opcode ,convert it to decimal so the result is (37),as shown in the figure the result also (37) so this part is true.

→ The next five bits (11000000100111111111011001110111) , identify the address one ,convert it to decimal get the result is (19),as shown in the figure the result also (19) so this part is true.

→ The next five bits (11000000100111111111011001110111), identify the address two. convert it to decimal get the result is (1e), as shown in the figure the result also (1e) so this part is true.

→ The next five bits (11000000100111111111011001110111), identify the address three. convert it to decimal get the result is (1f), as shown in the figure the result also (1f) so this part is true.

All the result above same as the result shown in the waveform so this machine construction works correctly.

Notes that the final 11 bits (11000000100111111111011001110111) are unused.

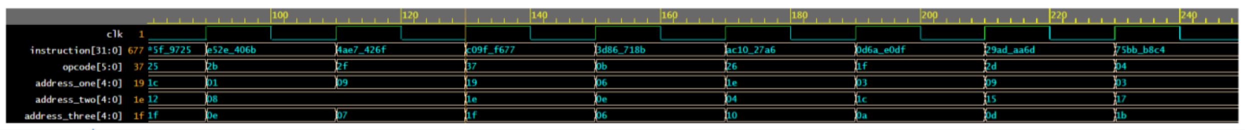

100 120 140 160 180 200 220 240
clk 1
instruction[31:0] 677 *5f_9725 e52e_406b 4ae7_426f c09f_f677 3d86_718b ac10_27a6 0d6a_e0df 29ad_aa6d 75bb_b8c4
opcode[5:0] 37 25 2b 2f 37 0b 26 1f 2d 04
address_one[4:0] 19 1c 01 09 19 06 1e 03 09 03
address_two[4:0] 1e 12 08 1e 0e 04 1c 15 17
address_three[4:0] 1f 1f 0e 07 1f 06 10 0a 0d 1b

Figure 15 : Simulation snapshot machine instruction

Creating the core of the microprocessor

In this part will connect the ALU and Register File to form a simple microprocessor.

For ensure this part is work correctly at first it will take the instruction at positive edge of clock and valid opcode is one for as shown on the figure below (15) the instruction 000c_db8f convert it to binary number to select the value of opcode and address for one ,two, and three ,the instruction represent

(00000000000011001101101110001111) .

→ The first six bits (00000000000011001101101110001111), identify the opcode ,convert it to decimal so the result is (0f),as shown in the figure the result also (0f) so this part is true.

→ The next five bits (00000000000011001101101110001111) , identify the address one ,convert it to decimal get the result is (0e),as shown in the figure the result also (0e) so this part is true.

→ The next five bits (00000000000011001101101110001111) , identify the address two. convert it to decimal get the result is (1b), as shown in the figure the result also (1b) so this part is true.

→ The next five bits (00000000000011001101101110001111) , identify the address three. convert it to decimal get the result is (0c), as shown in the figure the result also (0c) so this part is true.

After take the value of address , then will be make sure if output one produces the item within the register file that is address by address one. The address one is (0e) in Hexadecimal it must be converted it to decimal referring to location 15 in memory the value is 0_0002_0008 as same as shown in the waveform, so the core of microprocessor is work correctly.

If it was repeated all previous operation for other opcode will be get the true result.

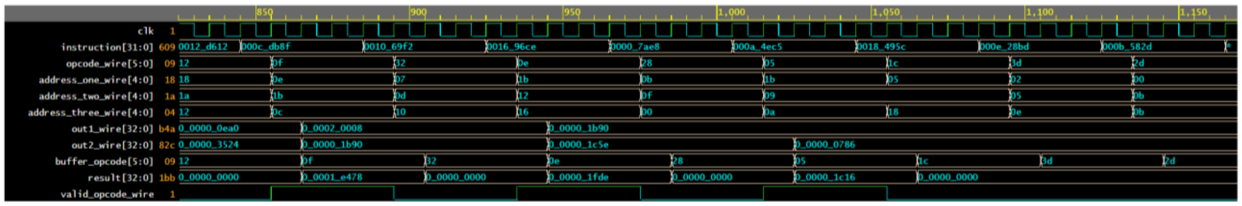


Figure 16 : core of the microprocessor

Test _Bench For Design

→ mp _top is a module instantiated with the name DUT (Design Under Test). ports are connected in a certain order which is determined by the position of that port in the port list of the module declaration, A better way to connect ports is by explicitly linking ports on both the sides using their port name. The dot (.) indicates that the port name following the dot belongs to the design. The signal to which the design port has to be connected is given next within parentheses ().

→ Initialize values to input value like instruction and clock, and it can be initializing this value inside an initial block.

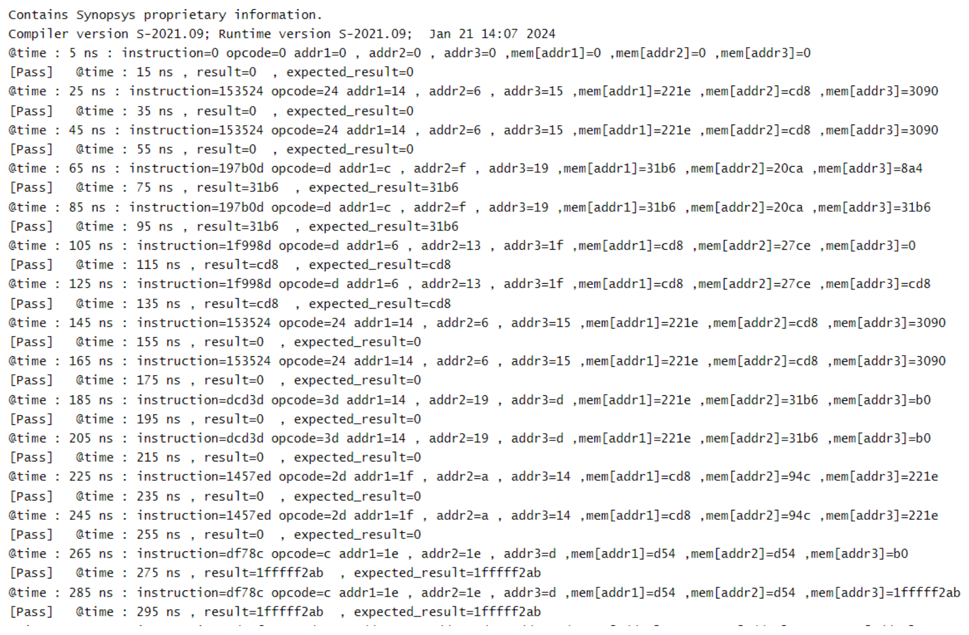
→ Print Report if the test pass or fail when compare result and expected result.

The input instruction and result (output)

System verifying :

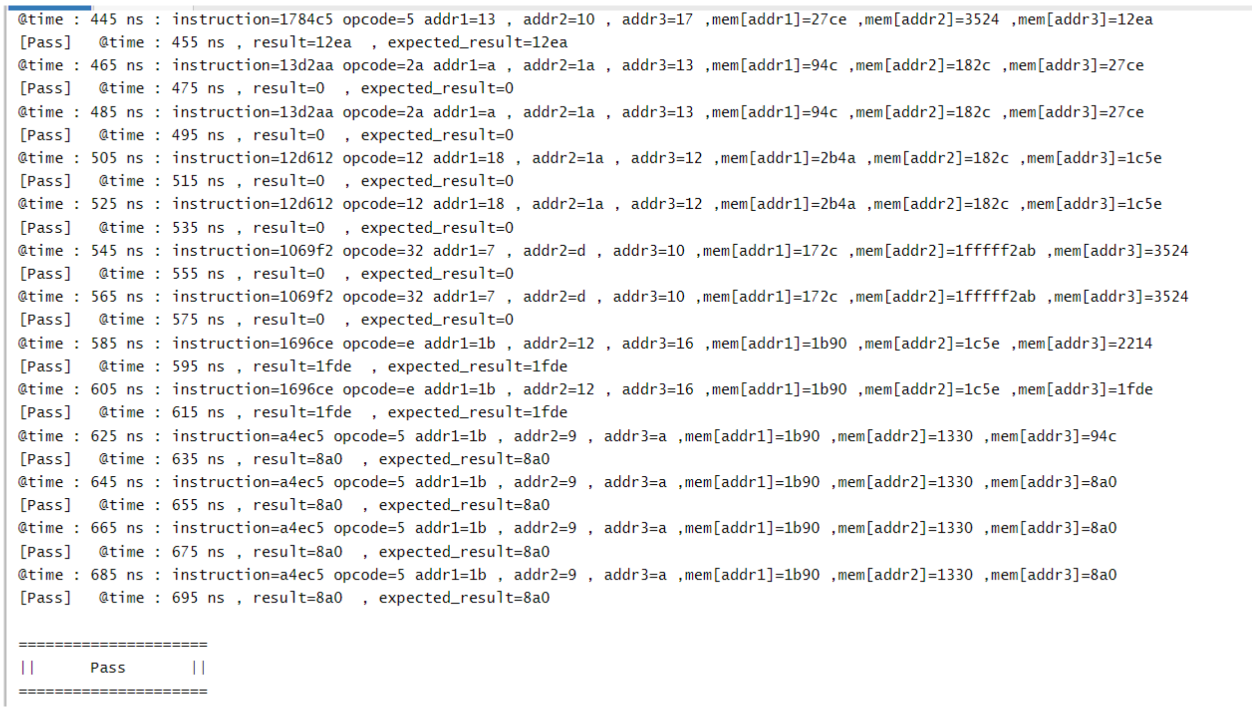
Aiming to verify the system that is working perfectly in right way as in Figure , generates a lot of cases of the instruction and send to the design as inputs, Also from this input generated expected result , then compare the actual result (output from design) with expected result, if there any mis-match between the expected results and the actual result, we put Fail message to assert to user there is problem , on other case generate pass .

As shown in the below figures , The log or console that show the time ,instruction , opcode value , address one , address two , address three , value of location of memory address one , value of location of memory address two and value of location of memory address three , Also I when comparing the result with expected result print pass if the value equal otherwise print fall . Also in the end of the test print test report status pass or fall depend on all comparison between generated and expected result.



```
Contains Synopsys proprietary information.
Compiler version S-2021.09; Runtime version S-2021.09;  Jan 21 14:07 2024
@time : 5 ns : instruction=0 opcode=0 addr1=0 , addr2=0 , addr3=0 ,mem[addr1]=0 ,mem[addr2]=0 ,mem[addr3]=0
[Pass]   @time : 15 ns , result=0  , expected_result=0
@time : 25 ns : instruction=153524 opcode=24 addr1=14 , addr2=6 , addr3=15 ,mem[addr1]=221e ,mem[addr2]=cd8 ,mem[addr3]=3090
[Pass]   @time : 35 ns , result=0  , expected_result=0
@time : 45 ns : instruction=153524 opcode=24 addr1=14 , addr2=6 , addr3=15 ,mem[addr1]=221e ,mem[addr2]=cd8 ,mem[addr3]=3090
[Pass]   @time : 55 ns , result=0  , expected_result=0
@time : 65 ns : instruction=197b0d opcode=d addr1=c , addr2=f , addr3=19 ,mem[addr1]=31b6 ,mem[addr2]=20ca ,mem[addr3]=8a4
[Pass]   @time : 75 ns , result=31b6  , expected_result=31b6
@time : 85 ns : instruction=197b0d opcode=d addr1=c , addr2=f , addr3=19 ,mem[addr1]=31b6 ,mem[addr2]=20ca ,mem[addr3]=31b6
[Pass]   @time : 95 ns , result=31b6  , expected_result=31b6
@time : 105 ns : instruction=1f998d opcode=d addr1=6 , addr2=13 , addr3=1f ,mem[addr1]=cd8 ,mem[addr2]=27ce ,mem[addr3]=0
[Pass]   @time : 115 ns , result=cd8  , expected_result=cd8
@time : 125 ns : instruction=1f998d opcode=d addr1=6 , addr2=13 , addr3=1f ,mem[addr1]=cd8 ,mem[addr2]=27ce ,mem[addr3]=cd8
[Pass]   @time : 135 ns , result=cd8  , expected_result=cd8
@time : 145 ns : instruction=153524 opcode=24 addr1=14 , addr2=6 , addr3=15 ,mem[addr1]=221e ,mem[addr2]=cd8 ,mem[addr3]=3090
[Pass]   @time : 155 ns , result=0  , expected_result=0
@time : 165 ns : instruction=153524 opcode=24 addr1=14 , addr2=6 , addr3=15 ,mem[addr1]=221e ,mem[addr2]=cd8 ,mem[addr3]=3090
[Pass]   @time : 175 ns , result=0  , expected_result=0
@time : 185 ns : instruction=dcd3d opcode=3d addr1=14 , addr2=19 , addr3=d ,mem[addr1]=221e ,mem[addr2]=31b6 ,mem[addr3]=b0
[Pass]   @time : 195 ns , result=0  , expected_result=0
@time : 205 ns : instruction=dcd3d opcode=3d addr1=14 , addr2=19 , addr3=d ,mem[addr1]=221e ,mem[addr2]=31b6 ,mem[addr3]=b0
[Pass]   @time : 215 ns , result=0  , expected_result=0
@time : 225 ns : instruction=1457ed opcode=2d addr1=1f , addr2=a , addr3=14 ,mem[addr1]=cd8 ,mem[addr2]=94c ,mem[addr3]=221e
[Pass]   @time : 235 ns , result=0  , expected_result=0
@time : 245 ns : instruction=1457ed opcode=2d addr1=1f , addr2=a , addr3=14 ,mem[addr1]=cd8 ,mem[addr2]=94c ,mem[addr3]=221e
[Pass]   @time : 255 ns , result=0  , expected_result=0
@time : 265 ns : instruction=df78c opcode=c addr1=1e , addr2=1e , addr3=d ,mem[addr1]=d54 ,mem[addr2]=d54 ,mem[addr3]=b0
[Pass]   @time : 275 ns , result=1fffff2ab  , expected_result=1fffff2ab
@time : 285 ns : instruction=df78c opcode=c addr1=1e , addr2=1e , addr3=d ,mem[addr1]=d54 ,mem[addr2]=d54 ,mem[addr3]=1fffff2ab
[Pass]   @time : 295 ns , result=1fffff2ab  , expected_result=1fffff2ab
```

Figure 17 : Snapshot (1) for the input instruction and result (output)

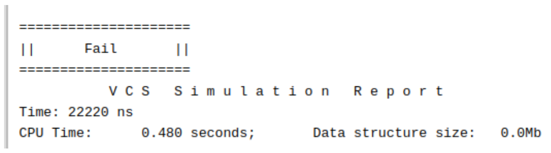


```
@time : 445 ns : instruction=1784c5 opcode=5 addr1=13 , addr2=10 , addr3=17 ,mem[addr1]=27ce ,mem[addr2]=3524 ,mem[addr3]=12ea
[Pass]   @time : 455 ns , result=12ea  , expected_result=12ea
@time : 465 ns : instruction=13d2aa opcode=2a addr1=a , addr2=1a , addr3=13 ,mem[addr1]=94c ,mem[addr2]=182c ,mem[addr3]=27ce
[Pass]   @time : 475 ns , result=0  , expected_result=0
@time : 485 ns : instruction=13d2aa opcode=2a addr1=a , addr2=1a , addr3=13 ,mem[addr1]=94c ,mem[addr2]=182c ,mem[addr3]=27ce
[Pass]   @time : 495 ns , result=0  , expected_result=0
@time : 505 ns : instruction=12d612 opcode=12 addr1=18 , addr2=1a , addr3=12 ,mem[addr1]=2b4a ,mem[addr2]=182c ,mem[addr3]=1c5e
[Pass]   @time : 515 ns , result=0  , expected_result=0
@time : 525 ns : instruction=12d612 opcode=12 addr1=18 , addr2=1a , addr3=12 ,mem[addr1]=2b4a ,mem[addr2]=182c ,mem[addr3]=1c5e
[Pass]   @time : 535 ns , result=0  , expected_result=0
@time : 545 ns : instruction=1069f2 opcode=32 addr1=7 , addr2=d , addr3=10 ,mem[addr1]=172c ,mem[addr2]=1fffff2ab ,mem[addr3]=3524
[Pass]   @time : 555 ns , result=0  , expected_result=0
@time : 565 ns : instruction=1069f2 opcode=32 addr1=7 , addr2=d , addr3=10 ,mem[addr1]=172c ,mem[addr2]=1fffff2ab ,mem[addr3]=3524
[Pass]   @time : 575 ns , result=0  , expected_result=0
@time : 585 ns : instruction=1696ce opcode=e addr1=1b , addr2=12 , addr3=16 ,mem[addr1]=1b90 ,mem[addr2]=1c5e ,mem[addr3]=2214
[Pass]   @time : 595 ns , result=1fde  , expected_result=1fde
@time : 605 ns : instruction=1696ce opcode=e addr1=1b , addr2=12 , addr3=16 ,mem[addr1]=1b90 ,mem[addr2]=1c5e ,mem[addr3]=1fde
[Pass]   @time : 615 ns , result=1fde  , expected_result=1fde
@time : 625 ns : instruction=a4ec5 opcode=5 addr1=1b , addr2=9 , addr3=a ,mem[addr1]=1b90 ,mem[addr2]=1330 ,mem[addr3]=94c
[Pass]   @time : 635 ns , result=8a0  , expected_result=8a0
@time : 645 ns : instruction=a4ec5 opcode=5 addr1=1b , addr2=9 , addr3=a ,mem[addr1]=1b90 ,mem[addr2]=1330 ,mem[addr3]=8a0
[Pass]   @time : 655 ns , result=8a0  , expected_result=8a0
@time : 665 ns : instruction=a4ec5 opcode=5 addr1=1b , addr2=9 , addr3=a ,mem[addr1]=1b90 ,mem[addr2]=1330 ,mem[addr3]=8a0
[Pass]   @time : 675 ns , result=8a0  , expected_result=8a0
@time : 685 ns : instruction=a4ec5 opcode=5 addr1=1b , addr2=9 , addr3=a ,mem[addr1]=1b90 ,mem[addr2]=1330 ,mem[addr3]=8a0
[Pass]   @time : 695 ns , result=8a0  , expected_result=8a0

=====================
||      Pass       ||
=====================
```

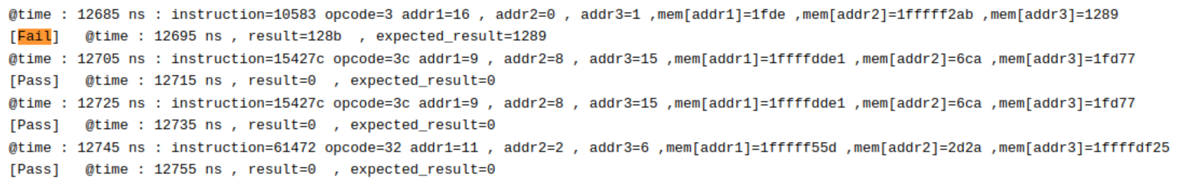
Figure 18 : Snapshot (2) for the input instruction and result(output)

I make some errors intentionally to discover that there is an error see . As show the opcode 3 means the current operation is add , and when add the a and b the correct answer as expected will be 1289 but the actual is 128b.



```
=====================
||      Fail       ||
=====================
           V C S   S i m u l a t i o n   R e p o r t
Time: 22220 ns
CPU Time:      0.480 seconds;       Data structure size:   0.0Mb
```

Figure 19 : Snapshot (3) for the input instruction fail



```
@time : 12685 ns : instruction=10583 opcode=3 addr1=16 , addr2=0 , addr3=1 ,mem[addr1]=1fde ,mem[addr2]=1fffff2ab ,mem[addr3]=1289
[Fail]   @time : 12695 ns , result=128b  , expected_result=1289
@time : 12705 ns : instruction=15427c opcode=3c addr1=9 , addr2=8 , addr3=15 ,mem[addr1]=1ffffdde1 ,mem[addr2]=6ca ,mem[addr3]=1fd77
[Pass]   @time : 12715 ns , result=0  , expected_result=0
@time : 12725 ns : instruction=15427c opcode=3c addr1=9 , addr2=8 , addr3=15 ,mem[addr1]=1ffffdde1 ,mem[addr2]=6ca ,mem[addr3]=1fd77
[Pass]   @time : 12735 ns , result=0  , expected_result=0
@time : 12745 ns : instruction=61472 opcode=32 addr1=11 , addr2=2 , addr3=6 ,mem[addr1]=1fffff55d ,mem[addr2]=2d2a ,mem[addr3]=1ffffdf25
[Pass]   @time : 12755 ns , result=0  , expected_result=0
```

Figure 20 : snapshot (4) for the input instruction fail

Conclusion

In this project learn a lot of concept about microprocessor , how to create simple one that contains two main block the ALU and register file, also how to build ALU that perform basic arithmetic like addition and subtraction and logic operation like and ,or, xor and etc .Also learn how to build project and how to use the simulator tool (tool that is used for using Verilog language), and how to write a full synchronous with asynchronous component.
Another things learn how to test the design and how to create expected logic to compare it with the design value. The system was verified perfectly as mention in above section with no error.

About Suggestions, increase the width of the result to make it 33 bit to catch overflow case. Also I prefer to put read or write signal in register file because to give the user more control on the design and can use 1 bit of unused bit in instruction to add it. Also I suggest to put load singal in register file , to give user more control to put the initial value (depend on ID) or to make it all zero. Last suggestion I prefer to add opcode to subtract b from a .

Finally, Verilog language is a full-power and great Hardware language to design , also it’s an easy language like C and it’s contains a lot of feature that discover it when dive during work on this project .

Reference

[1] : https://byjus.com/gate/alu-notes/

Accessed on Monday 15,2024.

[2] : https://en.wikibooks.org/wiki/Microprocessor_Design/Register_File

Accessed on Tuesday 16,2024.

[3] : https://www.chipverify.com/verilog/verilog-testbench

Accessed on Thursday 18,2024.

Appendix A

ALU

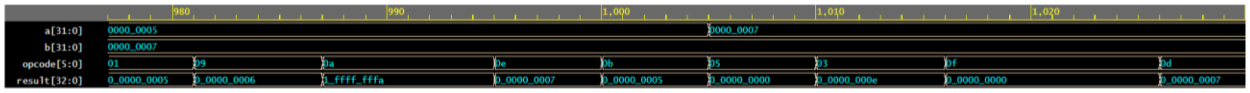

980 990 1,000 1,010 1,020
a[31:0] 0000_0005 0000_0007
b[31:0] 0000_0007
opcode[5:0] 01 09 0a 0e 0b 05 03 0f 0d
result[32:0] 0_0000_0005 0_0000_0006 1_ffff_fffa 0_0000_0007 0_0000_0005 0_0000_0000 0_0000_000e 0_0000_0000 0_0000_0007

Figure 21 : Simulation snapshot ALU (1)

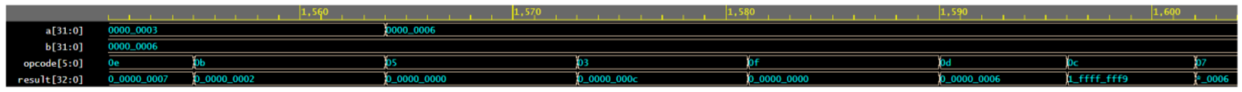

1,560 1,570 1,580 1,590 1,600
a[31:0] 0000_0003 0000_0006
b[31:0] 0000_0006
opcode[5:0] 0e 0b 05 03 0f 0d 0c 07
result[32:0] 0_0000_0007 0_0000_0002 0_0000_0000 0_0000_000c 0_0000_0000 0_0000_0006 1_ffff_fff9 *_0006

Figure 22 : Simulation snapshot ALU (2)

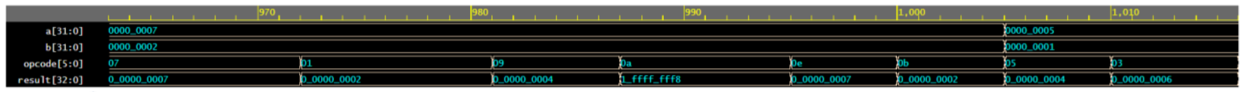

970 980 990 1,000 1,010
a[31:0] 0000_0007 0000_0005
b[31:0] 0000_0002 0000_0001
opcode[5:0] 07 01 09 0a 0e 0b 05 03
result[32:0] 0_0000_0007 0_0000_0002 0_0000_0004 1_ffff_fff8 0_0000_0007 0_0000_0002 0_0000_0004 0_0000_0006

Figure 23 : Simulation snapshot ALU (3)

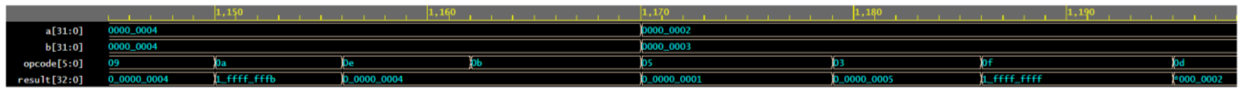

1,150 1,160 1,170 1,180 1,190
a[31:0] 0000_0004 0000_0002
b[31:0] 0000_0004 0000_0003
opcode[5:0] 09 0a 0e 0b 05 03 0f 0d
result[32:0] 0_0000_0004 1_ffff_fffb 0_0000_0004 0_0000_0001 0_0000_0005 1_ffff_ffff *000_0002

Figure 24 : Simulation snapshot ALU (4)

Register File

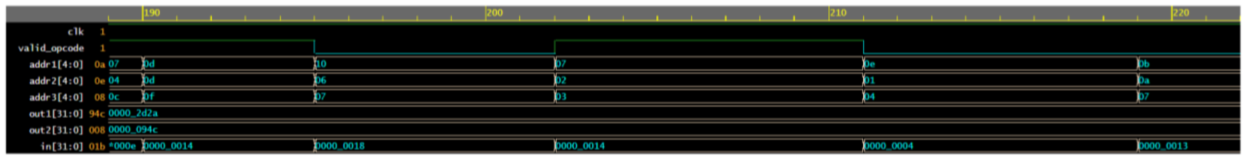

190 200 210 220
clk 1
valid_opcode 1
addr1[4:0] 0a 07 0d 10 07 0e 0b
addr2[4:0] 0e 04 0d 06 02 01 0a
addr3[4:0] 08 0c 0f 07 03 04 07
out1[31:0] 94c 0000_2d2a
out2[31:0] 008 0000_094c
in[31:0] 01b *000e 0000_0014 0000_0018 0000_0014 0000_0004 0000_0013

Figure 25 : Simulation snapshot Register File (1)

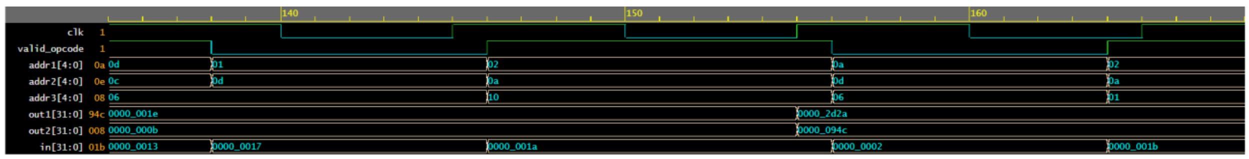

140 150 160
clk 1
valid_opcode 1
addr1[4:0] 0a 0d 01 02 0a 02
addr2[4:0] 0e 0c 0d 0a 0d 0a
addr3[4:0] 08 06 10 06 01
out1[31:0] 94c 0000_001e 0000_2d2a
out2[31:0] 008 0000_000b 0000_094c
in[31:0] 01b 0000_0013 0000_0017 0000_001a 0000_0002 0000_001b

Figure 26 : Simulation snapshot Register File (2)

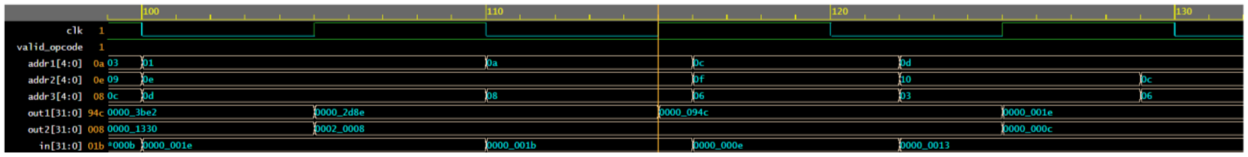


Figure 27 : Figure 23 : Simulation snapshot Register File (3)

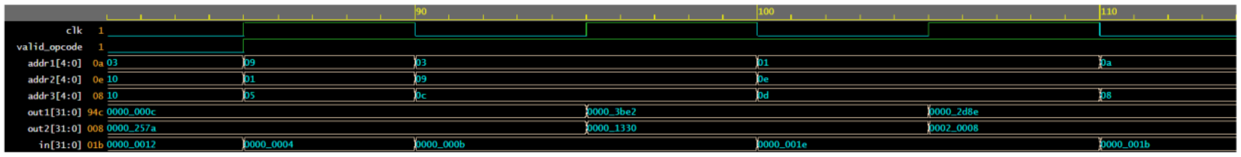


Figure 28 : Figure 23 : Simulation snapshot Register File (4)

The core of the microprocessor

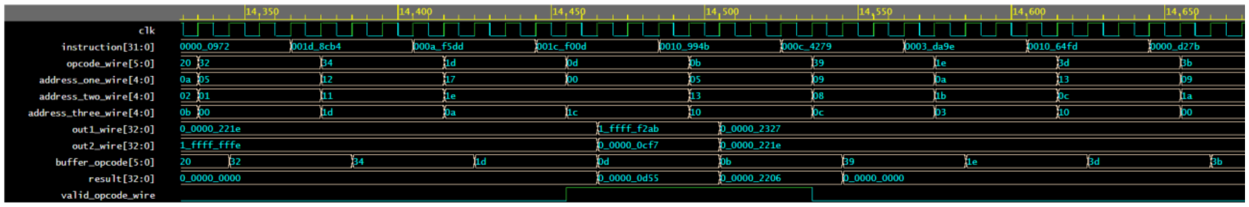


Figure 29 : Simulation snapshot The core of the microprocessor (1)

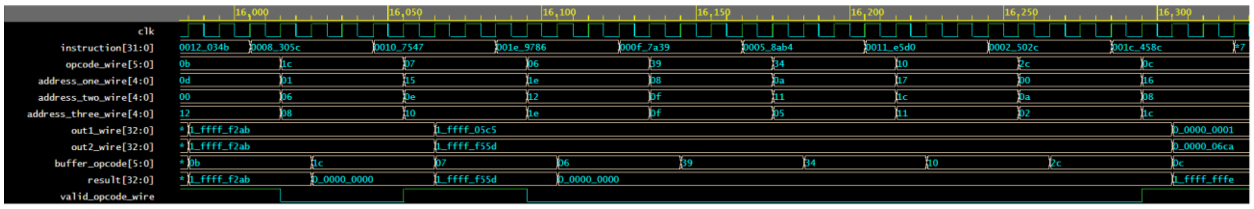


Figure 30 : Simulation snapshot The core of the microprocessor (2)

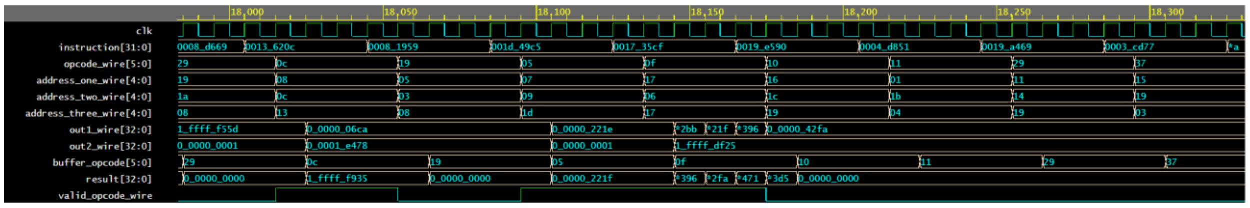


Figure 31 : Simulation snapshot The core of the microprocessor (3)

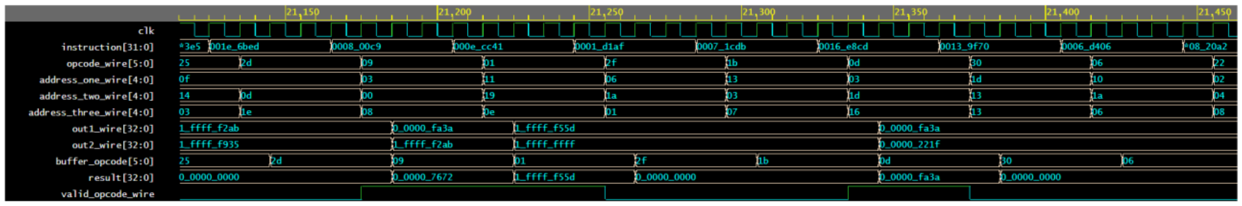


Figure 32 : Simulation snapshot The core of the microprocessor (4)

Machine Instructions

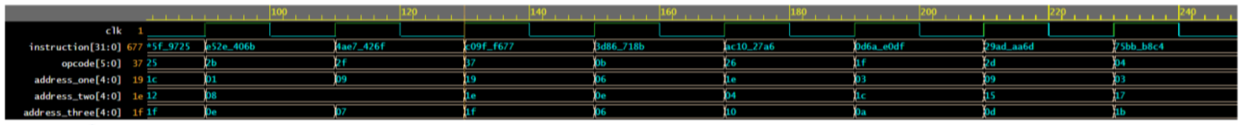


Figure 33 : Simulation snapshot machine instruction(1)

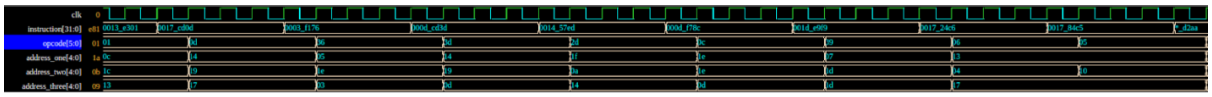


Figure 34 : Simulation snapshot machine instruction(2)

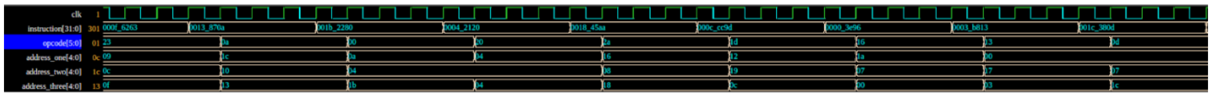


Figure 35 : Simulation snapshot machine instruction(3)

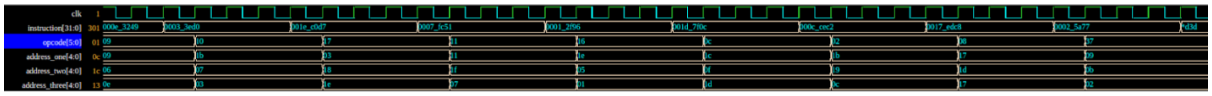
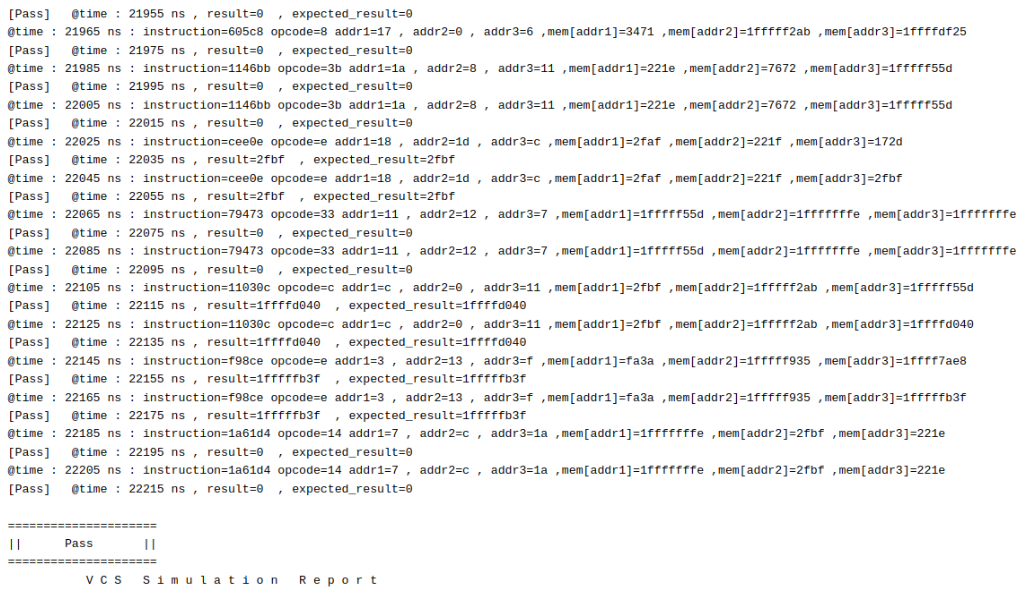


Figure 36 : Simulation snapshot machine instruction(4)

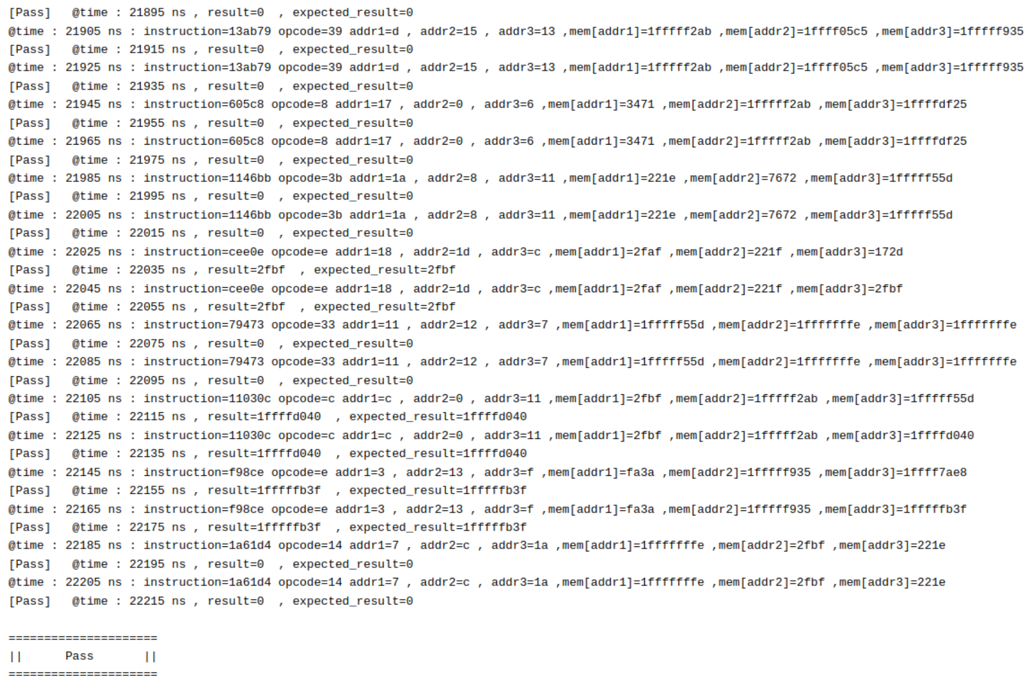
The input instruction and result (output)



```
[Pass]   @time : 21955 ns , result=0  , expected_result=0
@time : 21965 ns : instruction=605c8 opcode=8 addr1=17 , addr2=0 , addr3=6 ,mem[addr1]=3471 ,mem[addr2]=1fffff2ab ,mem[addr3]=1ffffdf25
[Pass]   @time : 21975 ns , result=0  , expected_result=0
@time : 21985 ns : instruction=1146bb opcode=3b addr1=1a , addr2=8 , addr3=11 ,mem[addr1]=221e ,mem[addr2]=7672 ,mem[addr3]=1fffff55d
[Pass]   @time : 21995 ns , result=0  , expected_result=0
@time : 22005 ns : instruction=1146bb opcode=3b addr1=1a , addr2=8 , addr3=11 ,mem[addr1]=221e ,mem[addr2]=7672 ,mem[addr3]=1fffff55d
[Pass]   @time : 22015 ns , result=0  , expected_result=0
@time : 22025 ns : instruction=cee0e opcode=e addr1=18 , addr2=1d , addr3=c ,mem[addr1]=2faf ,mem[addr2]=221f ,mem[addr3]=172d
[Pass]   @time : 22035 ns , result=2fbf  , expected_result=2fbf
@time : 22045 ns : instruction=cee0e opcode=e addr1=18 , addr2=1d , addr3=c ,mem[addr1]=2faf ,mem[addr2]=221f ,mem[addr3]=2fbf
[Pass]   @time : 22055 ns , result=2fbf  , expected_result=2fbf
@time : 22065 ns : instruction=79473 opcode=33 addr1=11 , addr2=12 , addr3=7 ,mem[addr1]=1fffff55d ,mem[addr2]=1ffffffe ,mem[addr3]=1ffffffe
[Pass]   @time : 22075 ns , result=0  , expected_result=0
@time : 22085 ns : instruction=79473 opcode=33 addr1=11 , addr2=12 , addr3=7 ,mem[addr1]=1fffff55d ,mem[addr2]=1ffffffe ,mem[addr3]=1ffffffe
[Pass]   @time : 22095 ns , result=0  , expected_result=0
@time : 22105 ns : instruction=11030c opcode=c addr1=c , addr2=0 , addr3=11 ,mem[addr1]=2fbf ,mem[addr2]=1fffff2ab ,mem[addr3]=1fffff55d
[Pass]   @time : 22115 ns , result=1ffffd040  , expected_result=1ffffd040
@time : 22125 ns : instruction=11030c opcode=c addr1=c , addr2=0 , addr3=11 ,mem[addr1]=2fbf ,mem[addr2]=1fffff2ab ,mem[addr3]=1ffffd040
[Pass]   @time : 22135 ns , result=1ffffd040  , expected_result=1ffffd040
@time : 22145 ns : instruction=f98ce opcode=e addr1=3 , addr2=13 , addr3=f ,mem[addr1]=fa3a ,mem[addr2]=1fffff935 ,mem[addr3]=1ffff7ae8
[Pass]   @time : 22155 ns , result=1fffffb3f  , expected_result=1fffffb3f
@time : 22165 ns : instruction=f98ce opcode=e addr1=3 , addr2=13 , addr3=f ,mem[addr1]=fa3a ,mem[addr2]=1fffff935 ,mem[addr3]=1fffffb3f
[Pass]   @time : 22175 ns , result=1fffffb3f  , expected_result=1fffffb3f
@time : 22185 ns : instruction=1a61d4 opcode=14 addr1=7 , addr2=c , addr3=1a ,mem[addr1]=1ffffffe ,mem[addr2]=2fbf ,mem[addr3]=221e
[Pass]   @time : 22195 ns , result=0  , expected_result=0
@time : 22205 ns : instruction=1a61d4 opcode=14 addr1=7 , addr2=c , addr3=1a ,mem[addr1]=1ffffffe ,mem[addr2]=2fbf ,mem[addr3]=221e
[Pass]   @time : 22215 ns , result=0  , expected_result=0

=====================
||      Pass       ||
=====================
           V C S   S i m u l a t i o n   R e p o r t
```

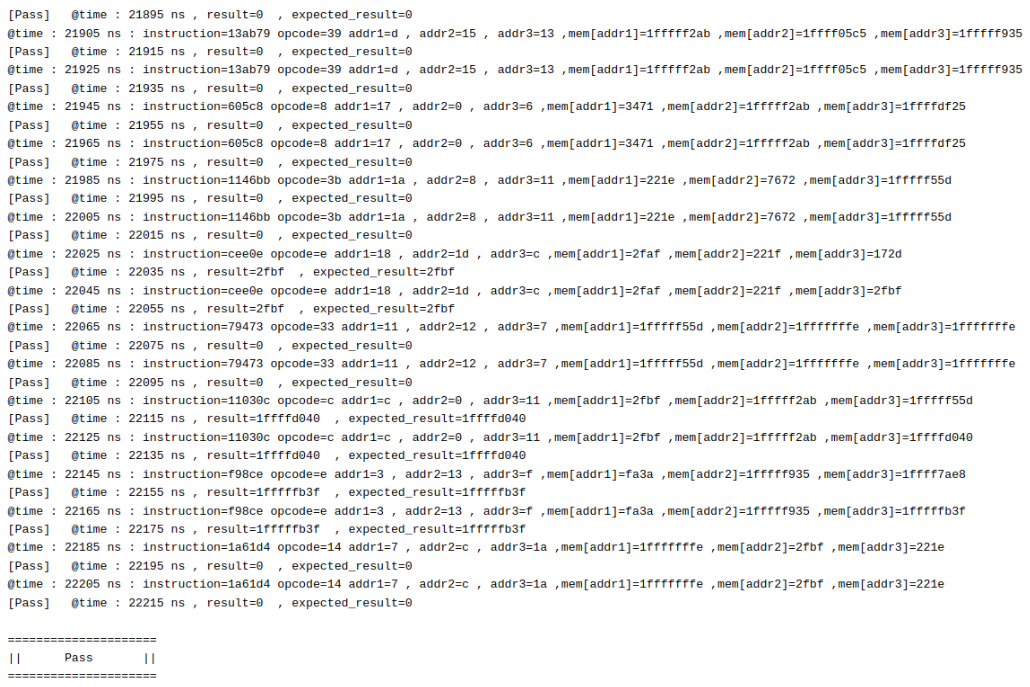
Figure 37 : Snapshot (1) for the input instruction and result (output)



```
[Pass]   @time : 21895 ns , result=0  , expected_result=0
@time : 21905 ns : instruction=13ab79 opcode=39 addr1=d , addr2=15 , addr3=13 ,mem[addr1]=1fffff2ab ,mem[addr2]=1ffff05c5 ,mem[addr3]=1fffff935
[Pass]   @time : 21915 ns , result=0  , expected_result=0
@time : 21925 ns : instruction=13ab79 opcode=39 addr1=d , addr2=15 , addr3=13 ,mem[addr1]=1fffff2ab ,mem[addr2]=1ffff05c5 ,mem[addr3]=1fffff935
[Pass]   @time : 21935 ns , result=0  , expected_result=0
@time : 21945 ns : instruction=605c8 opcode=8 addr1=17 , addr2=0 , addr3=6 ,mem[addr1]=3471 ,mem[addr2]=1fffff2ab ,mem[addr3]=1ffffdf25
[Pass]   @time : 21955 ns , result=0  , expected_result=0
@time : 21965 ns : instruction=605c8 opcode=8 addr1=17 , addr2=0 , addr3=6 ,mem[addr1]=3471 ,mem[addr2]=1fffff2ab ,mem[addr3]=1ffffdf25
[Pass]   @time : 21975 ns , result=0  , expected_result=0
@time : 21985 ns : instruction=1146bb opcode=3b addr1=1a , addr2=8 , addr3=11 ,mem[addr1]=221e ,mem[addr2]=7672 ,mem[addr3]=1fffff55d
[Pass]   @time : 21995 ns , result=0  , expected_result=0
@time : 22005 ns : instruction=1146bb opcode=3b addr1=1a , addr2=8 , addr3=11 ,mem[addr1]=221e ,mem[addr2]=7672 ,mem[addr3]=1fffff55d
[Pass]   @time : 22015 ns , result=0  , expected_result=0
@time : 22025 ns : instruction=cee0e opcode=e addr1=18 , addr2=1d , addr3=c ,mem[addr1]=2faf ,mem[addr2]=221f ,mem[addr3]=172d
[Pass]   @time : 22035 ns , result=2fbf  , expected_result=2fbf
@time : 22045 ns : instruction=cee0e opcode=e addr1=18 , addr2=1d , addr3=c ,mem[addr1]=2faf ,mem[addr2]=221f ,mem[addr3]=2fbf
[Pass]   @time : 22055 ns , result=2fbf  , expected_result=2fbf
@time : 22065 ns : instruction=79473 opcode=33 addr1=11 , addr2=12 , addr3=7 ,mem[addr1]=1fffff55d ,mem[addr2]=1ffffffe ,mem[addr3]=1ffffffe
[Pass]   @time : 22075 ns , result=0  , expected_result=0
@time : 22085 ns : instruction=79473 opcode=33 addr1=11 , addr2=12 , addr3=7 ,mem[addr1]=1fffff55d ,mem[addr2]=1ffffffe ,mem[addr3]=1ffffffe
[Pass]   @time : 22095 ns , result=0  , expected_result=0
@time : 22105 ns : instruction=11030c opcode=c addr1=c , addr2=0 , addr3=11 ,mem[addr1]=2fbf ,mem[addr2]=1fffff2ab ,mem[addr3]=1fffff55d
[Pass]   @time : 22115 ns , result=1ffffd040  , expected_result=1ffffd040
@time : 22125 ns : instruction=11030c opcode=c addr1=c , addr2=0 , addr3=11 ,mem[addr1]=2fbf ,mem[addr2]=1fffff2ab ,mem[addr3]=1ffffd040
[Pass]   @time : 22135 ns , result=1ffffd040  , expected_result=1ffffd040
@time : 22145 ns : instruction=f98ce opcode=e addr1=3 , addr2=13 , addr3=f ,mem[addr1]=fa3a ,mem[addr2]=1fffff935 ,mem[addr3]=1ffff7ae8
[Pass]   @time : 22155 ns , result=1fffffb3f  , expected_result=1fffffb3f
@time : 22165 ns : instruction=f98ce opcode=e addr1=3 , addr2=13 , addr3=f ,mem[addr1]=fa3a ,mem[addr2]=1fffff935 ,mem[addr3]=1fffffb3f
[Pass]   @time : 22175 ns , result=1fffffb3f  , expected_result=1fffffb3f
@time : 22185 ns : instruction=1a61d4 opcode=14 addr1=7 , addr2=c , addr3=1a ,mem[addr1]=1ffffffe ,mem[addr2]=2fbf ,mem[addr3]=221e
[Pass]   @time : 22195 ns , result=0  , expected_result=0
@time : 22205 ns : instruction=1a61d4 opcode=14 addr1=7 , addr2=c , addr3=1a ,mem[addr1]=1ffffffe ,mem[addr2]=2fbf ,mem[addr3]=221e
[Pass]   @time : 22215 ns , result=0  , expected_result=0

=====================
||      Pass       ||
=====================
```

Figure 38 : Snapshot (2) for the input instruction and result (output)



```
[Pass]   @time : 21895 ns , result=0  , expected_result=0
@time : 21905 ns : instruction=13ab79 opcode=39 addr1=d , addr2=15 , addr3=13 ,mem[addr1]=1fffff2ab ,mem[addr2]=1ffff05c5 ,mem[addr3]=1fffff935
[Pass]   @time : 21915 ns , result=0  , expected_result=0
@time : 21925 ns : instruction=13ab79 opcode=39 addr1=d , addr2=15 , addr3=13 ,mem[addr1]=1fffff2ab ,mem[addr2]=1ffff05c5 ,mem[addr3]=1fffff935
[Pass]   @time : 21935 ns , result=0  , expected_result=0
@time : 21945 ns : instruction=605c8 opcode=8 addr1=17 , addr2=0 , addr3=6 ,mem[addr1]=3471 ,mem[addr2]=1fffff2ab ,mem[addr3]=1ffffdf25
[Pass]   @time : 21955 ns , result=0  , expected_result=0
@time : 21965 ns : instruction=605c8 opcode=8 addr1=17 , addr2=0 , addr3=6 ,mem[addr1]=3471 ,mem[addr2]=1fffff2ab ,mem[addr3]=1ffffdf25
[Pass]   @time : 21975 ns , result=0  , expected_result=0
@time : 21985 ns : instruction=1146bb opcode=3b addr1=1a , addr2=8 , addr3=11 ,mem[addr1]=221e ,mem[addr2]=7672 ,mem[addr3]=1fffff55d
[Pass]   @time : 21995 ns , result=0  , expected_result=0
@time : 22005 ns : instruction=1146bb opcode=3b addr1=1a , addr2=8 , addr3=11 ,mem[addr1]=221e ,mem[addr2]=7672 ,mem[addr3]=1fffff55d
[Pass]   @time : 22015 ns , result=0  , expected_result=0
@time : 22025 ns : instruction=cee0e opcode=e addr1=18 , addr2=1d , addr3=c ,mem[addr1]=2faf ,mem[addr2]=221f ,mem[addr3]=172d
[Pass]   @time : 22035 ns , result=2fbf  , expected_result=2fbf
@time : 22045 ns : instruction=cee0e opcode=e addr1=18 , addr2=1d , addr3=c ,mem[addr1]=2faf ,mem[addr2]=221f ,mem[addr3]=2fbf
[Pass]   @time : 22055 ns , result=2fbf  , expected_result=2fbf
@time : 22065 ns : instruction=79473 opcode=33 addr1=11 , addr2=12 , addr3=7 ,mem[addr1]=1fffff55d ,mem[addr2]=1fffffffe ,mem[addr3]=1fffffffe
[Pass]   @time : 22075 ns , result=0  , expected_result=0
@time : 22085 ns : instruction=79473 opcode=33 addr1=11 , addr2=12 , addr3=7 ,mem[addr1]=1fffff55d ,mem[addr2]=1fffffffe ,mem[addr3]=1fffffffe
[Pass]   @time : 22095 ns , result=0  , expected_result=0
@time : 22105 ns : instruction=11030c opcode=c addr1=c , addr2=0 , addr3=11 ,mem[addr1]=2fbf ,mem[addr2]=1fffff2ab ,mem[addr3]=1fffff55d
[Pass]   @time : 22115 ns , result=1ffffd040  , expected_result=1ffffd040
@time : 22125 ns : instruction=11030c opcode=c addr1=c , addr2=0 , addr3=11 ,mem[addr1]=2fbf ,mem[addr2]=1fffff2ab ,mem[addr3]=1ffffd040
[Pass]   @time : 22135 ns , result=1ffffd040  , expected_result=1ffffd040
@time : 22145 ns : instruction=f98ce opcode=e addr1=3 , addr2=13 , addr3=f ,mem[addr1]=fa3a ,mem[addr2]=1fffff935 ,mem[addr3]=1ffff7ae8
[Pass]   @time : 22155 ns , result=1fffffb3f  , expected_result=1fffffb3f
@time : 22165 ns : instruction=f98ce opcode=e addr1=3 , addr2=13 , addr3=f ,mem[addr1]=fa3a ,mem[addr2]=1fffff935 ,mem[addr3]=1fffffb3f
[Pass]   @time : 22175 ns , result=1fffffb3f  , expected_result=1fffffb3f
@time : 22185 ns : instruction=1a61d4 opcode=14 addr1=7 , addr2=c , addr3=1a ,mem[addr1]=1fffffffe ,mem[addr2]=2fbf ,mem[addr3]=221e
[Pass]   @time : 22195 ns , result=0  , expected_result=0
@time : 22205 ns : instruction=1a61d4 opcode=14 addr1=7 , addr2=c , addr3=1a ,mem[addr1]=1fffffffe ,mem[addr2]=2fbf ,mem[addr3]=221e
[Pass]   @time : 22215 ns , result=0  , expected_result=0

=====================
||      Pass       ||
=====================
```

Figure 39 : Snapshot (3) for the input instruction and result (output)