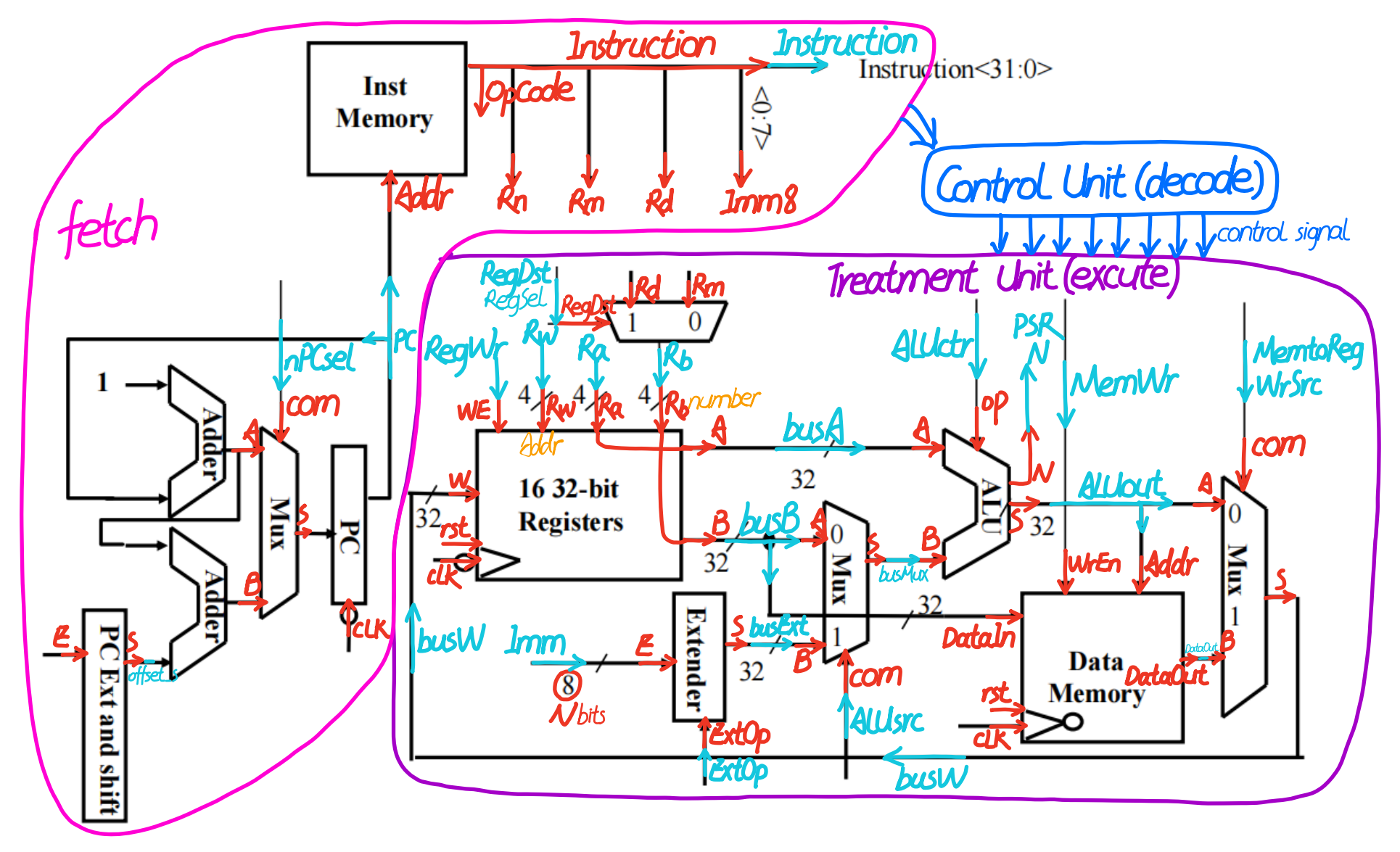
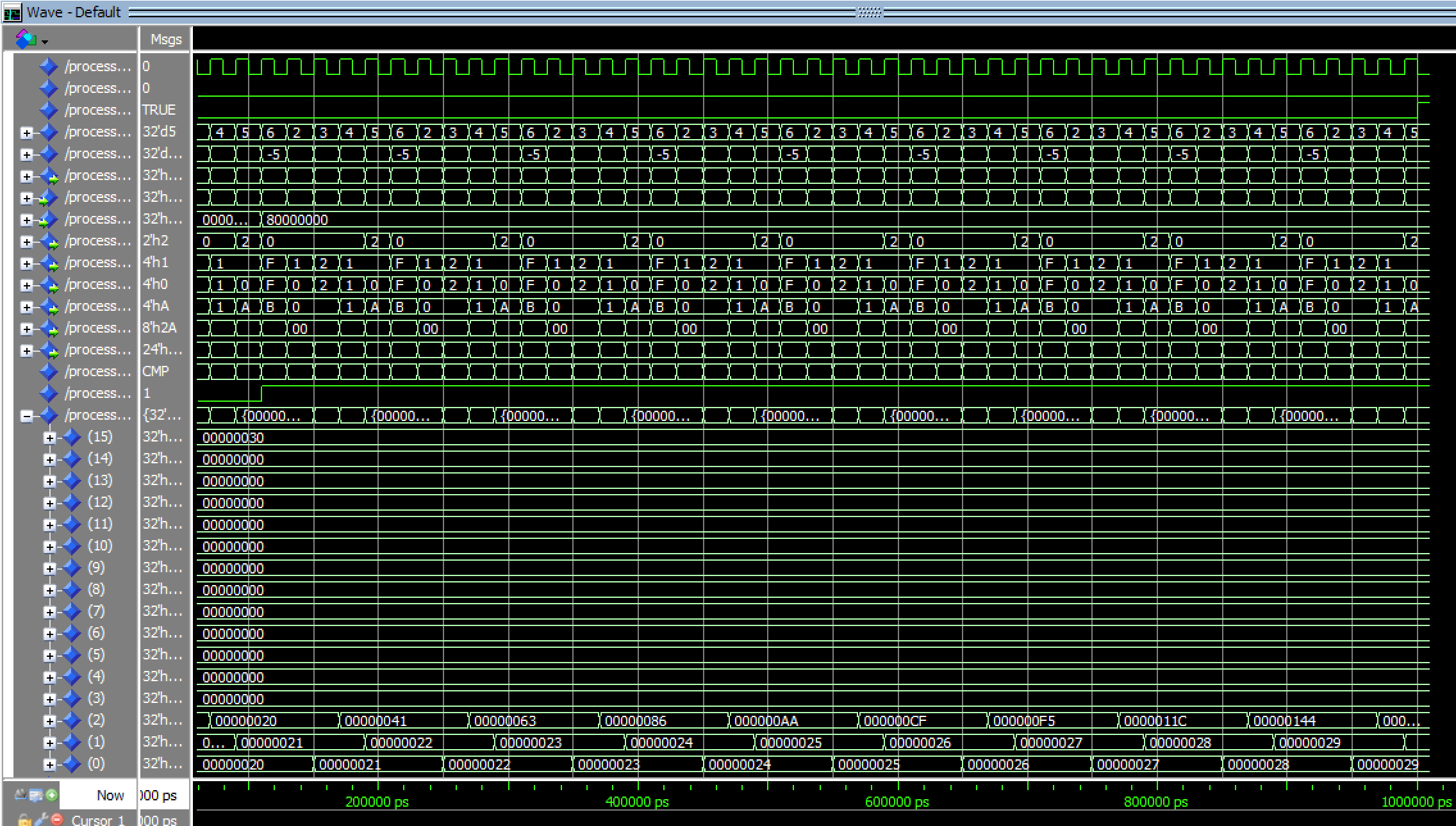
# PROCESSEUR MONO-CYCLE : RAPPORT

***Yuan Li - 19022100022***

***[16663781008@163.com](mailto:16663781008@163.com) Instrcuted by : Mr.Douze, Mr.Zong.***





## INTRODUCTION

The purpose of this project is to design and simulate a processor core. First, it will be divided into three parts: Treatment Unit, Instructions Fetch Unit and Control Unit. Secondly, each unit is composed of different modules, such as registers, multiplexers, memory and ALU, etc. Finally, each part is simulated and verified by the test bench.

Note: the.vhd and.do source code for each section are in the same file, check that each part of the project code works well if necessary. This report and code are completed by Yuan Li.

**PART 1 - TREATMENT UNIT**

The treatment unit mainly consists of arithmetic logic unit(ALU), register bank, data memory, multiplexer and signe extension.

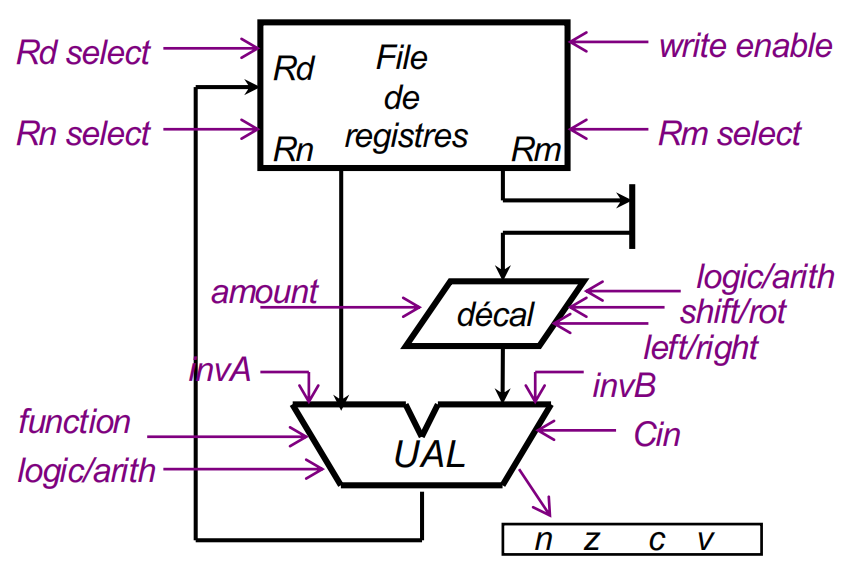
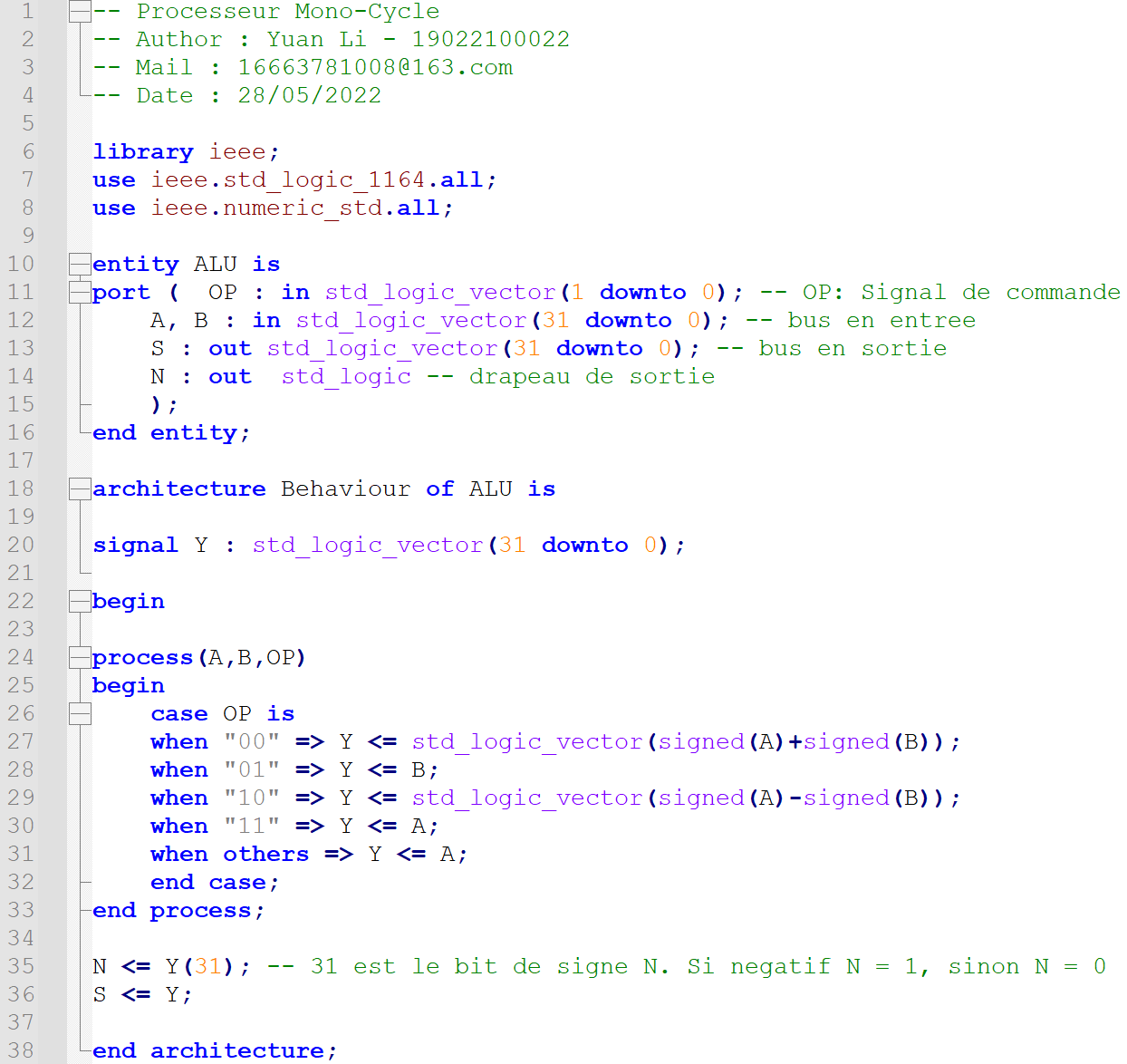
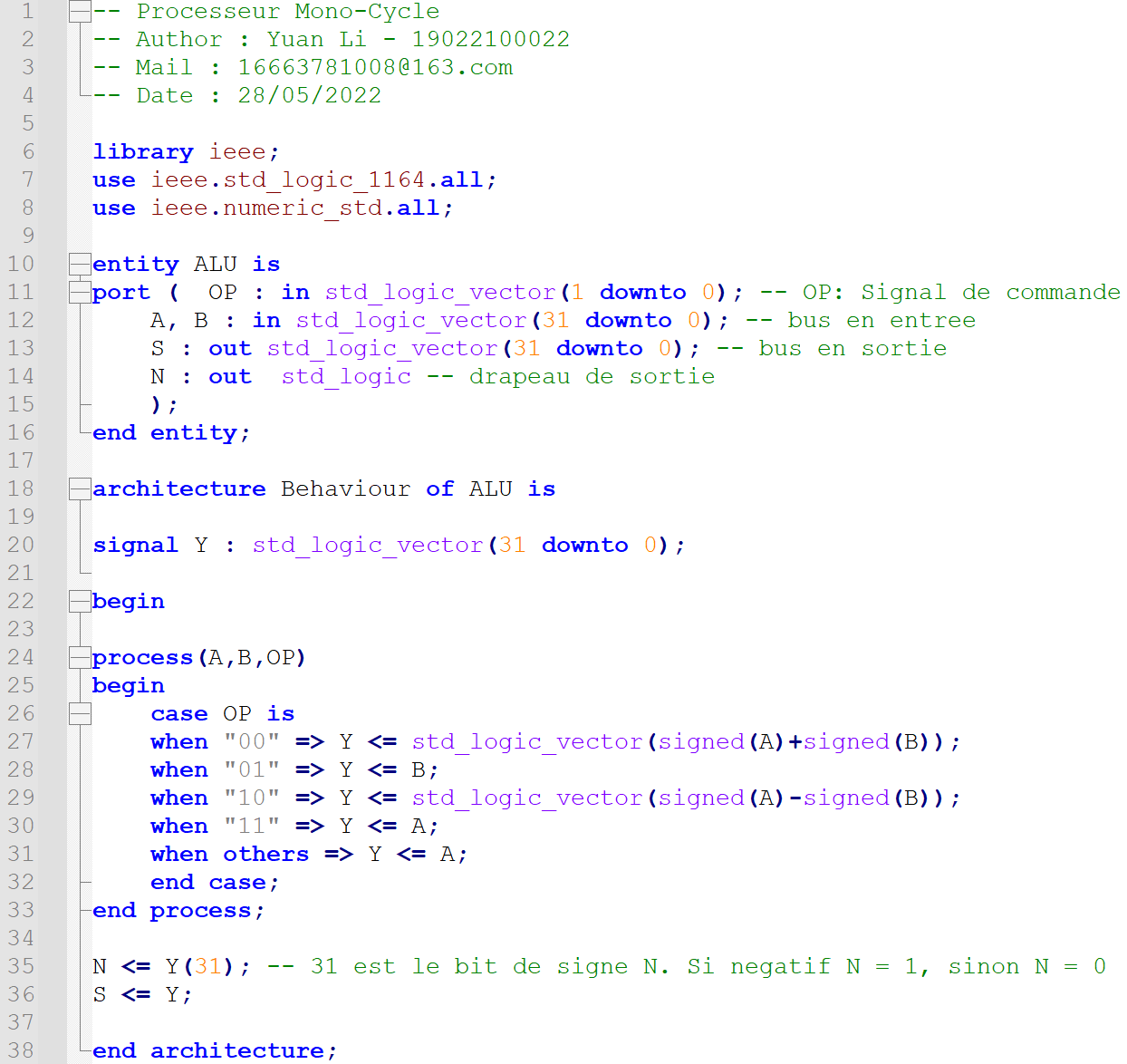


Figure 1 : Treatment Unit

**- ALU**

Arithmetic Logic Unit

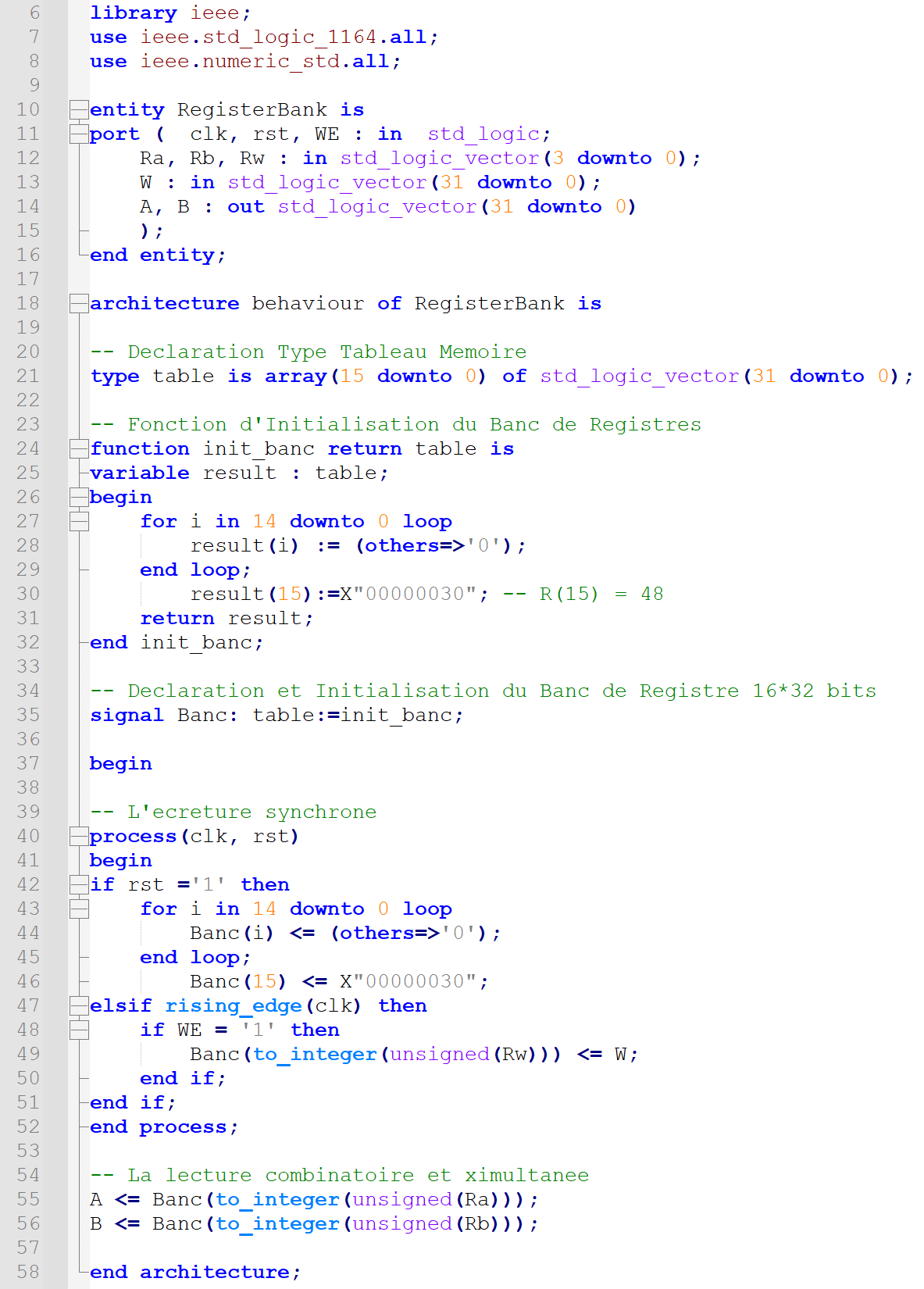
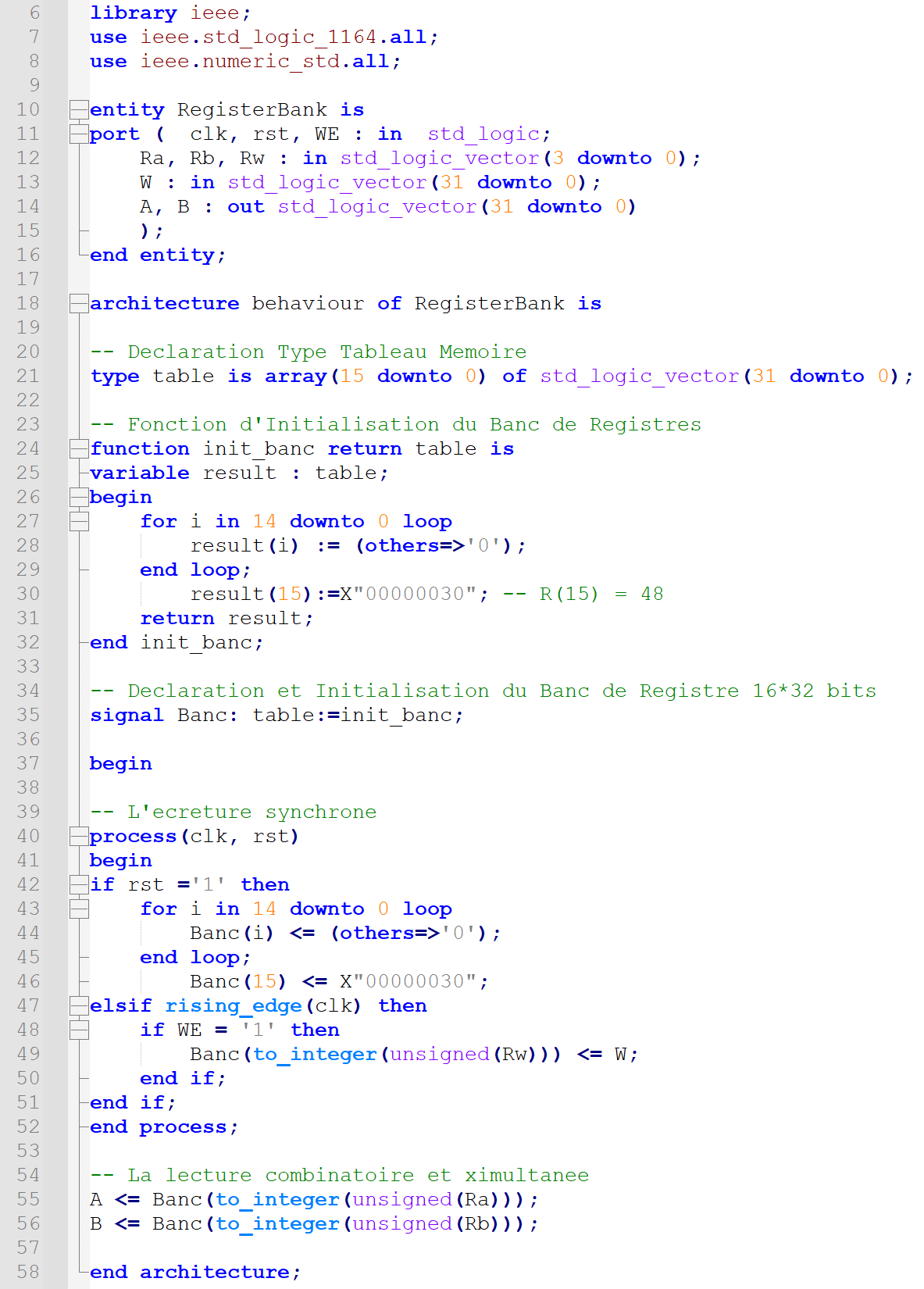
VHDL code implementation :



**- Register Bank**

The Register Bank consists of 16 32-bit registers.

VHDL code implementation :



**- Test Bench of Simple Treatment Unit**

Combine the ALU and Register Bank into a simple processing unit.

-- R(1) = R(15) = X"00000030"

Ra<= "1111"; -- A = R(15)

Rb <="0000"; -- par default

OP <= "11"; -- Y = A

WE <='1'; -- write enable

Rw<="0001"; -- R(1) <= W

wait for 20 ns;

-- R(1) = R(1) + R(15) = X"00000060"

Ra<= "0001"; -- A = R(1)

Rb <="1111"; -- B = R(15)

OP <= "00"; -- Y = A + B

WE <='1'; -- write enable

Rw<="0001"; -- R(1) <= W

wait for 20 ns;

-- R(2) = R(1) + R(15) = X"00000090"

Ra<= "0001"; -- A = R(1)

Rb <="1111"; -- B = R(15)

OP <= "00"; -- Y = A + B

WE <='1'; -- write enable

Rw<="0010"; -- R(2) <= W

wait for 20 ns;

-- R(3) = R(1) - R(15) = X"00000030"

Ra<= "0001"; -- A = R(1)

Rb <="1111"; -- B = R(15)

OP <= "10"; -- Y = A - B

WE <='1'; -- write enable

Rw<="0011"; -- R(2) <= W

wait for 20 ns;

-- R(5) = R(7) - R(15)= X"FFFFFFD0"

Ra<= "0111"; -- A = R(7)

Rb <="1111"; -- B = R(15)

OP <= "10"; -- Y = A - B

WE <='1'; -- write enable

Rw<="0101"; -- R(2) <= W

wait for 20 ns;

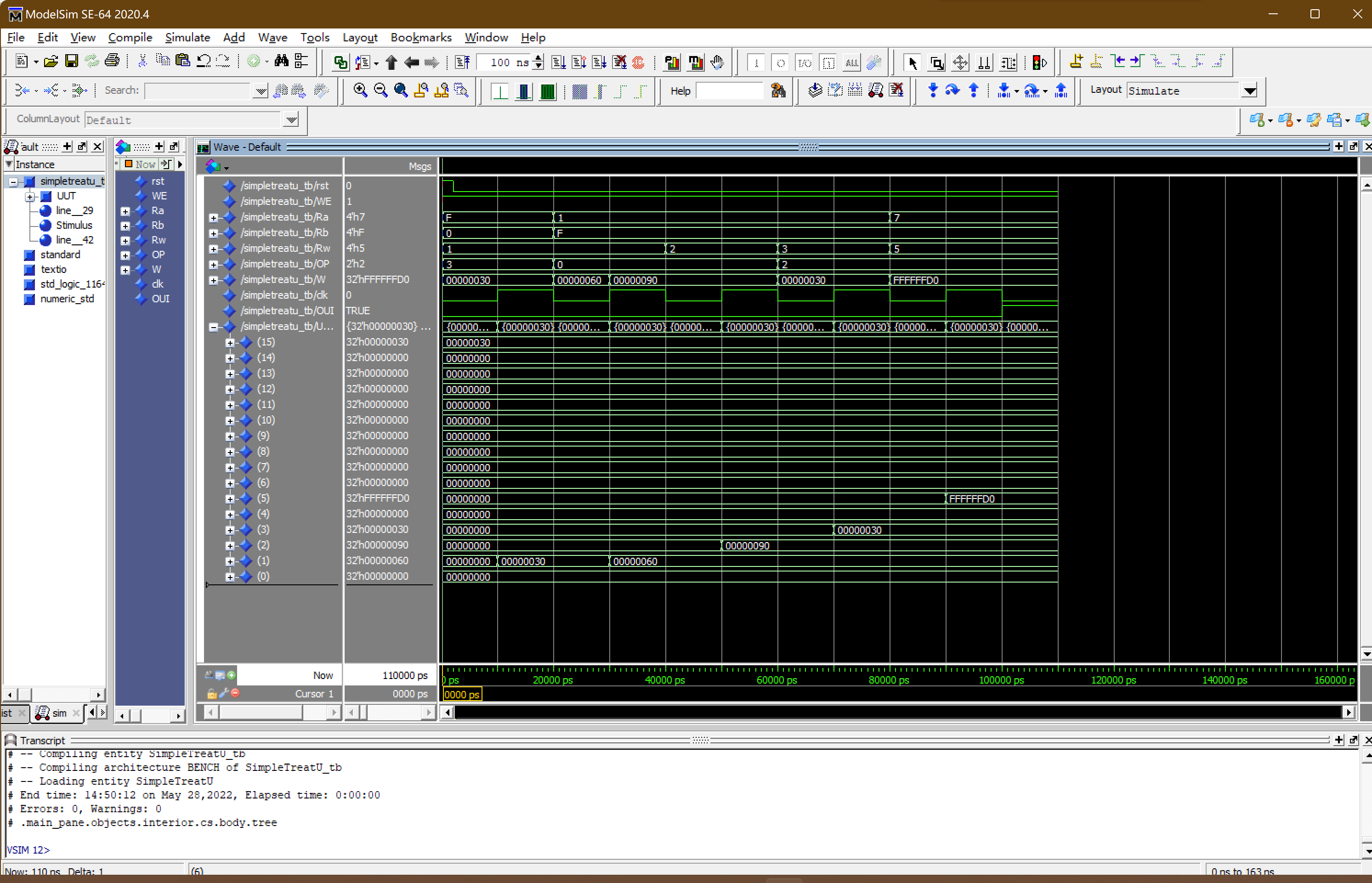


Figure 2 : Test Bench of Simple Treatment Unit

R(1) = R(1) + R(15) = X"00000060"

R(1) = R(1) + R(15) = X"00000060"

R(2) = R(1) + R(15) = X"00000090"

R(3) = R(1) - R(15) = X"00000030"

R(5) = R(7) - R(15)= X"FFFFFFD0"

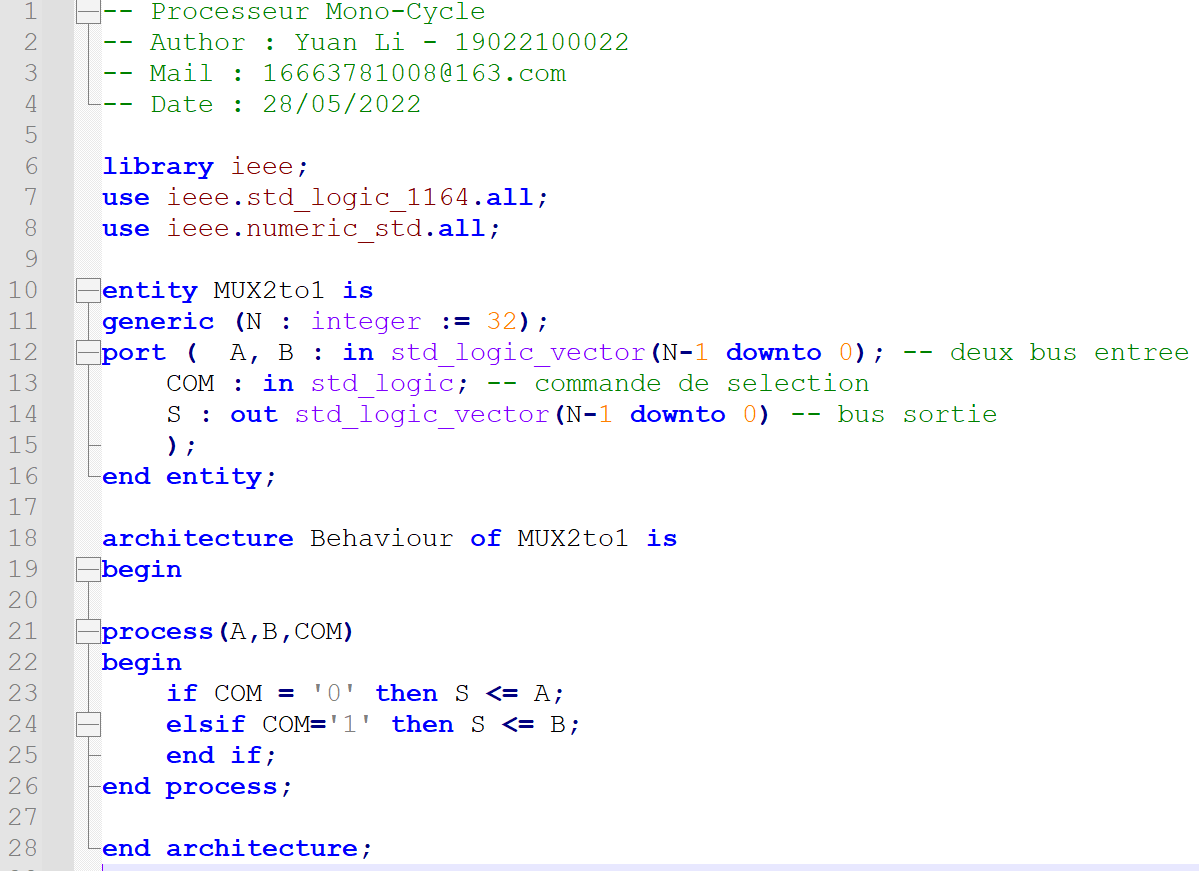
The simulation results show that the data in each register works well with time.

**- Multiplexer 2 to 1**

S = A, if COM = 0

S = B, if COM = 1

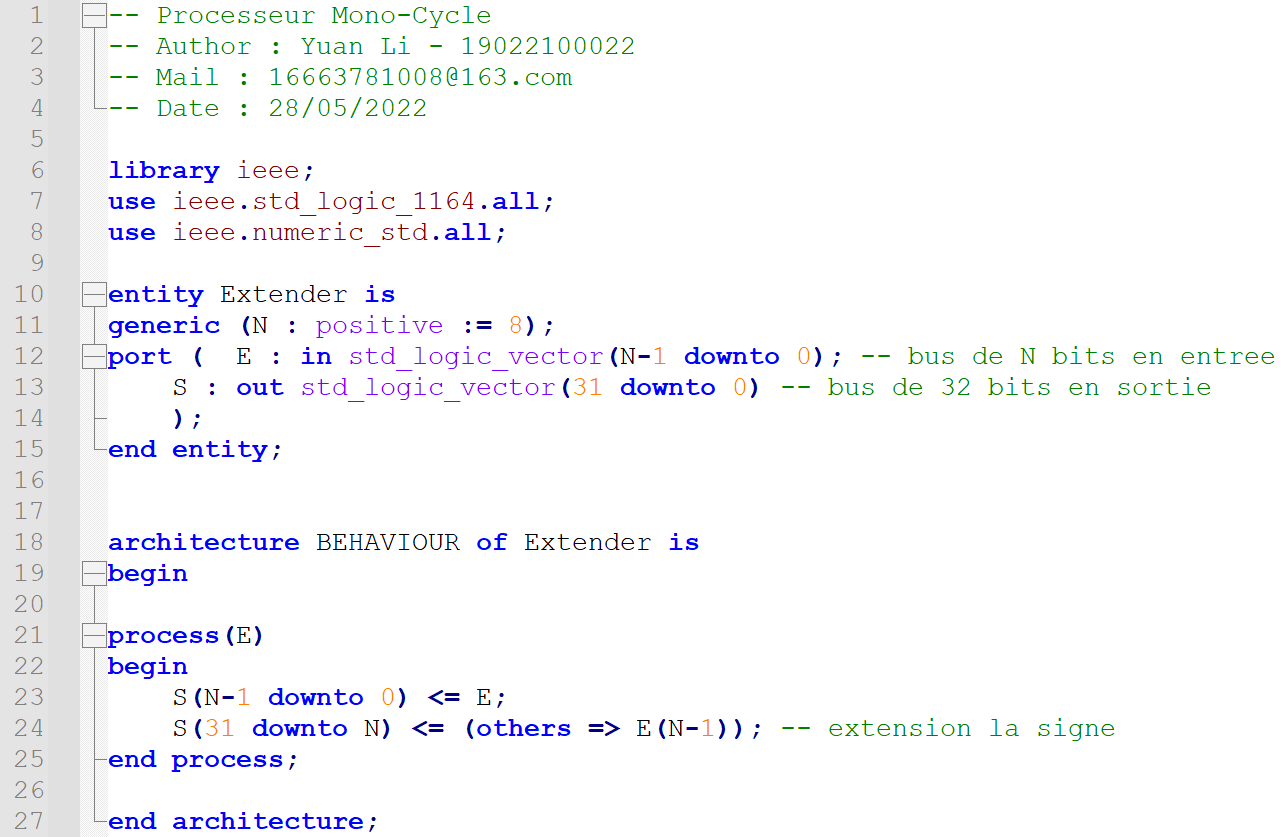
VHDL code implementation :



**- Sign Extender**

The extender outputs N bits of data to 32 bits.

VHDL code implementation :



**- Data Memory**

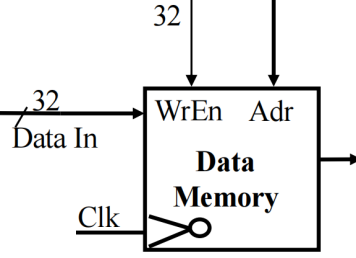
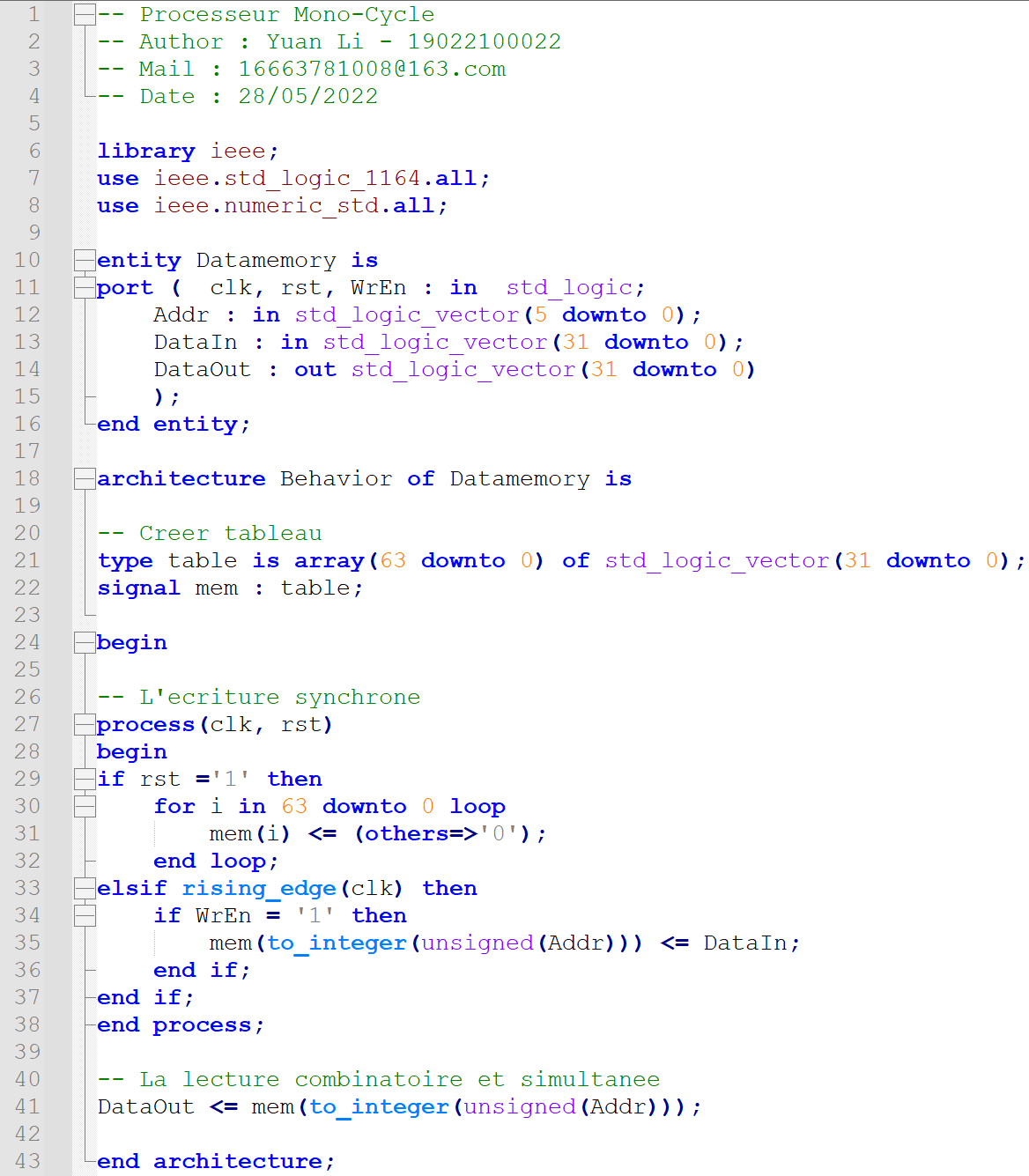


Figure 3 : Data Memory

Data Memory reads are combined circuits and writes are synchronous.

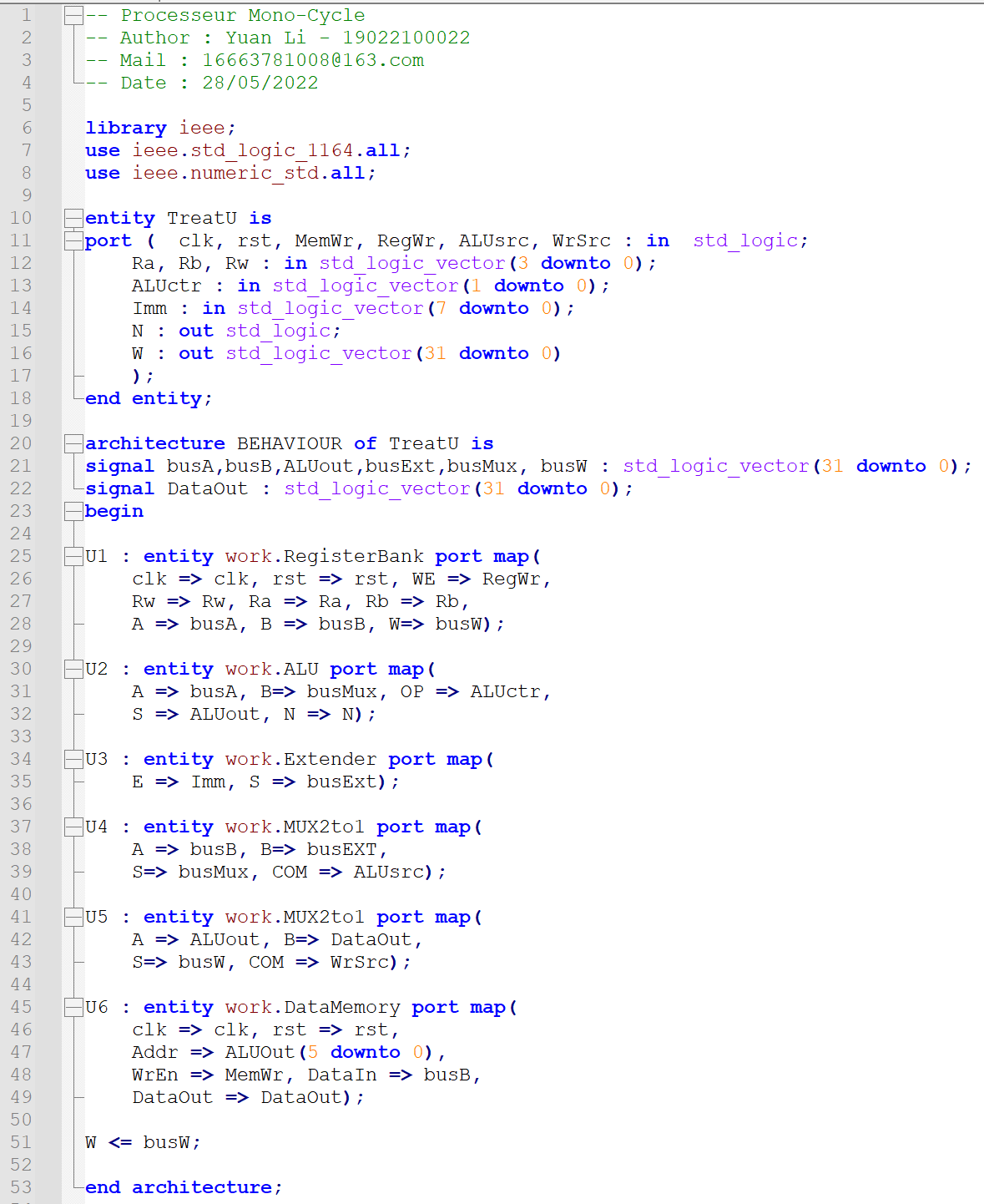
VHDL code implementation :



**- Assemblage Treatment Unit**

Assemble the previously generated components into a Treatment Unit.

VHDL code implementation :



**- Test Bench of Treatment Unit**

-- R(1) = R(15) = 48

Ra <= "1111"; -- busA = R(15)

Rb <= "0000"; -- par defaut

Imm <= "00000000"; -- par defaut

ALUsrc <= '0'; -- select busB

ALUctr <= "11"; -- ALUout = S = busA

WrSrc <= '0'; -- busW = S = ALUout

RegWr <='1'; -- write

Rw <= "0001"; -- R(1) <= bus W

MemWr <= '0'; -- non

wait for 20 ns;

-- R(2) = R(1) + R(15) = 96

Ra <= "0001"; -- busA = R(1)

Rb <= "1111"; -- busB = R(15)

Imm <= "00000000"; -- par defaut

ALUsrc <= '0'; -- select busA

ALUctr <= "00"; -- ALUout = S = A + B

WrSrc <= '0'; -- busW = S = ALUout

RegWr <='1'; -- write

Rw<="0010"; -- R(2) <= bus W

MemWr <= '0'; -- non

wait for 20 ns;

-- R(3) = R(1) + Imm(15) = 56

Ra <= "0001"; -- busA = R(1)

Rb <= "0000"; -- par defaut

Imm <= "00001000"; -- Imm = 8

ALUsrc <= '1'; -- select busB

ALUctr <= "00"; -- ALUout = S = A + B

WrSrc <= '0'; -- busW = S = ALUout

Rw <= "0011"; -- R(3) <= bus W

RegWr <='1'; -- write

MemWr <= '0'; -- non

wait for 20 ns;

-- R(4) = R(1) - R(2) = -48

Ra <= "0001"; -- busA = R(1)

Rb <= "0010"; -- busB = R(2)

Imm <= "00000000"; -- par defaut

ALUsrc <= '0'; -- select busA

ALUctr <= "10"; -- S = A - B

WrSrc <= '0'; -- write

Rw <= "0100"; -- R(4) <= bus W

RegWr <='1';

MemWr <= '0';

wait for 20 ns;

-- R(5) = R(1) - Imm (2) = 46

Ra <= "0001"; -- busA = R(1)

Rb <= "0000"; -- par defaut

Imm <= "00000010"; -- Imm = 2

ALUsrc <= '1';

AlUctr <= "10"; -- S = A - B

WrSrc <= '0';

Rw <= "0101"; -- R(5) <= bus W

RegWr <='1';

MemWr <= '0';

wait for 20 ns;

-- R(6) = R(1) = 48

Ra <= "0001"; -- busA = R(1)

Rb <= "0000"; -- par defaut

Imm <= "00000000"; -- par defaut

ALUsrc <= '0'; -- select busB

ALUctr <= "11"; -- ALUout = S = busA

WrSrc <= '0'; -- busW = S = ALUout

RegWr <='1'; -- write

Rw <= "0110"; -- R(6) <= bus W

MemWr <= '0'; -- non

wait for 20 ns;

-- Mem(R1) = R(3) = 56

Ra<= "0001"; -- Addr = R(1)

Rb <="0011"; --R(3)

Imm <= "00000000"; -- par defaut

ALUctr <= "11"; -- S = A

ALUsrc <= '0';

Rw<="0000"; -- par defaut

WrSrc <= '0';

RegWr <= '0'; -- no

MemWr <= '1'; -- write memory

wait for 20 ns;

-- R(6) = Mem(R1) = 56

Ra<= "0001"; -- par defaut

Rb <= "0000"; -- par defaut

Imm <= "00000000"; -- par defaut

Rw<="0110"; --R(6)

RegWr <='1';

ALUctr <= "11"; -- S = A

ALUsrc <= '0';

WrSrc <= '1'; -- S = Dataout

MemWr <= '0';

wait for 20 ns;

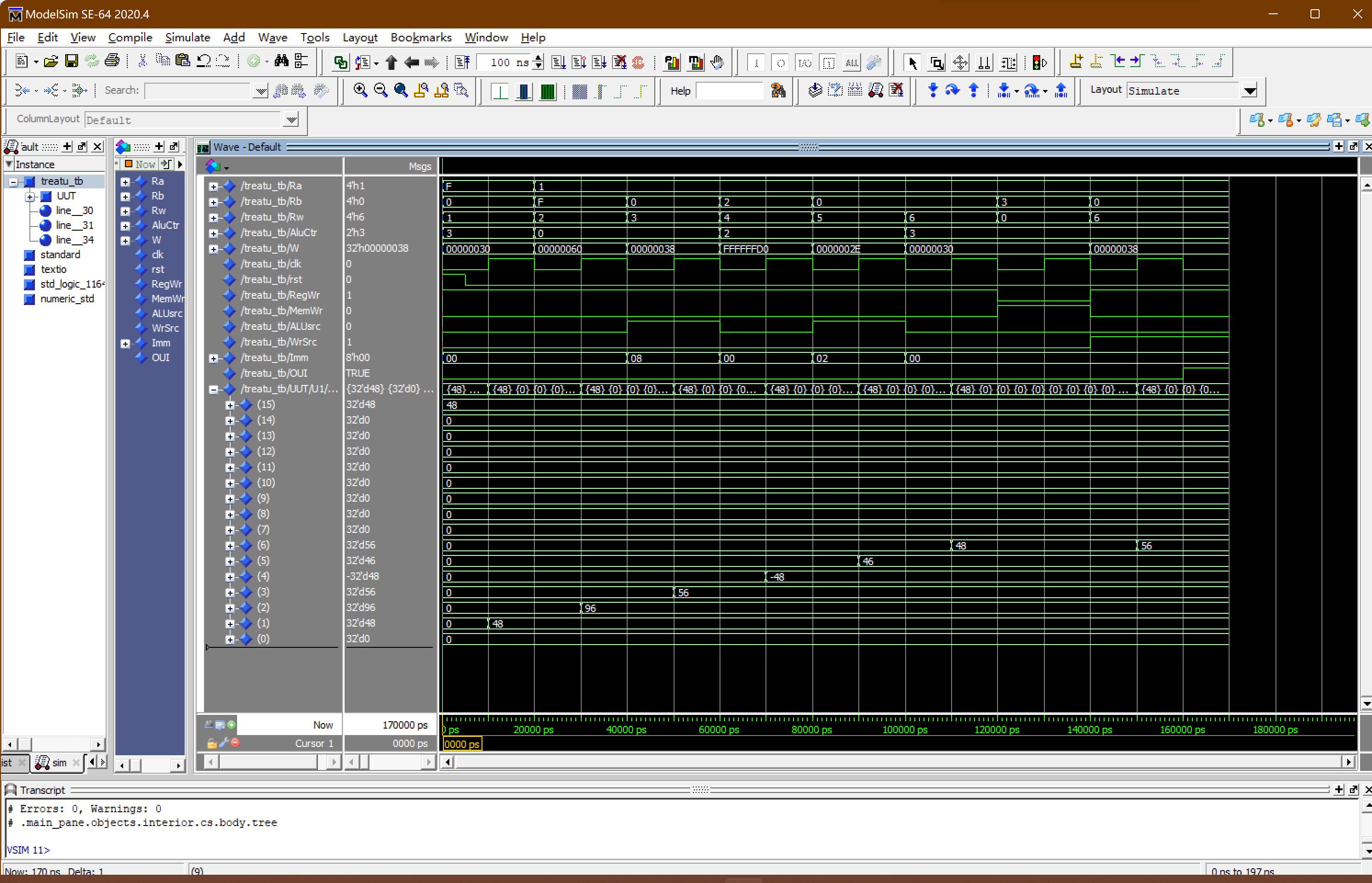


Figure 4 : Test Bench of Treatment Unit

R(1) = R(15) = 48 (decimal)

R(2) = R(1) + R(15) = 96

R(3) = R(1) + Imm(15) = 56

R(4) = R(1) - R(2) = -48

R(5) = R(1) - Imm (2) = 46

R(6) = R(1) = 48

Mem(R1) = R(3) = 56

R(6) = Mem(R1) = 56

The simulation results show that the data in each register works well with time.

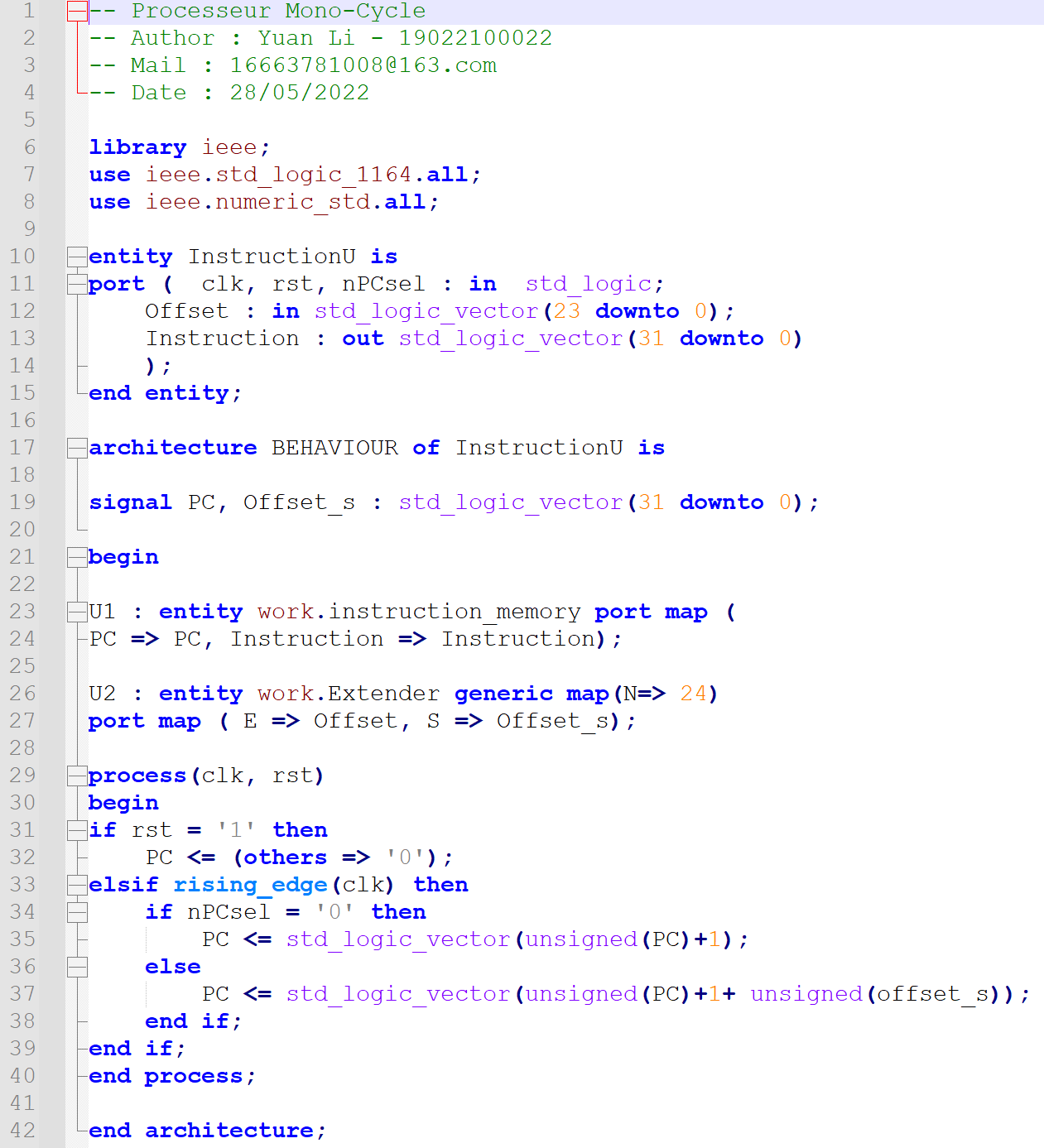
**Part 2 - Instruction Fetch Unit**

**- Instruction Fetch Unit**

PC = PC + 1, if nPCsel = 0

PC = PC + 1 + Offset, if nPCsel = 1

VHDL code implementation :



**- Test Bench of Instruction Fetch Unit**

-- PC <= PC + 1 = 1

nPCsel<= '0';

Offset <= (others => '0');

wait for 20 ns;

-- PC <= PC + 1 = 2

nPCsel<= '0';

Offset <= (others => '0');

wait for 20 ns;

-- PC <= PC + 1 = 3

nPCsel<= '0';

Offset <= (others => '0');

wait for 20 ns;

-- PC <= PC + 1 + Offset of 5 = 9

nPCsel<= '1';

Offset <= x"000005";

wait for 20 ns;

-- PC <= PC + 1 Offset of 1 = B

nPCsel<= '1';

Offset <= x"000001";

wait for 20 ns;

-- PC <= PC + 1 + Offset of -4 = 8

nPCsel<= '1';

Offset <= x"FFFFFC";

wait for 20 ns;

-- PC <= PC + 1 = 9

nPCsel<= '0';

Offset <= (others => '0');

wait for 20 ns;

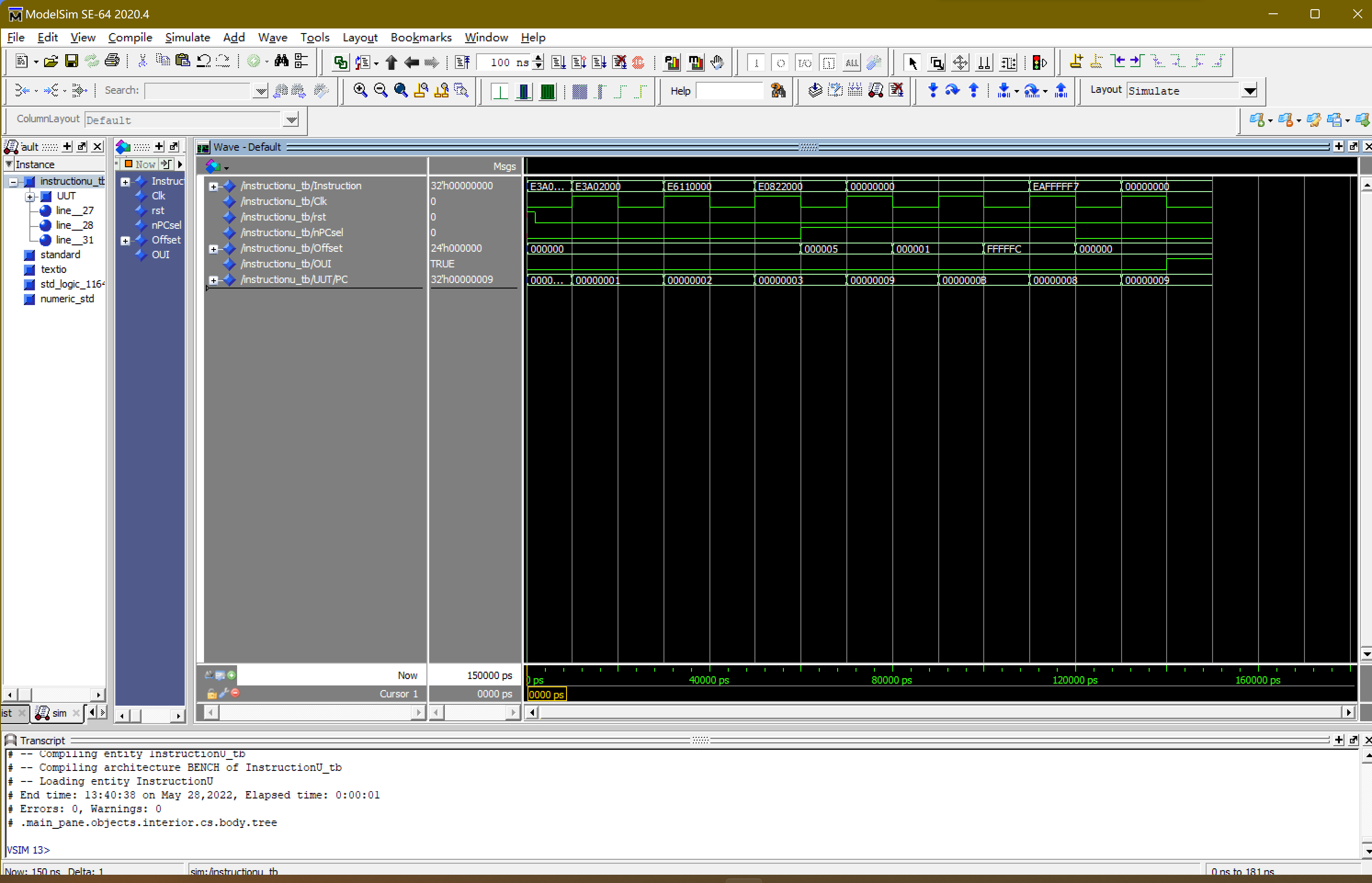


Figure 5 : Test Bench of Instructions Fetch Unit

PC <= PC + 1 = 1

PC <= PC + 1 = 2

PC <= PC + 1 = 3

PC <= PC + 1 + Offset of 5 = 9

PC <= PC + 1 Offset of 1 = B

PC <= PC + 1 + Offset of -4 = 8

PC <= PC + 1 = 9

The simulation results show that the data in each register works well with time.

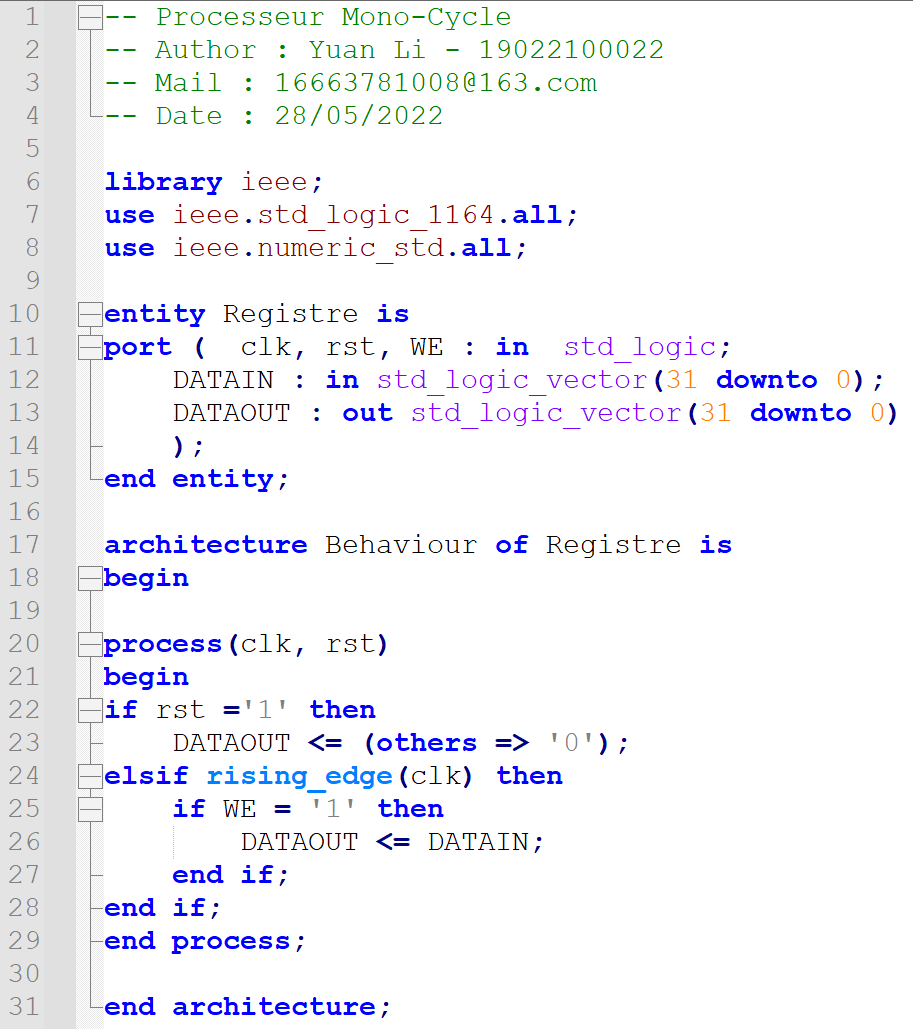
**Part3 - Control Unit**

The control unit consists of 32 Bits Register with Charge Command and Instructions Decoder.

**- 32 Bits Register with Charge Command**

This module stores the state of the processor(CPSR).

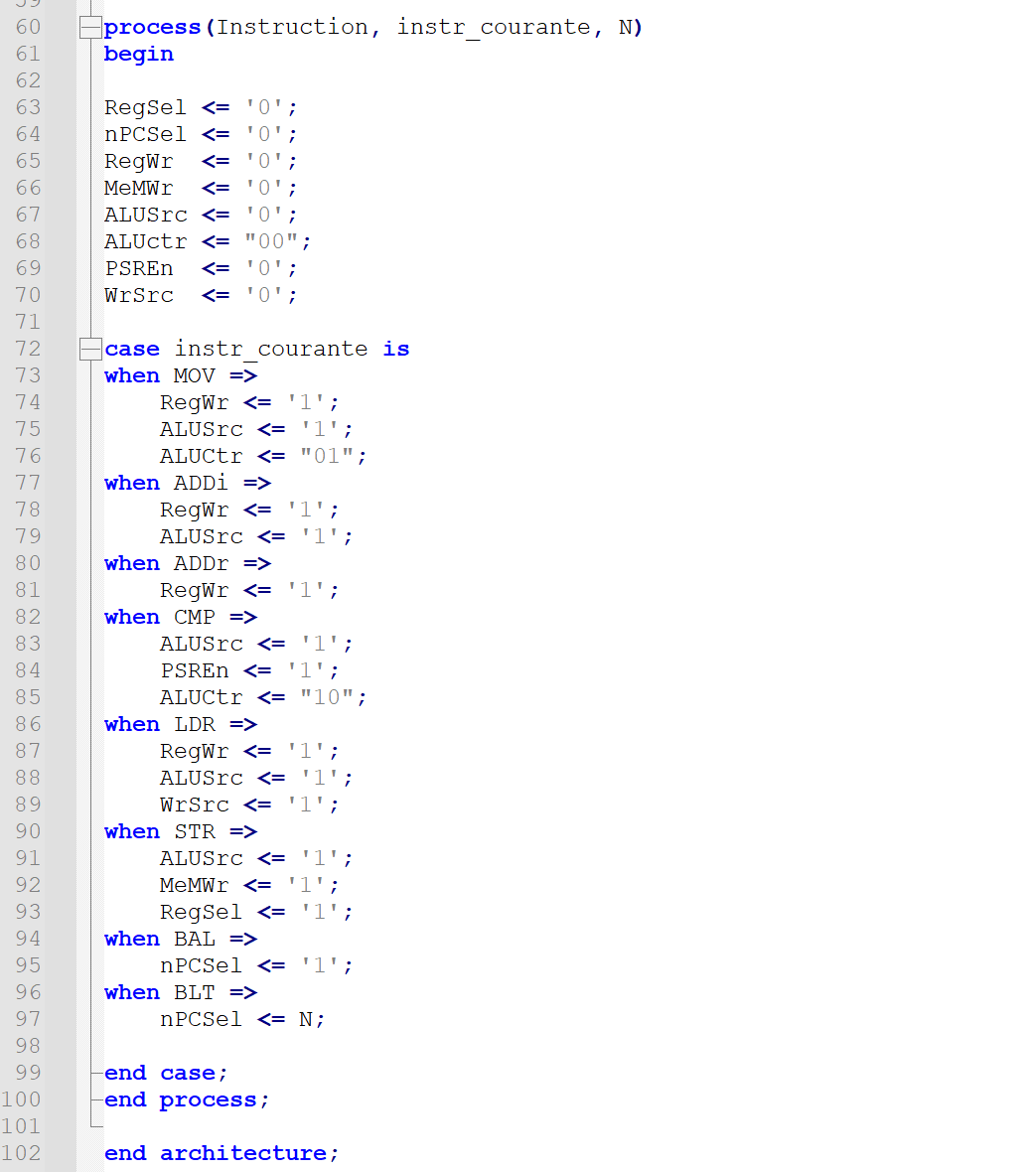
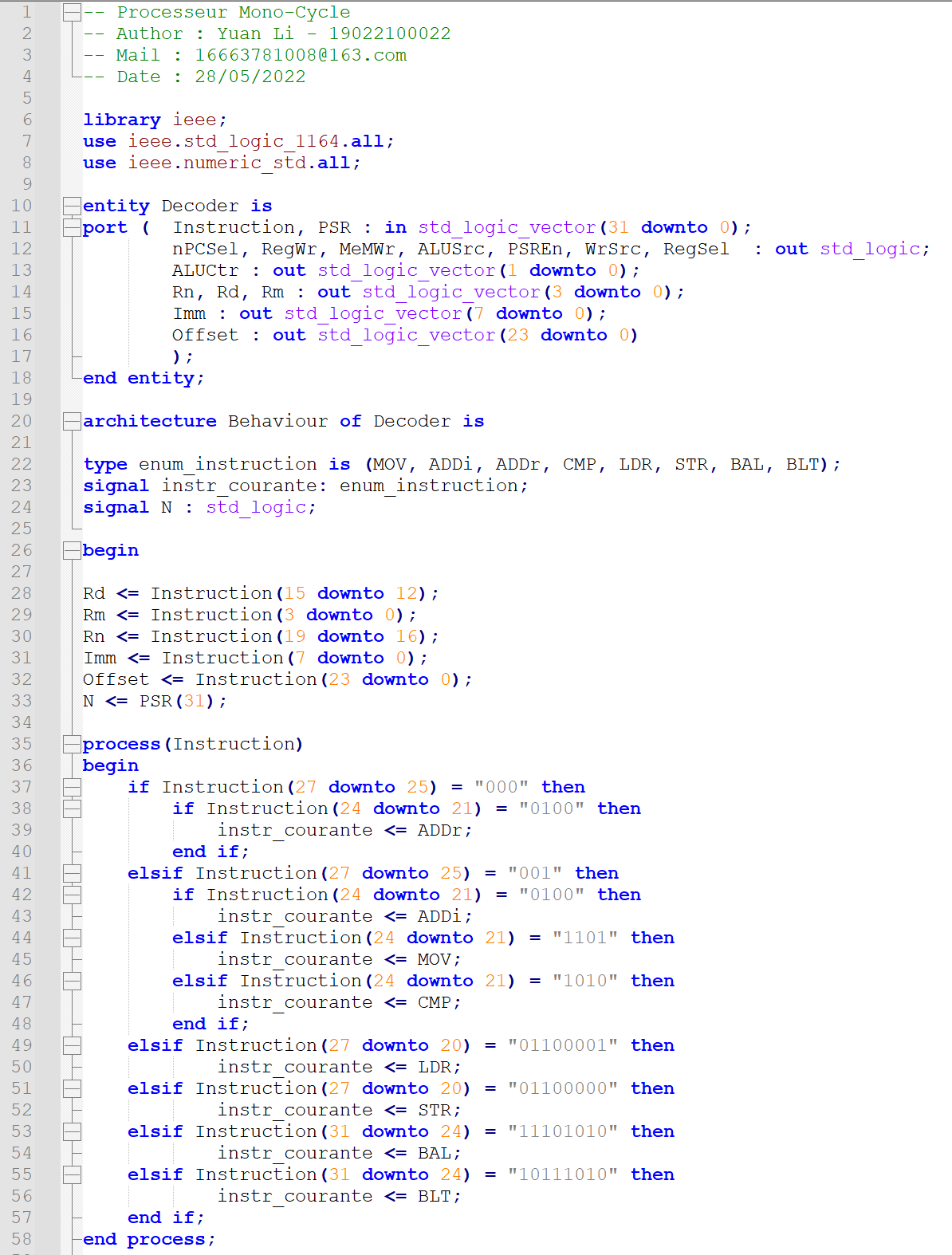
VHDL code implementation :



**- Instructions Decoder**

The decoder generates signals that control Treatment Unit, Instructions Fetch Unit, and CPSR.

VHDL code implementation :



For each different instruction, a corresponding control signal can be generated.

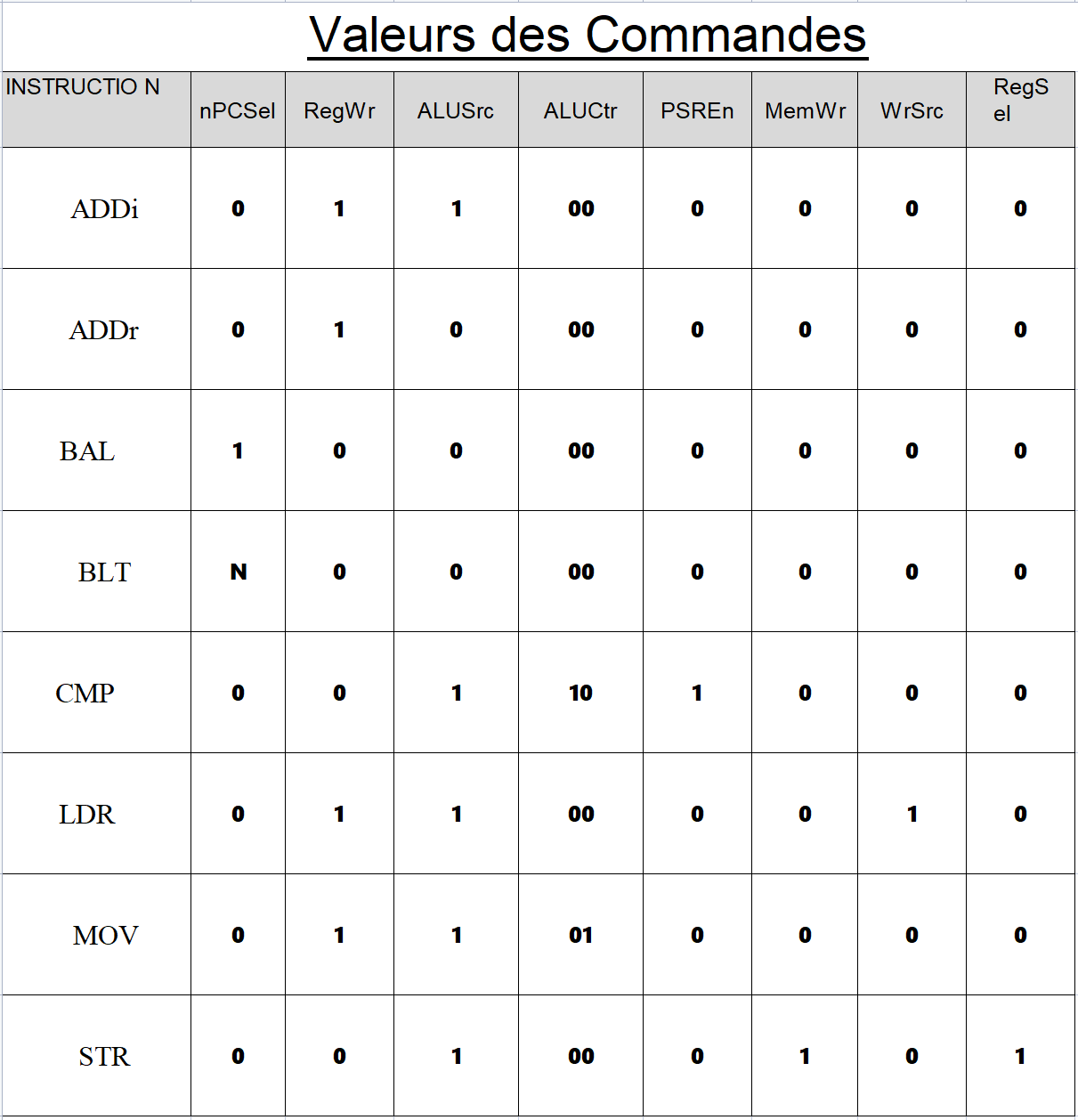


Table 1 : Control Values

**Part 4 - Combination Processor and Test**

In this part, the three previously generated units(Treatment Unit, Instructions Fetch Unit and Control Unit) are combined to form a processor.

**- Processor**

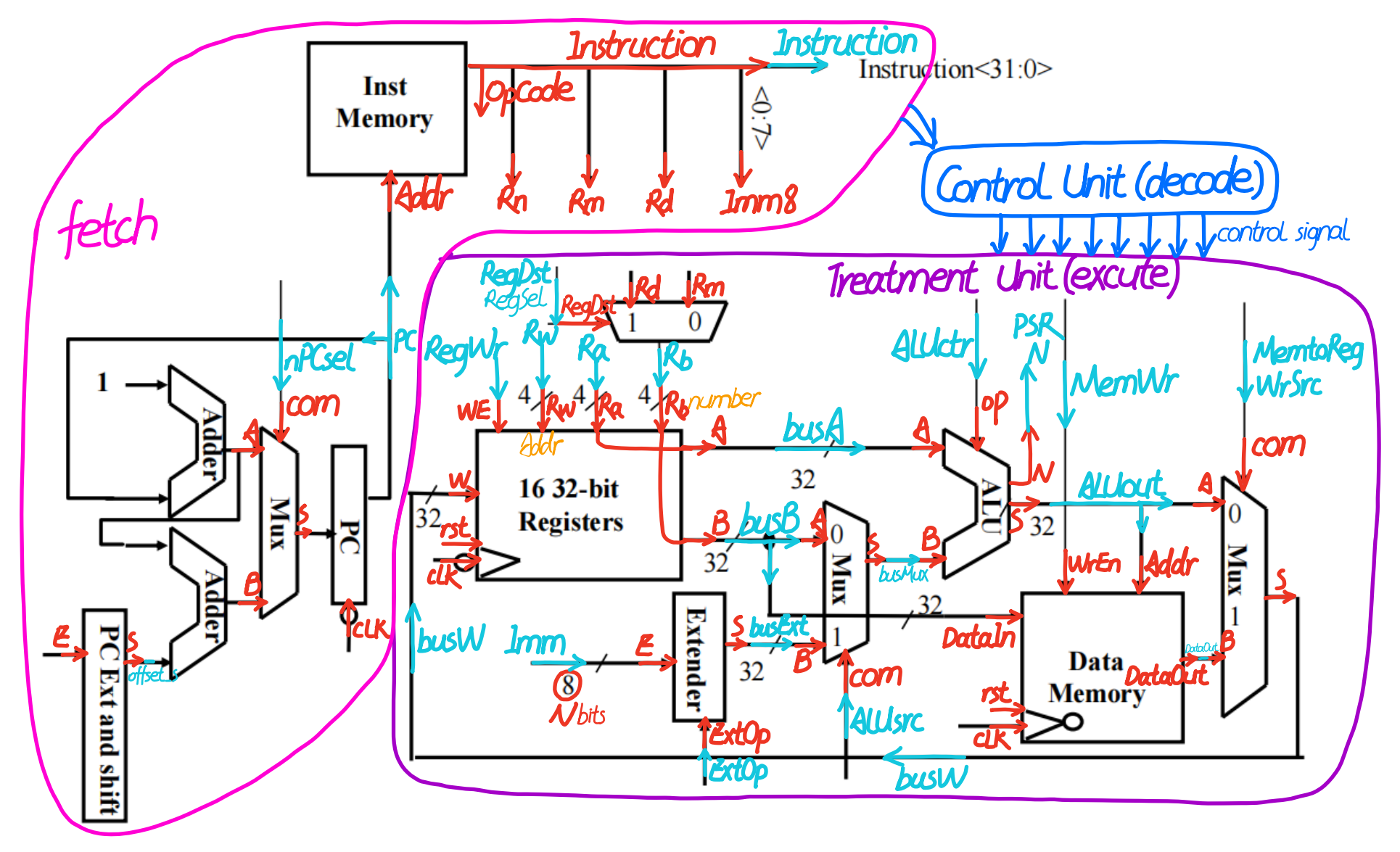
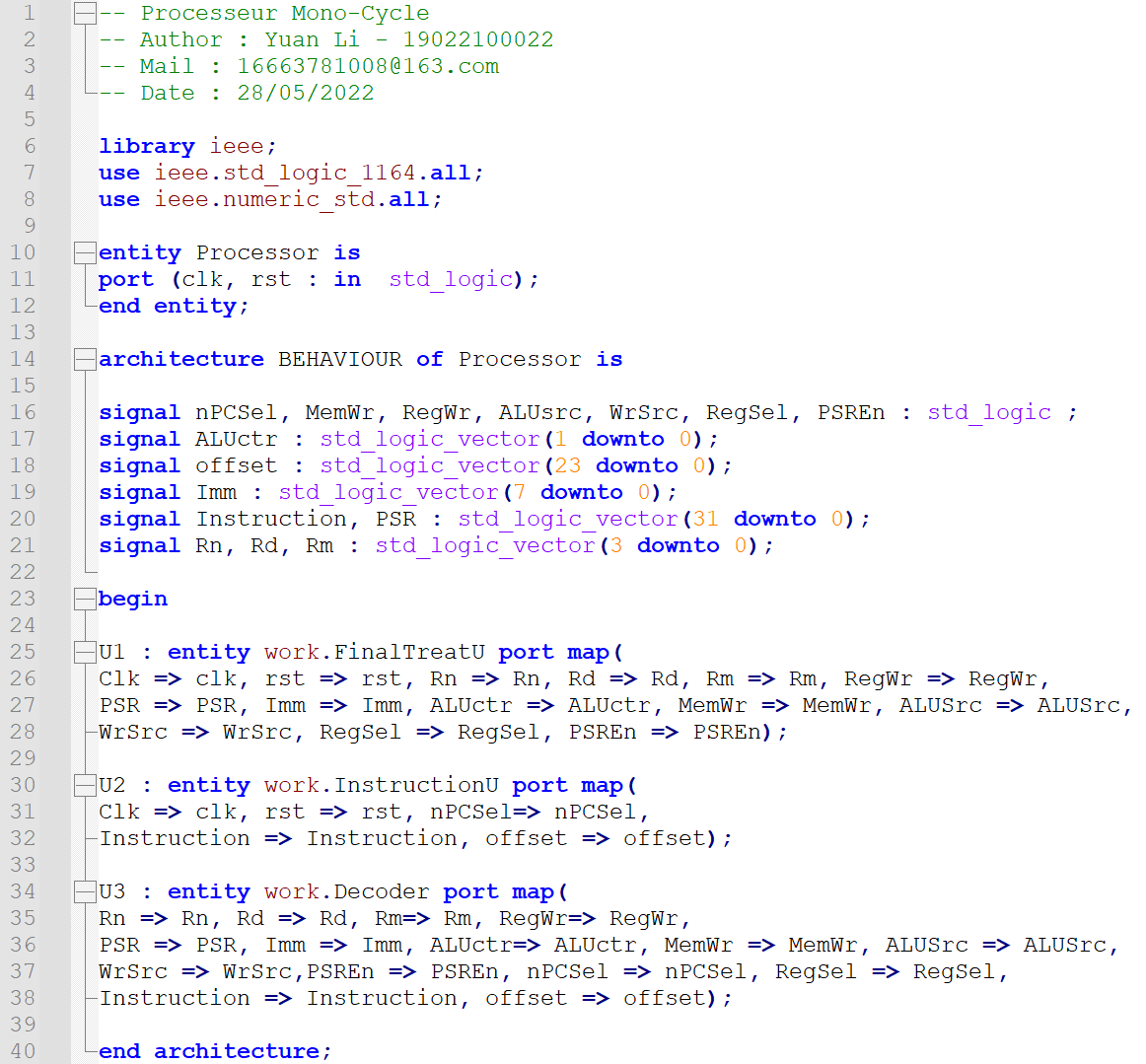


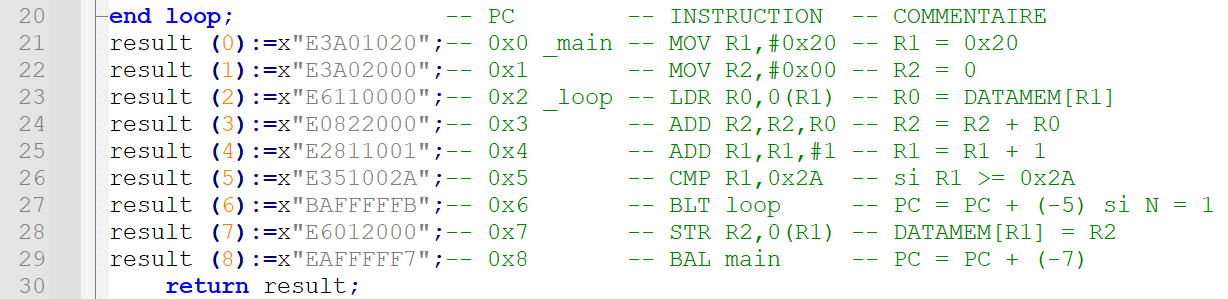
Figure 6 : Combination Processor and Data Path (Annotated by Yuan Li)

VHDL code implementation :



**- Test Bench of Processor（instruction of memory）**

Verify whether the processor runs well through simulation.



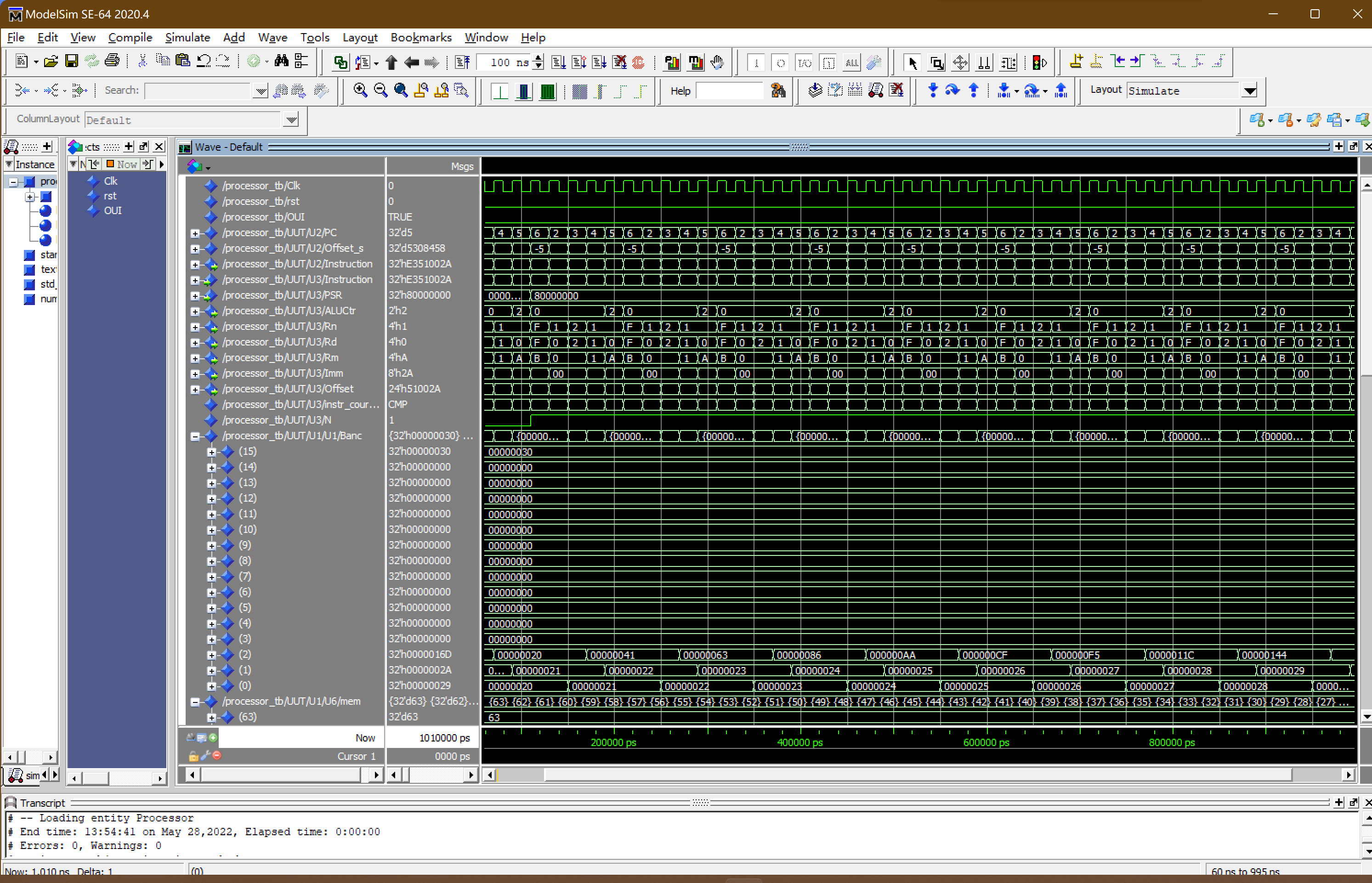
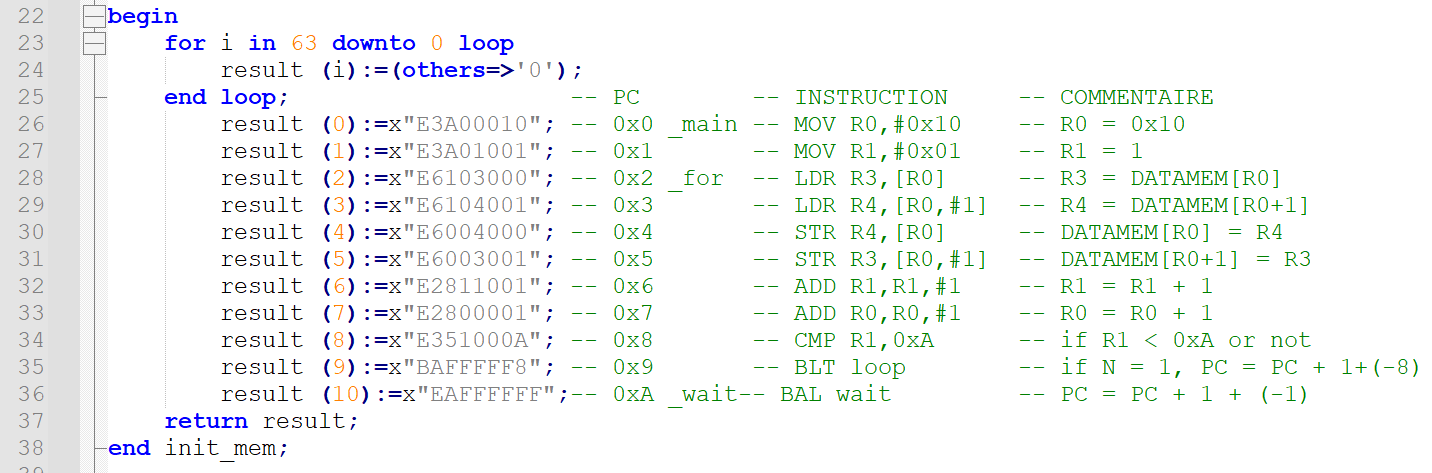


Figure 7 : Test Bench of Processor

By observing R(0), R(1) and R(2) in the simulation results, the code in the test bench works well. The value of R(0), R(1) and R(2) changes over time in accordance with the source code.

**Part 5 - Processor Test**

**- Test Bench of Processor（LOAD, STORE with Offset）**



By modifying some code in the instruction memory to verify the operation of LOAD, STORE with Offset, create a new file called instructions\_memory2.vhd, then the simulation results are verified to be correct.

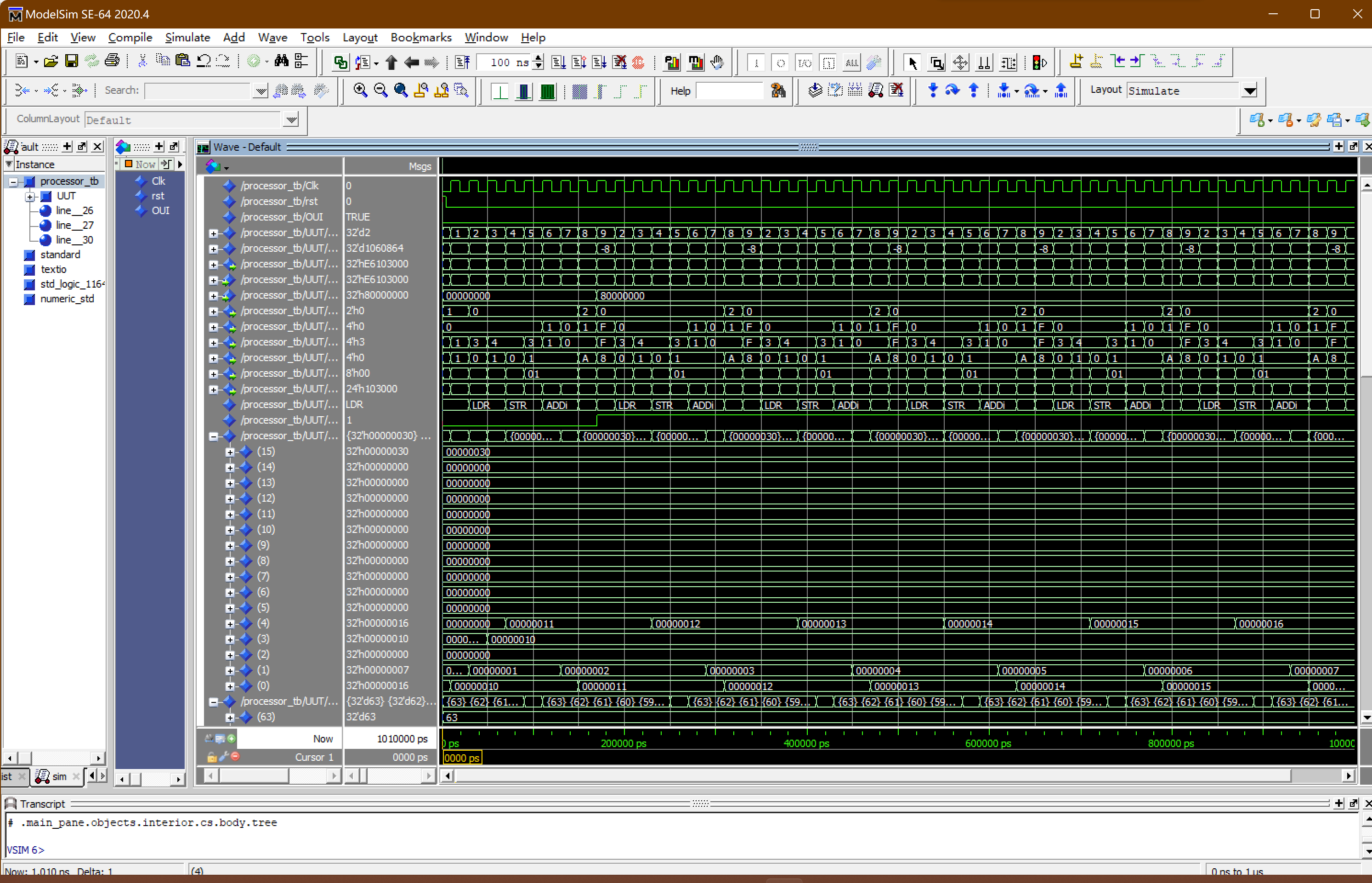


Figure 8 : Test Bench of Processor(LOAD, STORE with Offset)

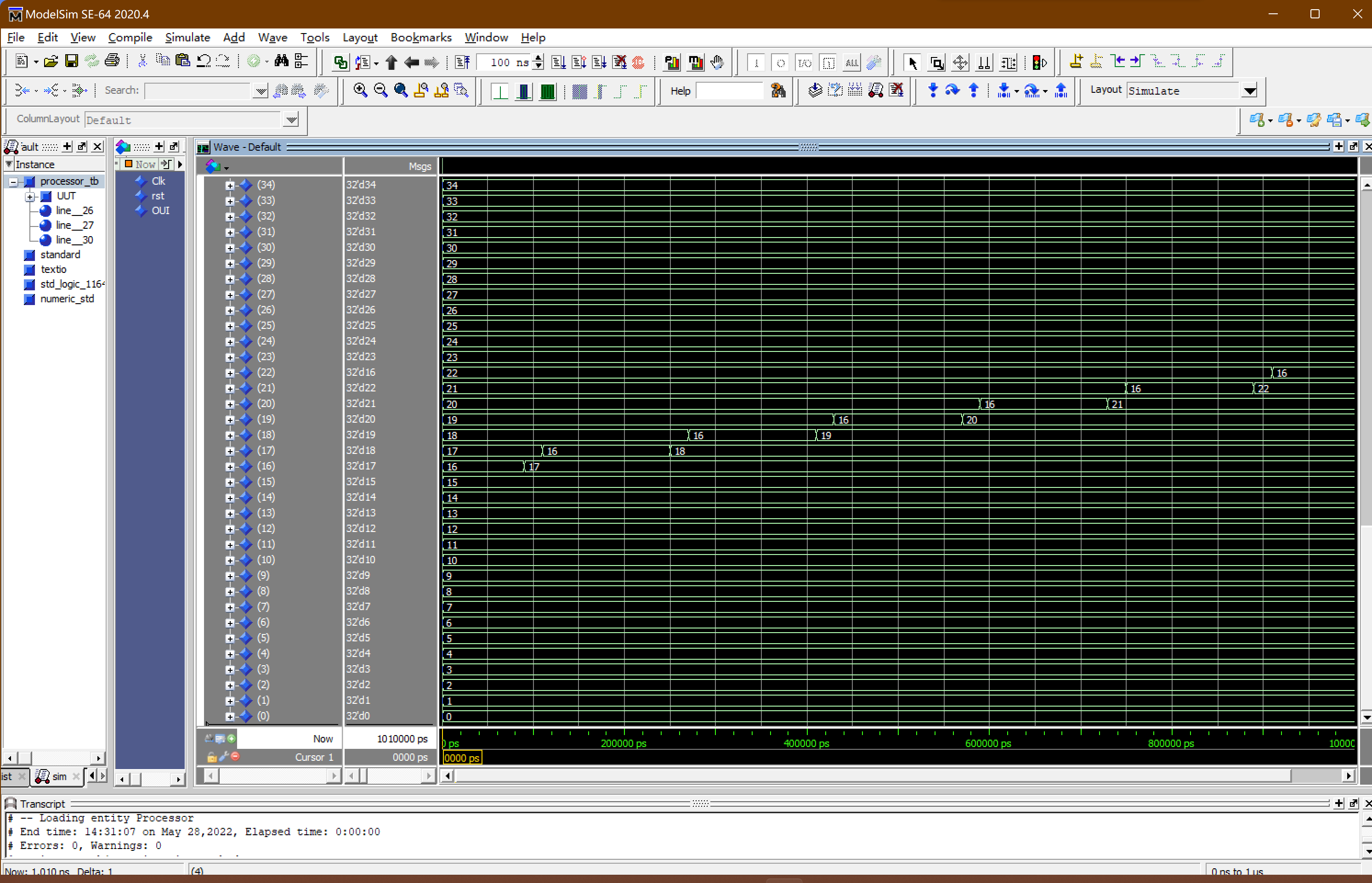


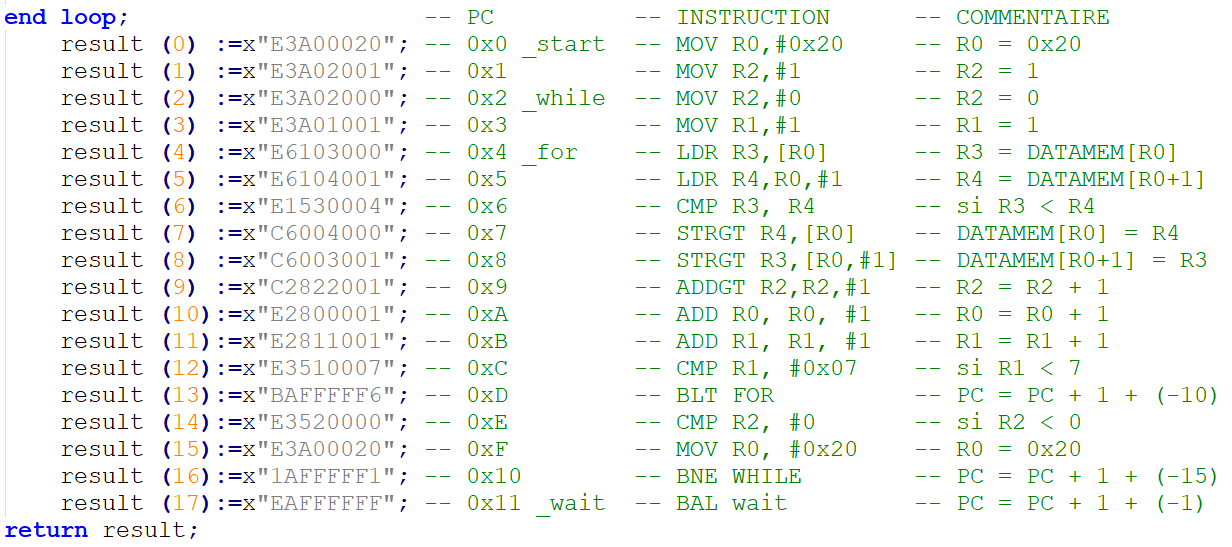
Figure 9 : Test Bench of Processor(LOAD, STORE with Offset)

By observing R(0), R(1), R(3), R(4) and Memory in the simulation results, the code in the test bench works well. The value of R(0), R(1), R(3), R(4) and Memory changes over time in accordance with the source code.

**Part 6 - Processor Test**

**- Test Bench of Processor（BNE、STRGT and ADDGT）**

By converting the bubble sort algorithm into binary code, the code in the instruction memory is modified to create a file named instruction\_memory3.vhd, and then the operation of the BNE, STRGT and ADDGT instruction is verified by simulation



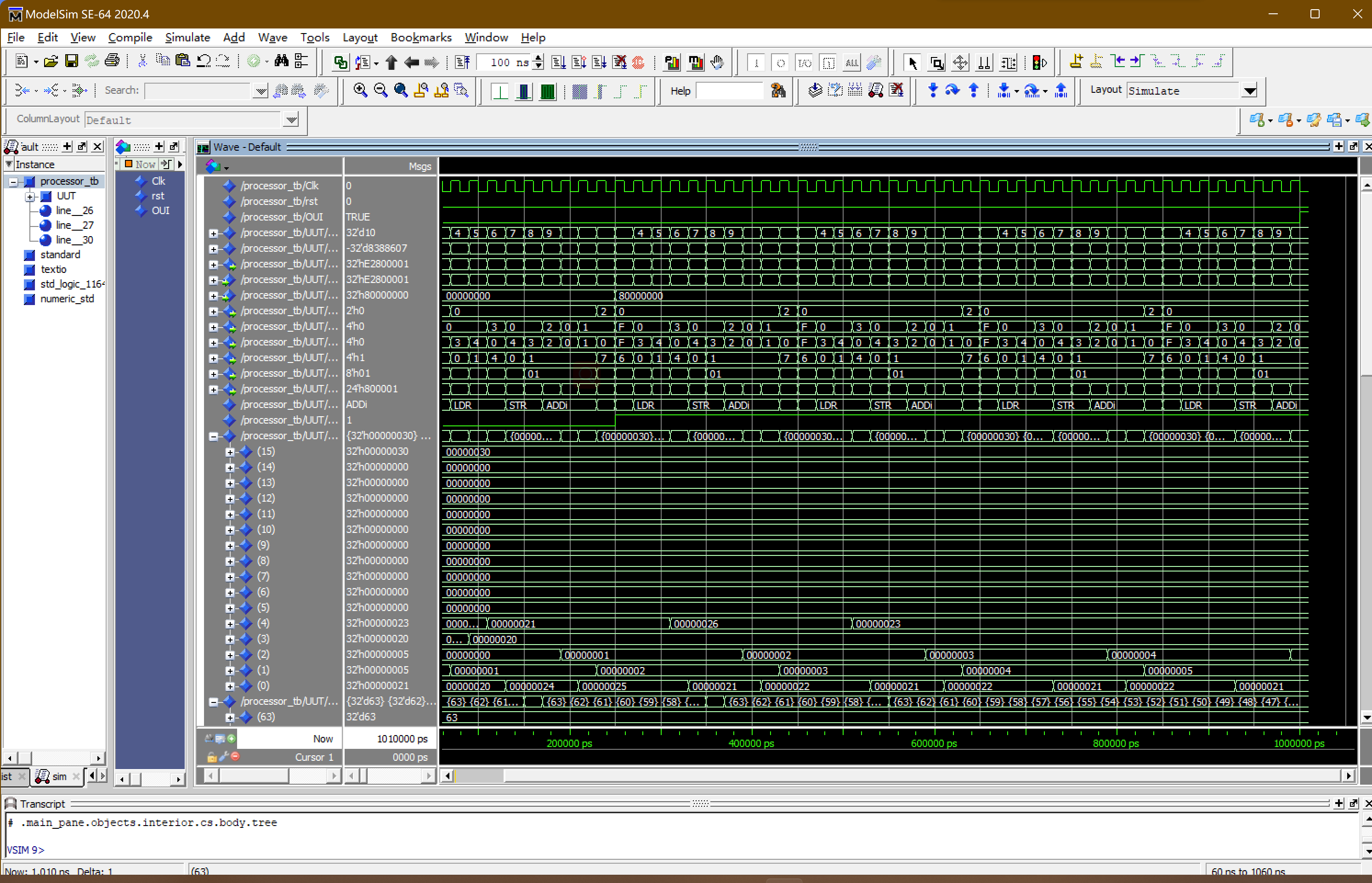


Figure 10 : Test Bench of Processor(BNE, STRGT with ADDGT)

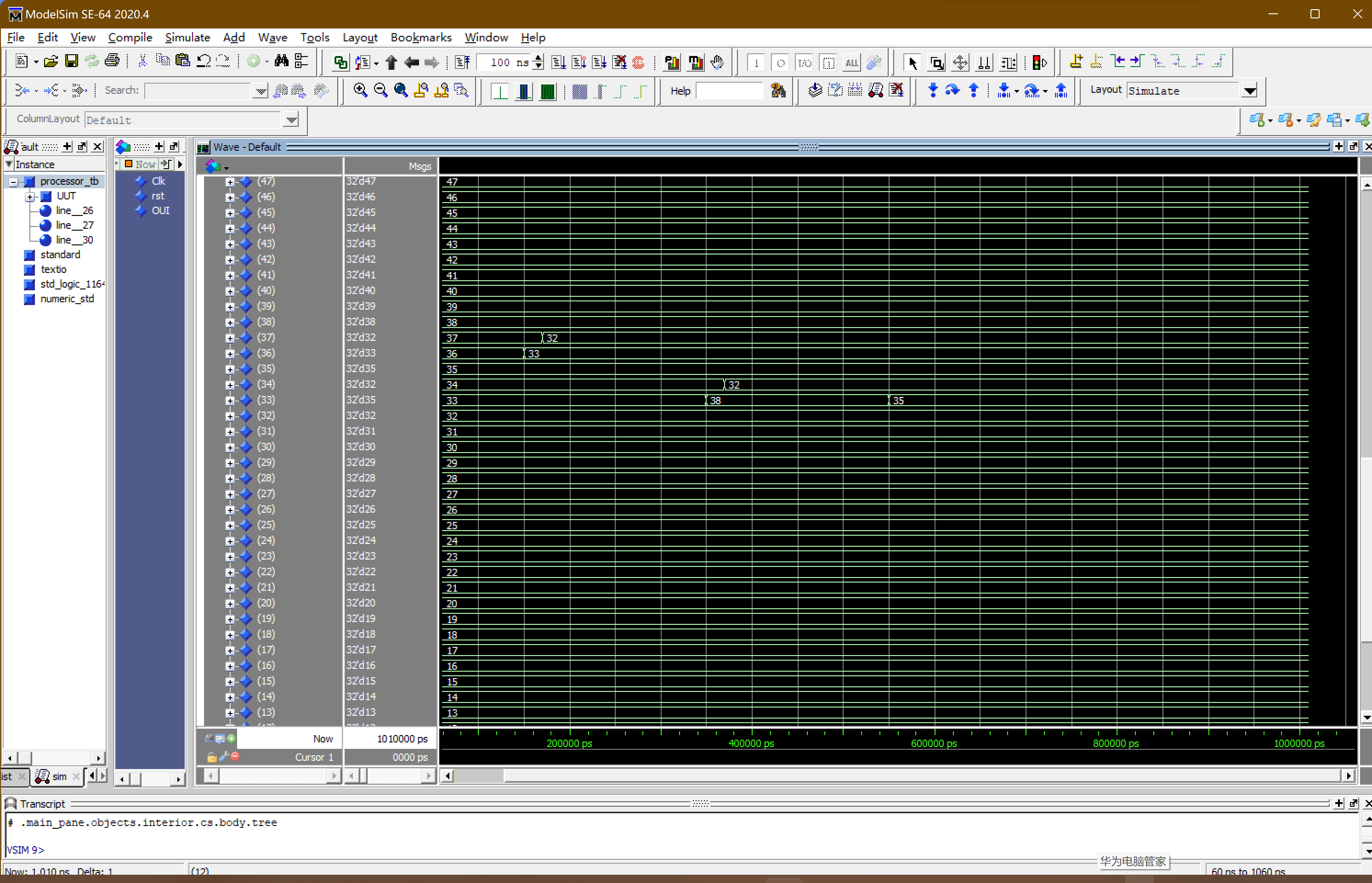


Figure 10 : Test Bench of Processor(BNE, STRGT with ADDGT)

By observing R(0), R(1), R(2), R(3), R(4) and Memory in the simulation results, the code in the test bench works well. The value of R(0), R(1), R(3), R(4) and Memory changes over time in accordance with the source code.

**In summary :**

The processor consists of three parts, namely, the Treatment Unit, the Instructions Fetch Unit and the Control Unit. Each unit consists of a number of different components. Through this project, I got familiar with and master the data transmission process inside the processor, and have a deeper understanding and application of 32-bit signal instructions. A complete processor needs not only each component to work correctly, but also the data paths between them. By completing this project, I not only made good use of the knowledge about VHDL and processor taught by the professor in class, but also mastered the application of Modelsim software.

Finally, I would like to thank you for your cultivation in this semester, which enabled me to learn a lot of knowledge about VHDL, Modelsim software and processor. Thank you!