Digital Circuits and Logic Design Assignment Report

Wang Anran 19022100074

Han Bingying 19022100071

Hello, Prof. DOUZE, Mr. Zongru.

We have completed all parts of our project in this semester's VHDL major assignment. Here is the report of our assignment.

PART 1——PROCESSING UNIT（UT)

We have written the *ALU.vhd* , and its functionality is complete and working. Then,we wrote the *Banc_de_registres.vhd .* Implementation of the register function.

Mission I.

1） Describe and simulate these modules in VHDL behavioral

We will put the simulation and the ALU in this part together.

2）Assemble UAL and the bench as in the diagram below to validate the processing unit.

We wrote the *Re_and_ALU.vhd* to link the register and ALU circuits as shown in the figure.

3) Write a test bench to validate by simulation the correct functioning of the following following operations:

-R(1) = R(15)

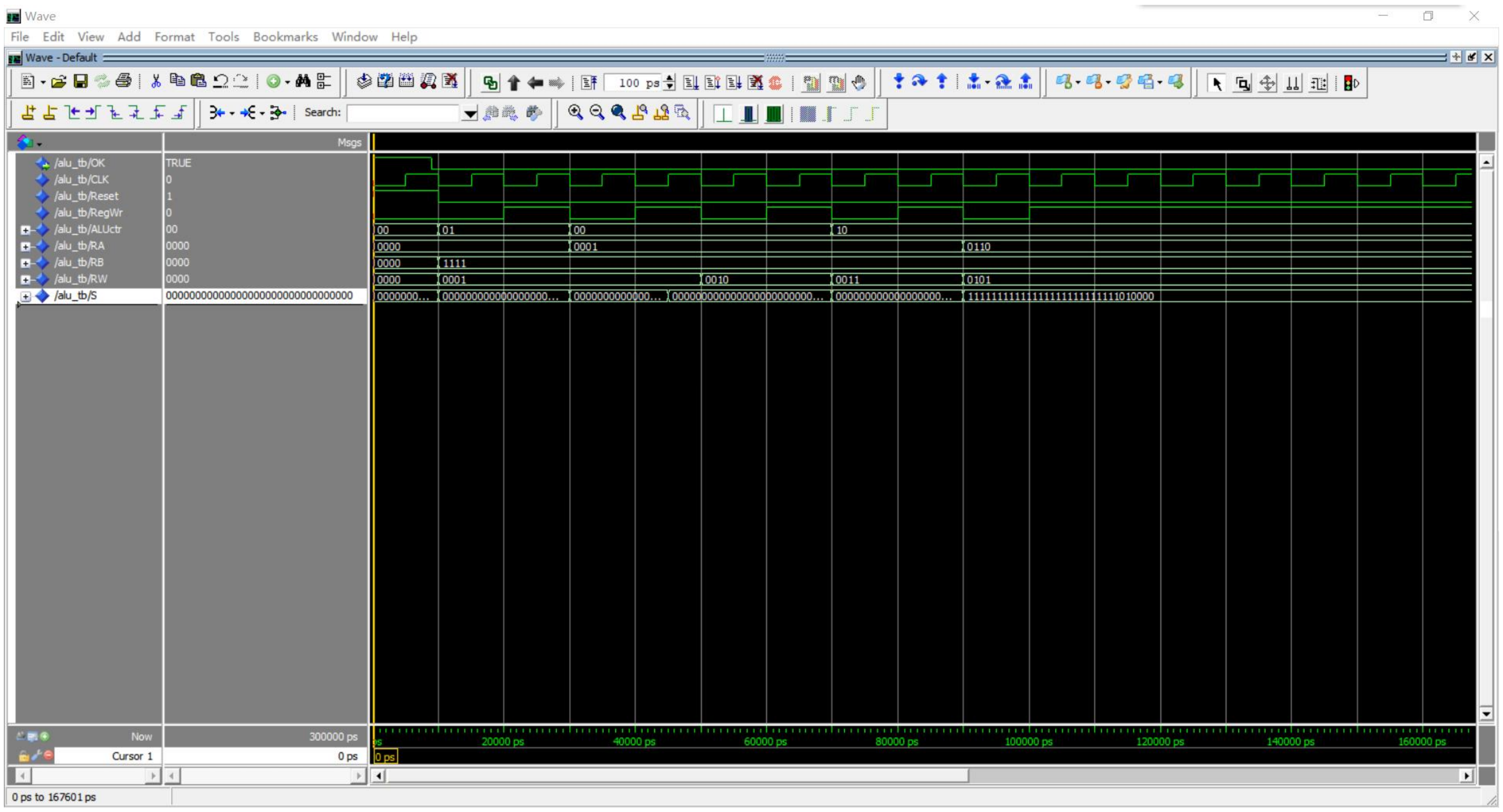
- R(1) = R(1) + R(15)

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

- R(3) = R(1) – R(15)

- R(5) = R(7) – R(15)

We wrote the *ALU_TB.vhd* for functional testing and the simulation results are as follows :


```
/alu_tb/OK      TRUE
/alu_tb/CLK     0
/alu_tb/Reset   1
/alu_tb/RegWr   0
/alu_tb/ALUctr  00    | 00 | 01 | 00 | 10
/alu_tb/RA      0000  | 0000 | 0001 | 0110
/alu_tb/RB      0000  | 0000 | 1111
/alu_tb/RW      0000  | 0000 | 0001 | 0010 | 0011 | 0101
/alu_tb/S       00000000000000000000000000000000  | 0000000... | 000000000000000000... | 0000000000000... | 0000000000000000000000000... | 000000000000000000... | 11111111111111111111111111010000
Now      300000 ps
Cursor 1 0 ps
0 ps to 167601 ps
```

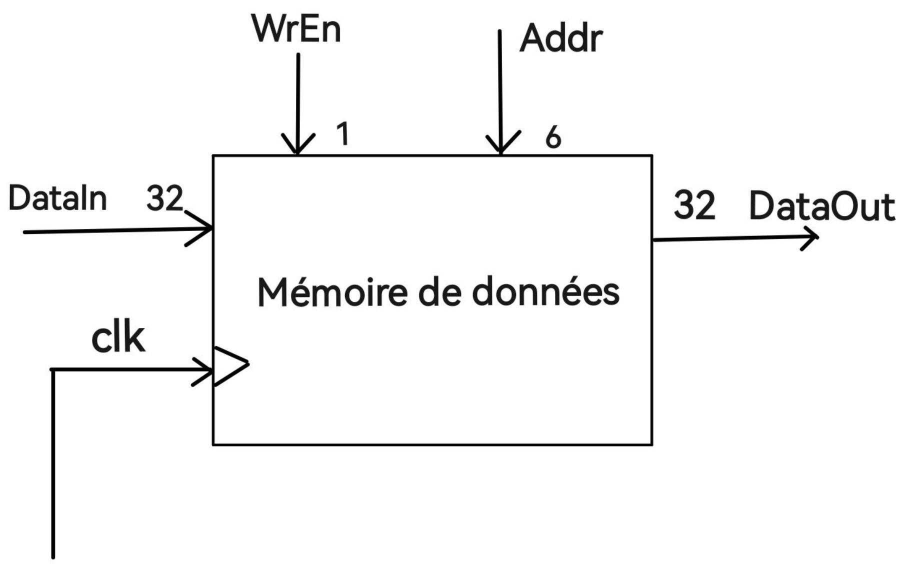
Next, we wrote the Multiplexeur2to1.vhd file to implement the multiplexer functionality.

We have written *SE.vhd* to implement the symbolic expansion function.

We have written DM.vhd toload and store 64 words of 32 bits.

Mission II.

1) Give the block diagram of these modules

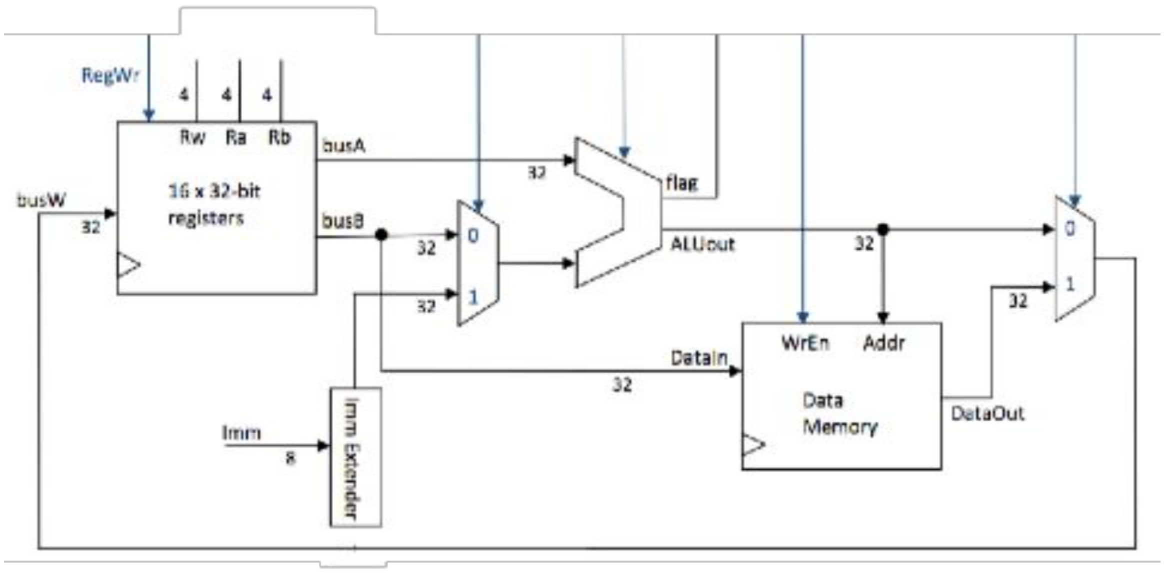


2) Describe and simulate these modules in VHDL behavioral

We put this part of the simulation into the simulation of the processing unit of the final component processor for presentation.

Mission III.

1) Write a VHDL module that performs the assembly of the processing unit.



Based on the diagram given in the topic, we wrote *UT.vhd* to link the whole circuit.

2) Write a testbench to validate by simulation the correct operation of :

- The addition of 2 registers - The addition of 1 register with an immediate value

- The subtraction of 2 registers

- The subtraction of 1 immediate value to 1 register

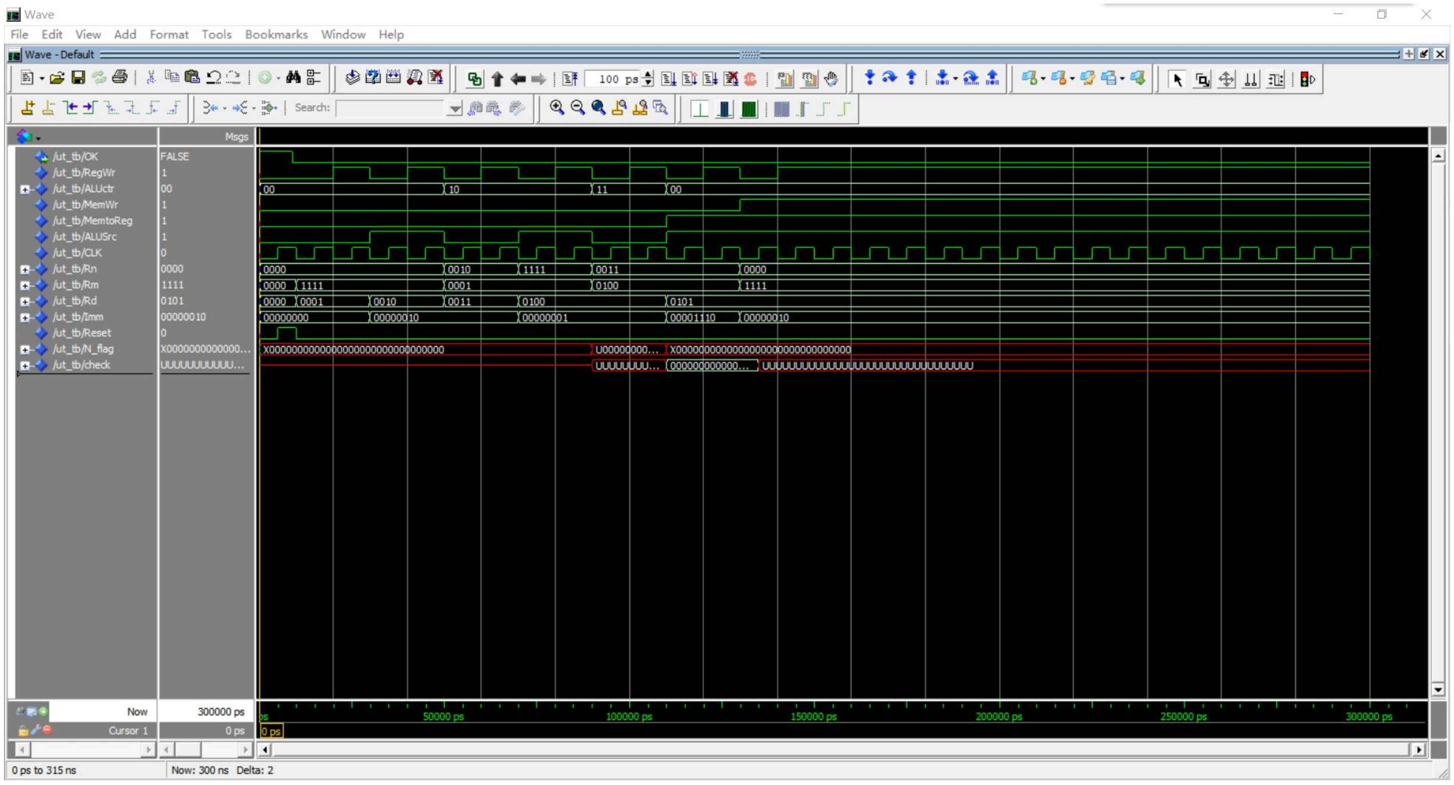
- The copy of the value of a register in another register

- The writing of a register in a word of the memory.

- The reading of a word of the memory in a register.

We have written *UT_tb.vhd* to simulate whether the above items will work properly.

The simulation results are shown in the figure below.



PART 2 - INSTRUCTION MANAGEMENT UNIT

To implement this section, we divided it into separate VHDL code for the following modules:

1. *PC_R.vhd* : A 32-bit register (PC register).
2. *PC_E.vhd* : An extension unit from 24 to 32 signed bits .
3. *instruction_memory.vhd* : A 64-word, 32-bit instruction memory similar to that of the processing unit.
4. *ADD1.vhd* : If nPCsel = 0 , PC = PC + 1;

5. *ADD2.vhd* : If nPCsel = 1 , PC = PC + 1 + SignExt(offset)

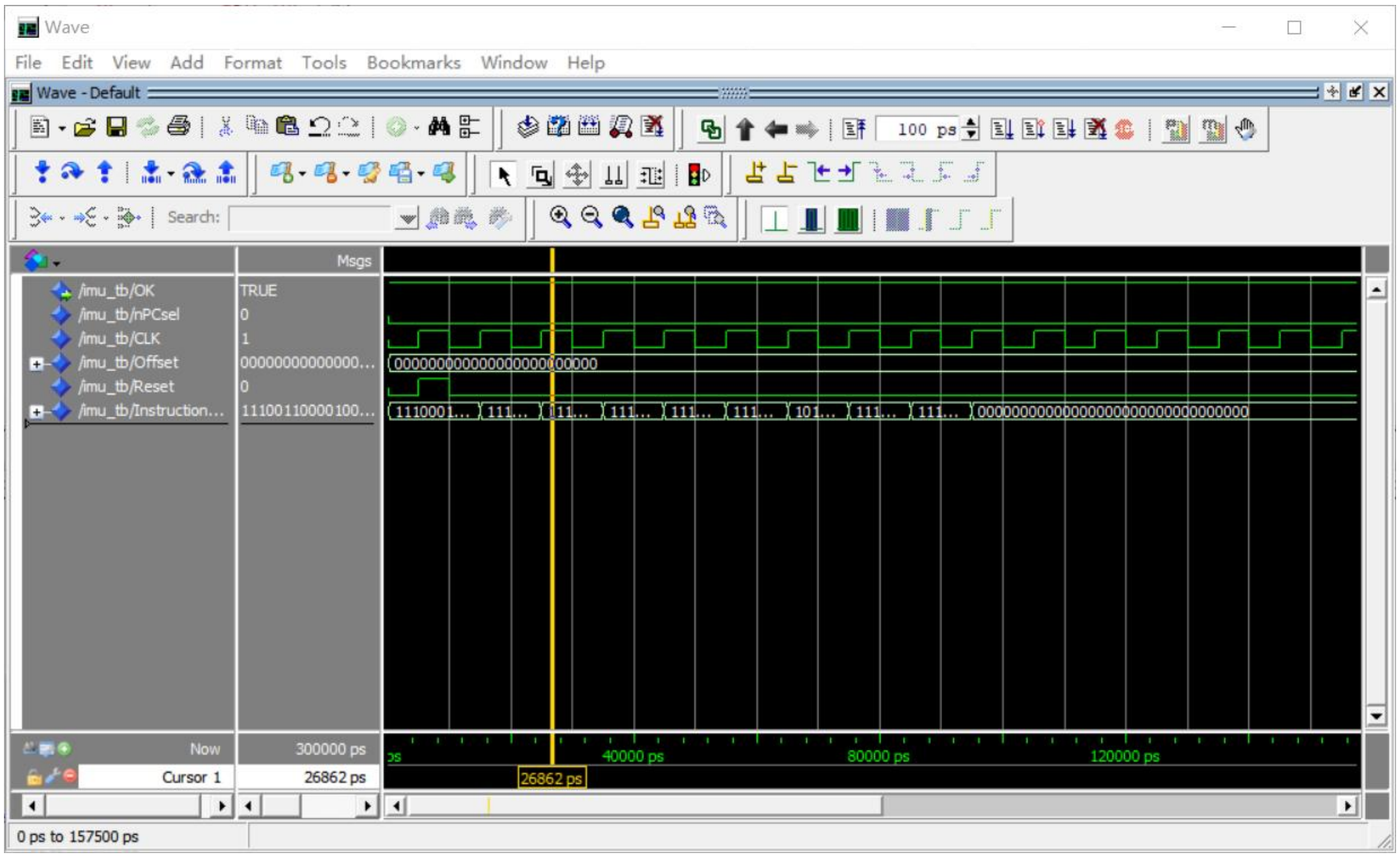
6. *MUL2to1.vhd*

Finally, we wrote the *IMU.vhd* file to connect the above circuits together for simulation.

Mission

1) Describe and simulate the instruction management unit

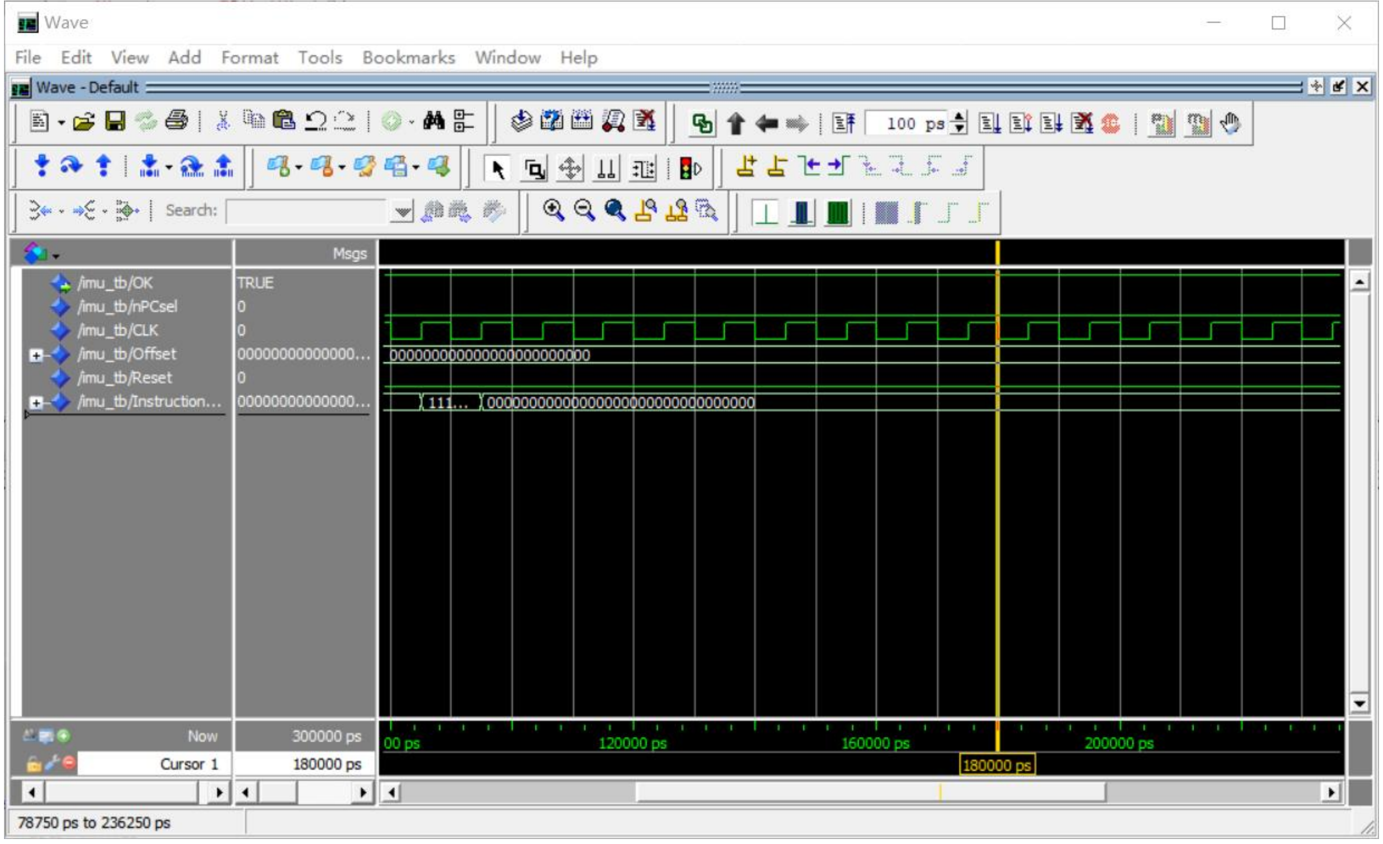
We wrote the *IMU_tb.vhd* file for testing and the results are shown in the figure below.



	Msgs
/imu_tb/OK	TRUE
/imu_tb/nPCsel	0
/imu_tb/CLK	1
/imu_tb/Offset	00000000000000...
/imu_tb/Reset	0
/imu_tb/Instruction...	11100110000100...

Now	300000 ps
Cursor 1	26862 ps

0 ps to 157500 ps



```
/imu_tb/OK              TRUE
/imu_tb/nPCsel          0
/imu_tb/CLK             0
/imu_tb/Offset          00000000000000...   00000000000000000000000
/imu_tb/Reset           0
/imu_tb/Instruction...  00000000000000...   111...  00000000000000000000000000000000

Now       300000 ps
Cursor 1  180000 ps
78750 ps to 236250 ps
```

PART 3 - CONTROL UNIT

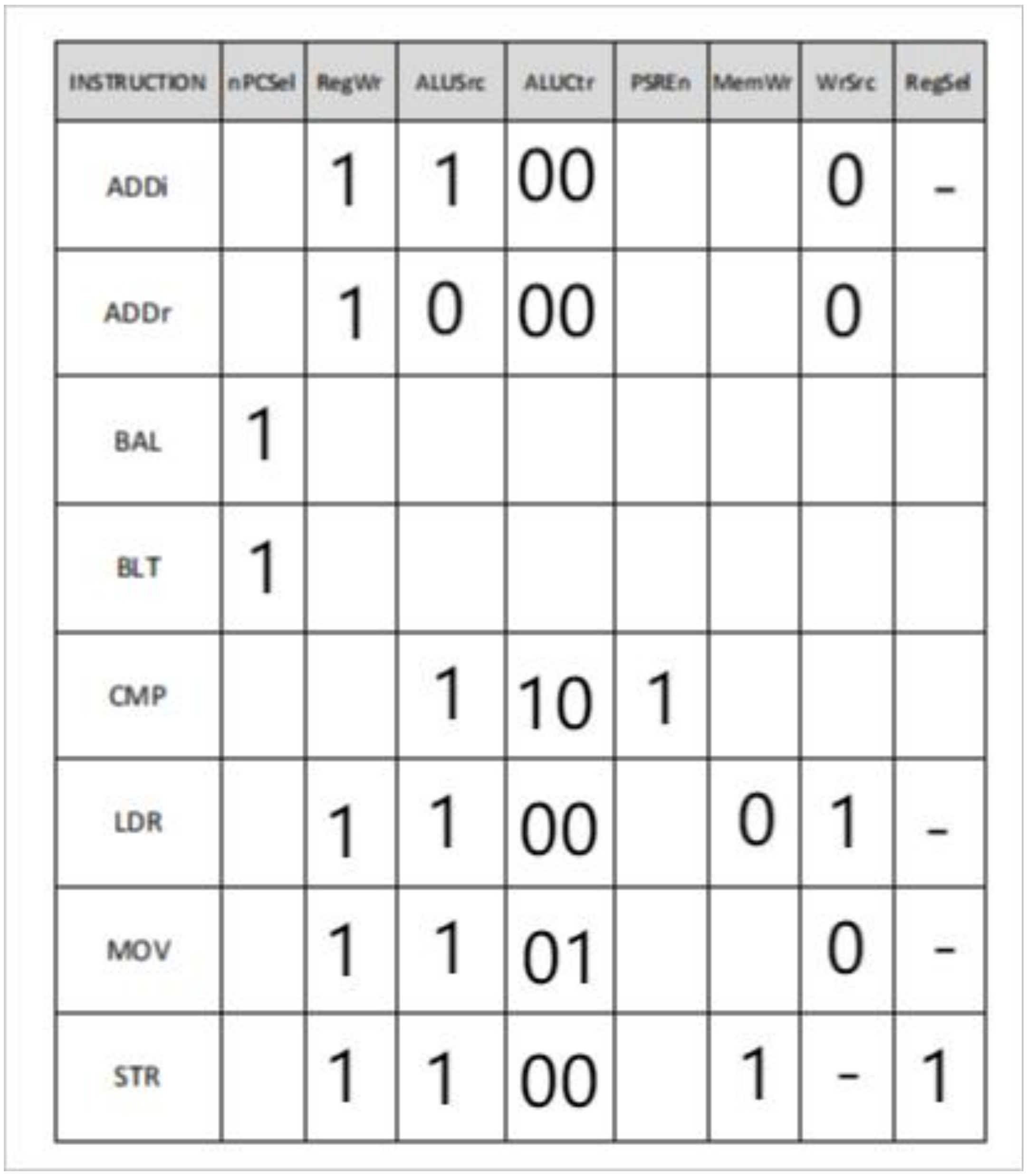
According to the requirements of the topic, we wrote the following VHDL file:

1. *PSR.vhd* : 32 bit register with load control;If WE=1, the register stores the value placed on the DATAIN bus. If WE=0, the register keeps its previous value.

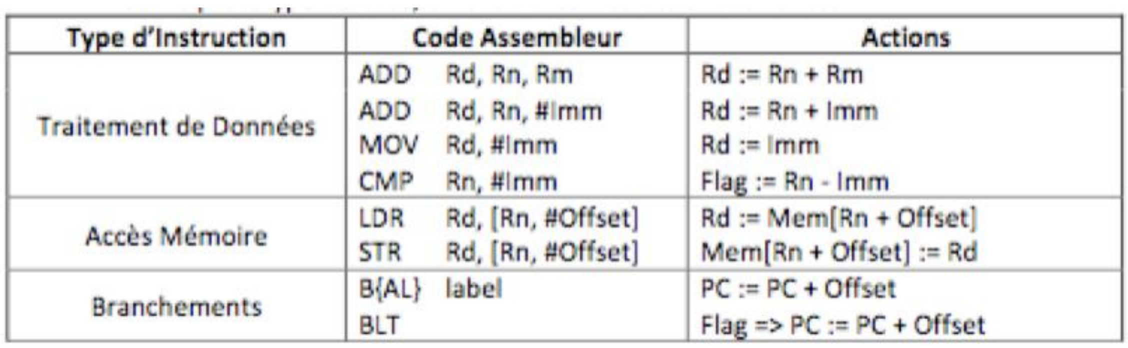
2. *Instruction_Decoder.vhd* : This combinatorial module generates the control signals for the processing unit, the instruction the instruction management unit, as well as the PSR register, all described previously described.

Mission

1) Complete the table "command values" in the attachment, which will summarize the actions of the decoder according to the instructions.



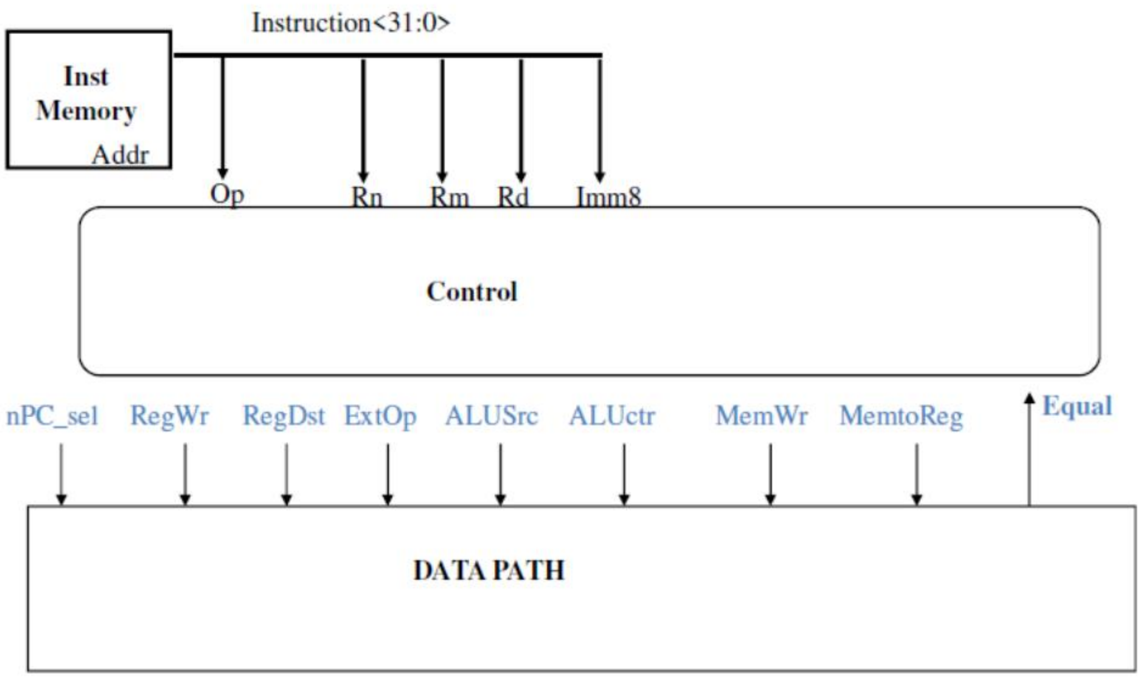
INSTRUCTION	nPCSel	RegWr	ALUSrc	ALUCtr	PSREn	MemWr	WrSrc	RegSel
ADDi		1	1	00			0	-
ADDr		1	0	00			0	
BAL	1							
BLT	1							
CMP			1	10	1			
LDR		1	1	00		0	1	-
MOV		1	1	01			0	-
STR		1	1	00		1	-	1



Type d'Instruction	Code Assembleur	Actions
Traitement de Données	ADD Rd, Rn, Rm	Rd := Rn + Rm
	ADD Rd, Rn, #Imm	Rd := Rn + Imm
	MOV Rd, #Imm	Rd := Imm
	CMP Rn, #Imm	Flag := Rn - Imm
Accès Mémoire	LDR Rd, [Rn, #Offset]	Rd := Mem[Rn + Offset]
	STR Rd, [Rn, #Offset]	Mem[Rn + Offset] := Rd
Branchements	B{AL} label	PC := PC + Offset
	BLT	Flag => PC := PC + Offset

2) Describe in VHDL the two modules of the control unit. The 32-bit PSR register, if it receives a load command, will loading, will acquire the N flag of the ALU on its low weight, and 31 bits to 0 on its high weights.

We have written the *CU.vhd* file to implement the functionality described in the topic. This file contains the PC registers and decoders written previously. We use the following image to illustrate the functionality of the instructions executed by this code.



PART 4 - ASSEMBLY AND VALIDATION OF THE PROCESSOR

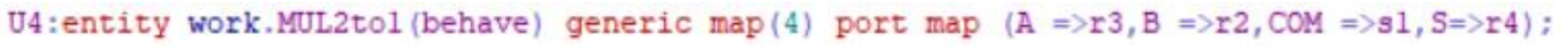
We must now finalize the modeling of the processor by assembling its three main units

- The instruction management unit
- The processing unit
- The control unit

Misson

1) Complete the previously designed processing unit by adding a multiplexer with 2 inputs on 4 bits driven by the control signal RegSel control signal generated by the control unit. This multiplexer will be placed at the input of the address address of the register bank, as it is represented on the diagram of the processor in the appendix.

This part we have embodied in the code.



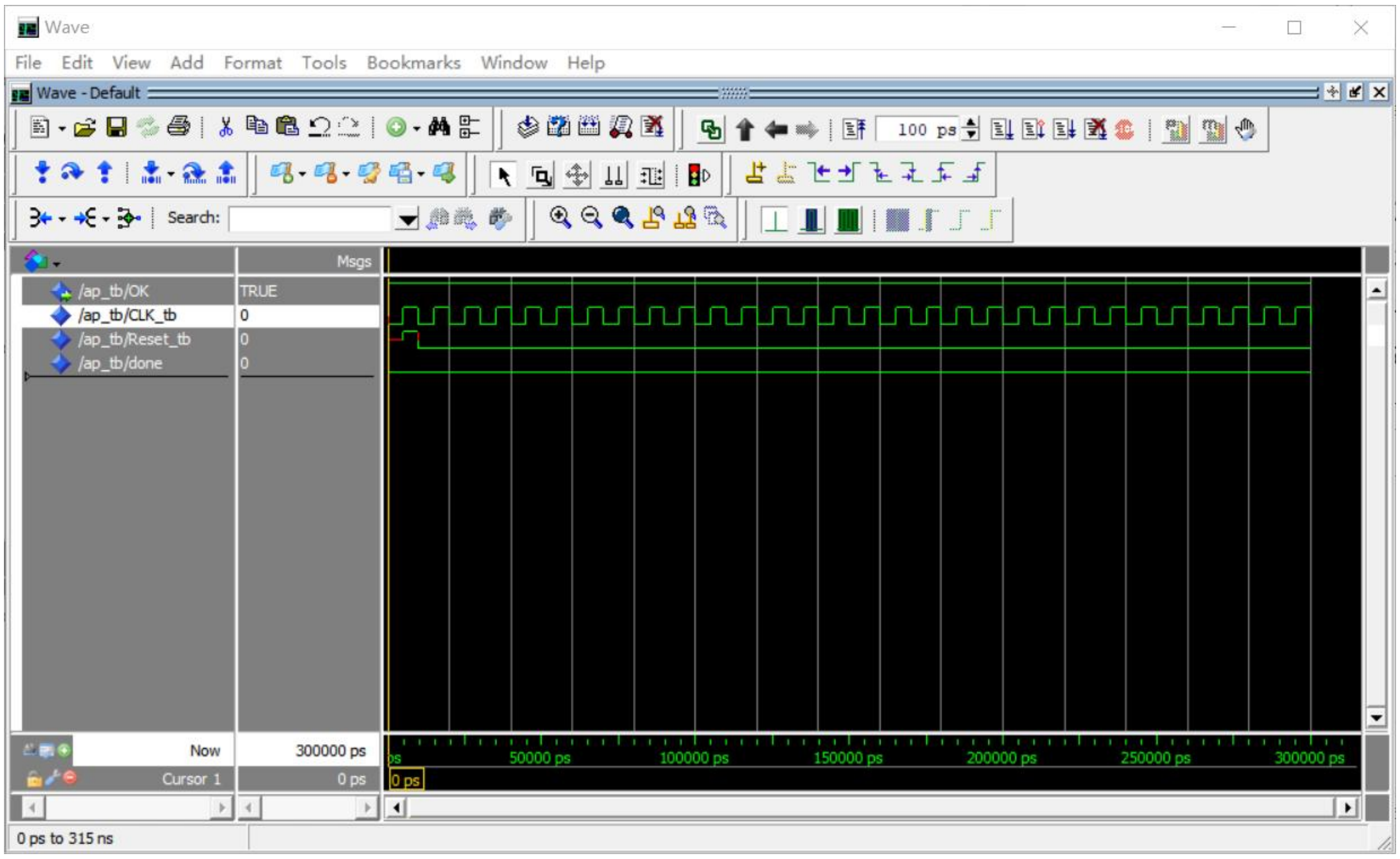
```
U4:entity work.MUL2to1(behave) generic map(4) port map (A =>r3,B =>r2,COM =>s1,S=>r4);
```

2) Assemble the processor from its three units.

We wrote the *AP.vhd* file to link the first three parts together.

3) Simulate the execution of the test program by the processor and verify its correct operation. It may be necessary to initialize some data in the Data Memory (from 0x20 to 0x2A)

We wrote the *AP_tb.vhd* file to perform the test, and the output simulation waveform is shown below.



The overall circuit is shown in the following figure.

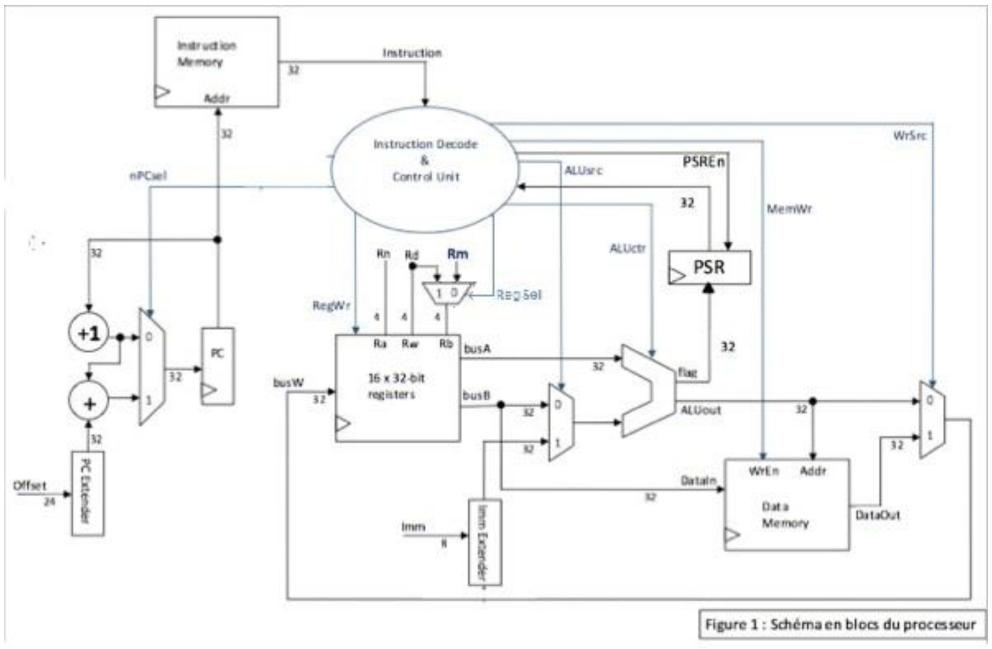


Figure 1 : Schéma en blocs du processeur

PART 5 - COMPLETE PROCESSOR TEST

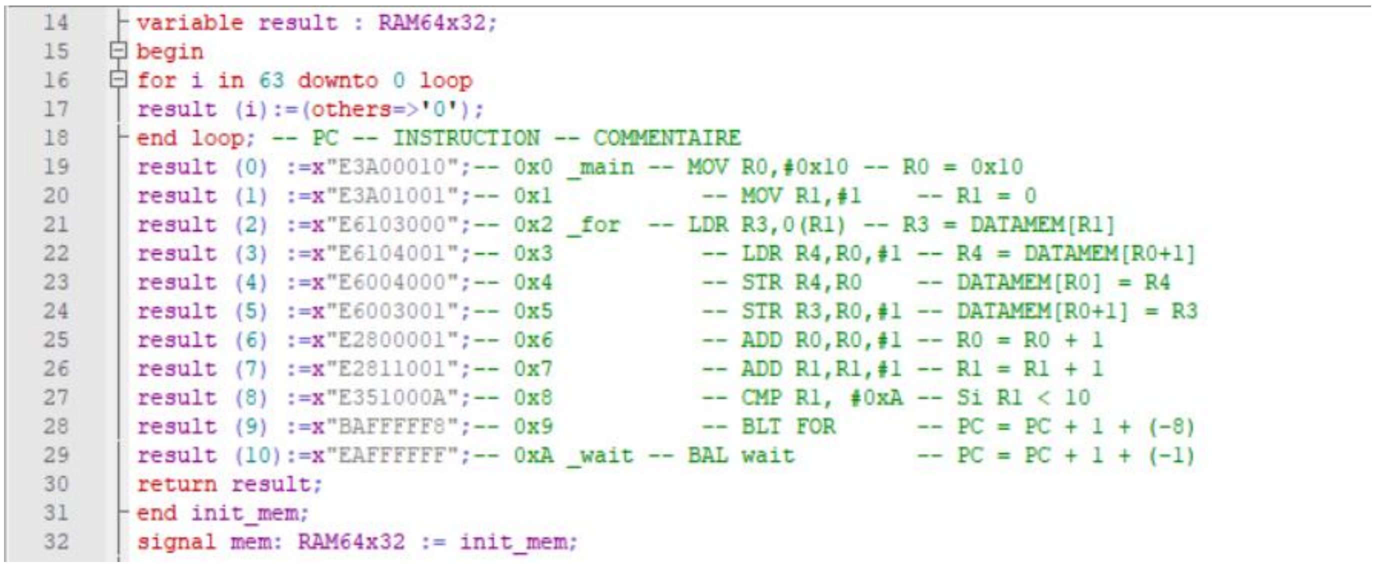
Misson

1) Find the binary code of this small program from the operating code of the code of the ARM7TDMI instructions given in the Appendix

2) Modify the VHDL code of the instruction memory by creating a new component called *instruction_memory2.vhd*

3) Simulate the execution of this second test program by the processor and check its correct operation. It will be necessary to initialize some data in the Data Me mory (from 0x20 to 0x2A)

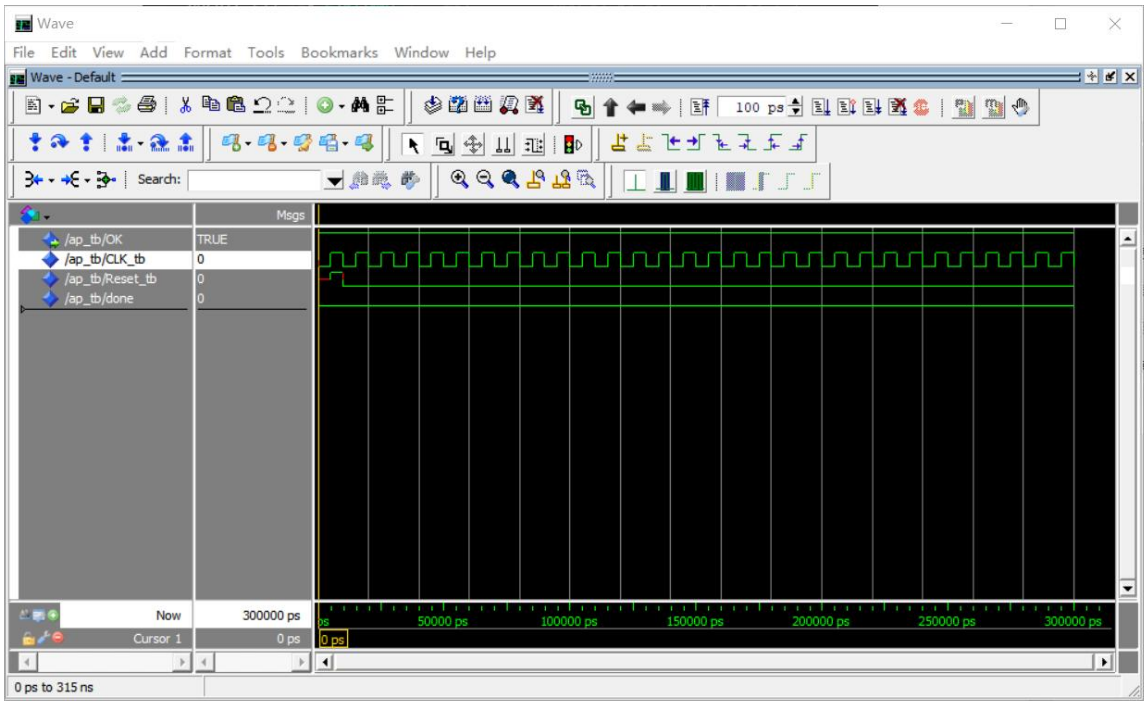
The modified code is as follows.



```
variable result : RAM64x32;
begin
for i in 63 downto 0 loop
result (i):=(others=>'0');
end loop; -- PC -- INSTRUCTION -- COMMENTAIRE
result (0) :=x"E3A00010";-- 0x0 _main -- MOV R0,#0x10 -- R0 = 0x10
result (1) :=x"E3A01001";-- 0x1          -- MOV R1,#1    -- R1 = 0
result (2) :=x"E6103000";-- 0x2 _for  -- LDR R3,0(R1) -- R3 = DATAMEM[R1]
result (3) :=x"E6104001";-- 0x3          -- LDR R4,R0,#1 -- R4 = DATAMEM[R0+1]
result (4) :=x"E6004000";-- 0x4          -- STR R4,R0    -- DATAMEM[R0] = R4
result (5) :=x"E6003001";-- 0x5          -- STR R3,R0,#1 -- DATAMEM[R0+1] = R3
result (6) :=x"E2800001";-- 0x6          -- ADD R0,R0,#1 -- R0 = R0 + 1
result (7) :=x"E2811001";-- 0x7          -- ADD R1,R1,#1 -- R1 = R1 + 1
result (8) :=x"E351000A";-- 0x8          -- CMP R1, #0xA -- Si R1 < 10
result (9) :=x"BAFFFFF8";-- 0x9          -- BLT FOR      -- PC = PC + 1 + (-8)
result (10):=x"EAFFFFFF";-- 0xA _wait -- BAL wait        -- PC = PC + 1 + (-1)
return result;
end init_mem;
signal mem: RAM64x32 := init_mem;
```

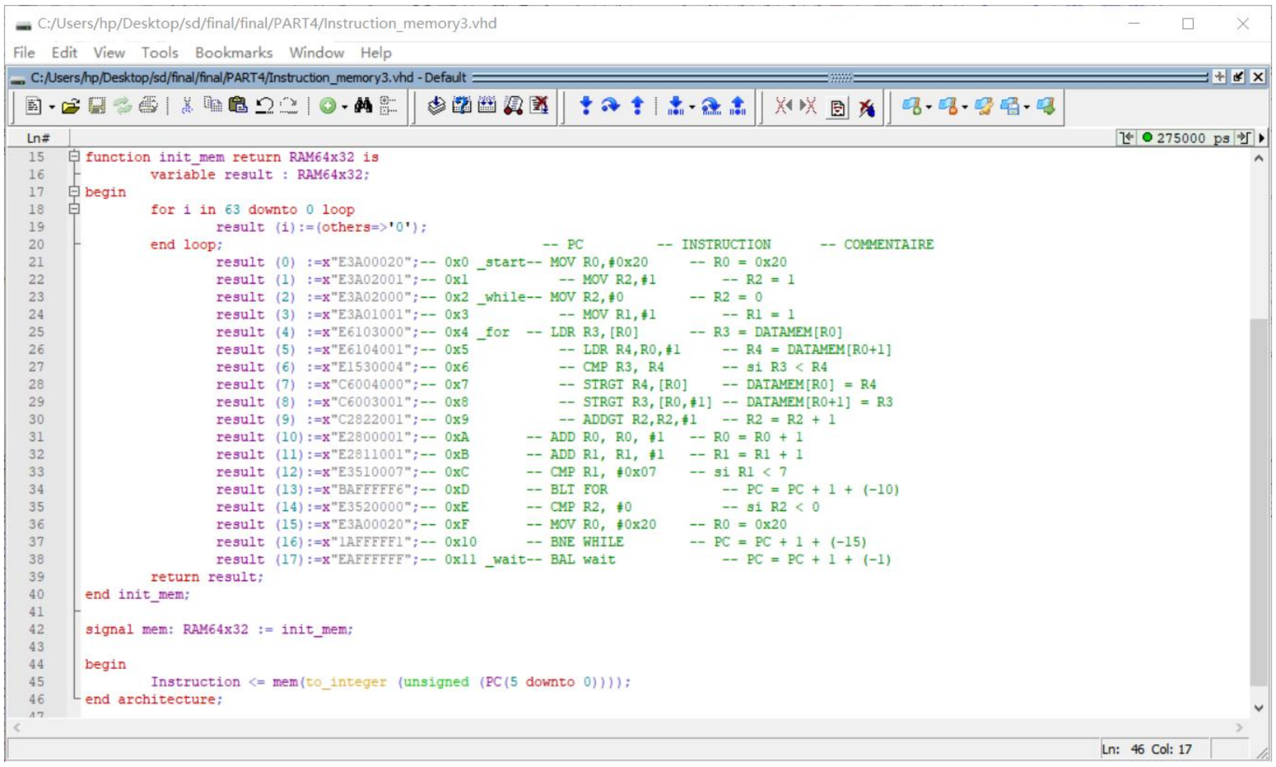
The simulation diagram is as

follows.



PART 6 - INCREASING THE INSTRUCTION SET

The modified code is as follows.

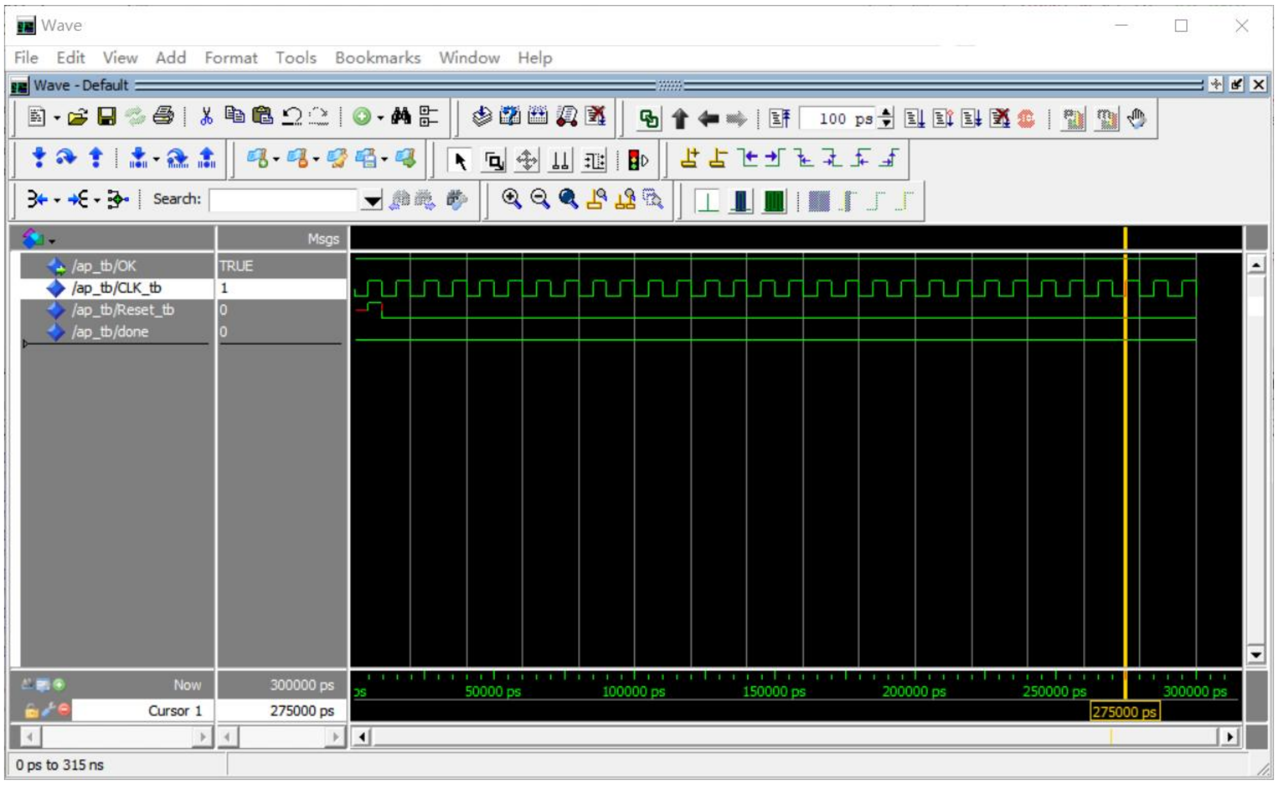

C:/Users/hp/Desktop/sd/final/final/PART4/Instruction_memory3.vhd
File Edit View Tools Bookmarks Window Help
C:/Users/hp/Desktop/sd/final/final/PART4/Instruction_memory3.vhd - Default
275000 ps
Ln: 46 Col: 17

```vhdl
function init_mem return RAM64x32 is
        variable result : RAM64x32;
begin
        for i in 63 downto 0 loop
                result (i):=(others=>'0');
        end loop;                                       -- PC         -- INSTRUCTION      -- COMMENTAIRE
                result (0) :=x"E3A00020";-- 0x0 _start-- MOV R0,#0x20      -- R0 = 0x20
                result (1) :=x"E3A02001";-- 0x1           -- MOV R2,#1        -- R2 = 1
                result (2) :=x"E3A02000";-- 0x2 _while-- MOV R2,#0         -- R2 = 0
                result (3) :=x"E3A01001";-- 0x3           -- MOV R1,#1        -- R1 = 1
                result (4) :=x"E6103000";-- 0x4 _for  -- LDR R3,[R0]       -- R3 = DATAMEM[R0]
                result (5) :=x"E6104001";-- 0x5           -- LDR R4,R0,#1     -- R4 = DATAMEM[R0+1]
                result (6) :=x"E1530004";-- 0x6           -- CMP R3, R4       -- si R3 < R4
                result (7) :=x"C6004000";-- 0x7           -- STRGT R4,[R0]    -- DATAMEM[R0] = R4
                result (8) :=x"C6003001";-- 0x8           -- STRGT R3,[R0,#1] -- DATAMEM[R0+1] = R3
                result (9) :=x"C2822001";-- 0x9           -- ADDGT R2,R2,#1   -- R2 = R2 + 1
                result (10):=x"E2800001";-- 0xA       -- ADD R0, R0, #1   -- R0 = R0 + 1
                result (11):=x"E2811001";-- 0xB       -- ADD R1, R1, #1   -- R1 = R1 + 1
                result (12):=x"E3510007";-- 0xC       -- CMP R1, #0x07    -- si R1 < 7
                result (13):=x"BAFFFFF6";-- 0xD       -- BLT FOR              -- PC = PC + 1 + (-10)
                result (14):=x"E3520000";-- 0xE       -- CMP R2, #0           -- si R2 < 0
                result (15):=x"E3A00020";-- 0xF       -- MOV R0, #0x20    -- R0 = 0x20
                result (16):=x"1AFFFFF1";-- 0x10      -- BNE WHILE        -- PC = PC + 1 + (-15)
                result (17):=x"EAFFFFFF";-- 0x11 _wait-- BAL wait             -- PC = PC + 1 + (-1)
        return result;
end init_mem;

signal mem: RAM64x32 := init_mem;

begin
        Instruction <= mem(to_integer (unsigned (PC(5 downto 0))));
end architecture;
```

The simulation diagram is as follows.



```
/ap_tb/OK        TRUE
/ap_tb/CLK_tb    1
/ap_tb/Reset_tb  0
/ap_tb/done      0
Now       300000 ps
Cursor 1  275000 ps
0 ps to 315 ns
```

Summary

Through this project, I was able to build a unicycle processor thanks to three sub-units, themselves composed of sub-blocks. Thanks to the increase of the instruction set, the processor was able to realize a sorting algorithm. To make it even more efficient and able to accommodate a larger number of programs, we could add more instructions.