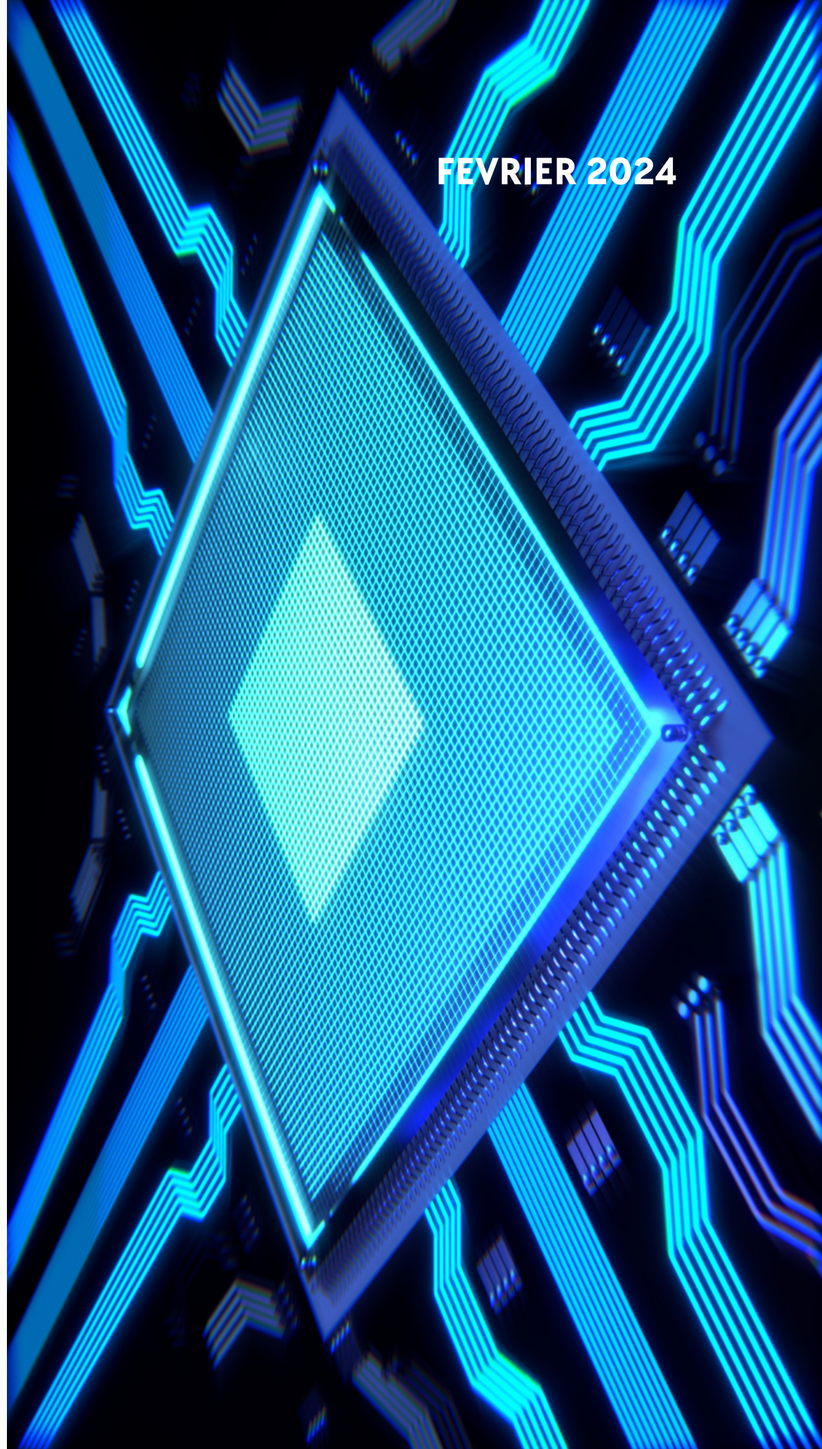
ARCHITECTURE DES SYSTÈMES NUMÉRIQUES

CONCEPTION RISC

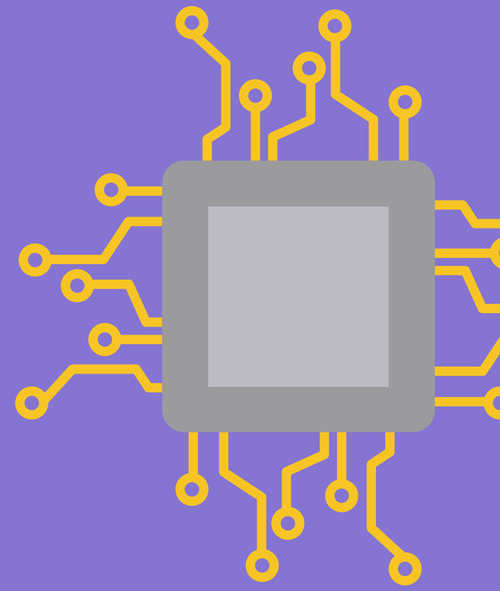
LOUIS VAUTERIN & HUGUES D'HARDEMARE



FEVRIER 2024

SOMMAIRE

OBJECTIFS & CAHIER DES CHARGES



Objectif : Concevoir un microprocesseur de type RISC 4 bits

Conception d’un microprocesseur simple pouvant réaliser des opérations basiques.

Instruction Set Architecture

Format 1:

Instruction de taille 16 bits

Format (2 bit)	Op (3bits)	Rdest (3bits)	RB (3bits)	RA (3bits)	Mnemonic	Function
00	000	RID	RID	RID	ADD	Rdest=RB+RA
00	001	RID	RID	RID	SUB	Rdest=RB-RA
00	010	RID	RID	RID	RSHIFT	Rdest = RB>> f(RA) avec f(RA) = i si RA= i
00	011	RID	RID	RID	LSHIFT	Rdest = RB << f(RA) avec f(RA) = i si RA= i
00	100	RID	RID	RID	AND	Rdest = RA AND RB bit à bit
00	101	RID	RID	RID	OR	Rdest = RA OR RB bit à bit
00	110	RID	RID	NA	MOV	Rdest=RB

Instruction Set Architecture

Format 2:

Format (2 bit)	Op (3bits)	Rdest (3bits)	RB (3bits)	Imm (5 bits)	Mnemonic	Function
01	000	RID	RID	Imm	ADD	Rdest=RB+Imm
01	001	RID	RID	Imm	SUB	Rdest=RB-Imm
01	010	RID	RID	Imm	RSHIFT	Rdest = RB>> f(Imm) avec f(Imm) = i si Imm= i
01	011	RID	RID	Imm	LSHIFT	Rdest = RB << f(Imm) avec f(Imm) = i si Imm= i
01	100	RID	RID	Imm	AND	Rdest = Imm AND RB bit à bit
01	101	RID	RID	Imm	OR	Rdest = Imm OR RB bit à bit
01	110	RID	RID	Imm	MOV	Rdest=Imm

Instruction Set Architecture

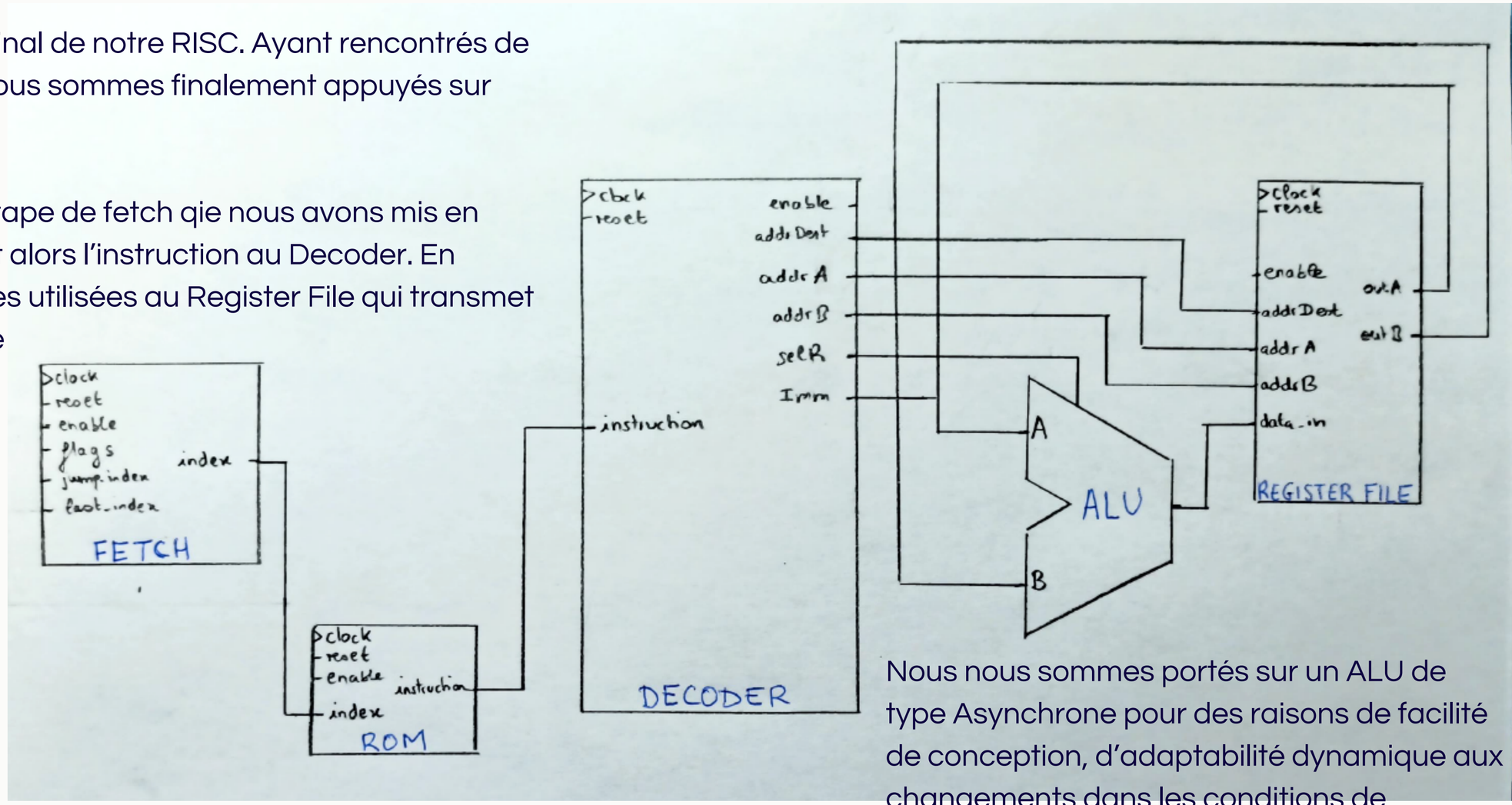
Format 3:

Format (2 bit)	Op (3bits)	Rdest (3bits)	Imm (5 bits)	Mnemonic	Function
10	000	RID	Imm	MOV	Rdest=Imm
10	001	RID	Imm	NJMP	PC=Imm if N=1

Schéma architectural

Voici le schéma architectural final de notre RISC. Ayant rencontrés de nombreuses difficultés, nous nous sommes finalement appuyés sur une architecture simple.

Nous avons commencé par l'étape de fetch qie nous avons mis en série avec la ROM qui transmet alors l'instruction au Decoder. En décodant, il indique les adresses utilisées au Register File qui transmet les valeurs des registres pour le calcul de l'ALU dont le résultat sera stocké dans RDest.
Choix d'un fonctionnement asynchrone pour l'ALU.



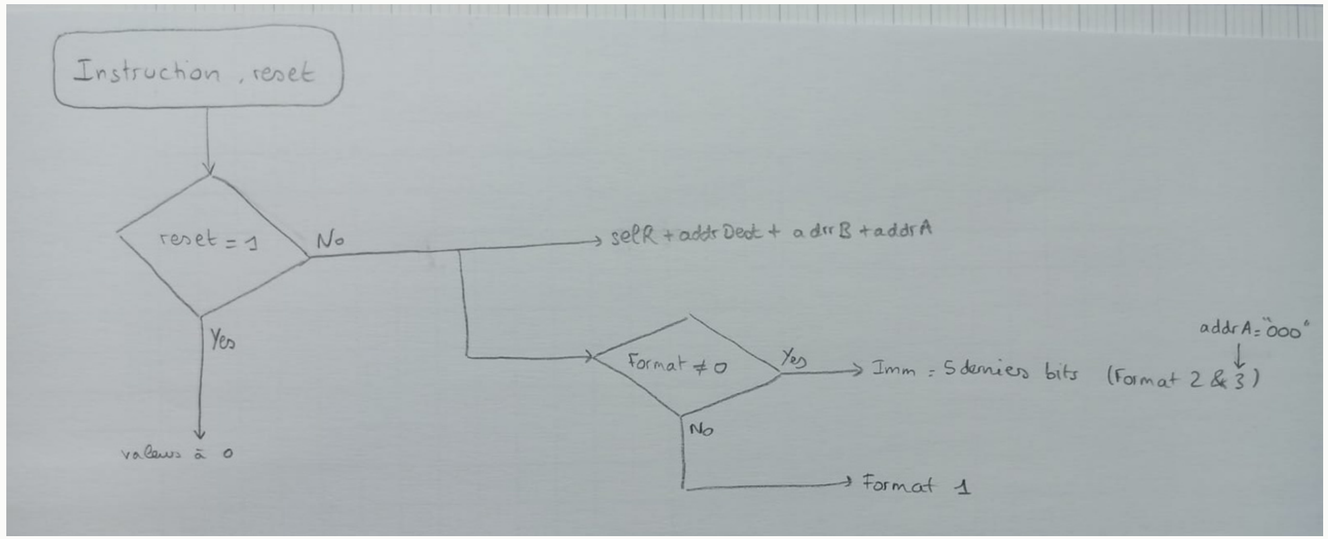
Nous nous sommes portés sur un ALU de type Asynchrone pour des raisons de facilité de conception, d'adaptabilité dynamique aux changements dans les conditions de fonctionnement sans dépendre d'une horloge fixe ainsi que pour réduire les problèmes de synchronisation.

Algorigramme du decoder

```
begin
    process(clock, reset)
      begin
      if reset = '1' then
        opcode <= (others => '0');
        selR <= (others => '0');
        addrDest <= (others => '0');
        addrA <= (others => '0');
        addrB <= (others => '0');
        imm <= (others => '0');
      elsif rising_edge(clock) then

      -- decode instruction
      opcode <= instruction(15 downto 11);
      selR <= instruction(15 downto 11);
      addrDest <= instruction(10 downto 8);
      addrA <= instruction(4 downto 2);
      addrB <= instruction(7 downto 5);
      if instruction(15 downto 14) /= "00" then
      imm <= instruction(4 downto 0);
      end if;
    end if;
    end process;
```

Algorigramme du decoder



Le décodeur est utilisé pour déchiffrer les instructions du programme. Il analyse le code machine ou op-code et active les différentes parties du processeur en fonction de l'instruction.

Exemple de code

Opérations

A = 10
B = 5
C =8
D = A + B
E = C -3
A = D & E
B = D || E
C = B // 2

Assembler

MOV 000 10
MOV 001 5
MOV 010 8
ADD 100 001 000
SUB 101 010 3
AND 000 100 101
OR 001 100 101
RSHIFT 010 001 1

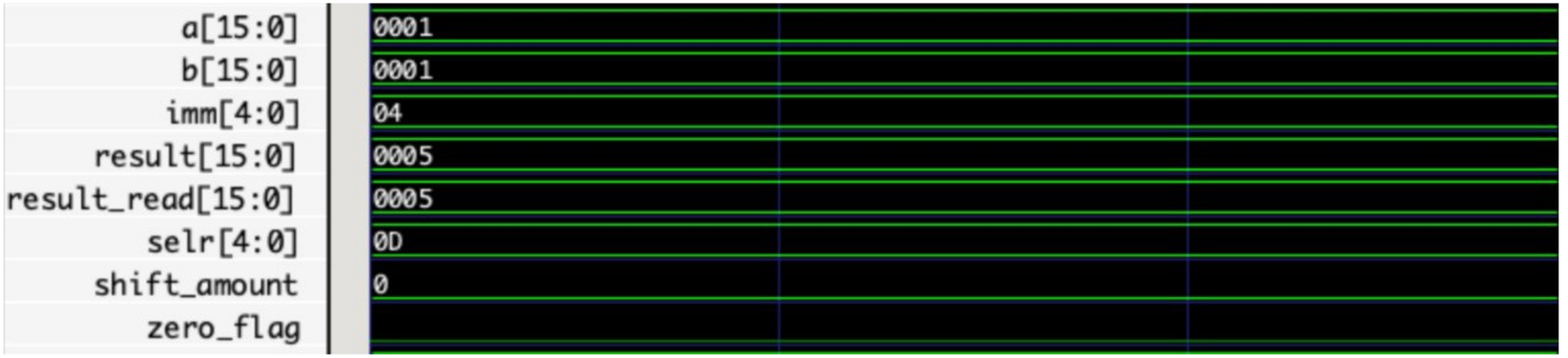
Instructions

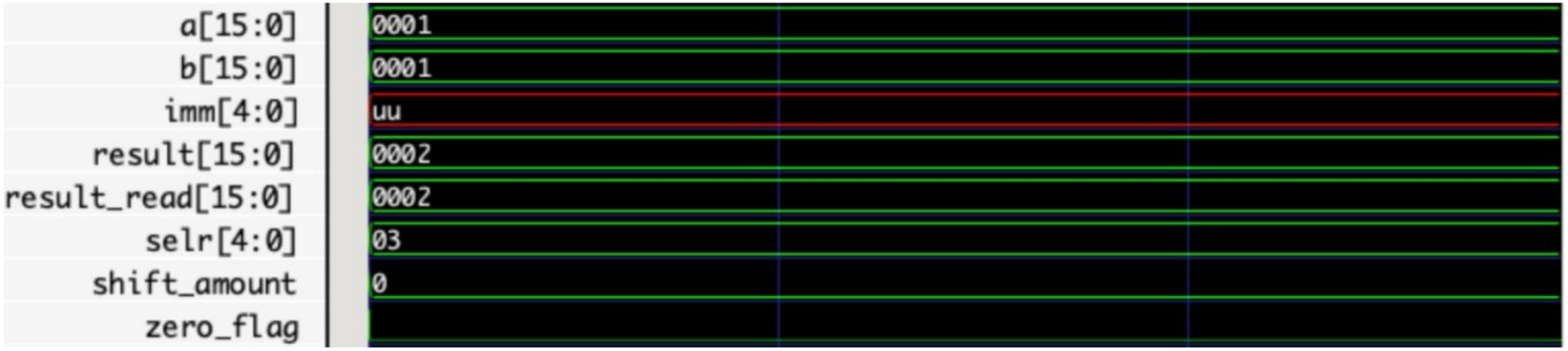
1000000000001010
1000000000100101
1000000001001000
0000010000100000
0100110101010011
0010000010010100
0010100110010100
0101001000100001

Les instructions sont chargées dans la ROM(fait dans notre code pour cet exemple);

Simulations

Nous avons pu tester individuellement les parties dont voici les résultats des “Tests Benchs”. Après la validation de ces composants, nous avons rencontré beaucoup de difficultés à assembler les différentes parties de notre processeur et donc à passer l’ensemble en simulation sur Quartus. Les testbench ont été réalisés grâce à GHDL sur MacOs.

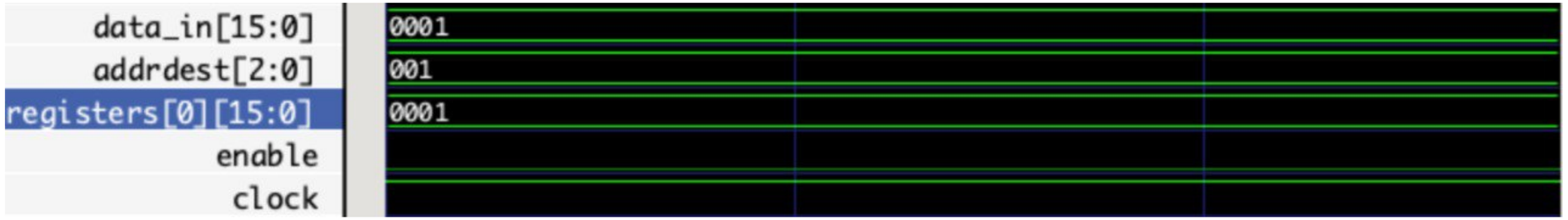


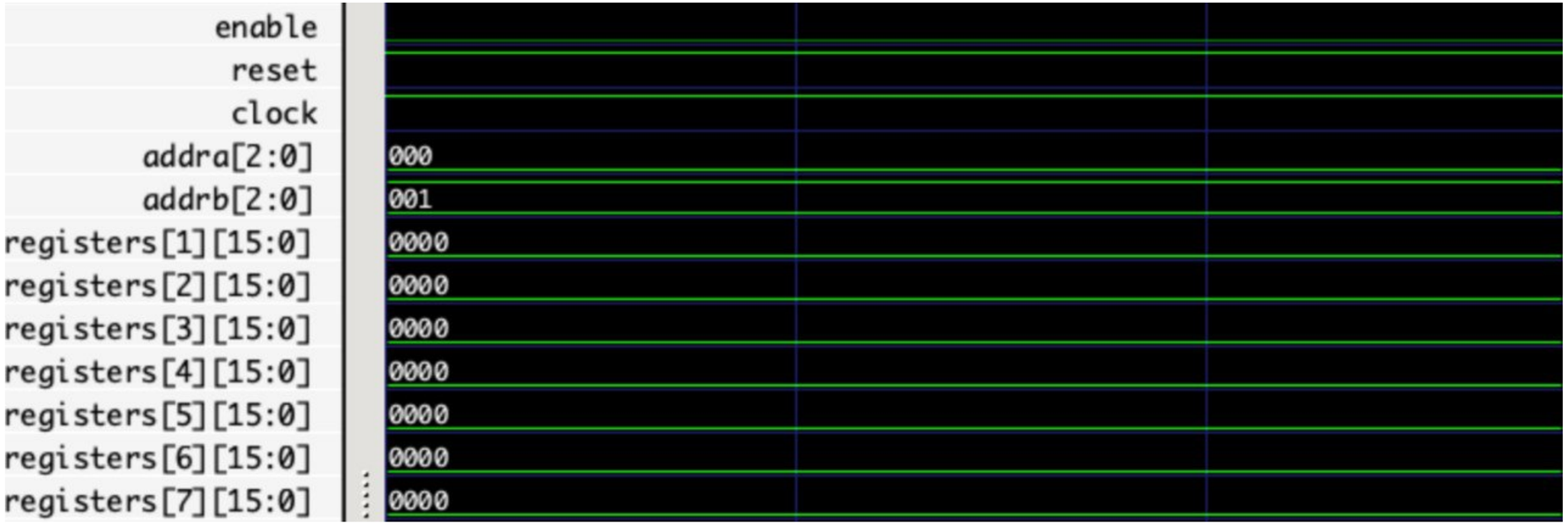



```
process is
    begin
    -- test format 1
    A <= "0000000000000001";
    B <= "0000000000000001";
    selR <= "00011"; --LSHIFT
    wait for 10 ns;

    -- test format 2
    IMM <= "00100";
    B <= "0000000000000001";
    selR <= "01101"; --OR
    wait for 10 ns;

    -- test format 3
    IMM <= "00100";
    selR <= "10000"; --MOV
    wait for 10 ns;
    end process;
```





Pour run les simulations voici les commandes utilisées:

ghdl -a *.vhd

ghdl -r test_alu --stop-time=10000ns --wave=wave.ghw

gtkwave wave.ghw