Pràctica 4:

Ampliació del conjunt d'instruccions de la CPU

Eloi Galan Badiella

Índex

Introducció **2**

Instrucció MUL **3**

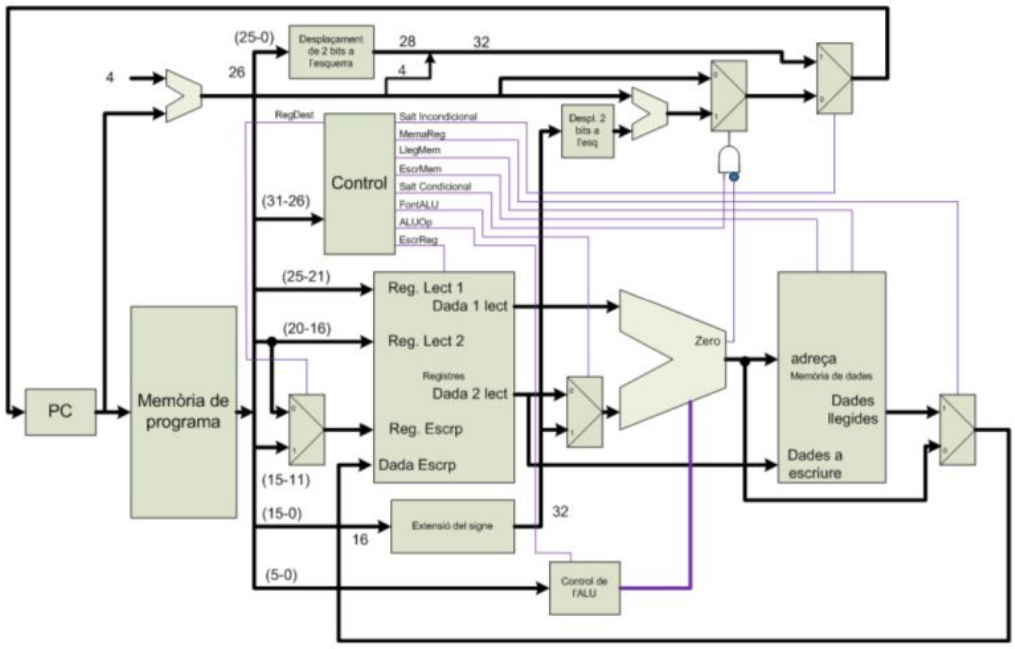
Implementació **4**
- Modificació ALU32 4
- Modificació ControlALU 6
- Modificació UnitatControl 7

Demostració del funcionament **8**

Codi complet **11**
- VHD 11
  - ALU32 11
  - ControlALU 12
  - UnitatControl 13
- VHT 15
  - ALU32 15

1. Introducció

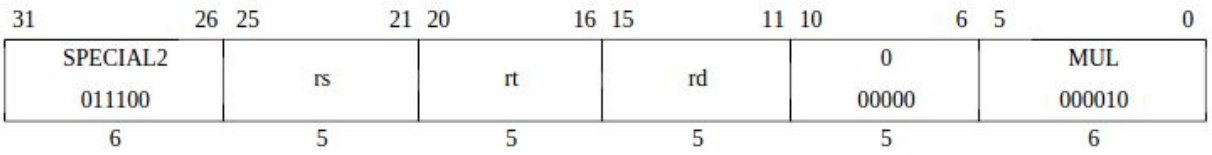
En aquesta pràctica realitzaré les modificacions necessàries en els mòduls que sigui necessaris per tal d'implementar la nova funció que se m'ha adjudicat: **MUL**



2. Instrucció MUL

Aquesta instrucció és bàsicament una multiplicació dels registres de 32 bits **rs** i **rt** on es guarda els 32 bits de menys pes del resultat al registre **rd.**

```
rd ← rs × rt
```



3. Implementació

Per a la implementació de instrucció MUL al nostre microprocessador he hagut de modificar diferents mòduls. Les modificacions fetes estan explicades en els següents punts de la pràctica.

3.1. Modificació ALU32

Per poder implementar la multiplicació a l'ALU32 he agut de decidir un valor de la senyal d'entrada Control_ALU corresponent a la instrucció MUL. El valor que he decidit ha estat el 011.

Basicament per poder afegir la instrucció MUL he hagut d'augmentar el valor de la variable on s'emmagatzema el resultat fins a 64 bits.

```
variable result: signed(63 downto 0);
```

La resta de codi es exactament igual excepte que en les altres valors de Control_ALU he hagut de restringir els bits d'emmagatzament per tal de que coincidís el numero de bits del resultat obtingut amb el espai esperat per a guardar.

```
when "000" => result(32 downto 0) := ('0' & Op1) AND ('0' & Op2); --and
when "001" => result(32 downto 0) := ('0' & Op1) OR ('0' & Op2); --or
when "010" => result(32 downto 0) := ('0' & Op1) + ('0' & Op2); --sum
when "110" => result(32 downto 0) := ('0' & Op1) - ('0' & Op2); --rest
when "011" => result := (Op1) * (Op2); --mul
when "111" =>  if (('0' & Op1) < ('0' & Op2)) then --Condicional
                  result(32 downto 0) := '0' & x"00000001";
               else
                  result(32 downto 0) := '0' & x"00000000";
               end if;
when others => result(32 downto 0) := ('0' & x"00000000");
```

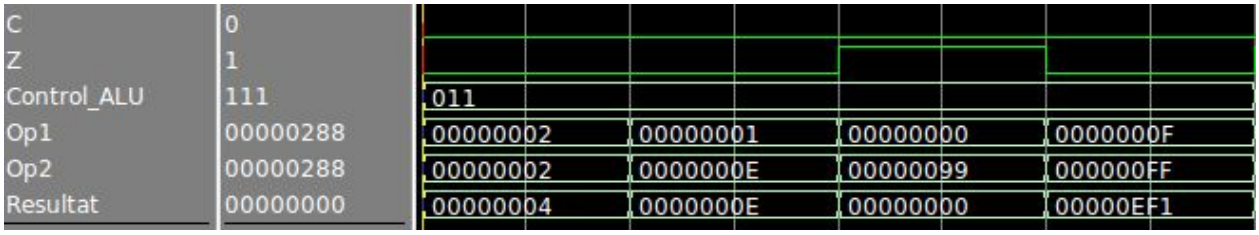
Simulació (ALU32)

La següent simulació correspon a 4 diferents proves de multiplicació.

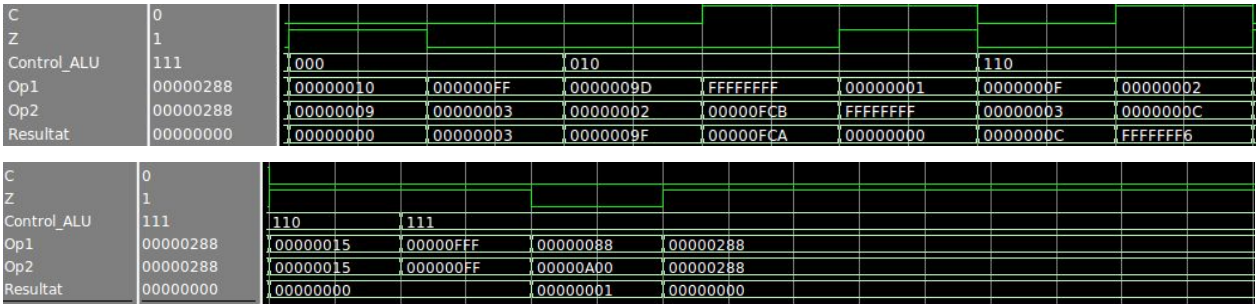
```
Op1 <= x"00000002";
Op2 <= x"00000002";
Control_ALU <= "011"; --mul 1
wait for 1 us;
Op1 <= x"00000001";
Op2 <= x"0000000E";
Control_ALU <= "011"; --mul 2
wait for 1 us;

Op1 <= x"00000000";
Op2 <= x"00000099";
Control_ALU <= "011"; --mul 3
wait for 1 us;
Op1 <= x"0000000F";
Op2 <= x"000000FF";
Control_ALU <= "011"; --mul 4
wait for 1 us;
```

Com es pot observar els resultats de la simulació són els esperats i ens activa els flags corresponents.



Per tal de comprovar de que segueix realitzant correctament els altres càlculs he realitzat més proves en el test bench i el resultat obtingut ha estat el correcte.

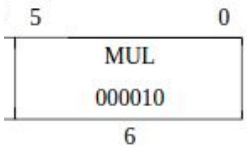


3.2. Modificació ControlALU

El que he modificat d'aquest mòdul ha estat afegir una nova sortida del ControlALU per a realitzar la nova instrucció MUL a l'ALU32. Aquesta nova ordre per a l'ALU l'he definit com a "011".

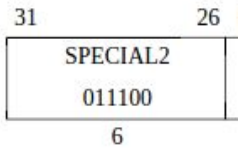
```
architecture behav of ControlALU is
begin
   process (funcio,CodiOP)
   variable Control: signed(2 downto 0);
   begin
      case funcio is
         when "000010" =>
            case CodiOP is
               when "10" => Control := "011";
                            Operacio <= Control(2 downto 0);
               when others => Control := "000";
                              Operacio <= Control(2 downto 0);
            end case;
         when others =>
            Control(0) := ((funcio(3) OR funcio(0)) AND CodiOP(1));
            Control(1) := ((NOT funcio(2)) OR (NOT CodiOP(1)));
            Control(2) := ((funcio(1) AND CodiOP(1)) OR codiOP(0));
            Operacio <= Control(2 downto 0);
      end case;
   end process;
end behav;
```

Com es pot observar en el codi que he adjuntat lo únic que he hagut de fer ha estat afegir un nou case per comprovar que els primers 6 bits de la instrucció corresponguin a la funció de MUL, que són "000010". I també comprovar que l'ALUop sigui la corresponent, en aquest cas ja que és una operació de tipus R seria "10".



3.3. Modificació UnitatControl

En aquest modul he afegit les noves sortides de flags per la nova instrucció MUL. Com es pot observar detectarem que és aquesta funció quan els primers 6 bits de la instrucció siguin "011100".



Els flags que posarem a 1 seran RegDest i EscReg per poder guardar el valor de la operació al Rd.
L'ALUOP serà "10" com tots els tipus R.

```
when "011100" =>  --Tipus R -> mul
 RegDest <= '1';
 SaltInc <= '0';
 MemaReg <= '0';
 LlegMem <= '0';
 EscrMem <= '0';
 SaltCon <= '0';
 FontALU <= '0';
 ALUOP  <= "10";
 EscrReg <= '1';
```

4. Demostració del funcionament

Per comprovar que tot el conjunt funciona perfectament he creat noves instruccions a la Memòria d'Instruccions.

```
X"20070040",X"20080048",X"20090060",X"8cea0000",
X"71072802",X"70C72802",X"71284002",X"00000000",
```

Instruccions per guardar valors a registres per poder operar:

1- X"20070040"
addi $7, $0, 0x40
0010 00|00 000|0 0111| 0000 0000 0100 0000

2- X"20080048"
addi $8, $0, 0x48
0010 00|00 000|0 1000| 0000 0000 0100 1000

3- X"20090060"
addi $9, $0, 0x60
0010 00|00 000|0 1001| 0000 0000 0110 0000

4- X"8CEA0000"
lw $10, 0($7)
1000 11|00 111|0 1010| 0000 0000 0000 0000

Instruccións que fan servir la multiplicació implementada:

5- X"71072802"
mul $5, $8, $7
r5←r8×r7

Special2 r8 r7 r5 MUL
0111 00|01 000|0 0111| 0010 1|000 00|00 0010

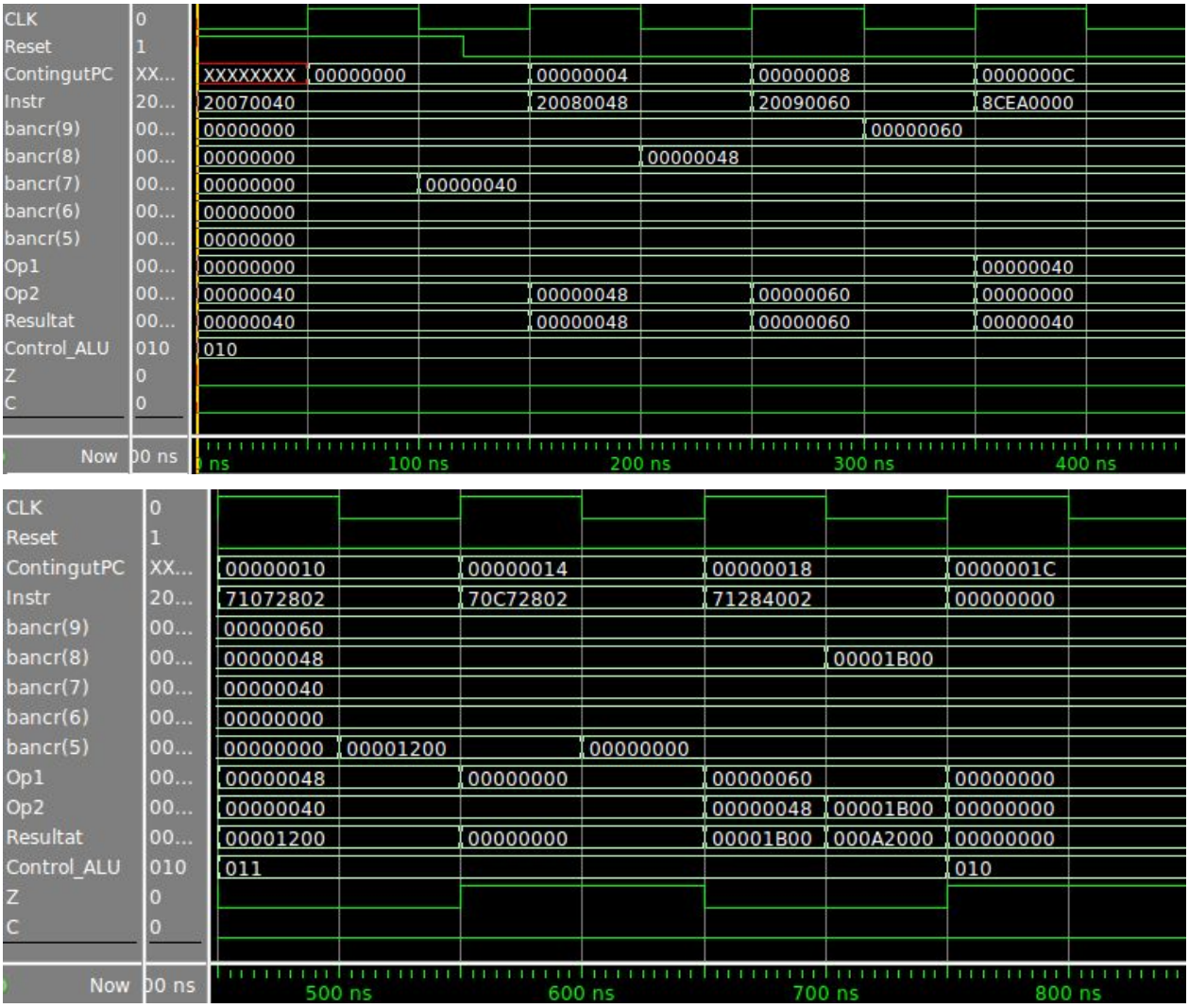
6- X"70C72802"
mul $5, $6, $7
r5←r6×r7

Special2 r6 r7 r5 MUL
0111 00|00 110|0 0111| 0010 1|000 00|00 0010

7- X"71284002"
mul $8, $9, $8
r8←r9×r8

Special2 r9 r8 r8 MUL
0111 00|01 001|0 1000| 0100 0|000 00|00 0010

Simulació de les instruccions esmentades:



Com podem comprovar mirant la simulació, ens realitza la multiplicació perfetament sense afectar a les altres instruccions més bàsiques. Per tant concluire dient que el programa ja és capaç de realitzar la nova instrucció MUL en tota perfecció.

5. Codi complet

En aquest apartat adjuntaré el codi de els mòduls modificats per a la realització d'aquesta pràctica. La resta segueixen exactament igual que a la practica 3.

5.1. VHD

ALU32

```
LIBRARY ieee;
USE ieee.std_logic_1164.all; --RECORDAR: no diferencia majúscules de minúscules
use ieee.numeric_std.all; --en qualsevol de les comandes vhdl

entity ALU32 is
   port (
         Op1, Op2 : in signed (31 downto 0);
         Control_ALU : in signed (2 downto 0);
         Resultat : out signed (31 downto 0);
         Z : out STD_LOGIC;
         C : out STD_LOGIC);
end ALU32;

architecture behavior of ALU32 is
begin
   process (Control_ALU, Op1, Op2) is
      variable result: signed(63 downto 0);

   begin
      case Control_ALU is
                                    when "000" => result(32 downto 0) := ('0' & Op1) AND ('0' & Op2); --and
                                    when "001" => result(32 downto 0) := ('0' & Op1) OR ('0' & Op2); --or
         when "010" => result(32 downto 0) := ('0' & Op1) + ('0' & Op2); --sum
         when "110" => result(32 downto 0) := ('0' & Op1) - ('0' & Op2); --rest
                                    when "011" => result := (Op1) * (Op2); --mul
                                    when "111" =>     if (('0' & Op1) < ('0' & Op2)) then      --Condicional
                                                                                               result(32 downto 0) := '0' &
x"00000001";
                                                                                     else
                                                                                               result(32 downto 0) := '0' &
x"00000000";
                                                                                     end if;
         when others => result(32 downto 0) := ('0' & x"00000000");
   end case;

   if(result(31 downto 0) = x"00000000") then
      Z <= '1';
   else
```

```
    Z <= '0';
  end if;
  C <= result(32);
        resultat <= result(31 downto 0);
  end process;
end architecture;
```

ControlALU

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

entity ControlALU is
      Port (
            funcio : in STD_LOGIC_VECTOR (5 downto 0);
            CodiOP : in STD_LOGIC_VECTOR (1 downto 0); -- ALUOP
            Operacio : out signed (2 downto 0) -- Sortida control alu
      );
end ControlALU;



architecture behav of ControlALU is
begin
      process (funcio,CodiOP)
      variable Control: signed(2 downto 0);
      begin
            case funcio is
                  when "000010" =>
                        case CodiOP is
                              when "10" => Control := "011";
                                                Operacio <= Control(2 downto 0);
                              when others => Control := "000";
                                                      Operacio <= Control(2 downto 0);
                        end case;
                  when others =>
                        Control(0) := ((funcio(3) OR funcio(0)) AND CodiOP(1));
                        Control(1) := ((NOT funcio(2)) OR (NOT CodiOP(1)));
                        Control(2) := ((funcio(1) AND CodiOP(1)) OR codiOP(0));
                        Operacio <= Control(2 downto 0);
            end case;
      end process;
end behav;
```

UnitatControl

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

entity UnitatControl is
 Port (
  co : in STD_LOGIC_VECTOR (5 downto 0);
  --z : in STD_LOGIC; -- colocarem el flag z i c fora de la Unitat de Control tal com esta a l'esquema
  --c : in STD_LOGIC;
  RegDest : out STD_LOGIC;
  SaltInc : out STD_LOGIC;
  MemaReg : out STD_LOGIC;
  LlegMem : out STD_LOGIC;
  EscrMem : out STD_LOGIC;
  SaltCon : out STD_LOGIC;
  FontALU : out STD_LOGIC;
  ALUOP : out STD_LOGIC_VECTOR(1 downto 0) ;
  EscrReg : out STD_LOGIC
 );
end UnitatControl;



architecture behav of UnitatControl is
begin
 process (co)
 begin
  case co is
   when "000000" =>  --Tipus R -> add, sub, and, or, nop, mul fet
    RegDest <= '1';
    SaltInc <= '0';
    MemaReg <= '0';
    LlegMem <= '0';
    EscrMem <= '0';
    SaltCon <= '0';
    FontALU <= '0';
    ALUOP  <= "10";
    EscrReg <= '1';

   when "001000" =>  --Tipus I -> addi fet
    RegDest <= '0';
    SaltInc <= '0';
    MemaReg <= '0';
    LlegMem <= '0';
    EscrMem <= '0';
    SaltCon <= '0';
```

```
 FontALU <= '1';
 ALUOP  <= "00";
 EscrReg <= '1';

when "100011" =>  --Tipus I -> lw fet
 RegDest <= '0';
 SaltInc <= '0';
 MemaReg <= '1';
 LlegMem <= '1';
 EscrMem <= '0';
 SaltCon <= '0';
 FontALU <= '1';
 ALUOP  <= "00";
 EscrReg <= '1';

when "101011" =>  --Tipus I -> sw fet
 RegDest <= '0';
 SaltInc <= '0';
 MemaReg <= '0';
 LlegMem <= '0';
 EscrMem <= '1';
 SaltCon <= '0';
 FontALU <= '1';
 ALUOP  <= "00";
 EscrReg <= '0';

when "000101" =>  --Tipus I -> bne fet
 RegDest <= '0';
 SaltInc <= '0';
 MemaReg <= '0';
 LlegMem <= '0';
 EscrMem <= '0';
 SaltCon <= '1';
 FontALU <= '0';
 ALUOP  <= "01";
 EscrReg <= '1';

when "001010" =>  --Tipus I -> slti fet
 RegDest <= '0';
 SaltInc <= '0';
 MemaReg <= '0';
 LlegMem <= '0';
 EscrMem <= '0';
 SaltCon <= '0';
 FontALU <= '1';
 ALUOP  <= "11";
 EscrReg <= '1';

when "000010" =>  --Tipus J -> J fet
```

```
  RegDest <= '0';
  SaltInc <= '1';
  MemaReg <= '0';
  LlegMem <= '0';
  EscrMem <= '0';
  SaltCon <= '0';
  FontALU <= '0';
  ALUOP  <= "00";
  EscrReg <= '0';

        when "011100" =>  --Tipus R -> mul NNNNNOOOOUUUUUU
  RegDest <= '1';
  SaltInc <= '0';
  MemaReg <= '0';
  LlegMem <= '0';
  EscrMem <= '0';
  SaltCon <= '0';
  FontALU <= '0';
  ALUOP  <= "10";
  EscrReg <= '1';

 when others =>
  RegDest <= '0';
  SaltInc <= '0';
  MemaReg <= '0';
  LlegMem <= '0';
  EscrMem <= '0';
  SaltCon <= '0';
  FontALU <= '0';
  ALUOP  <= "00";
  EscrReg <= '0';
 end case;
end process;
end behav;
```

5.2. VHT

ALU32

```
Op1 <= x"00000002";
Op2 <= x"00000002";
Control_ALU <= "011"; --mul 1
wait for 1 us;
Op1 <= x"00000001";
Op2 <= x"0000000E";
Control_ALU <= "011"; --mul 2
wait for 1 us;
```

```
Op1 <= x"00000000";
Op2 <= x"00000099";
Control_ALU <= "011"; --mul 3
wait for 1 us;
Op1 <= x"0000000F";
Op2 <= x"000000FF";
Control_ALU <= "011"; --mul 4
wait for 1 us;


Op1 <= x"00000010";
Op2 <= x"00000009";
Control_ALU <= "000"; --and 1
wait for 1 us;
Op1 <= x"000000FF";
Op2 <= x"00000003";
Control_ALU <= "000"; --and 2
wait for 1 us;

Op1 <= x"0000009D";
Op2 <= x"00000002";
Control_ALU <= "010"; --sum 1
wait for 1 us;
Op1 <= x"FFFFFFFF";
Op2 <= x"00000FCB";
Control_ALU <= "010"; --sum 2 -> carry
wait for 1 us;
Op1 <= x"00000001";
Op2 <= x"FFFFFFFF";
Control_ALU <= "010"; --sum 3 -> carry + Z
wait for 1 us;

Op1 <= x"0000000F";
Op2 <= x"00000003";
Control_ALU <= "110"; --rest 1
wait for 1 us;
Op1 <= x"00000002";
Op2 <= x"0000000C";
Control_ALU <= "110"; --rest 2 -> carry
wait for 1 us;
Op1 <= x"00000015";
Op2 <= x"00000015";
Control_ALU <= "110"; --rest 3 -> Z
wait for 1 us;

Op1 <= x"00000FFF";
Op2 <= x"000000FF";
Control_ALU <= "111"; --condicional 1 -> op1>op2 = false (0)
```

```
wait for 1 us;
Op1 <= x"00000088";
Op2 <= x"00000A00";
Control_ALU <= "111"; --condicional 2 -> op1<op2 = true (1)
wait for 1 us;
Op1 <= x"00000288";
Op2 <= x"00000288";
Control_ALU <= "111"; --condicional 3 -> op1==op2 = false (0)
wait for 1 us;
```