# Pràctica 1:

## Disseny de la unitat Aritmètica i lògica (ALU)

*Arnau Plans Castelló*
*Marc Estiarte Salisi*
*Eloi Galan Badiella*

## 1. Introducció

En aquesta primera pràctica realitzarem el disseny de la ALU (Unitat Aritmètica Lògica) mostrada a la *Figura 1.*

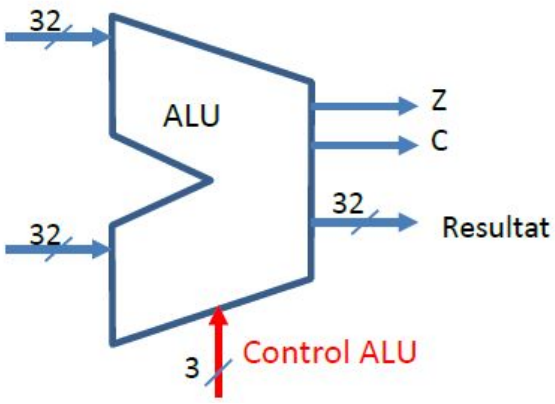



*Figura 1: ALU*

La implementació la duem a terme fent servir el programa Quartus II i la simulació fent servir el ModelSim.

## 2. Disseny ALU 32

En aquest apartat explicarem com hem dissenyat la entitat ALU 32, mostrarem el codi vhdl que hem programat i el comentarem per fer veure el procés i la lògica del codi.

### *Codi:*

Arquitectura:

```
architecture behavior of ALU32 is
begin
    process (Control_ALU, Op1, Op2) is
        variable result: signed(32 downto 0);

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

### *Explicació:*

A l'inici de l'arquitectura podem observar com hem designat els operands esmentats (Control_ALU) a la pràctica a cada una de les diferents operacions: and, or, suma, resta, condició menor que.
A continuació s'hi pot observar el funcionament dels dos flags: Z (s'activa quan el resultat és 0), C (s'activa quan apareix carry).
I finalment hi trobem la assignació del resultat.

## 2.1. Simulació

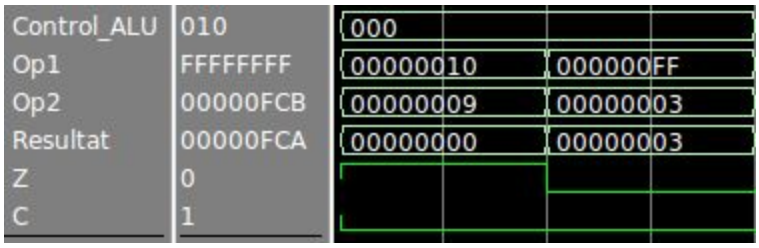
En aquest apartat mostrarem el test bench i la seva simulació amb els seus respectius càlculs.

- **Operació/Simulació AND:**

***Codi:***

```
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;
```

***Simulació:***




***Explicació:***

En aquest cas hem fet dos proves per a la instrucció AND, ja que no és necessari fer-ne moltes més al no poder activar el flag C (Carry) de cap manera. Aleshores hem provat posant 2 nombres aleatoris als 2 operands en les dos proves de tal manera que pogués sim verificar el funcionament de l'instrucció. Com podem veure els resultats són els esperats pel comportament de l'instrucció AND.

10 and 9→ b00000000000000000000000000010000 → x10
b00000000000000000000000000001001 → x9
resultat: b00000000000000000000000000000000 → x0 (amb Z)

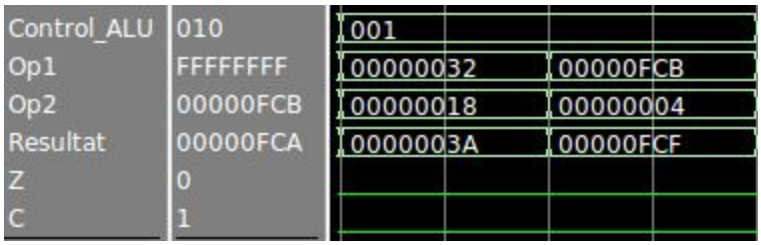
FF and 3 → b00000000000000000000000011111111 → xFF
b00000000000000000000000000000011 → x3
resultat: b00000000000000000000000000000011 → x3

- **Operació/Simulació OR:**

***Codi:***

```
Op1 <= x"00000032";
Op2 <= x"00000018";
Control_ALU <= "001"; --or 1
wait for 1 us;
Op1 <= x"00000FCB";
Op2 <= x"00000004";
Control_ALU <= "001"; --or 2
wait for 1 us;
```

***Simulació:***




***Explicació:***

Per a aquesta operació tambe hem fet servir nomes dos proves.
Aleshores hem provat posant 2 nombres aleatoris als 2 operants en les dos proves de tal manera que pogués sim verificar el funcionament de l'instrucció.
Com podem veure els resultats són els esperats pel comportament de l'instrucció OR.

32 or 18 → b00000000000000000000000000110010 → x32
b00000000000000000000000000011000 → x18
resultat: b00000000000000000000000000111010 → x3A

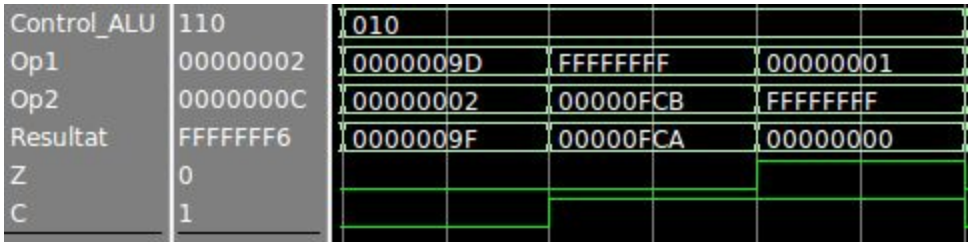
FCB or 4 → b00000000000000000000111111001011 → xFCB
b00000000000000000000000000000100 → x4
resultat: b00000000000000000000111111001111 → xFCF

- **Operació/Simulació Suma:**

***Codi:***

```
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;
```

***Simulació:***




***Explicació:***

En aquest cas hem fet tres proves per a la instrucció SUMA, ja que hi ha la possibilitat d'activar el flag C (Carry) i el flag Z (Zero). El resultat de les tres proves ha estat l'esperat ja sigui el resultat o el valor dels flags.

9D + 2 → b00000000000000000000000010011101 → x9D
b00000000000000000000000000000010 → x2
resultat: b00000000000000000000000010011111 → x9F

FFFFFFFF + FCB → b11111111111111111111111111111111 → xFFFFFFFF
b00000000000000000000111111001011 → xFCB
resultat: b00000000000000000000111111001010 → xFCA (amb Carry)

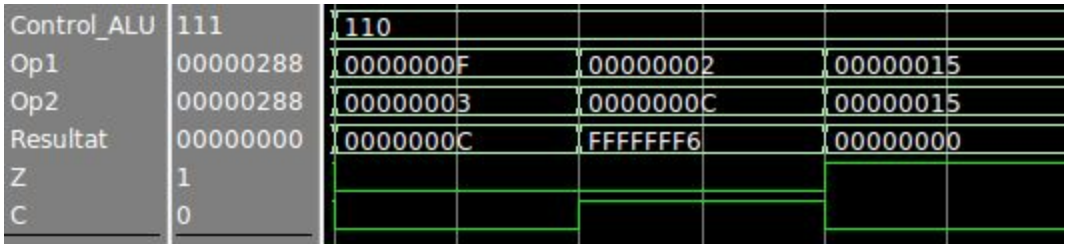
1 + FFFFFFFF→ b00000000000000000000000000000001 → x1
b11111111111111111111111111111111 → xFFFFFFFF
resultat: b00000000000000000000000000000000 → x0 (amb Carry i Z)

- **Operació/Simulació Resta:**

***Codi:***

```
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;
```

***Simulació:***




***Explicació:***

En el cas de la RESTA hem tornat a fer tres proves ja que hi ha la possibilitat d'activar el flag C (Carry) i el flag Z (Zero). El resultat de les tres proves ha estat l'esperat ja sigui el resultat o el valor dels flags.

F - 3 → b00000000000000000000000000001111 → xF
b00000000000000000000000000000011 → x3
resultat: b00000000000000000000000000001100 → xC

2 - C → b00000000000000000000000000000010 → x2
b00000000000000000000000000001100 → xC
resultat: b11111111111111111111111111110110 → xFFFFFFF6 (amb Carry)

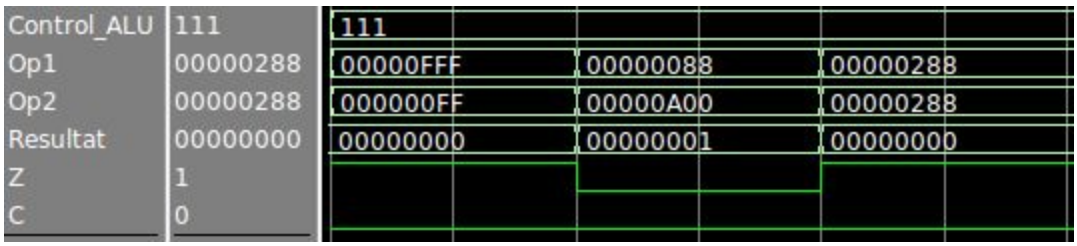
15 - 15 → b00000000000000000000000000010101 → x15
b00000000000000000000000000010101 → x15
resultat: b00000000000000000000000000000000 → x0 (amb Z)

- **Operació/Simulació Comparació:**

***Codi:***

```
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;
```

***Simulació:***




***Explicació:***

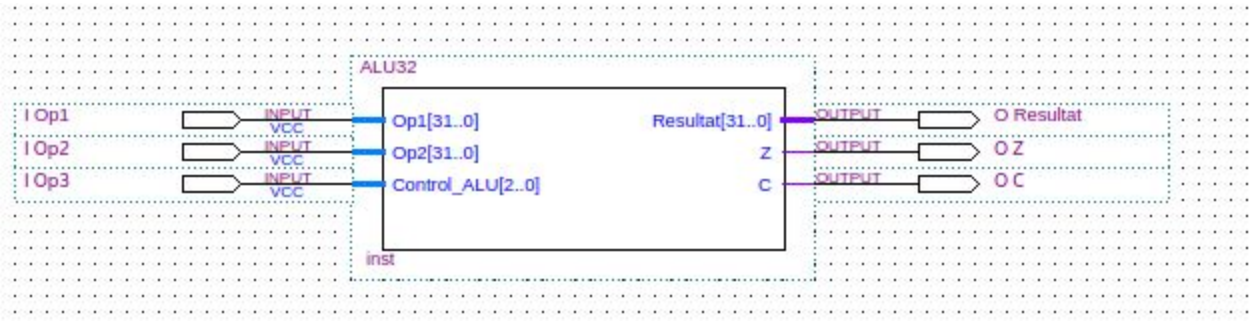
Pel que fa el CONDICIONAL hem fet 3 proves, una en el cas de que Op1 sigui més gran que Op2, una altre en el cas de que Op1 sigui més petit que Op2, i finalment en el cas de que els dos operands (Op1, i Op2) siguin iguals.

Per a poder aconseguir un 0 o un 1 en binari com a resultat hem fet un condicional ("if statement") el qual comparava els dos operands i per tant com a resultat d'aquests obteníem un true/false, per acabar en binari hem hagut d'analitzar si la resposta a la operació era true, mostrarem un 1, en cas contrari mostrem un 0. D'aquesta manera si el Op1 es més petit que Op2 es mostra un 1, si el Op1 és més gran o igual que Op2 es mostrara un 0.

Op1 > Op2 => b00000000000000000000000000000000
Op1 < Op2 => b00000000000000000000000000000001
Op1 = Op2 => b00000000000000000000000000000000

## 3. Símbol ALU 32

En aquest apartat hem realitzar el que seria el símbol de la ALU 32 dissenyada a l'apartat anterior. L'hi hem col·locat els 3 pins d'entrada i els 3 pins de sortida per a que el disseny estigues complert.




El resultat de la simulació hauria de ser el mateix.

## 4. Preguntes

a) Heu pogut fer la simulació dels dos dissenys de la mateixa forma? Quines heu utilitzat i per quin motiu?
La simulació del disseny de l'ALU de 32 bits l'hem realitzar amb el ModelSim ja que és el que se'ns a recomanat a classe, i la simulació del Símbol de l'ALU de 32 bits no l'hem pogut realitzar ja que no sabíem com fer-ho.

b) Hi ha un temps mínim per poder fer modificacions als senyals d'entrada?
Per esbrinar-ho hem reduït el temps de canvi fins a 0.1 ps i no ha donat cap error. Per tant suposem que no hi ha temps mínim per poder fer modificacions als senyals d'entrada.

c) Quin és el temps de resposta dels senyals de sortida?
Segons les simulacions el temps de resposta dels senyals de sortida és insignificant.

## 5. Codi complet

**VHDL**

```
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(32 downto 0);

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

## VHT

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.numeric_std.all; --en qualsevol de les comandes vhdl

ENTITY ALU32_vhd_tst IS
END ALU32_vhd_tst;
ARCHITECTURE ALU32_arch OF ALU32_vhd_tst IS
-- constants
-- signals
SIGNAL C : STD_LOGIC;
SIGNAL Control_ALU : Signed(2 DOWNTO 0);
SIGNAL Op1 : signed(31 DOWNTO 0);
SIGNAL Op2 : signed(31 DOWNTO 0);
SIGNAL Resultat : Signed(31 DOWNTO 0);
SIGNAL Z : STD_LOGIC;
COMPONENT ALU32
        PORT (
        C : OUT STD_LOGIC;
        Control_ALU : IN signed(2 DOWNTO 0);
        Op1 : IN signed(31 DOWNTO 0);
        Op2 : IN signed(31 DOWNTO 0);
        Resultat : OUT signed(31 DOWNTO 0);
        Z : OUT STD_LOGIC
        );
END COMPONENT;
BEGIN
        i1 : ALU32
        PORT MAP (
-- list connections between master ports and signals
        C => C,
        Control_ALU => Control_ALU,
        Op1 => Op1,
        Op2 => Op2,
        Resultat => Resultat,
        Z => Z
        );
init : PROCESS
-- variable declarations
BEGIN
    -- code that executes only once
WAIT;
END PROCESS init;
always : PROCESS
-- optional sensitivity list
-- (       )
-- variable declarations
BEGIN
```

```
-- code executes for every event on sensitivity list
        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"00000032";
        Op2 <= x"00000018";
        Control_ALU <= "001"; --or 1
        wait for 1 us;
        Op1 <= x"00000FCB";
        Op2 <= x"00000004";
        Control_ALU <= "001"; --or 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;

WAIT;
END PROCESS always;
END ALU32_arch;
```