Pràctica 3:

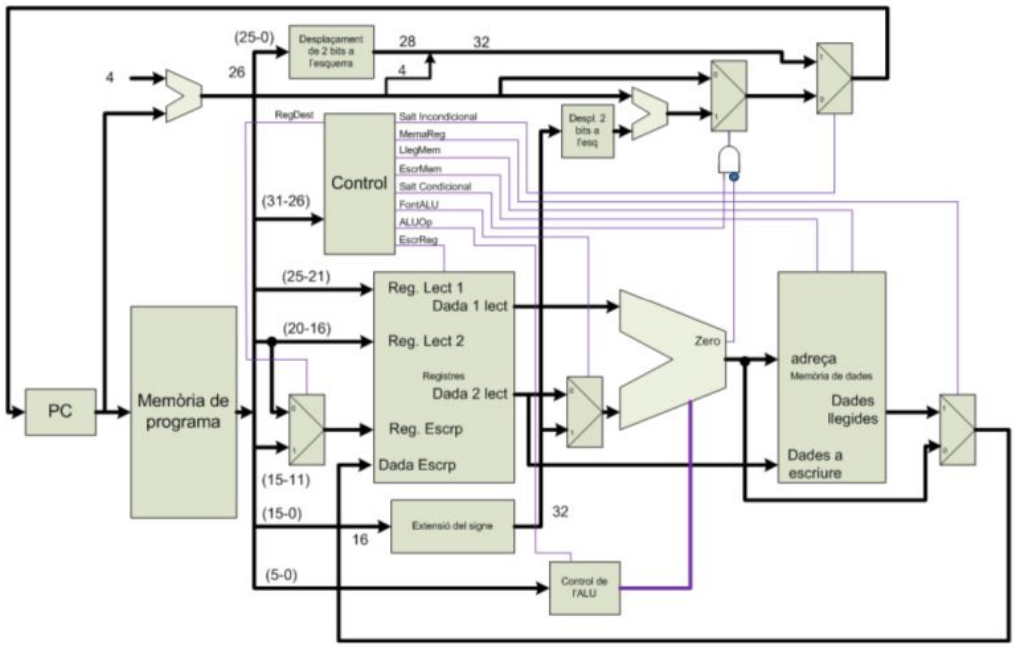
Disseny de la CPU

Arnau Plans Castelló
Marc Estiarte Salisi
Eloi Galan Badiella

Índex

1. Introducció

En aquesta pràctica realitzarem el disseny de la Unitat de Control iel Control de l'ALU pel processador monocicle i el camí complert de dades del processador.



2. Exercicis

2.1. Ex 1 - Unitat de Control de la CPU

Simulació (UnitatControl de la CPU)

Explicació de la simulació de la UnitatControl de la CPU:

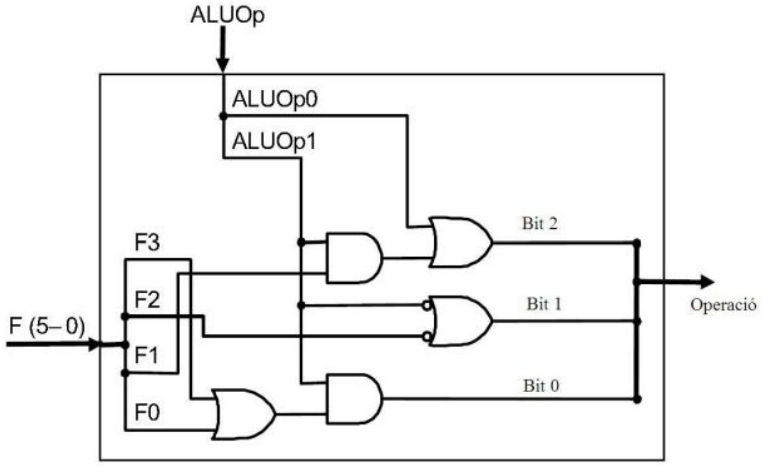
0 - 100 ns → Hem fet esperar 100ns per evitar errors a l'hora d'iniciar les proves.
100 - 200 ns → En aquest període de temps hem simulat el codi d'operació de la instrucció ADD, la qual ens activa els flags EscTeg i RegDest mentres els altres els manté a desactivats. L'ALUOP val 10 tal com diu la pràctica.
200 - 300 ns → En quest cas hem provat la instrucció LW, i els flags activats han estat: FontALU, LlegMem i MemaReg. El valor de l'ALUOP és 00.
300 - 400 ns → En l'ultim periode de 100ns hem comprovat que la operació BRNE actives els flags corresponents. Com podem veure a la simulació l'unic flag activat ha estat el de SaltCondicional, mentres que la ALUOP pren per valor 01.

2.2. Ex 2 - Unitat de Control de l'ALU

La seguent taula mostra el resultat de l'Operació qua hara de tenir la Unitat de Control de l'ALU.

ALUOp		Camp de la funció						Operació
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

Per aconseguir-ho hem posat en pràctica l'implementació següent:

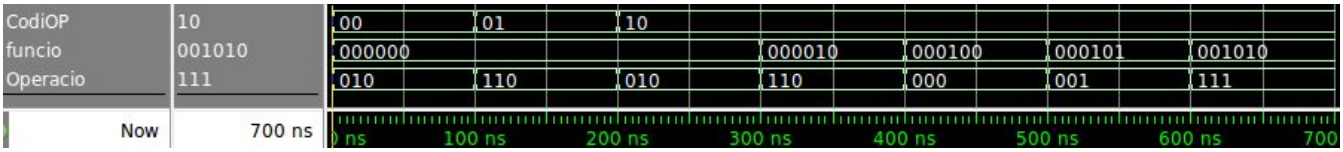


La implementació en codi de l'esquema anterior seria la següent:

```
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);
```

Simulació (ControlALU)



Explicació de la simulació del ControlALU:

Com podem observar a la simulació l'implementació de l'esquema és correcte ja que el resultat de l'Operació en la simulació concorda amb els resultats de la taula anterior.

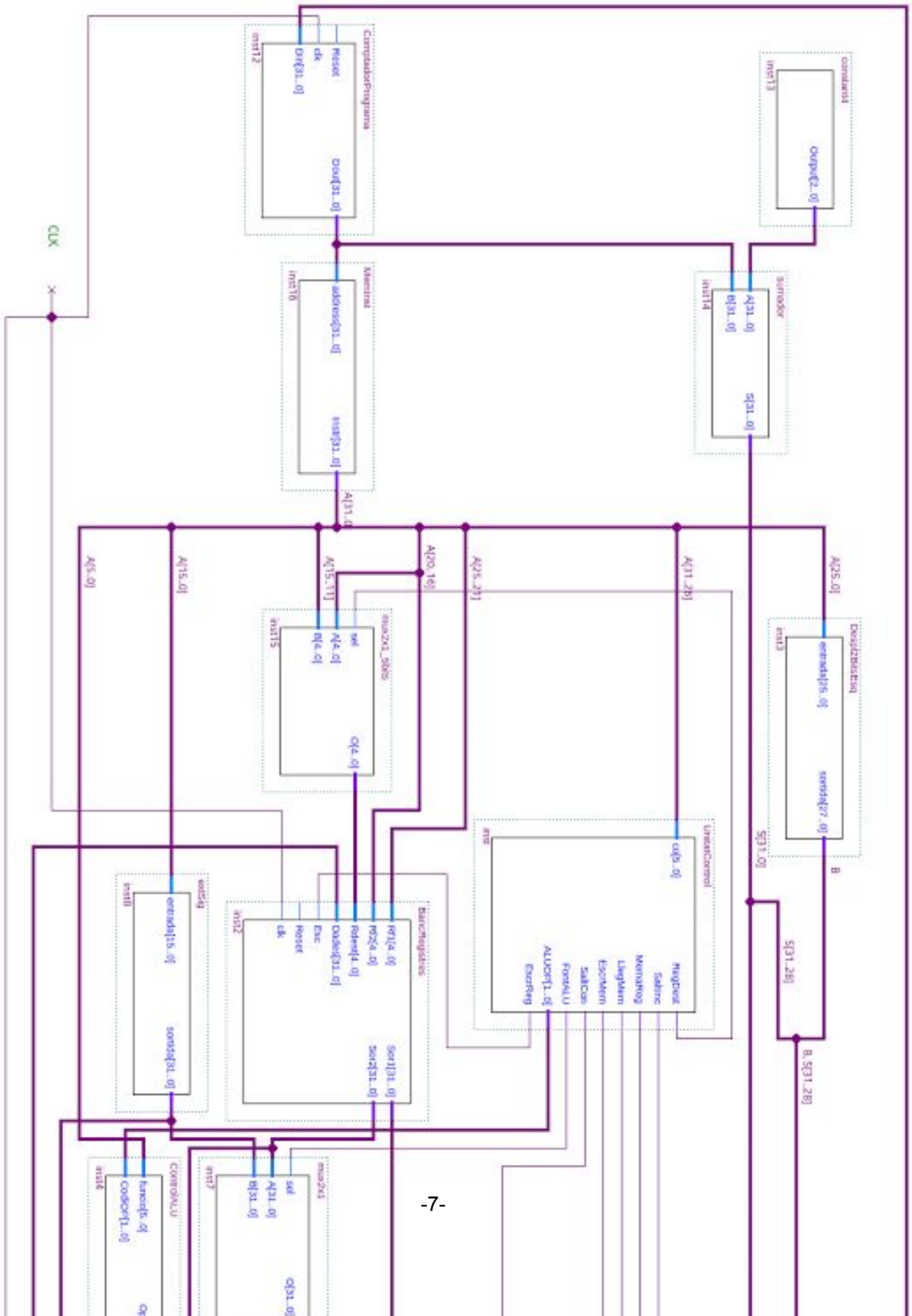
2.3. Ex 3 - Passos previs a seguir

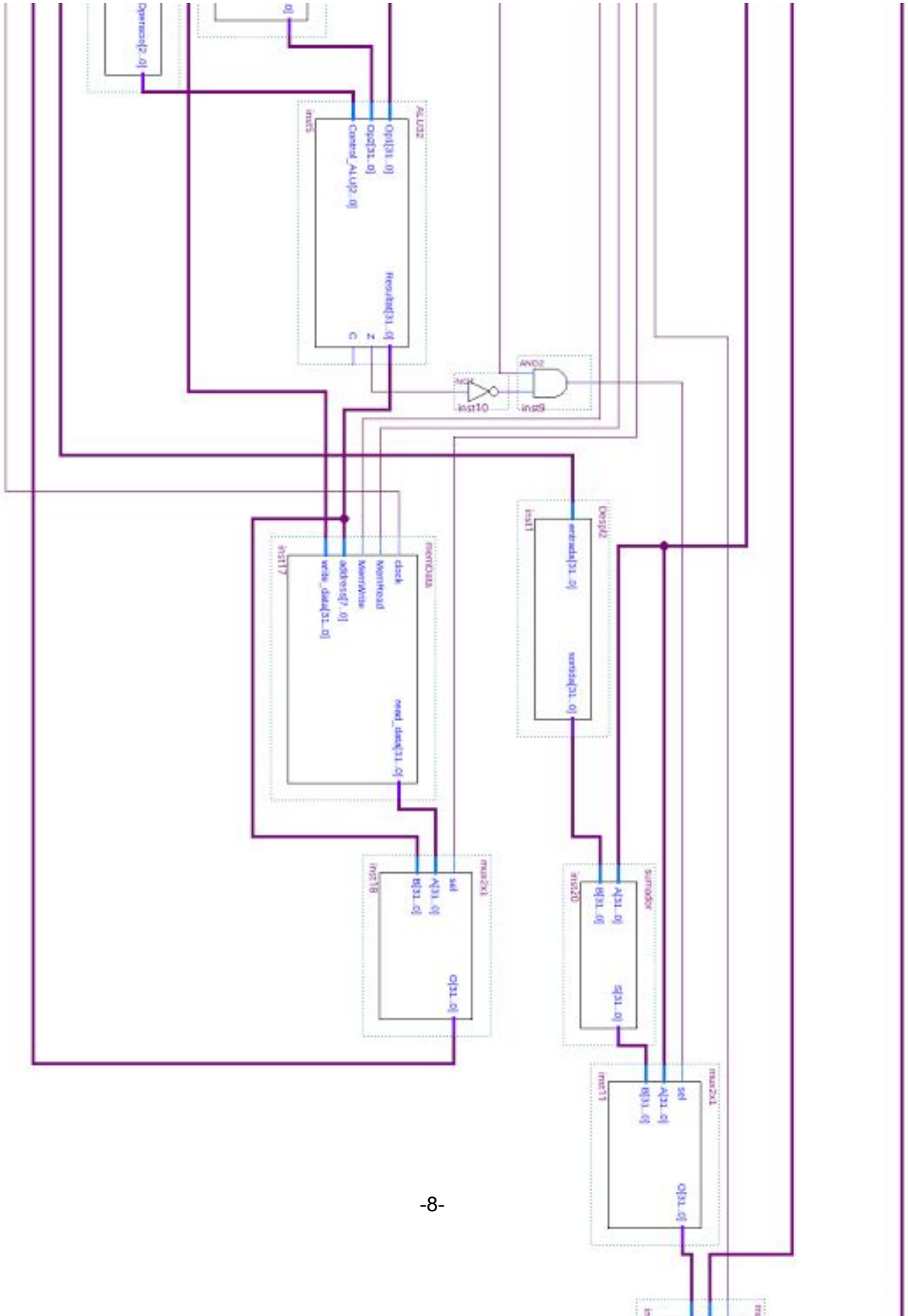
Fet.

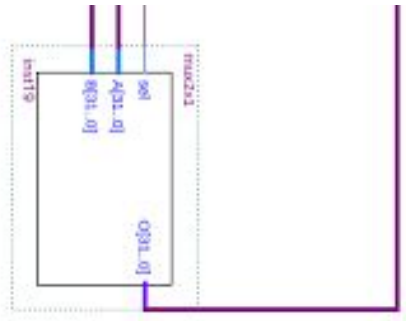
2.4. Ex 4 - Disseny del microprocessador

Per a realitzar el disseny del microprocessador vam començar creant els simbols de tots els moduls i creant un block de simbols. Pero ens vam trobar amb molts problemes a l'hora de simular. Per aquest motiu vam redissenyarlo a partir d'un nou modul VHD que col·lecciona tots els elements del microprocessador i els relaciona entre ells.
En el seguent apartat "Disseny amb simbols" hem adjuntat una imatge de com ens ha quedat el disseny amb simbols no finalitzar, i mes endebant a l'apartat "disseny amb codi" expliquem com hem realitzat el codi base del microprocessador i les seves simulacions corresponents.

Disseny amb símbols

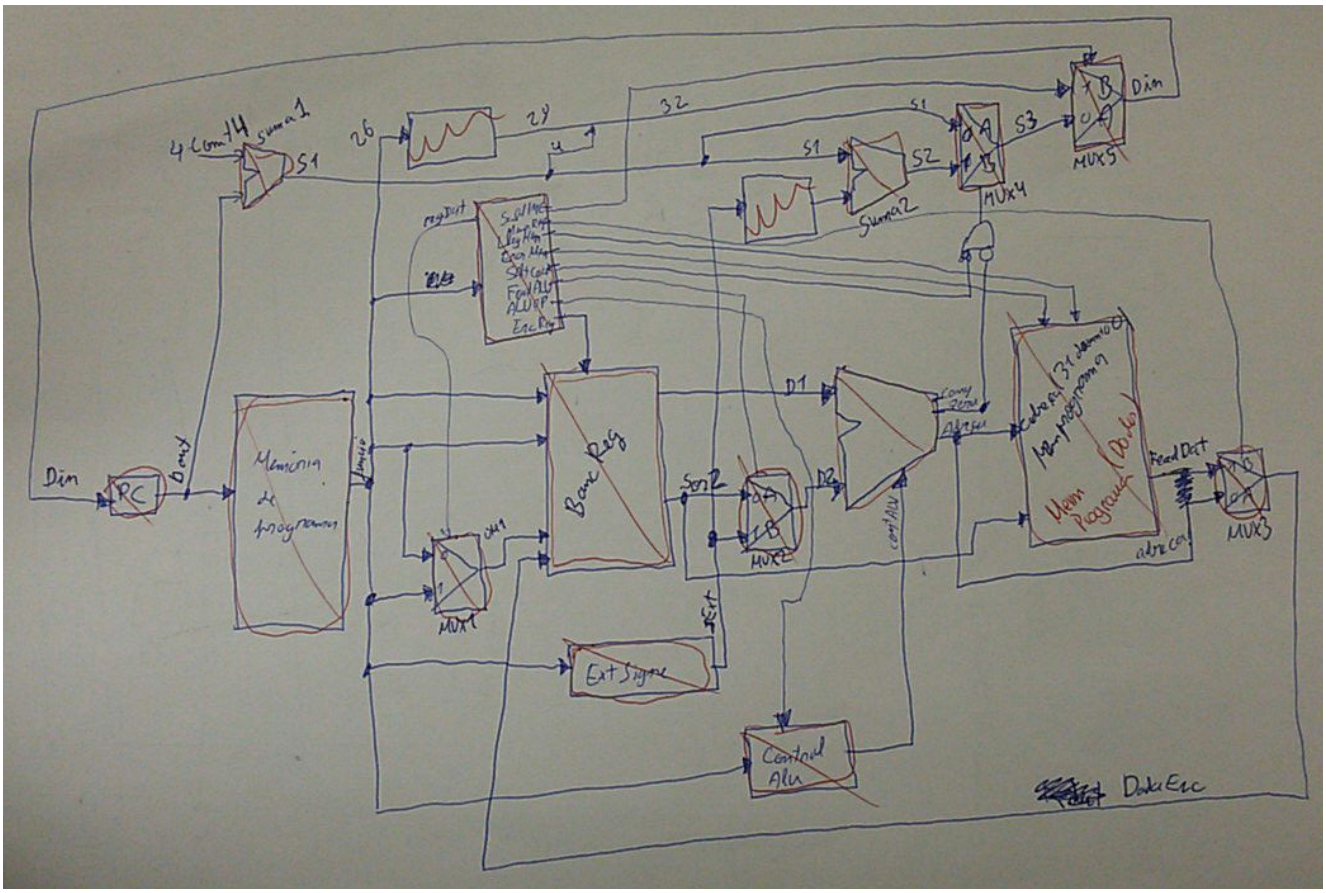

CLK
inst12
ComputadorPrograma
Reset
clk
Dir[31..0]
Dout[31..0]
inst13
constant
Ouput[2..0]
inst16
Memtmat
address[31..0]
mdata[31..0]
inst14
sumador
B[31..0]
A[31..0]
S[31..0]
A[31..0]
A[5..0]
A[15..0]
A[15..11]
A[20..16]
A[25..21]
A[31..26]
A[25..0]
inst15
mux2x5_5bits
sel
A[4..0]
B[4..0]
Q[4..0]
inst3
Desp2BitsIzq
entrada[26..0]
sortida[27..0]
S[31..0]
B
inst
UnitatControl
Op[5..0]
RegDest
Salsinc
MemaRReg
LlegMem
EscMem
SalCon
FontALU
ALUOP[1..0]
EscReg
inst1
extSig
entrada[15..0]
sortida[31..0]
inst2
BancRegistres
Dades[31..0]
Rdreg[4..0]
Rf1[4..0]
Rf2[4..0]
Esc
Reset
clk
Sor1[31..0]
Sor2[31..0]
S[31..26]
B_S[31..26]
inst4
ControlALU
CodiOp[1..0]
funcio[5..0]
inst7
mux2x5
sel
A[31..0]
B[31..0]
Q[31..0]


ALU32
inst5
Control_ALU[2..0]
Opd[31..0]
Result[31..0]
Z
C
AND2
inst9
inst10
memoria
inst17
clock
MemRead
MemWrite
address[7..0]
write_data[31..0]
read_data[31..0]
Desp2
inst11
entrada[31..0]
salida[31..0]
mux2x1
inst18
sel
A[31..0]
B[31..0]
O[31..0]
Sumador
inst20
inst11


inst1@
B[31..0]
A[31..0]
sel
mux2x1
O[31..0]

Disseny amb Codi

Per a la realització del modul microprocessador amb codi hem hagut d'assignar noms a tots els busos de dades i bits que relacionen uns moduls amb altres. En la seguent imatge es pot observar el nom de tots els busos que farem servir per l'implementació del codi.



2.5. Ex 5 - Simulació del projecte

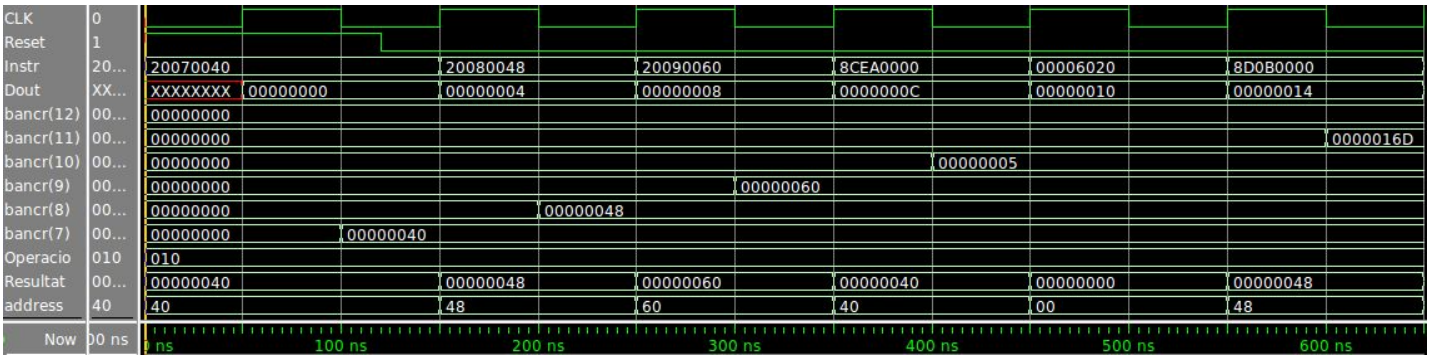
En aquest apartat explicarem com s'ha comportat els elements del nostre microprocessador a partir de les següents instruccions donades.

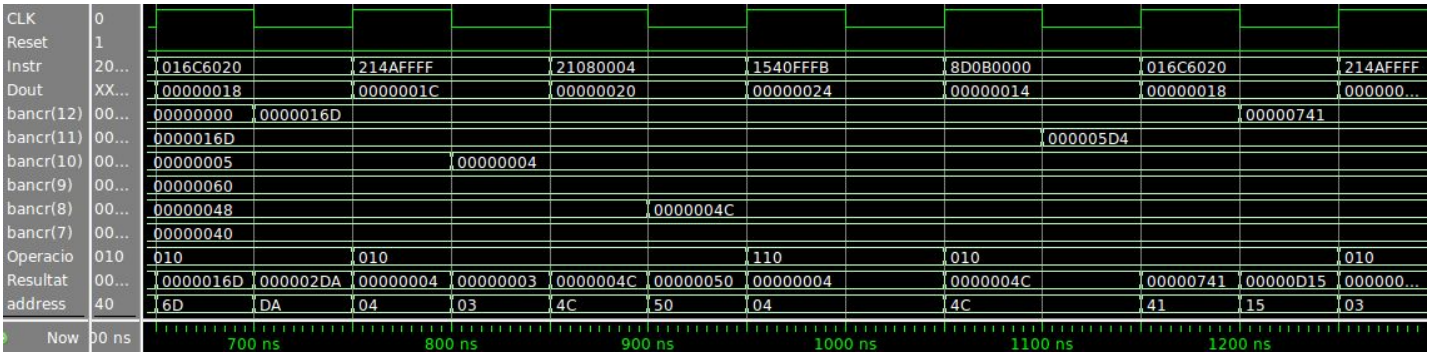
```
00: 20070040  addi $7, $0, 0x40
04: 20080048  addi $8, $0, 0x48
08: 20090060  addi $9, $0, 0x60
0c: 8cea0000  lw $10, 0($7)
10: 00006020  add $12, $0, $0
14: 8d0b0000  lw $11, 0($8)
18: 016c6020  add $12, $11, $12
1c: 214affff  addi $10, $10, -1
20: 21080004  addi $8, $8, 4
24: 1540fffb  bne $10, $0, -16
28: ad2C0000  sw $12, 0($9)
```

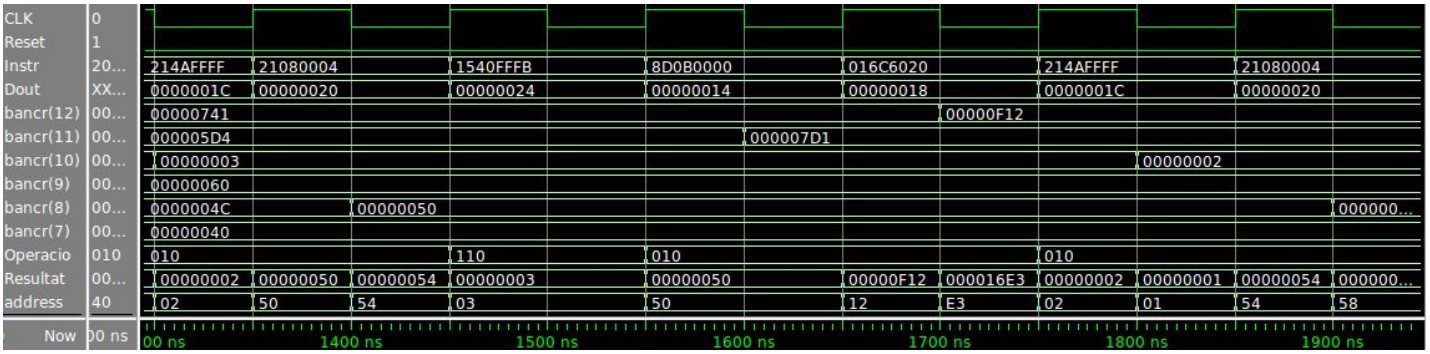
Memòria de dades a nivel de byte

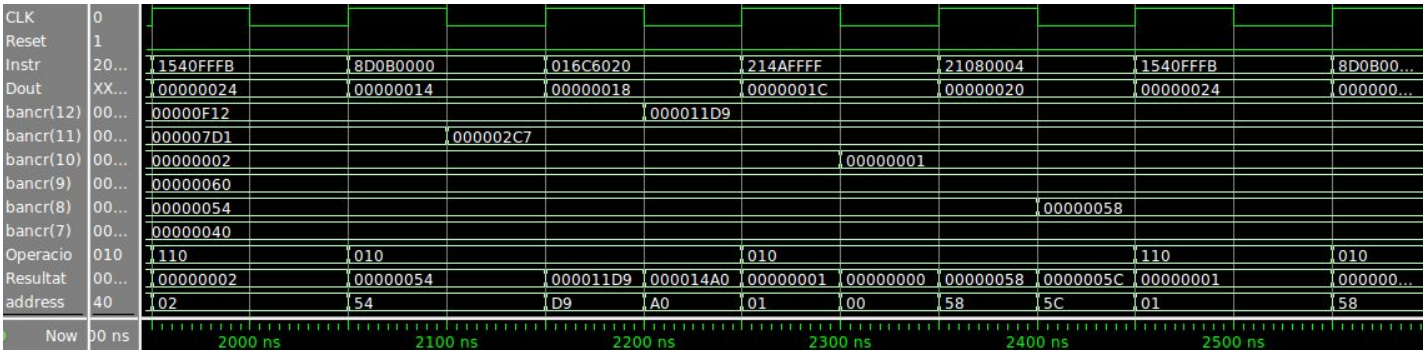
```
40H: 00000005        5       0x40: 00; 0x41: 00; 0x42:00; 0x43: 05
44H: 00000000
48H: 0000016d        365
4CH: 000005d4        1492
50H: 000007d1        2001
54H: 000002c7        711
58H: 00000799        1945
5CH: 00000000
60H: 00000000        0
```

Com podem observar a les imatges que hem adjuntat a continuació, el microprocessador es comporta de la manera desitjada. En les primeres instruccions podem veure com realitza correctament el carregament de diferents valors als registres 7, 8 i 9. A continuació ens realitza el load d'una posició de la memòria en el registre 10.
Anant comparant arribaríem a comprovar com es desenvolupen correctament totes les instruccions.









Per ultim en aquesta pràctica se'ns demanava que omplíssim els valors de la següent taula corresponents a l'ultima fase de l'última instrucció:

	Valor (hex)
PC	0x00000028
Instrucció	0xAD2C0000
Registre 7	0x00000040
Registre 8	0x0000005C
Registre 9	0x00000060
Registre 10	0x00000000
Registre 11	0x00000799
Registre 12	0x00001972
Registre 13	0x00000000

Adreces de memòria	
0x40	0x00000005
0x44	0x00000000
0x48	0x0000016D
0x4C	0x000005D4
0x50	0x000007D1
0x54	0x000002C7
0x58	0x00000799
0x5C	0x00000000
0x60	0x00001972

3. Codi complet

3.1. VHD

UnitatControl CPU

```
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 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       <= "10";
        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 "000100" =>  --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 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;
```

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 STD_LOGIC_VECTOR (2 downto 0) -- Sortida control alu
    );
end ControlALU;
```

```
architecture behav of ControlALU is
begin
        process (funcio,CodiOP)
        variable Control: STD_LOGIC_VECTOR(2 downto 0);
        begin
                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 process;
end behav;
```

microprocessador

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

entity microprocessador2 is
   port (
                        Reset : in STD_LOGIC;
                        CLK : in STD_LOGIC);
end microprocessador2;


---------------------------------------------------------------------------------------------

architecture Behavioral of microprocessador2 is
   component ALU32 is --fet
      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 component;

   component UnitatControl is --fet
      Port (
                        co : in STD_LOGIC_VECTOR (5 downto 0);
                        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 component;

component ControlALU is --fet
        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 component;


component sumador is --fet suma1
        Port (
                        A, B: in std_logic_vector(31 downto 0);
                        S: out std_logic_vector(31 downto 0)
    );
end component;

component mux2x1_5bits is --fet mux1
        Port (
                sel: in std_logic;
                A, B: in std_logic_vector(4 downto 0);
                O: out std_logic_vector(4 downto 0)
    );
end component;

component mux2x1 is --fet mux2
        Port (
                        sel: in std_logic;
                        A, B: in std_logic_vector(31 downto 0);
                        O: out std_logic_vector(31 downto 0)
    );
end component;


component MemInst is --fet
        Port (
                        address : in  STD_LOGIC_VECTOR (31 downto 0);
                        Instr : out  STD_LOGIC_VECTOR (31 downto 0)
    );
end component;

component memData is --fet
```

```
        Port (
                clock, MemRead, MemWrite: in std_logic;
                address: in std_logic_vector(7 downto 0);
                write_data: in std_logic_vector(31 downto 0);
                read_data: out std_logic_vector(31 downto 0)
    );
end component;

component extSig is
        Port (
            entrada: in std_logic_vector(15 downto 0);
            sortida: out std_logic_vector(31 downto 0)
    );
end component;

component ComptadorPrograma is
        Port (
                Reset : in STD_LOGIC;
                clk : in STD_LOGIC;
                Din : in STD_LOGIC_VECTOR(31 downto 0);
                Dout : out STD_LOGIC_VECTOR(31 downto 0)
    );
end component;


    component BancRegistres is
        Port (
                Rf1: in STD_LOGIC_VECTOR(4 downto 0);
                Rf2: in STD_LOGIC_VECTOR(4 downto 0);
                Rdest: in STD_LOGIC_VECTOR(4 downto 0);

                Sor1 : out STD_LOGIC_VECTOR(31 downto 0);
                Sor2 : out STD_LOGIC_VECTOR(31 downto 0);

                Dades: in STD_LOGIC_VECTOR(31 downto 0);

                Esc : in STD_LOGIC;
                Reset : in STD_LOGIC;

                clk : in STD_LOGIC
    );
end component;

-----------------------------------------------------------------------------------------------
    -- Senyals ALU32
signal D1 : SIGNED (31 downto 0);
signal D2 : SIGNED (31 downto 0);
signal carry : STD_LOGIC;
    signal zero : STD_LOGIC;
```

```
        signal adreca : SIGNED (31 downto 0);
        signal contALU : SIGNED (2 downto 0);

        -- Senyals UnitatControl
        signal RegDest : STD_LOGIC;
        signal SaltInc : STD_LOGIC;
        signal MemaReg : STD_LOGIC;
        signal LlegMem : STD_LOGIC;
        signal EscrMem : STD_LOGIC;
        signal SaltCon : STD_LOGIC;
        signal FontALU : STD_LOGIC;
        signal ALUOP : STD_LOGIC_VECTOR(1 downto 0) ;
        signal EscrReg : STD_LOGIC;

        -- senyals controlALU
        signal funcio : STD_LOGIC_VECTOR (31 downto 0);
        signal CodiOP : STD_LOGIC_VECTOR (1 downto 0);
        --signal contALU: signed (2 downto 0);

        --suma1 4+Dout
        signal Dout : std_logic_vector(31 downto 0); --inicialitzar a
0????????????????????????????????????????????
        signal S1 : std_logic_vector(31 downto 0);
        signal Const4 : std_logic_vector(31 downto 0) := "00000000000000000000000000000100";

        --mux1
        signal om1 : std_logic_vector(4 downto 0);
        --          signal funcio : STD_LOGIC_VECTOR (31 downto 0);

        --MemoriaPrograma
        --          signal Dout : std_logic_vector(31 downto 0);
        --          signal funcio : STD_LOGIC_VECTOR (31 downto 0);

        --ComptadorPrograma
        signal Din : STD_LOGIC_VECTOR(31 downto 0);
        --signal Dout : STD_LOGIC_VECTOR(31 downto 0);

        --BancRegistres
        --signal om1: STD_LOGIC_VECTOR(4 downto 0);
        signal sor2 : STD_LOGIC_VECTOR(31 downto 0);
        signal DadaEsc: STD_LOGIC_VECTOR(31 downto 0);

        --Extsigne
        signal sExt : std_logic_vector(31 downto 0);

        --mux2
        --signal D2: std_logic_vector(31 downto 0);

        --MemoriaPrograma
```

```
        signal readDat :std_logic_vector(31 downto 0);

        --Suma2
        signal S2 : std_logic_vector(31 downto 0);
        signal B2 : std_logic_vector(31 downto 0);

        --mux4
        signal S3 : std_logic_vector(31 downto 0);
        signal selmux4 : std_logic;

        --mux5
        signal B3: std_logic_vector (31 downto 0);

  begin

-------------------------------------------------------------------------------------------------

        selmux4 <= (SaltCon AND (not zero));
        B3 <= ((S1(31 downto 28))&(funcio(25 downto 0) & "00"));
        B2 <= (sExt(29 downto 0) & "00");

-------------------------------------------------------------------------------------------------

        ALU: ALU32 port map (
                Op1 => D1,
                Op2 => D2,
                Control_ALU => contALU,
                Resultat => adreca,
                Z => zero,
                C => carry);

        UC: UnitatControl port map(
                co => funcio(31 downto 26),
                RegDest => RegDest,
                SaltInc => SaltInc,
                MemaReg => MemaReg,
                LlegMem => LlegMem,
                EscrMem => EscrMem,
                SaltCon => SaltCon,
                FontALU => FontALU,
                ALUOP => ALUOP,
                EscrReg => EscrReg);

        ContrALU: ControlALU port map(
                funcio => funcio(5 downto 0),
                CodiOP => ALUOP,
                Operacio => contALU);

  suma1: sumador port map(
```

```
            A => Const4,
            B => Dout,
            S => S1);

        mux1: mux2x1_5bits port map(
            sel => RegDest,
            A => funcio(20 downto 16),
            B => funcio(15 downto 11),
            O => om1);

        PC: ComptadorPrograma port map(
            Reset => Reset,
            clk => CLK,
            Din => Din,
            Dout => Dout);

        BancReg: BancRegistres port map(
            Rf1 => funcio(25 downto 21),
            Rf2 => funcio(20 downto 16),
            Rdest => om1,
            signed(Sor1) => D1,
            Sor2 => sor2,
            Dades=> DadaEsc,
            Esc => EscrReg,
            Reset => Reset,
            clk => CLK
        );

ExSig : extSig port map(
            entrada => funcio(15 downto 0),
            sortida => sExt
        );

         mux2 : mux2x1 port map(
            sel => FontALU,
            A => Sor2,
            B => sExt,
            signed(O) => D2
        );

        MemoriaPrograma : MemInst port map(
            address => Dout,
            Instr => funcio
        );

        MemDades : memData port map(
            clock => clk,
            write_data => sor2,
            read_data => ReadDat,
```

```
                MemRead => LlegMem,
                MemWrite => EscrMem,
                address => std_logic_vector(adreca(7 downto 0))
        );

        mux3 : mux2x1 port map(
                sel => MemaReg,
                A => std_logic_vector(adreca),
                B => ReadDat,
                O => DadaEsc
        );

        suma2: sumador port map(
                A => S1,
                B => B2,
                S => S2
        );

        mux4 : mux2x1 port map(
                sel => selmux4,
                A => S1,
                B => S2,
                O => S3
        );

        mux5 : mux2x1 port map(
                sel => SaltInc,
                A => S3,
                B => B3,
                O => Din
        );

end architecture;
```

3.2. VHT (TestBench)

Clock de 100ns de periode.

UnitatControl CPU

```
wait for 100 ns;
co <= "000000"; --Add

wait for 100 ns;
co <= "100011"; --lw

wait for 100 ns;
co <= "000100"; --brne
```

ControlALU

```
CodiOP <= "00";
funcio <= "000000";
wait for 100 ns; -- 010

CodiOP <= "01";
funcio <= "000000";
wait for 100 ns; -- 110

CodiOP <= "10";
funcio <= "000000";
wait for 100 ns; --010

CodiOP <= "10";
funcio <= "000010";
wait for 100 ns; --110

CodiOP <= "10";
funcio <= "000100";
wait for 100 ns; --000

CodiOP <= "10";
funcio <= "000101";
wait for 100 ns; --001

CodiOP <= "10";
funcio <= "001010";
```

```
        wait for 100 ns; --111
```

microprocessador

En el testbench d'aquest modul nomes hem afegit el clock que sens proporciona a la pràctica i un reset inicial amb la durada que especifica la pràctica.

```
clock : PROCESS
-- variable declarations
BEGIN
        clk <= '0';
        wait for 50 ns;
        clk <= '1';
        wait for 50 ns;
END PROCESS clock;

always : PROCESS
-- optional sensitivity list
-- (        )
-- variable declarations
BEGIN
        reset <= '1';
        wait for 120 ns;
        reset <= '0';
        wait for 3200 ns;
WAIT;
END PROCESS always;
```