DISEÑO DE MICRO MIPS

* Registro MICRO
* **Librería:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

* **Entidad:**

entity RegsMIPS is

port (

**Clk :** in std\_logic; -- Reloj

**NRst** : in std\_logic; -- Reset asíncrono a nivel bajo

**A1** : in unsigned(4 downto 0); -- Dirección para el puerto Rd1

**Rd1** : out signed(31 downto 0); -- Dato del puerto Rd1

**A2** : in unsigned(4 downto 0); -- Dirección para el puerto Rd2

**Rd2** : out signed(31 downto 0); -- Dato del puerto Rd2

**A3** : in unsigned(4 downto 0); -- Dirección para el puerto Wd3

**Wd3** : in signed(31 downto 0); -- Dato de entrada Wd3

**We3** : in std\_logic -- Habilitación del banco de registros

);

end RegsMIPS;

* **Arquitectura:**

architecture Practica of RegsMIPS is

type regs\_t is array (0 to 31) of signed(31 downto 0);

signal regs : regs\_t;

begin

process (all)

begin

if NRst = '0' then

for i in 0 to 31 loop

regs(i) <= (others => '0');

end loop;

elsif (rising\_edge(clk) and We3 = '1' and A3 /= 0) then

regs(to\_integer(A3)) <= Wd3;

end if;

end process;

Rd1 <= regs(to\_integer(A1));

Rd2 <= regs(to\_integer(A2));

end Practica;

* ALU MICRO
* **Librería:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

* **Entidad:**

entity ALUMIPS is

port(

**Op1 :** in signed(31 downto 0);

**Op2 :** in signed(31 downto 0);

**ALUControl :** in std\_logic\_vector (2 downto 0);

**Res :** out signed(31 downto 0);

**Z :** out std\_logic

);

end ALUMIPS;

* **Arquitectura:**

architecture Practica of ALUMIPS is

begin

Process(all)

begin

case ALUControl is

when "000" => Res <= Op1 and Op2;

when "001" => Res <= Op1 or Op2;

when "010" => Res <= Op1 + Op2;

when "011" => Res <= Op1 xor Op2;

when "101" => Res <= Op1 nor Op2;

when "110" => Res <= Op1 - Op2;

when "111" =>

if Op1 < Op2 then

Res <= (0 => '1', others => '0');

else

Res <= (others => '0');

end if;

when others => res <= (others => '1');

end case;

end process;

Z <= '1' when Res = 0 else '0';

end Practica;

* UNIDAD DE CONTROL
* **Librería:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

* **Entidad:**

entity UnidadControl is port(

**OPCode :** in std\_logic\_vector (5 downto 0);

**Funct** : in std\_logic\_vector(5 downto 0);

**Jump** : out std\_logic;

**RegToPC** : out std\_logic;

**Branch** : out std\_logic;

**PCToReg** : out std\_logic;

**MemToReg** : out std\_logic;

**MemWrite** : out std\_logic;

**ALUSrc** : out std\_logic;

**ALUControl** : out std\_logic\_vector (2 downto 0);

**ExtCero** : out std\_logic;

**RegWrite** : out std\_logic;

**RegDest** : out std\_logic

);

end UnidadControl;

* **Arquitectura:**

architecture behavior of UnidadControl is

begin

ALUControl <= "000" **when** (OPCode = "000000" and Funct = "100100") **or** (OpCode = "001100") **else** -- and y andi

"001" **when** (OPCode = "000000" and Funct = "100101") **or** (OpCode = "001101") **else** -- or y ori

"010" **when** (OPCode = "000000" and Funct = "100000") or (OpCode = "001000") **or** (OpCode = "100011") **or** (OpCode = "101011") **else** -- add, addi, lw y sw

"011" **when** (OPCode = "000000" and Funct = "100110") **else** -- xor

"110" **when** (OPCode = "000000" and Funct = "100010") **or** (OpCode = "000100") **else** -- sub y beq

"111" **when** (OPCode = "000000" and Funct = "101010") or (OpCode = "001010") **else** - slt y slti

"100";

**Jump** <= '1' when OPCode = "000010" else '0';

**Branch** <= '1' when OPCode = "000100" else '0';

**MemToReg** <= '1' when OPCode = "100011" else '0';

**MemWrite** <= '1' when OPCode = "101011" else '0';

**ALUSrc** <= '0' when OPCode = "000000" or OPCode = "000100" else '1';

**ExtCero** <= '1' when OPCode = "001100" or OPCode = "001101" else '0';

**RegWrite** <= '0' when OPCode = "000010" or OPCode = "000100" or

**OPCode** = "101011" else '1';

**RegDest** <= '1' when OPCode = "000000" else '0';

end behavior;

* MICRO
* **Librería:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

* **Entidad:**

entity MicroMIPS is port (

**Clk** : in std\_logic; -- Reloj

**NRst** : in std\_logic; -- Reset activo a nivel bajo

**MemProgAddr** : out unsigned(31 downto 0); Ç

**MemProgData** : in unsigned(31 downto 0);

**MemDataAddr** : out unsigned(31 downto 0);

**MemDataDataRead** : in signed(31 downto 0);

**MemDataDataWrite :** out signed(31 downto 0);

**MemDataWE :** out std\_logic

);

end MicroMIPS;

* **Arquitectura:**

architecture Test OF MicroMIPS is

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

-- Declaración de los componentes y de las señales auxiliares

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

-- Banco Registros

component RegsMIPS is

port (

**Clk :** in std\_logic; CLK

**NRst** : in std\_logic; Nrst

**A1** : in unsigned(4 downto 0); MemProgData ( a downto b)

**Rd1** : out signed(31 downto 0); AUX ALUA

**A2** : in unsigned(4 downto 0); MemProgData ( a downto b)

**Rd2** : out signed(31 downto 0); MemDataDataWrite

**A3** : in unsigned(4 downto 0); AUX A3\_aux

**Wd3** : in signed(31 downto 0); AUX Wd3\_aux

**We3** : in std\_logic regwrite

);

end component;

-- ALU

component ALUMIPS is

port(

**Op1** : in signed(31 downto 0); AUX ALUA

**Op2** : in signed(31 downto 0); AUX ALUB

**ALUControl** : in std\_logic\_vector (2 downto 0); AUX alucontrol

**Res** : out signed(31 downto 0); AUX ALUC

**Z** : out std\_logic AUX ALUZ

);

end component;

-- Unidad de Control

component UnidadControl is port(

**OPCode** : in std\_logic\_vector (5 downto 0);

**Funct** : in std\_logic\_vector(5 downto 0);

**Jump** : out std\_logic;

**Branch** : out std\_logic;

**MemToReg** : out std\_logic;

**MemWrite** : out std\_logic;

**ALUSrc** : out std\_logic;

**ALUControl** : out std\_logic\_vector (2 downto 0);

**ExtCero** : out std\_logic;

**RegWrite** : out std\_logic;

**RegDest** : out std\_logic

);

end component;

-- Señales auxiliares

**signal** Instr\_Sig, ALUA , ALUB, ALUC, Wd3\_aux, ExtSigno, ExtCero\_aux: signed(31 downto 0);

**signal** PC, pcmas4, BTA, JTA: unsigned(31 downto 0);

**signal** Instr\_Slv : std\_logic\_vector(31 downto 0);

**signal** A3\_aux : unsigned(4 downto 0);

**signal** jump, branch, memtoreg, memwrite, alusrc, extcero, regwrite, regdst, z, pcsrc : std\_logic;

**signal** alucontrol : std\_logic\_vector(2 downto 0);

begin

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

-- Instanciar los componentes usados

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

Instr\_Slv <= std\_logic\_vector(MemProgData);

Instr\_Sig <= signed(MemProgData);

**GPR**: RegsMIPS port map (

**Clk**=>clk, **NRst**=>nRst, **A1**=>MemProgData(25 downto 21), **Rd1**=>ALUA, **A2**=>MemProgData(20 downto 16), **Rd2**=>MemDataDataWrite, **A3**=>A3\_aux, **Wd3**=>Wd3\_aux, **We3**=>regwrite

);

**ALU**: ALUMIPS port map (

**Op1**=>ALUA, **Op2**=>ALUB, **ALUControl**=>alucontrol, **Res**=>ALUC, **Z**=>z

);

**UC**: UnidadControl port map (

**OPCode**=>Instr\_Slv(31 downto 26), **Funct**=>Instr\_Slv(5 downto 0), **Jump**=>jump, **Branch**=>branch, **MemToReg**=>memtoreg, **MemWrite**=>memwrite, **ALUSrc**=>alusrc, **ALUControl**=>alucontrol, **ExtCero**=>extcero, **RegWrite**=>regwrite, **RegDest**=>regdst

);

-- Diseño de la extension de signo

**ExtSigno** <= resize(Instr\_Sig(15 downto 0), 32);

-- Diseño de la extension de cero

**ExtCero\_aux** <= signed(resize(MemProgData(15 downto 0), 32));

-- Diseño de los dos MUX para la entrada ALUB

**ALUB** <= MemDataDataWrite when alusrc = '0' else ExtCero\_aux when extcero = '1' else ExtSigno;

-- Diseño del MUX para la entrada A3

**A3\_aux** <= MemProgData(15 downto 11) when regdst = '1' else MemProgData(20 downto 16);

-- Diseño del MUX para la entrada Wd3

**Wd3\_aux** <= ALUC when memtoreg = '0' else MemDataDataRead;

-- Señales para la ruta del PC, secuencial y saltos condicionales e incondicionales

**pcsrc** <= branch and z ;

**BTA** <= pcmas4 + unsigned(ExtSigno(29 downto 0) & "00");

**pcmas4** <= PC + 4 ;

**JTA** <= pcmas4(31 downto 28) & MemProgData(25 downto 0) & "00" ;

-- Señales de acceso a la memoria de datos

**MemDataAddr** <= unsigned(ALUC) ;

**MemDataWE** <= memwrite;

-- Diseño de la ruta del PC

Process(all)

begin

if nRst = '0' then PC <= (others => '0');

elsif rising\_edge(clk) then

if jump = '1' then

PC <= JTA ;

elsif pcsrc = '1' then

PC <= BTA ;

else PC <= pcmas4 ;

end if;

end if;

end process;

MemProgAddr <= PC;

end Test;