Instituto Politécnico Nacional

Escuela Superior de Computo

Proyecto.

Procesador ESCOMips.

Nombre: Flores Castro Luis Antonio.

Arquitectura de Computadoras.

Profesora: Vega García Nayeli.

Link de Video de entrega del Proyecto.

https://drive.google.com/file/d/1h6rrtgVOoZrxmgsJPj5ObWNkRDJtDh7w/view?usp=sharing

Código de Paquete del Procesador ESCOMips.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

package paqueteRD is


component UnidadControl is
      Port(clk, clr: in STD_LOGIC;
            opCode: in STD_LOGIC_VECTOR(4 downto 0);
            funCode, banderas: in STD_LOGIC_VECTOR(3 downto 0);
            microInstruccion: out STD_LOGIC_VECTOR(19 downto 0));
end component;

component PilaH is
    Port ( pcIn : in STD_LOGIC_VECTOR (15 downto 0);
           clk, clr, wPC, UP, DW : in STD_LOGIC;
           pcOut : out STD_LOGIC_VECTOR (15 downto 0));
end component;

--memoria de programa
component dataprogram is
    Port ( PC : in STD_LOGIC_VECTOR (9 downto 0);
           ints : out STD_LOGIC_VECTOR (24 downto 0));
end component;

--Archivo de Registros
component FileReg is
      Port( writeReg, readReg1, readReg2, shamt: in STD_LOGIC_VECTOR(3
downto 0);
            writeData: in STD_LOGIC_VECTOR(15 downto 0);
            clk,clr,wr,she,dir: in STD_LOGIC;
            readData1, readData2: out STD_LOGIC_VECTOR(15 downto 0));
end component;

--ALU
component ALU4bits is
Port( a : in STD_LOGIC_VECTOR(15 downto 0);
      b : in STD_LOGIC_VECTOR(15 downto 0);
      Aluop : in STD_LOGIC_VECTOR(3 downto 0);
      Res : inout STD_LOGIC_VECTOR(15 downto 0);
      bandera : out STD_LOGIC_VECTOR(3 downto 0)
      --Cout : out STD_LOGIC
      );
end component;

--Memoria de Datos

component datamemory is
      Port(dir:in STD_LOGIC_VECTOR(9 downto 0);
      dataIn: in STD_LOGIC_VECTOR(15 downto 0);
      WD, clk: STD_LOGIC;
      dataOut: out STD_LOGIC_VECTOR(15 downto 0));
end component;
```

```
--Extensor de Signo
component ExtensorSigno is
      Port(sliti: in STD_LOGIC_VECTOR(11 downto 0);
            slito: out STD_LOGIC_VECTOR(15 downto 0));
end component;

--Extensor de Direccion
component ExtensorDireccion is
Port( dliti: in STD_LOGIC_VECTOR(11 downto 0);
      dlito: out STD_LOGIC_VECTOR(15 downto 0));
end component;

--mux de 4 bits
component mux4b is
Port(A,B: in std_logic_vector(3 downto 0);
      selector: in STD_LOGIC;--SOLO UN BIT
      salida: out STD_LOGIC_VECTOR(3 downto 0));
end component;

--mux de 16 bits
component mux16b is
Port(A,B: in std_logic_vector(15 downto 0);
      selector: in STD_LOGIC;--SOLO UN BIT
      salida: out STD_LOGIC_VECTOR(15 downto 0));
end component;



component muxSR2 is
    Port ( A, B : in STD_LOGIC_VECTOR (3 downto 0);
           selector : in STD_LOGIC;
           salida : out STD_LOGIC_VECTOR (3 downto 0));
end component;

end package;
```

Código de Implementación del Procesador ESCOMips.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.paqueteRD.all;

entity RutaDatos is
      Port(rclr, clk: in STD_LOGIC;
      pc: out STD_LOGIC_VECTOR(9 downto 0);
      Instruccion: out STD_LOGIC_VECTOR(24 downto 0);
      ReD1, ReD2, RESALU: out STD_LOGIC_VECTOR(15 downto 0));
end RutaDatos;

architecture Behavioral of RutaDatos is

signal AUXSDMP, AUXSWD, AUXSEXT, AUXSOP1, AUXSOP2, AUXSDMD, AUXSR,
AUXPCOUT:STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal AUXINST:STD_LOGIC_VECTOR(24 downto 0):=(others=>'0');
signal clr: STD_LOGIC:='0';
signal AUXBANDERAS, AUXSR2: STD_LOGIC_VECTOR(3 downto 0):=(others=>'0');
signal AUXMINSTRUCCION: STD_LOGIC_VECTOR(19 downto 0):=(others=>'0');
signal AUXREADDATA1, AUXREADDATA2: STD_LOGIC_VECTOR(15 downto
0):=(others=>'0');
signal AUXEXTDIR, AUXEXTSIG, AUXRESALU, AUXDATAOUT: STD_LOGIC_VECTOR(15
downto 0):=(others=>'0');

begin

process(clk)
begin
if(falling_edge(clk))then
      clr<=rclr;
end if;
end process;
--AQUI VAN LAS CONEXIONES YA

--entidad para Unidad de Control
UCtr: UnidadControl
Port map(
clk=>clk,
clr=>clr,
opCode=>AUXINST(24 downto 20),
funCode=>AUXINST(3 downto 0),
banderas=>AUXBANDERAS,
microInstruccion=>AUXMINSTRUCCION
);

--entidad para pila
sP: PilaH
Port map(
pcIn=>AUXSDMP,
clk=>clk,
clr=>clr,
wPC=>AUXMINSTRUCCION(16),
UP=>AUXMINSTRUCCION(18),
DW=>AUXMINSTRUCCION(17),
pcOut=>AUXPCOUT
```

```
);

--CONEXION MEMORIA DE PROGRAMA
mpro: dataprogram
Port map(
PC=>AUXPCOUT(9 downto 0),
ints=>AUXINST
);

--multiplexor SR2
SR2: muxsr2
Port map(
A=>AUXINST(11 downto 8),
B=>AUXINST(19 downto 16),
selector=>AUXMINSTRUCCION(15),
salida=>AUXSR2
);

--MULTIPLEXOR SWD
SWD: mux16b
Port map(
A=>AUXINST(15 downto 0),
B=>AUXSR,
selector=>AUXMINSTRUCCION(14),
salida=>AUXSWD
);

--CONEXION DE ARCHIVO DE REGISTROS
AR: FileReg
Port map(
wr=>AUXMINSTRUCCION(10),
dir=>AUXMINSTRUCCION(11),
she=>AUXMINSTRUCCION(12),
clk=>clk,
clr=>clr,
writeReg=>AUXINST(19 downto 16),
readReg1=>AUXINST(15 downto 12),
readReg2=>AUXSR2,
shamt=>AUXINST(7 downto 4),
writeData=>AUXSWD,
readData1=>AUXREADDATA1,
readData2=>AUXREADDATA2
);

--CONEXION DE EXTENSORES
ESIGNO: ExtensorSigno
Port map(
sliti=>AUXINST(11 downto 0),
slito=>AUXEXTSIG
);

EDIR : ExtensorDireccion
Port map(
dliti=>AUXINST(11 downto 0),
dlito=>AUXEXTDIR
);
```

```
--MULTIPLEXOR SEXT UNIDAD A EXTENSORES
SEXT: mux16b
Port map(
A=>AUXEXTSIG,
B=>AUXEXTDIR,
selector=>AUXMINSTRUCCION(13),
salida=>AUXSEXT
);

--MULTIPLEXOR SOP1 Y SOP2, UNIDAD A ARCHIVO DE REGISTROS Y ALU
SOP1: mux16b
Port map(
A=>AUXREADDATA1,
B=>AUXPCOUT,
selector=>AUXMINSTRUCCION(9),
salida=>AUXSOP1
);

--MULTIPLEXOR SOP1 UNIDAD A ARCHIVO DE REGISTROS Y ALU
SOP2: mux16b
Port map(
A=>AUXREADDATA2,
B=>AUXSEXT,
selector=>AUXMINSTRUCCION(8),
salida=>AUXSOP2
);

--CONEXION DE ALU

ALU: ALU4bits
Port map( a=>AUXSOP1,
      b=>AUXSOP2,
      Aluop=>AUXMINSTRUCCION(7 downto 4),
      Res=>AUXRESALU,
      bandera=>AUXBANDERAS
      );

--MULTIPLEXOR SDMD A MEMORIA DE DATOS
SDMD: mux16b
Port map(
A=>AUXRESALU,
B=>AUXINST(15 downto 0),
selector=>AUXMINSTRUCCION(3),
salida=>AUXSDMD
);

--CONEXION MEMORIA DE DATOS
mdata: datamemory
Port map(
dir=>AUXSDMD(9 downto 0),
dataIn=>AUXREADDATA2,
WD=>AUXMINSTRUCCION(2),
clk=>clk,
dataOut=>AUXDATAOUT
);

--MULTIPLEXOR SR SALIDA DE MEMORIA
```

```
SR: mux16b
Port map(
A=>AUXDATAOUT,
B=>AUXRESALU,
selector=>AUXMINSTRUCCION(1),
salida=>AUXSR
);

--MULTIPLEXOR SMDP
SDMP: mux16b
Port map(
A=>AUXINST(15 downto 0),
B=>AUXSR,
selector=>AUXMINSTRUCCION(19),
salida=>AUXSDMP
);

pc<=AUXPCOUT(9 downto 0);
Instruccion<=AUXINST;
ReD1<=AUXREADDATA1;
ReD2<=AUXREADDATA2;
RESALU<=AUXRESALU;



end Behavioral;
```

Código Test Bench del Procesador ESCOMips.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TbProyecto is
end TbProyecto;

architecture Behavioral of TbProyecto is

component RutaDatos is
      Port(rclr, clk: in STD_LOGIC;
      pc: out STD_LOGIC_VECTOR(9 downto 0);
      Instruccion: out STD_LOGIC_VECTOR(24 downto 0);
      ReD1, ReD2, RESALU: out STD_LOGIC_VECTOR(15 downto 0));
end component;

      signal rclr: STD_LOGIC:='0';
      signal clk: STD_LOGIC:='0';
      signal pc: STD_LOGIC_VECTOR(9 downto 0):=(others=>'0');
      signal Instruccion: STD_LOGIC_VECTOR(24 downto 0):=(others=>'0');
      signal ReD1: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
      signal ReD2: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
      signal RESALU: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');

begin

procesador: RutaDatos
      Port map(
      rclr=>rclr,
      clk=>clk,
      pc=>pc,
      Instruccion=>Instruccion,
      ReD1=>ReD1,
      ReD2=>ReD2,
      RESALU=>RESALU
      );

reloj: process
begin
      clk<='1';
      wait for 5 ns;
      clk<='0';
      wait for 5 ns;
end process;

reloj2: process
begin
      rclr<='1';
      wait for 10 ns;
      rclr<='0';
      wait;
end process;

end Behavioral;
```

Código de Implementación de Memoria de Programa para Programa 1.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity dataprogram is
    generic ( m : integer := 10;
              n : integer := 25);
    Port ( PC : in STD_LOGIC_VECTOR (m-1 downto 0);
           ints : out STD_LOGIC_VECTOR (n-1 downto 0));
end dataprogram;

architecture Behavioral of dataprogram is
    --Instrucciones tipo R
    constant TR : STD_LOGIC_VECTOR(4 downto 0):="00000";
    --Instrucciones de Carga y Almacenamiento
    constant LI:STD_LOGIC_VECTOR(4 downto 0):="00001";
    constant LWI:STD_LOGIC_VECTOR(4 downto 0):="00010";
    constant LW:STD_LOGIC_VECTOR(4 downto 0):="10111";
    constant SWI:STD_LOGIC_VECTOR(4 downto 0):="00011";
    constant SW:STD_LOGIC_VECTOR(4 downto 0):="00100";

    --Instrucciones Aritmeticas
    constant ADD:STD_LOGIC_VECTOR(3 downto 0):="0000";--R
    constant SUB:STD_LOGIC_VECTOR(3 downto 0):="0001";--R
    constant ADDI:STD_LOGIC_VECTOR(4 downto 0):="00101";
    constant SUBI:STD_LOGIC_VECTOR(4 downto 0):="00110";

    --Instrucciones Lógicas
    constant OPAND:STD_LOGIC_VECTOR(3 downto 0):="0010";
    constant OPOR:STD_LOGIC_VECTOR(3 downto 0):="0011";
    constant OPXOR:STD_LOGIC_VECTOR(3 downto 0):="0100";
    constant OPNAND:STD_LOGIC_VECTOR(3 downto 0):="0101";
    constant OPNOR:STD_LOGIC_VECTOR(3 downto 0):="0110";
    constant OPXNOR:STD_LOGIC_VECTOR(3 downto 0):="0111";
    constant OPNOT:STD_LOGIC_VECTOR(3 downto 0):="1000";
    constant ANDI:STD_LOGIC_VECTOR(4 downto 0):="00111";
    constant ORI:STD_LOGIC_VECTOR(4 downto 0):="01000";
    constant XORI:STD_LOGIC_VECTOR(4 downto 0):="01001";
    constant NANDI:STD_LOGIC_VECTOR(4 downto 0):="01010";
    constant NORI:STD_LOGIC_VECTOR(4 downto 0):="01011";
    constant XNORI:STD_LOGIC_VECTOR(4 downto 0):="01100";

    --Instrucciones de Corrimiento
    constant OPSLL:STD_LOGIC_VECTOR(3 downto 0):="1001";
    constant OPSRL:STD_LOGIC_VECTOR(3 downto 0):="1010";
    --constant OPR:STD_LOGIC_VECTOR(4 downto 0):="00000";

    --Instrucciones de Saltos Condicionales e Incondicionales
    constant BEQI:STD_LOGIC_VECTOR(4 downto 0):="01101";
    constant BNEI:STD_LOGIC_VECTOR(4 downto 0):="01110";
    constant BLTI:STD_LOGIC_VECTOR(4 downto 0):="01111";
    constant BLETI:STD_LOGIC_VECTOR(4 downto 0):="10000";
    constant BGTI:STD_LOGIC_VECTOR(4 downto 0):="10001";
    constant BGETI:STD_LOGIC_VECTOR(4 downto 0):="10010";
```

```
    constant B:STD_LOGIC_VECTOR(4 downto 0):="10011";

    --Instrucciones de Manejo de Subrutinas
    constant CALL:STD_LOGIC_VECTOR(4 downto 0):="10100";
    constant RET:STD_LOGIC_VECTOR(4 downto 0):="10101";

    --Otras Instrucciones
    constant NOP:STD_LOGIC_VECTOR(4 downto 0):="10110";
    constant SU:STD_LOGIC_VECTOR(3 downto 0):="0000"; -- Sin usar 4 bits

    --Registros
    constant R0 : STD_LOGIC_VECTOR (3 downto 0) := "0000";
    constant R1 : STD_LOGIC_VECTOR (3 downto 0) := "0001";
    constant R2 : STD_LOGIC_VECTOR (3 downto 0) := "0010";
    constant R3 : STD_LOGIC_VECTOR (3 downto 0) := "0011";
    constant R4 : STD_LOGIC_VECTOR (3 downto 0) := "0100";
    constant R5 : STD_LOGIC_VECTOR (3 downto 0) := "0101";
    constant R6 : STD_LOGIC_VECTOR (3 downto 0) := "0110";
    constant R7 : STD_LOGIC_VECTOR (3 downto 0) := "0111";
    constant R8 : STD_LOGIC_VECTOR (3 downto 0) := "1000";

    type mem is array (0 to (2**m)-1) of STD_LOGIC_VECTOR(n-1 downto 0);
    constant aux : mem := (--aqui nace la
      --PRIMER PROGRAMA
       LI & R0 & x"0001",
       LI & R1 & x"0007",
       TR & R1 & R1 & R0 & SU & ADD,
       SWI & R1 & x"0005",
       B & SU & x"0002",
    --PROMEDIO DE UN ARREGLO DE 8 NUMEROS
    --LI & R0 & x"0001",--0
    --LI & R1 & x"0008",--1
    --LI & R2 & x"0000",--2
    --SW & R0 & R2 & x"128",--3
    --ADDI & R0 & R0 & x"001",--4
    --ADDI & R2 & R2 & x"001",--5
    --BLTI & R1 & R2 & x"ffd",--6
    --LI & R2 & x"0000",--7
    --LI & R3 & x"0000",--8
    --LW & R4 & R2 & x"128",--9
    --TR & R3 & R3 & R4 & SU & ADD,--10
    --ADDI & R2 & R2 & x"001",--11
    --BLTI & R1 & R2 & x"ffd",--12
    --TR & R3 & R3 & SU & x"3" & OPSRL,--13
    --SWI & R3 & x"0260",--14
    --NOP & SU & SU & SU & SU & SU,--15
    --B & SU & x"000F",--16

       others => (others => '0')

    );
begin
    ints <= aux(conv_integer(PC));
end Behavioral;
```

Código de Implementación de Memoria de Programa para Programa 2 (Promedio de 8 números de un arreglo).

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity dataprogram is
    generic ( m : integer := 10;
              n : integer := 25);
    Port ( PC : in STD_LOGIC_VECTOR (m-1 downto 0);
           ints : out STD_LOGIC_VECTOR (n-1 downto 0));
end dataprogram;

architecture Behavioral of dataprogram is
    --Instrucciones tipo R
    constant TR : STD_LOGIC_VECTOR(4 downto 0):="00000";
    --Instrucciones de Carga y Almacenamiento
    constant LI:STD_LOGIC_VECTOR(4 downto 0):="00001";
    constant LWI:STD_LOGIC_VECTOR(4 downto 0):="00010";
    constant LW:STD_LOGIC_VECTOR(4 downto 0):="10111";
    constant SWI:STD_LOGIC_VECTOR(4 downto 0):="00011";
    constant SW:STD_LOGIC_VECTOR(4 downto 0):="00100";

    --Instrucciones Aritmeticas
    constant ADD:STD_LOGIC_VECTOR(3 downto 0):="0000";--R
    constant SUB:STD_LOGIC_VECTOR(3 downto 0):="0001";--R
    constant ADDI:STD_LOGIC_VECTOR(4 downto 0):="00101";
    constant SUBI:STD_LOGIC_VECTOR(4 downto 0):="00110";

    --Instrucciones Lógicas
    constant OPAND:STD_LOGIC_VECTOR(3 downto 0):="0010";
    constant OPOR:STD_LOGIC_VECTOR(3 downto 0):="0011";
    constant OPXOR:STD_LOGIC_VECTOR(3 downto 0):="0100";
    constant OPNAND:STD_LOGIC_VECTOR(3 downto 0):="0101";
    constant OPNOR:STD_LOGIC_VECTOR(3 downto 0):="0110";
    constant OPXNOR:STD_LOGIC_VECTOR(3 downto 0):="0111";
    constant OPNOT:STD_LOGIC_VECTOR(3 downto 0):="1000";
    constant ANDI:STD_LOGIC_VECTOR(4 downto 0):="00111";
    constant ORI:STD_LOGIC_VECTOR(4 downto 0):="01000";
    constant XORI:STD_LOGIC_VECTOR(4 downto 0):="01001";
    constant NANDI:STD_LOGIC_VECTOR(4 downto 0):="01010";
    constant NORI:STD_LOGIC_VECTOR(4 downto 0):="01011";
    constant XNORI:STD_LOGIC_VECTOR(4 downto 0):="01100";

    --Instrucciones de Corrimiento
    constant OPSLL:STD_LOGIC_VECTOR(3 downto 0):="1001";
    constant OPSRL:STD_LOGIC_VECTOR(3 downto 0):="1010";
    --constant OPR:STD_LOGIC_VECTOR(4 downto 0):="00000";

    --Instrucciones de Saltos Condicionales e Incondicionales
    constant BEQI:STD_LOGIC_VECTOR(4 downto 0):="01101";
    constant BNEI:STD_LOGIC_VECTOR(4 downto 0):="01110";
    constant BLTI:STD_LOGIC_VECTOR(4 downto 0):="01111";
    constant BLETI:STD_LOGIC_VECTOR(4 downto 0):="10000";
    constant BGTI:STD_LOGIC_VECTOR(4 downto 0):="10001";
```

```
    constant BGETI:STD_LOGIC_VECTOR(4 downto 0):="10010";
    constant B:STD_LOGIC_VECTOR(4 downto 0):="10011";

    --Instrucciones de Manejo de Subrutinas
    constant CALL:STD_LOGIC_VECTOR(4 downto 0):="10100";
    constant RET:STD_LOGIC_VECTOR(4 downto 0):="10101";

    --Otras Instrucciones
    constant NOP:STD_LOGIC_VECTOR(4 downto 0):="10110";
    constant SU:STD_LOGIC_VECTOR(3 downto 0):="0000"; -- Sin usar 4 bits

    --Registros
    constant R0 : STD_LOGIC_VECTOR (3 downto 0) := "0000";
    constant R1 : STD_LOGIC_VECTOR (3 downto 0) := "0001";
    constant R2 : STD_LOGIC_VECTOR (3 downto 0) := "0010";
    constant R3 : STD_LOGIC_VECTOR (3 downto 0) := "0011";
    constant R4 : STD_LOGIC_VECTOR (3 downto 0) := "0100";
    constant R5 : STD_LOGIC_VECTOR (3 downto 0) := "0101";
    constant R6 : STD_LOGIC_VECTOR (3 downto 0) := "0110";
    constant R7 : STD_LOGIC_VECTOR (3 downto 0) := "0111";
    constant R8 : STD_LOGIC_VECTOR (3 downto 0) := "1000";

    type mem is array (0 to (2**m)-1) of STD_LOGIC_VECTOR(n-1 downto 0);
    constant aux : mem := (--aqui nace la
      --PRIMER PROGRAMA
--       LI & R0 & x"0001",
--       LI & R1 & x"0007",
--       TR & R1 & R1 & R0 & SU & ADD,
--       SWI & R1 & x"0005",
--       B & SU & x"0002",
    --PROMEDIO DE UN ARREGLO DE 8 NUMEROS
    LI & R0 & x"0001",--0
    LI & R1 & x"0008",--1
    LI & R2 & x"0000",--2
    SW & R0 & R2 & x"128",--3
    ADDI & R0 & R0 & x"001",--4
    ADDI & R2 & R2 & x"001",--5
    BLTI & R1 & R2 & x"ffd",--6
    LI & R2 & x"0000",--7
    LI & R3 & x"0000",--8
    LW & R4 & R2 & x"128",--9
    TR & R3 & R3 & R4 & SU & ADD,--10
    ADDI & R2 & R2 & x"001",--11
    BLTI & R1 & R2 & x"ffd",--12
    TR & R3 & R3 & SU & x"3" & OPSRL,--13
    SWI & R3 & x"0260",--14
    NOP & SU & SU & SU & SU & SU,--15
    B & SU & x"000F",--16

       others => (others => '0')

    );
begin
    ints <= aux(conv_integer(PC));
end Behavioral;
```

Diagrama RTL del Procesador ESCOMips.

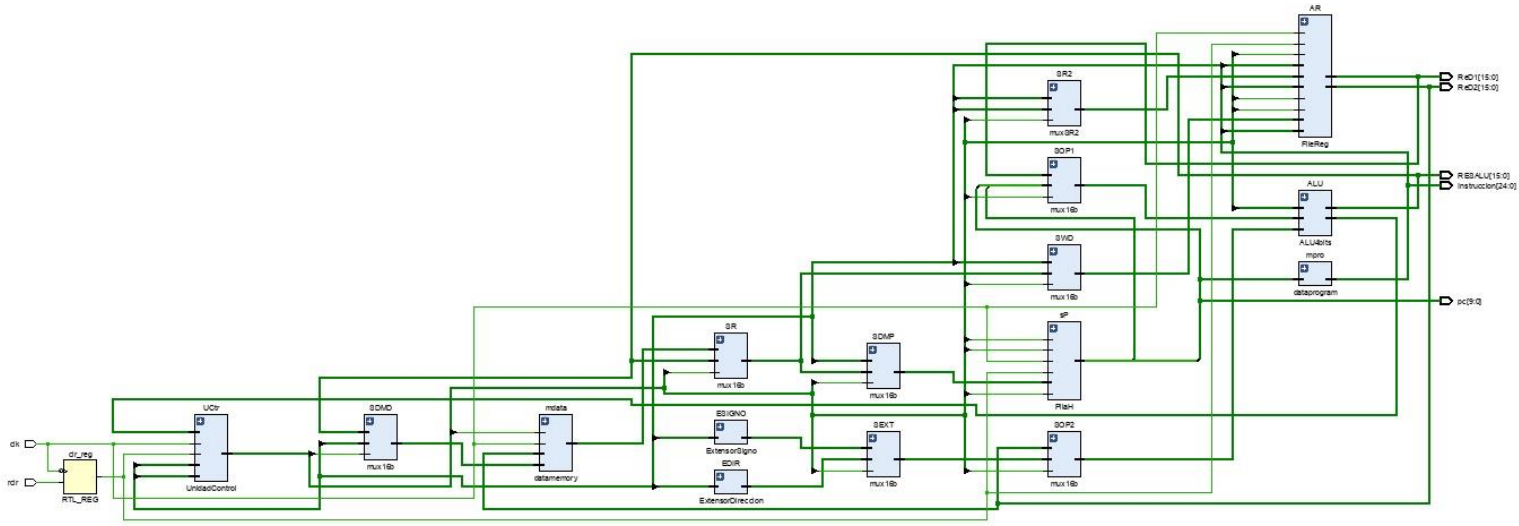

Figura 1. Diagrama RTL del Procesador ESCOMips (Vista General).

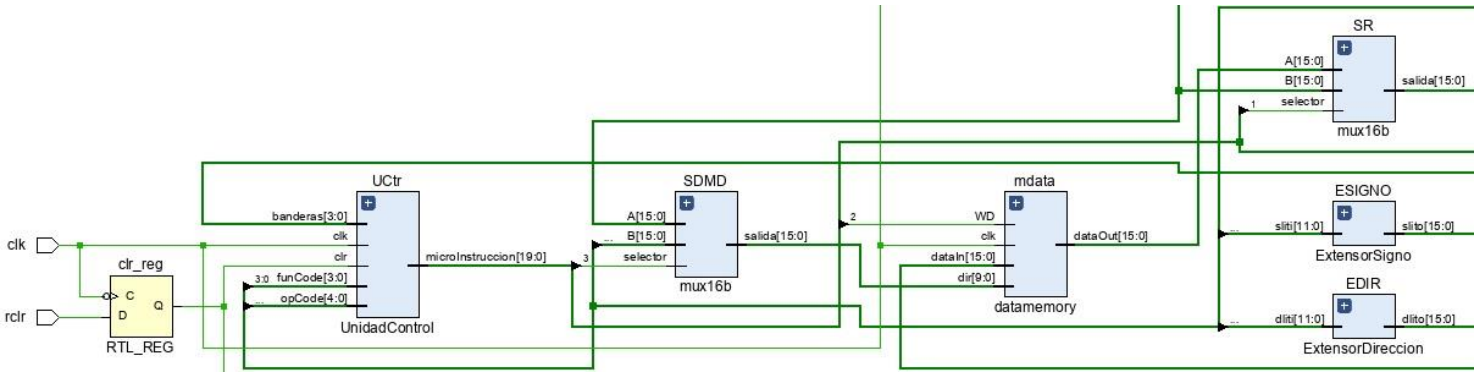

Figura 1.1. Diagrama RTL del Procesador ESCOMips (Vista Inicial).

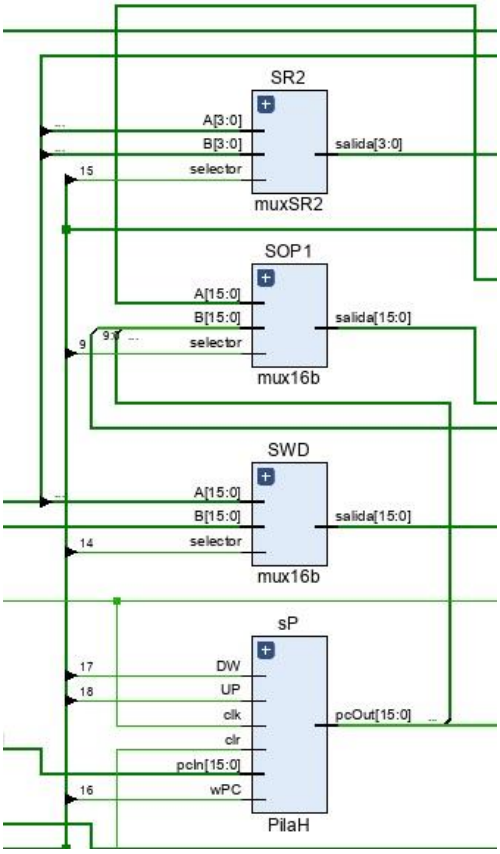

Figura 1.2. Diagrama RTL del Procesador ESCOMips (Vista Media).

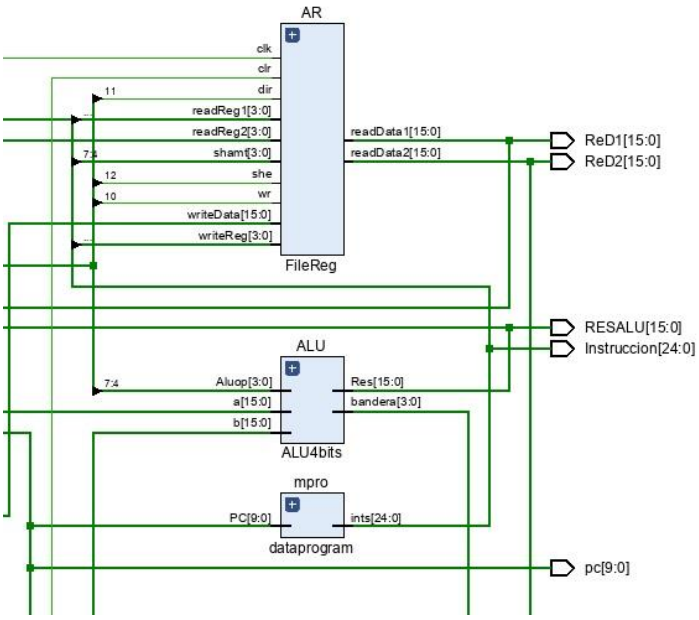

Figura 1.3. Diagrama RTL del Procesador ESCOMips (Vista Final).

Diagrama Logico del Procesador ESCOMips.

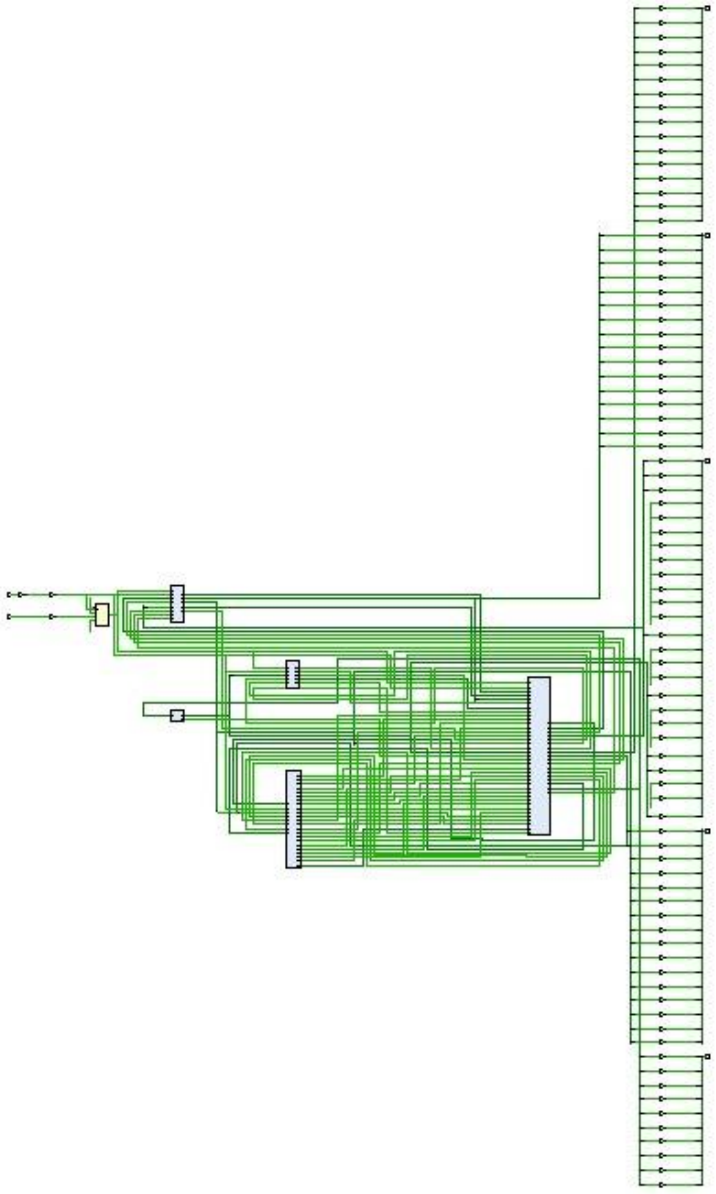

Figura 2. Diagrama Lógico del Procesador ESCOMips.

Memoria de Programa del Programa 2 (Promedio de 8 números de un arreglo).

PC	Instrucción	Código memoria de programa ESCOMips
0	LI R0, #1	LI & R0 & x”0001”
1	LI R1, #8	LI & R1 & x”0008”
2	LI R2, #0	LI & R2 & x”0000”
3	SW R0, (0X128) R2	SW & R0 & R2 & x”128”
4	ADDI R0, R0, #1	ADDI & R0 & R0 & x”001”
5	ADDI R2, R2, #1	ADDI & R2 & R2 & x”001”
6	BLTI R1, R2, # -3	BLTI & R1 & R2 & x”ffd”
7	LI R2, #0	LI & R2 & x”0000”
8	LI R3, #0	LI & R3 & x”0000”
9	LW R4, 0x128(R2)	LW & R4 & R2 & x”128”
10	ADD R3, R3, R4	TR & R3 & R3 & R4 & SU & ADD
11	ADDI R2, R2, #1	ADDI & R2 & R2 & x”001”
12	BLTI R1, R2, #-3	BLTI & R1 & R2 & x”ffd”
13	SRL R3, R3 #3	TR & R3 & R3 & SU & x”3” & OPSRL
14	SWI R3, 0x260	SWI & R3 & x”0260”
15	NOP	NOP & SU & SU & SU & SU & SU
16	B NOP	B & SU & x”000F”

Tabla 1. Pseudocódigo memoria de programa del programa 2.

PC	24 …. 20					19 … 16				15 … 12				11 … 8				7 … 4				3 … 0			
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
2	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0
4	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
5	0	0	1	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
6	0	1	1	1	1	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	0	1	0	0
7	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	0	1	1	1	0	1	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	1	1	1
10	0	0	0	0	0	0	0	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
11	0	0	1	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
12	0	1	1	1	1	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	0	1	0	0
13	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	1	1	1	0	1	0
14	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0
15	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Tabla 1.1 Memoria de programa del programa 2.

Diagrama de Onda del Programa 1.

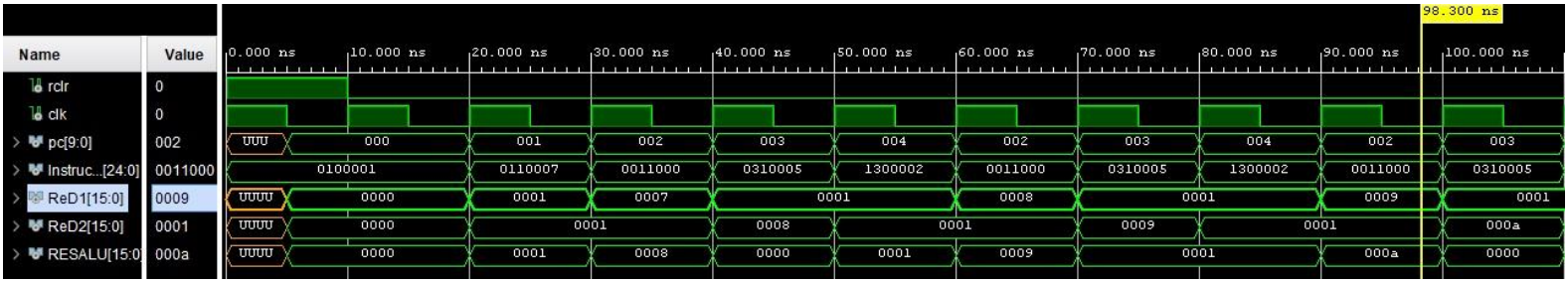

Figura 3. Diagrama de Onda del Programa 1.

Tabla del Programa 1.

Puerto	Ciclo 1	Ciclo 2	Ciclo 3	Ciclo 4	Ciclo 5	Ciclo 6	Ciclo 7	Ciclo 8	Ciclo 9	Ciclo 10
PC	UUU	000	001	002	003	004	002	003	004	002
Instrucción	0100001	0100001	0110007	0011000	0310005	1300002	0011000	0310005	1300002	0011000
ReadD1	UUU	0000	0001	0007	0001	0001	0008	0001	0001	0009
ReadD2	UUU	0000	0001	0001	0008	0001	0001	0009	0001	0001
ResALU	UUU	0000	0001	0008	0000	0001	0009	0001	0001	000a

Tabla 2. Tabla de Resultado del Programa 1.

Diagrama de Onda del Programa 2 (Promedio de 8 números de un arreglo).

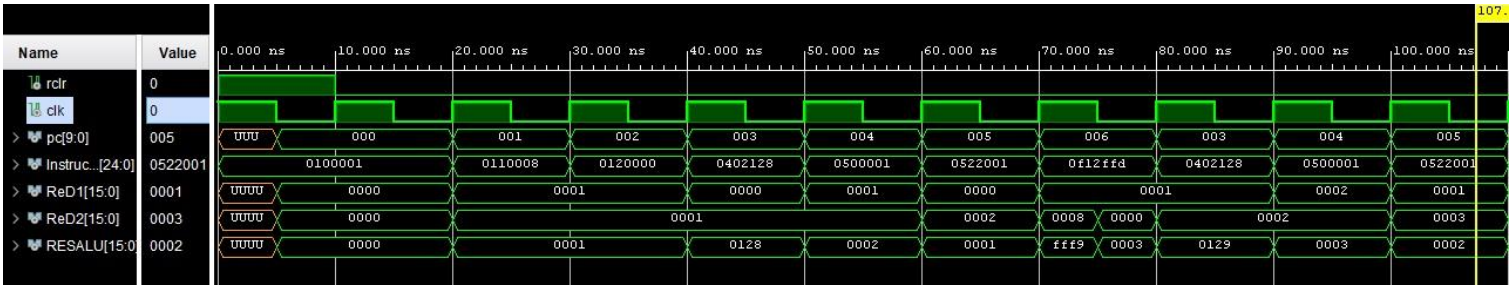

Figura 4. Diagrama de Onda del Programa 2 (Promedio de 8 números de un arreglo).

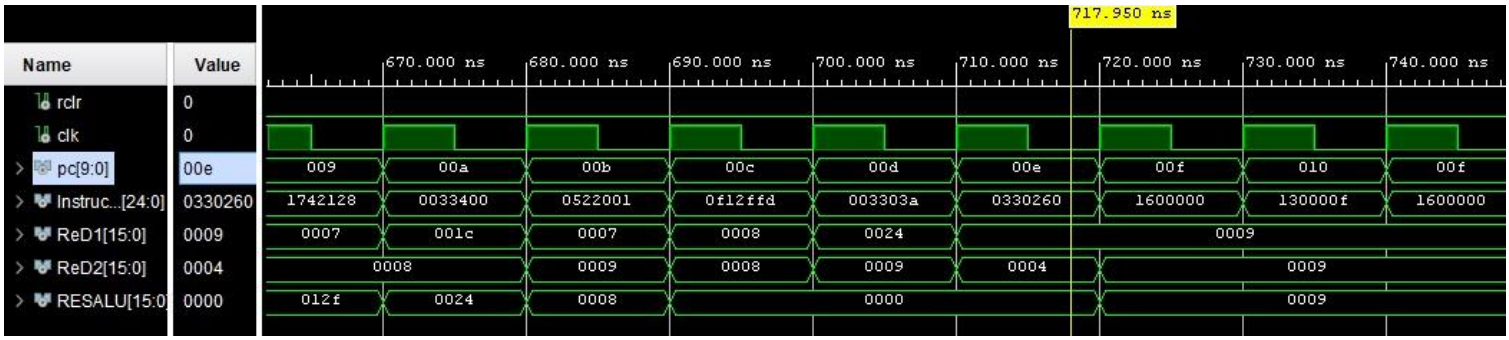

Figura 4.1. Diagrama de Onda del Programa 2 (Promedio de 8 números de un arreglo).

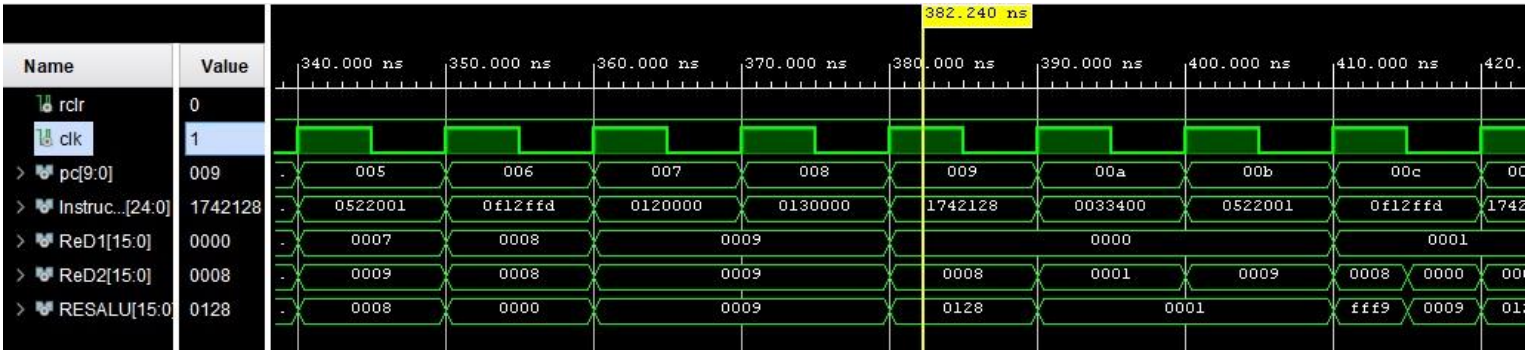

Figura 4.2 Diagrama de Onda del Programa 2 (Promedio de 8 números de un arreglo).

Tabla del Programa 2 (Promedio de 8 números de un arreglo).

Puerto	Ciclo 1	Ciclo 2	Ciclo 3	Ciclo 4	Ciclo 5	Ciclo 6	Ciclo 7	Ciclo 8	Ciclo 9	Ciclo 10
PC	UUU	000	001	002	003	004	005	006	003	004
Instrucción	0100001	0100001	0110008	0120000	0402128	0500001	0522001	0f12ffd	0402128	0500001
ReadD1	UUU	0000	0001	0001	0000	0001	0000	0001	0001	0002
ReadD2	UUU	0000	0001	0001	0001	0001	0002	0008 / 0000	0002	0002
ResALU	UUU	0000	0001	0001	0128	0002	0001	fff9 / 0003	0129	0003

Tabla 3. Tabla de Resultado del Programa 2 (Promedio de 8 números de un arreglo).

Diagrama de flujo del Programa 2 (Promedio de 8 números de un arreglo).