Instituto Politécnico Nacional

Escuela Superior de Cómputo

Arquitectura de Computadoras

ESCOMIPS

Nombre: *Sampayo Hernández Mauro*

Grupo: *3CV8*

Profesor: *Nayeli Vega García*

Fecha de entrega: *3 de julio del 2020*

Código de Implementación del procesador:

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

entity ESCOMIPS is
    generic ( d : integer := 25;
              x : integer := 16;
              n: integer := 4;
              tam_Microinstruccion: integer:=20;
              tam_OPCODE: integer:=5;
              tam_FUNCODE: integer:=4);
    Port ( clr0, clk: in STD_LOGIC;
           PC, RD1, RD2, resALU, BusSR: out STD_LOGIC_VECTOR (15 downto
0));
end ESCOMIPS;

architecture Behavioral of ESCOMIPS is

component Registro_clr is
    Port ( clk, rclr: in STD_LOGIC;
           clr: out STD_LOGIC);
end component;

component ALU_Nbits is
    Port ( a, b : in STD_LOGIC_VECTOR (x-1 downto 0);
           aluop : in STD_LOGIC_VECTOR (n-1 downto 0);--3 downto 0
           res : out STD_LOGIC_VECTOR (x-1 downto 0);
           bn, z, co, ov : out STD_LOGIC);
end component;

component ArchivodeRegistros is
    Port ( readReg1, readReg2, writeReg, shamt : in  STD_LOGIC_VECTOR (n-
1 downto 0);
           writeData : in  STD_LOGIC_VECTOR (x-1 downto 0);
           readData1, readData2 : inout  STD_LOGIC_VECTOR (x-1 downto 0);
           WR, SHE, dir, clk , clr : in  STD_LOGIC);
end component;

component MemoriaDatos is
    Port (dataIn: in  STD_LOGIC_VECTOR (x-1 downto 0);
          dir : in  STD_LOGIC_VECTOR (9 downto 0);
          clk, WD : in  STD_LOGIC;
          dataOut : out  STD_LOGIC_VECTOR (x-1 downto 0));
end component;

component Pila_MemoriaPrograma is
    Port( PC_in : in  STD_LOGIC_VECTOR (x-1 downto 0);
          clk, clr, UP, DW, WPC : in  STD_LOGIC;
          PC_out : out  STD_LOGIC_VECTOR (x-1 downto 0);
          Inst : out  STD_LOGIC_VECTOR (d-1 downto 0));
end component;

component UnidadControl is
    Port ( funCode : in  STD_LOGIC_VECTOR (tam_FUNCODE-1 downto 0);
           opCode : in  STD_LOGIC_VECTOR (tam_OPCODE-1 downto 0);
           banderas : in  STD_LOGIC_VECTOR (n-1 downto 0);
```

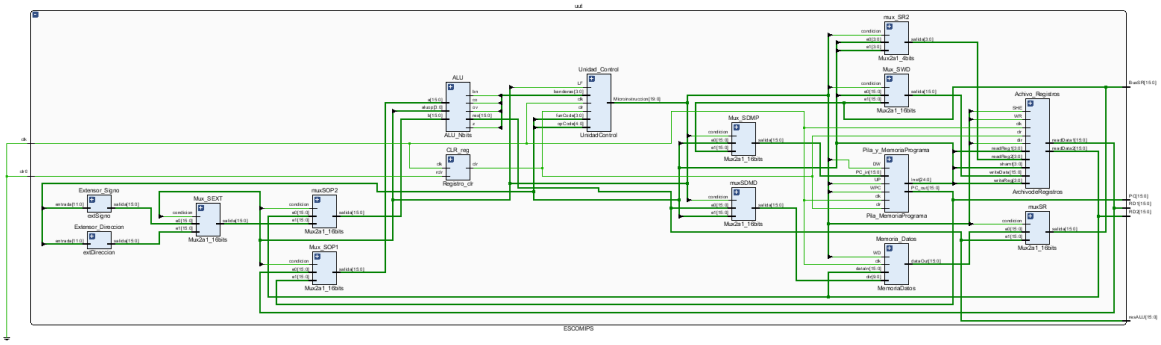
```
           clk, clr, LF: in  STD_LOGIC;
           Microinstruccion : out  STD_LOGIC_VECTOR
(tam_Microinstruccion-1 downto 0));
end component;

component Mux2a1_16bits is
    Port ( e0, e1 : in  STD_LOGIC_VECTOR (x-1 downto 0);
           condicion : in std_logic;
           salida : out  STD_LOGIC_VECTOR (x-1 downto 0));
end component;

component Mux2a1_4bits is
    Port ( e0, e1 : in  STD_LOGIC_VECTOR (n-1 downto 0);
           condicion : in  STD_LOGIC;
           salida : out  STD_LOGIC_VECTOR (n-1 downto 0));
end component;

component extSigno is
  Port ( entrada: in  STD_LOGIC_VECTOR (11 downto 0);
         salida: out STD_LOGIC_VECTOR (x-1 downto 0));
end component;

component extDireccion is
  Port ( entrada: in  STD_LOGIC_VECTOR (11 downto 0);
         salida: out STD_LOGIC_VECTOR (x-1 downto 0));
end component;

signal clr, bn, z, co, ov, LF, SHE, DIR, WR, WD, UP, DW, WPC: STD_LOGIC;
signal SDMP, SR2, SWD, SEXT, SOP1, SOP2, SDMD, SR: STD_LOGIC;
signal a, b, res, writeData, readData1, readData2, PC_in, PC_out,
sal_Signo, sal_Dir, extensor, dirMemData, dataOut, SR_Out:
STD_LOGIC_VECTOR(x-1 downto 0);
signal Inst: STD_LOGIC_VECTOR(d-1 downto 0);
signal readReg1, readReg2, writeReg, shamt, aluop: STD_LOGIC_VECTOR (n-1
downto 0);

begin
    CLR_reg: Registro_clr
        Port Map ( clk => clk,
                   rclr =>clr0,
                   clr => clr);

    Unidad_Control: UnidadControl --(0, N, Z, C)
        Port Map ( banderas(3) => ov,
                   banderas(2) => bn,
                   banderas(1) => z,
                   banderas(0) => co,
                   LF => LF,
                   clk => clk,
                   clr => clr,

                   opCode  => Inst(24 downto 20),
                   funCode => Inst(3 downto 0),

                   Microinstruccion(19) => SDMP,
                   Microinstruccion(18) => UP,
                   Microinstruccion(17) => DW,
```

```
                Microinstruccion(16) => WPC,
                Microinstruccion(15) => SR2,
                Microinstruccion(14) => SWD,
                Microinstruccion(13) => SEXT,
                Microinstruccion(12) => SHE,
                Microinstruccion(11) => DIR,
                Microinstruccion(10) => WR,
                Microinstruccion(9) => SOP1,
                Microinstruccion(8) => SOP2,
                Microinstruccion(7) => aluop(3),
                Microinstruccion(6) => aluop(2),
                Microinstruccion(5) => aluop(1),
                Microinstruccion(4) => aluop(0),
                Microinstruccion(3) => SDMD,
                Microinstruccion(2) => WD,
                Microinstruccion(1) => SR,
                Microinstruccion(0) => LF);

Pila_y_MemoriaPrograma: Pila_MemoriaPrograma
    Port map( PC_in => PC_in,
              clk => clk,
              clr => clr,
              UP => UP,
              DW => DW,
              WPC => WPC,
              PC_out => PC_out,
              Inst => Inst);

mux_SR2: mux2a1_4bits
    Port map( e1 => Inst(19 downto 16),
              e0 => Inst(11 downto 8),
              condicion => SR2,
              salida => readReg2);

Mux_SWD: Mux2a1_16bits
    Port map( e1 => SR_Out,
              e0 => Inst(15 downto 0),
              condicion => SWD,
              salida => writeData);

Extensor_Signo: extSigno
   Port Map ( entrada => Inst(11 downto 0),
              salida => sal_Signo);

Extensor_Direccion: extDireccion
   Port Map ( entrada => Inst(11 downto 0),
              salida => sal_Dir);

Mux_SEXT: Mux2a1_16bits
    Port map( e1 => sal_Dir,
              e0 => sal_Signo,
              condicion => sext,
              salida => extensor);

Achivo_Registros: ArchivodeRegistros
   Port map( readReg1 => Inst(15 downto 12),
             readReg2 => readReg2,
```

```
            writeReg => Inst(19 downto 16),
            shamt => Inst(7 downto 4),
            writeData => writeData,
            readData1 => readData1,
            readData2 => readData2,
            WR => WR,
            SHE => SHE,
            dir => dir,
            clk => clk,
            clr => clr);

Mux_SOP1: Mux2a1_16bits
    Port map( e1 => PC_out,
              e0 => readData1,
              condicion => SOP1,
              salida => a);

muxSOP2: Mux2a1_16bits
    Port map( e1 => extensor,
              e0 => readData2,
              condicion => SOP2,
              salida => b);

ALU: ALU_Nbits
   Port map( a => a,
             b => b,
             aluop => aluop,
             res => res,
             co => co,
             bn => bn,
             z => z,
             ov => ov);

muxSDMD: Mux2a1_16bits
    Port map( e1 => Inst(15 downto 0),
              e0 => res,
              condicion => SDMD,
              salida => dirMemData);

Memoria_Datos: MemoriaDatos
    Port map( dataIn => readData2,
              dir => dirMemData(9 downto 0),
              clk => clk,
              WD => WD,
              dataOut => dataOut);

muxSR: Mux2a1_16bits
    Port map( e1 => res,
              e0 => dataOut,
              condicion => SR,
              salida => SR_Out);

Mux_SDMP: Mux2a1_16bits
    Port map( e1 => SR_Out,
              e0 => Inst(15 downto 0),
              condicion => SDMP,
              salida => PC_in);
```

```
    --ASIGNACION DE SALIDAS
    PC <= PC_Out;
    RD1 <= readData1;
    RD2 <= readData2;
    resALU <= res;
    BusSr <= SR_Out;

end Behavioral;
```

Diagrama RTL:



Código para llenar a la memoria con el problema del punto 2:

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

entity MemoriaPrograma is
    generic ( d : integer := 25;
              a : integer := 10);
    Port (PC : in STD_LOGIC_VECTOR (a-1 downto 0);
          Inst : out STD_LOGIC_VECTOR (d-1 downto 0));
end MemoriaPrograma;

architecture Behavioral of MemoriaPrograma is

--INSTRUCCIONES
--Tipo I
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";
constant ADDI : std_logic_vector(4 downto 0):= "00101";
constant SUBI : std_logic_vector(4 downto 0):= "00110";
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";
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";

--Tipo R
constant TR : std_logic_vector(4 downto 0) := "00000";--Operaci□ipo R
constant ADD : std_logic_vector(3 downto 0) := "0000";
constant SUB : std_logic_vector(3 downto 0) := "0001";
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 OpSLL : std_logic_vector(3 downto 0) := "1001";
constant OpSRL : std_logic_vector(3 downto 0) := "1010";

--Tipo J
constant B: std_logic_vector(4 downto 0):= "10011";
constant CALL : std_logic_vector(4 downto 0):= "10100";

--Otros
constant RET : std_logic_vector(4 downto 0):= "10101";
constant NOP : std_logic_vector(4 downto 0):= "10110";

--Sin Uso
constant SU : std_logic_vector(3 downto 0) := "0000";--Sin Uso

--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";
constant R9 : std_logic_vector(3 downto 0) := "1001";
constant R10 : std_logic_vector(3 downto 0) := "1010";
constant R11 : std_logic_vector(3 downto 0) := "1011";
constant R12 : std_logic_vector(3 downto 0) := "1100";
constant R13 : std_logic_vector(3 downto 0) := "1101";
constant R14 : std_logic_vector(3 downto 0) := "1110";
constant R15 : std_logic_vector(3 downto 0) := "1111";

--COMANDOS :0
type banco is array (0 to (2**a)-1) of std_logic_vector(d-1 downto 0);
constant memProg : banco := (
--SUMA
LI & R0 & x"0001",            --0. LI R0, #1
LI & R1 & x"0007",            --1. LI R1, #7
TR & R1 & R1 & R0 & SU & ADD, --2. SUMA: ADD R1, R1, R0
SWI & R1 & x"0005",           --3. SWI R1, 5
```

```
B & SU & x"0002",              --4. B CICLO

begin
    Inst <= memProg(conv_integer(PC));
end Behavioral;
```

Simulación:

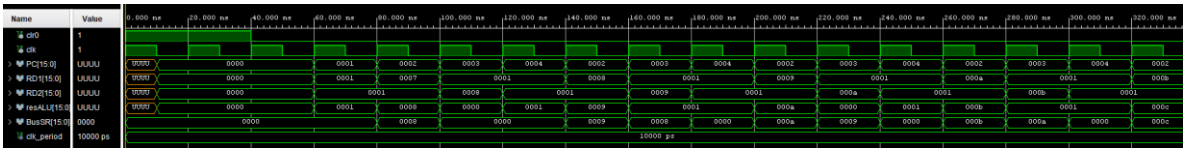


Tabla de resultados de la ejecución:

Bus	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
PC	0	1	2	3	4	2	3	4	2	3	4
Instrucción											
ReadData1	0	1	7	1	1	8	1	1	9	1	1
ReadData2	0	1	1	8	1	1	9	1	1	10	1
ResALU	0	1	8	0	1	9	1	1	10	0	1
BusSR	0	0	8	0	0	9	8	0	10	9	0

Código para llenar a la memoria con el problema del punto 5:

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

entity MemoriaPrograma is
    generic ( d : integer := 25;
              a : integer := 10);
    Port (PC : in STD_LOGIC_VECTOR (a-1 downto 0);
          Inst : out STD_LOGIC_VECTOR (d-1 downto 0));
end MemoriaPrograma;

architecture Behavioral of MemoriaPrograma is

--INSTRUCCIONES
--Tipo I
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";
constant ADDI : std_logic_vector(4 downto 0):= "00101";
constant SUBI : std_logic_vector(4 downto 0):= "00110";
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";
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";

--Tipo R
constant TR : std_logic_vector(4 downto 0) := "00000";--Operaci□ipo R
constant ADD : std_logic_vector(3 downto 0) := "0000";
constant SUB : std_logic_vector(3 downto 0) := "0001";
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 OpSLL : std_logic_vector(3 downto 0) := "1001";
constant OpSRL : std_logic_vector(3 downto 0) := "1010";

--Tipo J
constant B: std_logic_vector(4 downto 0):= "10011";
constant CALL : std_logic_vector(4 downto 0):= "10100";

--Otros
constant RET : std_logic_vector(4 downto 0):= "10101";
constant NOP : std_logic_vector(4 downto 0):= "10110";

--Sin Uso
constant SU : std_logic_vector(3 downto 0) := "0000";--Sin Uso

--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";
constant R9 : std_logic_vector(3 downto 0) := "1001";
constant R10 : std_logic_vector(3 downto 0) := "1010";
constant R11 : std_logic_vector(3 downto 0) := "1011";
constant R12 : std_logic_vector(3 downto 0) := "1100";
constant R13 : std_logic_vector(3 downto 0) := "1101";
constant R14 : std_logic_vector(3 downto 0) := "1110";
constant R15 : std_logic_vector(3 downto 0) := "1111";

--COMANDOS :0
type banco is array (0 to (2**a)-1) of std_logic_vector(d-1 downto 0);
constant memProg : banco := (
--FIBONACCI 15 ELEMENTOS
LI & R0 & x"0000",              --0.  LI R0, #0
LI & R1 & x"0001",          --1.  LI R1, #1
```

```
LI & R2 & x"0000",                --2.  LI R2, #0
LI & R3 & x"000f",                --3.  LI R3, #15
TR & R4 & R1 & R0 & SU & ADD,         --4.  SUMA: ADD R4, R1, R0
ADDI & R0 & R1 & x"000",          --5.  ADDI R0 R1 0
ADDI & R1 & R4 & x"000",          --6.  ADDI R1 R4 0
ADDI & R2 & R2 & x"001",          --7.  ADDI R2 R2 1
SWI & R1 & x"0026",           --8.  SWI R1, 26
BNEI & R2 & R3 & x"ffb",          --9.  BNEI R2 R3
NOP & SU & SU & SU & SU & SU,         --10. NOP
B &SU & x"000a",              --11. B NOP

begin
    Inst <= memProg(conv_integer(PC));
end Behavioral;
```

Simulación:

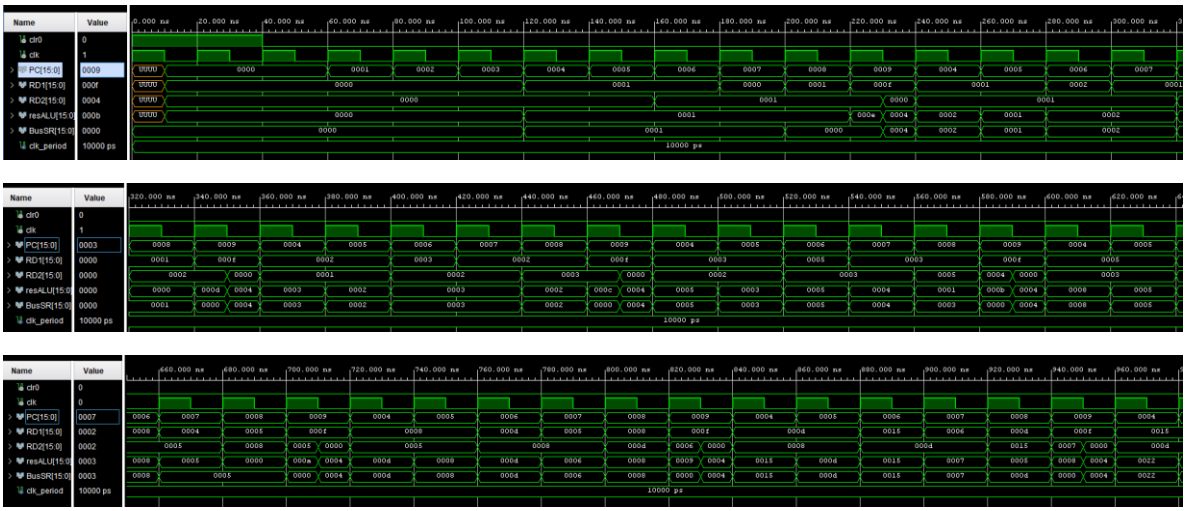


Tabla de resultados de la ejecución:

Bus	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
PC	0	1	2	3	4	5	6	7	8	9	4
Instrucción											
ReadData1	0	0	0	0	1	1	1	0	1	f	1
ReadData2	0	0	0	0	0	0	0	0	0	0	0
ResALU	0	0	0	0	1	1	1	1	1	e	2
BusSR	0	0	0	0	1	1	1	1	0	0	2