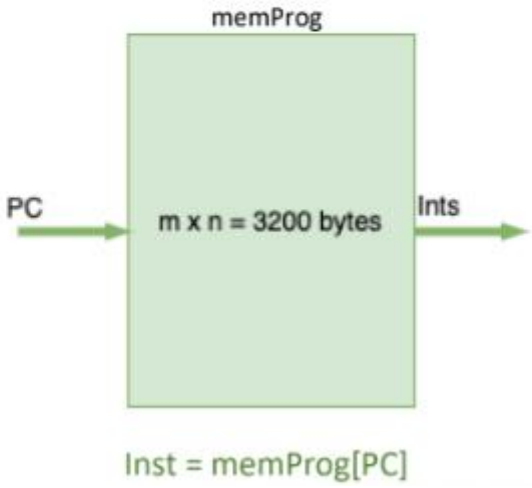
Práctica 8

Memoria de Programa

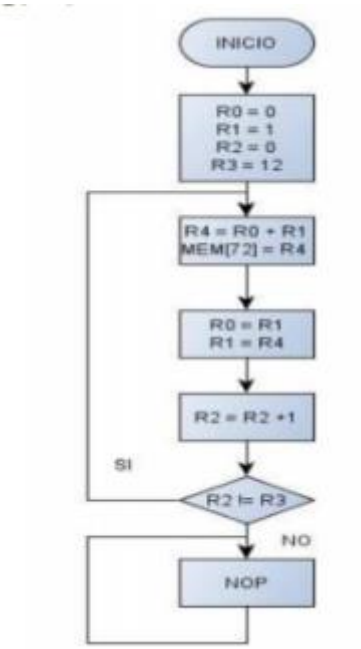
1. Implementar la memoria de programa para el ESCOMips. Tomar en cuenta que esta memoria debe estar parametrizada y que para esta práctica, la densidad de la memoria será de 3200 bytes
   a. Calcular el tamaño de los buses de datos y de direcciones
   b. Implementar la memoria con los tamaños calculados en el inciso a



Inst = memProg[PC]

Dado que tenemos una densidad expresada en bytes es necesario calcular la densidad en bits, lo cual es muy sencillo pues es el resultado de $3200\ x\ 8 = 25600\ bits$. Posteriormente, y dado que se conoce que la longitud de cada instrucción es de 25 bits, se realiza la división $\frac{25600}{25} = 1024$ siendo ese el número de palabras a almacenar. Ahora, para poder calcular el tamaño del bus de datos se requiere hacer el uso del logaritmo base 2, es decir: $\log_2 1024 = 10$. Por lo cual obtenemos que el **tamaño del bus de direcciones es de 10bits, y el de datos es de 25 bits**.

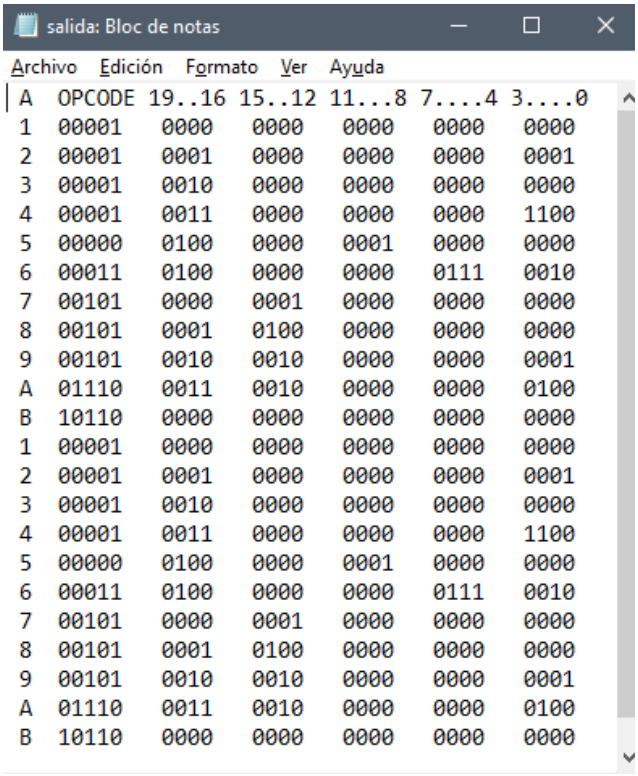
2. Inicializar la memoria con las instrucciones del siguiente programa, las localidades sin ocuparse deberán ser inicializadas con ceros.
3. Realizar la simulación desde testbench de tal manera que el programa se "ejecute" en su totalidad.



4. Escribir los resultados en un archivo de texto con el siguiente formato

PC	OPCODE	19..16	15..12	11...8	7....4	3....0

Al realizar la simulación se obtienen los siguientes resultados.



salida: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
A  OPCODE 19..16 15..12 11...8 7....4 3....0
1  00001   0000   0000   0000   0000   0000
2  00001   0001   0000   0000   0000   0001
3  00001   0010   0000   0000   0000   0000
4  00001   0011   0000   0000   0000   1100
5  00000   0100   0000   0001   0000   0000
6  00011   0100   0000   0000   0111   0010
7  00101   0000   0001   0000   0000   0000
8  00101   0001   0100   0000   0000   0000
9  00101   0010   0010   0000   0000   0001
A  01110   0011   0010   0000   0000   0100
B  10110   0000   0000   0000   0000   0000
1  00001   0000   0000   0000   0000   0000
2  00001   0001   0000   0000   0000   0001
3  00001   0010   0000   0000   0000   0000
4  00001   0011   0000   0000   0000   1100
5  00000   0100   0000   0001   0000   0000
6  00011   0100   0000   0000   0111   0010
7  00101   0000   0001   0000   0000   0000
8  00101   0001   0100   0000   0000   0000
9  00101   0010   0010   0000   0000   0001
A  01110   0011   0010   0000   0000   0100
B  10110   0000   0000   0000   0000   0000
```

Código de implementación

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

entity MemoPrograma is
    generic (
        m : integer := 10; --tama?o del bus de direcciones
        n : integer := 25 --tama?o de palabra
    );
    Port ( dir : in STD_LOGIC_VECTOR (m-1 downto 0);
           inst : out STD_LOGIC_VECTOR (n-1 downto 0));
end MemoPrograma;

architecture Behavioral of MemoPrograma is
----------------------Tipo R-------------------------
constant tipoR : 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";

---------Aritmeticas con registros----------------
constant add_app : std_logic_vector(3 downto 0) := "0000";
constant sub_app : std_logic_vector(3 downto 0) := "0001";

---------Aritmeticas-----------------------------------
constant addi : std_logic_vector (4 downto 0)  := "00101";
constant subi : std_logic_vector (4 downto 0)  := "00110";

---------.Logicas-------------------------------------
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";

---------Logicas con Registros--------------------
constant andr : std_logic_vector (3 downto 0) := "0010";
constant orr  : std_logic_vector (3 downto 0) := "0011";
constant xorr : std_logic_vector (3 downto 0) := "0100";
constant nandr: std_logic_vector (3 downto 0) := "0101";
constant norr : std_logic_vector (3 downto 0) := "0110";
constant xnorr: std_logic_vector (3 downto 0) := "0111";
constant notr : std_logic_vector (3 downto 0) := "1000";

--------Identificador Corrimiento R-------------------------
constant sllr : std_logic_vector (3 downto 0) := "1001";
constant srlr : std_logic_vector (3 downto 0) := "1010";

--------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";

---------------Manejo de Subrutinas---------------------------------
constant call : std_logic_vector (4 downto 0):= "10100";
constant ret : std_logic_vector (4 downto 0) := "10101";

--------------- Otros ------------------------------------------
constant nop : std_logic_vector (4 downto 0) := "10110";
```

```
constant su  : std_logic_vector (3 downto 0) := "0000";

-----------------  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";

type banco is array (0 to (2**m)-1) of std_logic_vector(n-1 downto 0);
constant aux : banco := (

    LI&R0&x"0000",
    LI&R1&x"0001",
    LI&R2&x"0000",
    LI&R3&x"000C",

    tipoR&R4&R0&R1&SU&ADD_APP,
    SWI&R4&x"0072",

    ADDI&R0&R1&x"000",
    ADDI&R1&R4&x"000",

    ADDI&R2&R2&x"001",

    BNEI&R3&R2&x"004",

    NOP&SU&SU&SU&SU&SU,

    others => (others => '0')
);
begin

    inst <= aux(conv_integer(dir));

end Behavioral;
```

Código de simulación

```
LIBRARY STD;
LIBRARY ieee;
USE STD.TEXTIO.ALL;
USE ieee.std_logic_TEXTIO.ALL;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_UNSIGNED.ALL;
USE ieee.std_logic_ARITH.ALL;

entity tb_MemoPrograma is
end tb_MemoPrograma;

architecture tb of tb_MemoPrograma is

    component MemoPrograma
        port (dir  : in std_logic_vector (9 downto 0);
              inst : out std_logic_vector (24 downto 0));
    end component;

    signal dir  : std_logic_vector (9 downto 0) := (others => '0');
    signal inst : std_logic_vector (24 downto 0);

begin

    dut : MemoPrograma
    port map (dir  => dir,
              inst => inst);

    stim_proc: process
   file ARCH_SAL : TEXT;
    variable LINEA_SAL : line;

    VARIABLE CADENA : STRING(1 TO 6);
    VARIABLE CADENA2 : STRING(1 TO 2);
    variable var_Instruccion: std_logic_vector(24 downto 0);
    variable var_operacion: std_logic_vector(4 downto 0);
    variable var_Parte1 : std_logic_vector(3 downto 0);
    variable var_Parte2 : std_logic_vector(3 downto 0);
    variable var_Parte3 : std_logic_vector(3 downto 0);
    variable var_Parte4 : std_logic_vector(3 downto 0);
    variable var_Parte5 : std_logic_vector(3 downto 0);
    variable ID : std_logic_vector(3 downto 0);

    begin
      file_open(ARCH_SAL, "D:\ESCOM\ARQUITECTURA\MemoriaPrograma\MemoriaP
rograma.srcs\sim_1\new\salida.txt", WRITE_MODE);

        --para los encabezados
```

```
    CADENA2:="A ";
    write(LINEA_SAL, CADENA2, right, CADENA2'LENGTH+1);
    CADENA:="OPCODE";
    write(LINEA_SAL, CADENA, right, CADENA'LENGTH+1);
    CADENA:="19..16";
    write(LINEA_SAL, CADENA, right, CADENA'LENGTH+1);
    CADENA:="15..12";
    write(LINEA_SAL, CADENA, right, CADENA'LENGTH+1);
    CADENA:="11...8";
    write(LINEA_SAL, CADENA, right, CADENA'LENGTH+1);
    CADENA:="7....4";
    write(LINEA_SAL, CADENA, right, CADENA'LENGTH+1);
    CADENA:="3....0";
    write(LINEA_SAL, CADENA, right, CADENA'LENGTH+1);
    writeline(ARCH_SAL,LINEA_SAL);

    for veces in 0 to 1 loop
        for i in 0 to 10 loop
                dir <= conv_std_logic_vector(i, 10);
                wait for 10 ps;
                var_Instruccion:= inst;

                for j in 24 downto 20 loop
                    var_operacion(j-20):=var_Instruccion(j);
                end loop;

                for j in 19 downto 16 loop
                    var_Parte1(j-16):=var_Instruccion(j);
                end loop;

                for j in 15 downto 12 loop
                    var_Parte2(j-12):=var_Instruccion(j);
                end loop;

                for j in 11 downto 8 loop
                    var_Parte3(j-8):=var_Instruccion(j);
                end loop;

                for j in 7 downto 4 loop
                    var_Parte4(j-4):=var_Instruccion(j);
                end loop;

                for j in 3 downto 0 loop
                    var_Parte5(j):=var_Instruccion(j);
                end loop;

                ID:=conv_std_logic_vector(i+1,4);
```

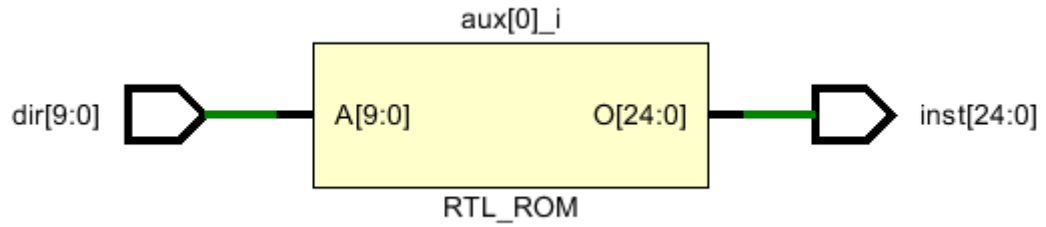
```
                Hwrite (LINEA_SAL,ID,RIGHT,2);
                write(LINEA_SAL,var_operacion,right,7);
                write(LINEA_SAL,var_Parte1,right,7);
                write(LINEA_SAL,var_Parte2,right,7);
                write(LINEA_SAL,var_Parte3,right,7);
                write(LINEA_SAL,var_Parte4,right,7);
                write(LINEA_SAL,var_Parte5,right,7);
                writeline(ARCH_SAL,LINEA_SAL);-
- escribe la linea en el archivo
        end loop;
    end loop;

            file_close(ARCH_SAL);  -- cierra el archivo

        wait;
    end process;

end tb;
```

Diagrama RTL



Forma de onda de salida