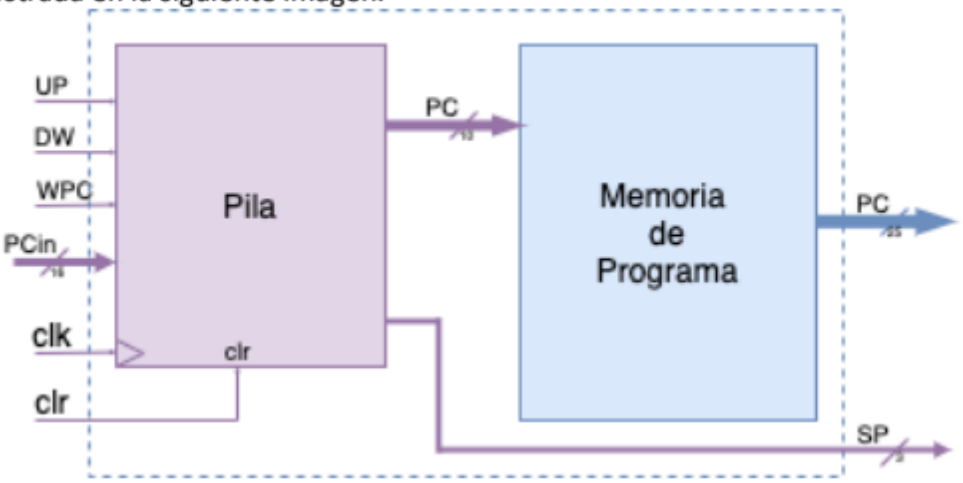
Práctica 11.

Pila de Memoria.

1. Implemente la etapa de búsqueda de una instrucción, con base en la arquitectura mostrada en la siguiente imagen.



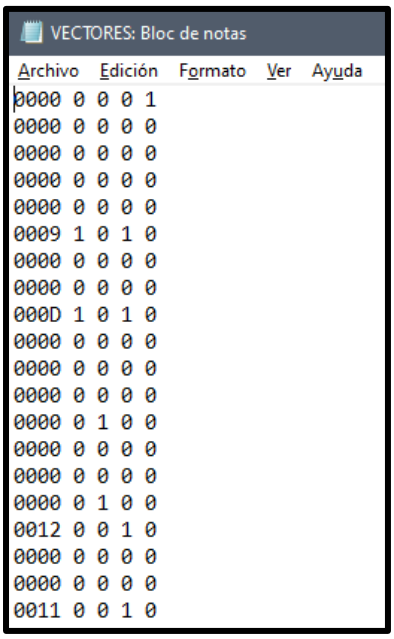
2. Cargue en la memoria de programa las siguientes instrucciones
   1. LI R6, #87
   2. LI R8, #90
   3. ADD R8, R2, R3
   4. SUB R1, R2, R3
   5. CALL 0x09
   6. LI R6, #87
   7. LI R8, #90
   8. CALL 13
   9. ADD R8, R2, R3
   10. SUB R1. R2. R3
   11. LI R6, #87
   12. RET
   13. SUB R1, R2, R3
   14. LI R6, #87
   15. RET
   16. B 18
   17. NOP
   18. NOP
   19. B 17

A continuación, se muestra como se han cargado todas las instrucciones a la memoria de programa.

```
type banco is array (0 to (2**num_p) - 1) of std_logic_vector((tam_p - 1) downto 0);
constant memoria : banco := (
op_li&r6&x"0057",               --1. li r6 #87
op_li&r8&x"005A",               --2. li r8 #90
op_add&r8&r2&r3&su&id_add,
op_sub&r1&r2&r3&su&id_sub,
op_call&su&x"0009",
op_li&r6&x"0057",               --li r6 #87
op_li&r8&x"005A",               --li r8 #90
op_call&su&x"000D",             --call 13
op_add&r8&r2&r3&su&id_add,
op_sub&r1&r2&r3&su&id_sub,
op_li&r6&x"0057",               --li r6 #87
op_ret&su&su&su&su&su,          --ret
op_sub&r1&r2&r3&su&id_sub,
op_li&r6&x"0057",               --li r6 #87
op_ret&su&su&su&su&su,          --ret
op_b&su&x"0012",                --b 18
op_nop&su&su&su&su&su,
op_nop&su&su&su&su&su,
op_b&su&x"0011",                --b 17
others => (others => '0')
);
```

3. Genere en un archivo de texto, los estímulos correspondientes para que el programa se "ejecute" una vez. Entiendase por ejecutar el hecho de hacer fetch a todas las instrucciones del programa.

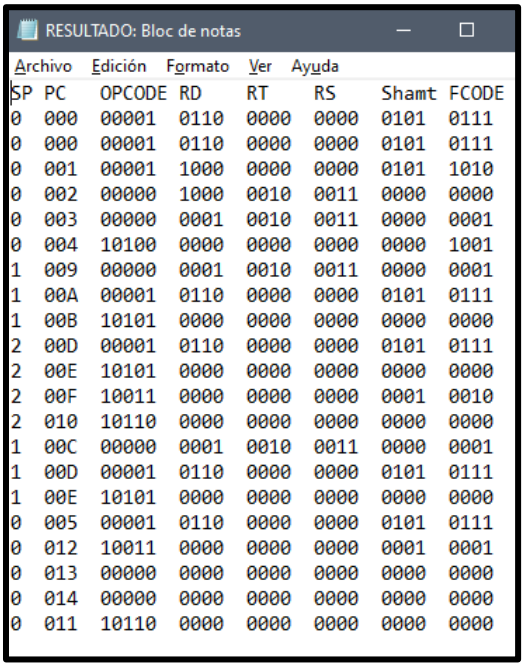
Se utilizo el siguiente archivo para la simulación del programa, en el se introducen los datos PC_in, UP, DOWN, WPC y CLR en ese orden.



4. Escriba el resultado en el archivo de texto con el siguiente formato

SP	PC	OP_CODE	Rd	Rt	Rs	Shamt	FUNC_CODE

A continuación, se muestra el archivo de salida, en el se distingue que se ejecutan las instrucciones previamente cargadas en la memoria de programa.



Código de simulación

```
LIBRARY ieee;
LIBRARY STD;
USE STD.TEXTIO.ALL;
USE ieee.std_logic_TEXTIO.ALL;  --PERMITE USAR STD_LOGIC

USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_UNSIGNED.ALL;
USE ieee.std_logic_ARITH.ALL;


ENTITY tb_pila_mem IS
END tb_pila_mem;

ARCHITECTURE behavior OF tb_pila_mem IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT main
    PORT(
         inst : OUT  std_logic_vector(24 downto 0);
         pc_in : IN  std_logic_vector(15 downto 0);
         pc_out : out  std_logic_vector(9 downto 0);
         sp_out : out std_logic_vector(2 downto 0);
         clk : IN  std_logic;
         clr : IN  std_logic;
         up : IN  std_logic;
         dw : IN  std_logic;
         wpc : IN  std_logic
        );
    END COMPONENT;


   --Inputs
   signal pc_in : std_logic_vector(15 downto 0) := (others => '0');
   signal clk : std_logic := '0';
   signal clr : std_logic := '0';
   signal up : std_logic := '0';
   signal dw : std_logic := '0';
   signal wpc : std_logic := '0';

    --Outputs
   signal inst : std_logic_vector(24 downto 0);
   signal pc_out : std_logic_vector(9 downto 0);
   signal sp_out : std_logic_vector(2 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;
```

```
BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: main PORT MAP (
          inst => inst,
          pc_in => pc_in,
          pc_out => pc_out,
          sp_out => sp_out,
          clk => clk,
          clr => clr,
          up => up,
          dw => dw,
          wpc => wpc
        );

   -- Clock process definitions
   clk_process :process
   begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   file ARCH_RES : TEXT;

    variable LINEA_RES : line;
   variable var_inst : std_logic_vector(24 downto 0);


    file ARCH_VEC : TEXT;
    variable LINEA_VEC : line;
    variable var_pc : std_logic_vector(15 downto 0);
    variable var_pc_out : std_logic_vector(9 downto 0);
    variable var_clk,var_clr,var_up,var_dw,var_wpc: std_logic;
    variable var_sp_out : std_logic_vector(2 downto 0);

    variable cadena2 : string(1 to 2);
    variable cadena5 : string(1 to 5);
    variable cadena6 : string(1 to 6);
   begin
        file_open(ARCH_RES, "D:\ESCOM\ARQUITECTURA\Practica11\pilaMem\pil
aMem.srcs\sim_1\new\RESULTADO.TXT", WRITE_MODE);
        file_open(ARCH_VEC, "D:\ESCOM\ARQUITECTURA\Practica11\pilaMem\pil
aMem.srcs\sim_1\new\VECTORES.TXT", READ_MODE);
```

```
        cadena2 := "SP";
        write(LINEA_RES, cadena2, left, 3);
        cadena2 := "PC";
        write(LINEA_RES, cadena2, left, 5);
        cadena6 := "OPCODE";
        write(LINEA_RES, cadena6, left, 7);
        cadena2 := "RD";
        write(LINEA_RES, cadena2, left, 6);
        cadena2 := "RT";
        write(LINEA_RES, cadena2, left, 6);
        cadena2 := "RS";
        write(LINEA_RES, cadena2, left, 6);
        cadena5 := "Shamt";
        write(LINEA_RES, cadena5, left, 6);
        cadena5 := "FCODE";
        write(LINEA_RES, cadena5, left, 6);
        writeline(ARCH_RES,LINEA_RES);-- escribe la linea en el archivo

        WAIT FOR 100 NS;
        FOR I IN 0 TO 20 LOOP
            readline(ARCH_VEC,LINEA_VEC); -- lee una linea completa
            Hread(LINEA_VEC, var_pc);
            pc_in <= var_pc;
            read(LINEA_VEC, var_up);
            up <= var_up;
            read(LINEA_VEC, var_dw);
            dw <= var_dw;
            read(LINEA_VEC, var_wpc);
            wpc <= var_wpc;
            read(LINEA_VEC, var_clr);
            clr <= var_clr;

            WAIT UNTIL RISING_EDGE(CLK);     --ESPERO AL FLANCO DE SUBIDA
            --wait for 100 ns;
            var_pc_out := pc_out;
            var_inst := inst;
            var_sp_out := sp_out;
            --wait for 100 ns;
            Hwrite(LINEA_RES, var_sp_out, left, 3);
            Hwrite(LINEA_RES, var_pc_out, left, 5); --
ESCRIBE EL CAMPO pc_out
            write(LINEA_RES, var_inst(24 downto 20), left, 7);
            write(LINEA_RES, var_inst(19 downto 16), left, 6);
            write(LINEA_RES, var_inst(15 downto 12), left, 6);
            write(LINEA_RES, var_inst(11 downto 8), left, 6);
            write(LINEA_RES, var_inst(7 downto 4), left, 6);
            write(LINEA_RES, var_inst(3 downto 0), left, 6);
```

```
            writeline(ARCH_RES,LINEA_RES);-
- escribe la linea en el archivo

        end loop;
        file_close(ARCH_VEC);   -- cierra el archivo
        file_close(ARCH_RES);   -- cierra el archivo

        --wait for clk_period*10;
      wait;
   end process;

END;
```

Código de implementación

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

entity main is
    Port ( inst : out  STD_LOGIC_VECTOR (24 downto 0);
           pc_in : in  STD_LOGIC_VECTOR (15 downto 0);
           pc_out: out std_logic_vector(9 downto 0);
           sp_out : out std_logic_vector(2 downto 0);
           clk,clr,up,dw,wpc : in  STD_LOGIC);
end main;

architecture Behavioral of main is

component pila is
    Port ( d : in  STD_LOGIC_VECTOR (15 downto 0);
           q : out  STD_LOGIC_VECTOR (9 downto 0);
           sp_out : out std_logic_vector(2 downto 0);
           clk, clr, up, dw, wpc : in  STD_LOGIC);
end component;
component mem_prog is
    Port ( pc : in  STD_LOGIC_VECTOR (9 downto 0);
           inst : out  STD_LOGIC_VECTOR (24 downto 0));
end component;
signal pc_out_aux : std_logic_vector (9 downto 0);

begin

pila1 : pila
     port map (
        d => pc_in,
        q => pc_out_aux,
        sp_out => sp_out,
        up => up,
```

```
        dw => dw,
        wpc => wpc,
        clk => clk,
        clr => clr
        );

mem_prog1 : mem_prog
     port map (
        pc => pc_out_aux,
        inst => inst
        );
pc_out <= pc_out_aux;
end Behavioral;
```

Forma de onda de la simulación

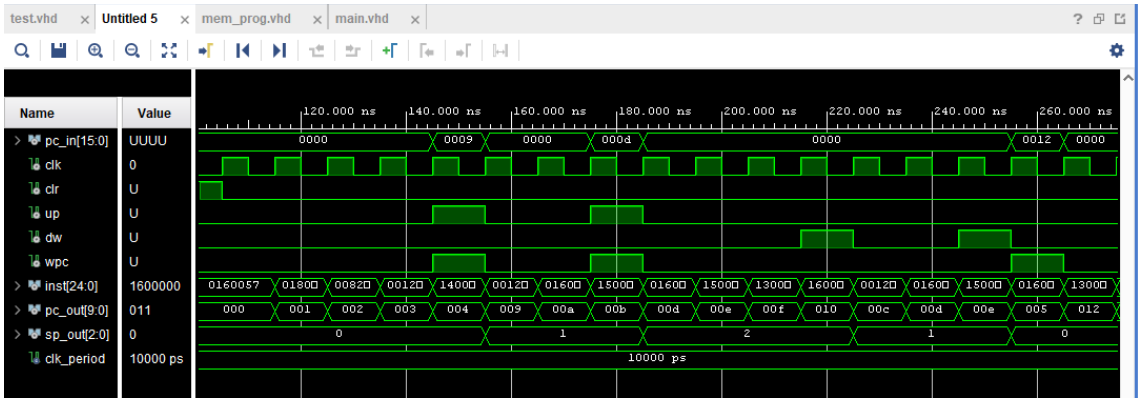


Diagrama RTL