Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica 11 - Pila Hardware y Memoria de Programa

Unidad de aprendizaje: Arquitectura de Computadoras

Grupo: 3CV1

Alumno(a):
Ramos Diaz Enrique

Profesor(a):
Vega García Nayeli

18 de mayo 2020

Índice

1. Código de implementación

1.1. Pila de Hardware

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

entity Pila is
    generic ( m : integer := 16; --tam PC
              n : integer := 3); --tam SP
    Port ( PCin : in STD_LOGIC_VECTOR (m-1 downto 0);
           clk, clr, wpc, up, dw : in STD_LOGIC;
           PCout : out STD_LOGIC_VECTOR (m-1 downto 0);
           SPout : out STD_LOGIC_VECTOR (n-1 downto 0));
end Pila;

architecture Behavioral of Pila is
    type banco is array (0 to (2**n)-1) of STD_LOGIC_VECTOR(m-1 downto 0);
    signal aux : banco;
begin
    process(clk, clr, aux)
        variable SP : integer range 0 to (2**n)-1;
    begin
        if (clr = '1') then
            SP := 0;
            aux <= (others => (others => '0'));
        elsif (rising_edge(clk)) then
            if (wpc = '0' and up = '0' and dw = '0') then
                aux(SP) <= aux(SP) + 1;
            elsif (wpc = '1' and up = '0' and dw = '0') then
                aux(SP) <= PCin;
            elsif (wpc = '1' and up = '1' and dw = '0') then
                SP := SP + 1;
                if(SP = 2**n) then
                    SP := 0;
                end if;
                aux(SP) <= PCin;
            elsif (wpc = '0' and up = '0' and dw = '1') then
                SP := SP - 1;
                if(SP = -1) then
                    SP := (2**n)-1;
                end if;
                aux(SP) <= aux(SP) + 1;
            end if;
        end if;
        PCout <= aux(SP);
        SPout <= conv_std_logic_vector(SP, n);
    end process;
end Behavioral;
```

1.2. Memoria de Programa

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

entity MemoriaPrograma is
    generic ( m : integer := 10; --tam PC
              n : integer := 25); --tam Instruccion
    Port ( pc : in STD_LOGIC_VECTOR (m-1 downto 0);
           inst : out STD_LOGIC_VECTOR (n-1 downto 0));
end MemoriaPrograma;

architecture Behavioral of MemoriaPrograma is
    --Instrucciones tipo R
    constant tipoR : STD_LOGIC_VECTOR (4 downto 0) := "00000";
    --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";
    --Aritmticas
    constant ADDI : STD_LOGIC_VECTOR (4 downto 0) := "00101";
    constant SUBI : STD_LOGIC_VECTOR (4 downto 0) := "00110";
    --Identificador Aritmticas R
    constant ADD : STD_LOGIC_VECTOR (3 downto 0) := "0000";
    constant SUB : STD_LOGIC_VECTOR (3 downto 0) := "0001";
    --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";
    --Identificador Logicas R
    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";
    constant OPR : STD_LOGIC_VECTOR (4 downto 0) := "00000";
    --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"; -- sin usar
    --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 := (
        "0000000000000000000000000",
        LI & R6 & x"0057",                  --LI    R6, #87
        LI & R8 & x"005a",                  --LI    R8, #90
        tipoR & R8 & R2 & R3 & SU & ADD,    --ADD   R8, R2, R3
        tipoR & R1 & R2 & R3 & SU & SUB,    --SUB   R1, R2, R3
        CALL & SU & x"0009",                --CALL  0x09
        LI & R6 & x"0057",                  --LI    R6, #87
        LI & R8 & x"005a",                  --LI    R8, #90
        CALL & SU & x"000D",                --CALL  13
        tipoR & R8 & R2 & R3 & SU & ADD,    --ADD   R8, R2, R3
        tipoR & R1 & R2 & R3 & SU & SUB,    --SUB   R1, R2, R3
        LI & R6 & x"0057",                  --LI    R6, #87
        RET & SU & SU & SU & SU & SU,       --RET
        tipoR & R1 & R2 & R3 & SU & SUB,    --SUB   R1, R2, R3
        LI & R6 & x"0057",                  --LI    R6, #87
        RET & SU & SU & SU & SU & SU,       --RET
        B & SU & x"0012",                   --B 18
        NOP & SU & SU & SU & SU & SU,       --NOP
        NOP & SU & SU & SU & SU & SU,       --NOP
        B & SU & x"0011",                   --B 17
        others => (others => '0')
    );
begin
    inst <= aux(conv_integer(pc));
end Behavioral;
```

1.3. Pila de Hardware y Memoria de Programa

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

entity Pila_MemPrograma is
    generic ( m : integer := 10; --tam PC
              m1 : integer := 16; --tam PCin
              n : integer := 3;  --tam SP
              tam_inst : integer := 25); --tam Instruccion
    Port ( PCin : in STD_LOGIC_VECTOR (m1-1 downto 0);
           clk, clr, wpc, up, dw : in STD_LOGIC;
           PCout : out STD_LOGIC_VECTOR (m1-1 downto 0);
           SPout : out STD_LOGIC_VECTOR (n-1 downto 0);
```

```vhdl
           inst : out STD_LOGIC_VECTOR (tam_inst-1 downto 0));
end Pila_MemPrograma;

architecture Behavioral of Pila_MemPrograma is
    component MemoriaPrograma is
        Port ( pc : in STD_LOGIC_VECTOR (m-1 downto 0);
               inst : out STD_LOGIC_VECTOR (tam_inst-1 downto 0));
    end component;

    component Pila is
        Port ( PCin : in STD_LOGIC_VECTOR (m1-1 downto 0);
               clk, clr, wpc, up, dw : in STD_LOGIC;
               PCout : out STD_LOGIC_VECTOR (m1-1 downto 0);
               SPout : out STD_LOGIC_VECTOR (n-1 downto 0));
    end component;

    signal aux_pc : STD_LOGIC_VECTOR (m1-1 downto 0);
begin
    stack : Pila Port map (
        PCin => PCin,
        clk => clk,
        clr => clr,
        wpc => wpc,
        up => up,
        dw => dw,
        PCout => aux_pc,
        SPout => SPout
    );

    mp : MemoriaPrograma Port map (
        pc => aux_pc(9 downto 0),
        inst => inst
    );

    PCout <= aux_pc;
end Behavioral;
```

2. Código de simulación

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.all;
use IEEE.STD_LOGIC_unsigned.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use STD.TEXTIO.ALL;

entity test_bench is
end test_bench;

architecture Behavioral of test_bench is
    component Pila_MemPrograma 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);
               SPout : out STD_LOGIC_VECTOR (2 downto 0);
               inst : out STD_LOGIC_VECTOR (24 downto 0));
    end component;

    signal PCin : STD_LOGIC_VECTOR (15 downto 0) := (others => '0');
    signal clk, clr, wpc, up, dw : STD_LOGIC;
    signal PCout : STD_LOGIC_VECTOR (15 downto 0) := (others => '0');
    signal SPout : STD_LOGIC_VECTOR (2 downto 0) := (others => '0');
    signal inst : STD_LOGIC_VECTOR (24 downto 0) := (others => '0');
begin
    stack_mp: Pila_MemPrograma Port map (
        PCin => PCin,
        clk => clk,
        clr => clr,
        wpc => wpc,
        up => up,
        dw => dw,
        PCout => PCout,
        SPout => SPout,
        inst => inst
    );

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

    process
        file arch_res : text;   --Apuntadores tipo
         ↪  txt
        variable linea_res : line;
        variable var_pc_out : STD_LOGIC_VECTOR (9 downto 0);
        variable var_sp_out : STD_LOGIC_VECTOR (2 downto 0);
        variable var_inst : STD_LOGIC_VECTOR (24 downto 0);

        file arch_en : text; --Apuntadores tipo txt
        variable linea_en: line;
        variable var_pc_in : STD_LOGIC_VECTOR (15 downto 0);
        variable var_clr : STD_LOGIC;
        variable var_wpc : STD_LOGIC;
        variable var_up : STD_LOGIC;
        variable var_dw : STD_LOGIC;
        variable cadena : string (1 to 6);
    begin
        --- PCIN CLR WPC UP DW
        file_open(arch_en, "Estimulos.txt", READ_MODE);

        --- SP PC OPCODE 19..16 15..12 11..8 7..4 3..0
        file_open(arch_res, "Resultado.txt", WRITE_MODE);

        cadena := "SP    ";
        write(linea_res, cadena, left, cadena'LENGTH+1);--ESCRIBE LA cadena "SP"
        cadena := "PC    ";
        write(linea_res, cadena, left, cadena'LENGTH+1);--ESCRIBE LA cadena "PC"
        cadena := "OPCODE";
        write(linea_res, cadena, left, cadena'LENGTH+1);--ESCRIBE LA cadena "OPCODE"
        cadena := "19..16";
        write(linea_res, cadena, left, cadena'LENGTH+1);--ESCRIBE LA cadena "19..16"
        cadena := "15..12";
        write(linea_res, cadena, left, cadena'LENGTH+1);--ESCRIBE LA cadena "15..12"
        cadena := " 11..8";
        write(linea_res, cadena, left, cadena'LENGTH+1);--ESCRIBE LA cadena "11..8"
        cadena := "  7..4";
        write(linea_res, cadena, left, cadena'LENGTH+1);--ESCRIBE LA cadena "7..4"
        cadena := "  3..0";
        write(linea_res, cadena, left, cadena'LENGTH+1);--ESCRIBE LA cadena "3..0"

        writeline(arch_res, linea_res);-- escribe la linea en el archivo

        for i in 1 to 20 loop
            readline(arch_en, linea_en); -- lee una linea completa
            --- PCIN CLR WPC UP DW

            --Lee PCIN
            Hread(linea_en, var_pc_in);
            PCin <= var_pc_in;

            --Lee CLR
            read(linea_en, var_clr);
            clr <= var_clr;

            --Lee WPC
            read(linea_en, var_wpc);
            wpc <= var_wpc;

            --Lee UP
            read(linea_en, var_up);
            up <= var_up;

            --Lee DW
            read(linea_en, var_dw);
            dw <= var_dw;

            wait until rising_edge(clk); --ESPERA AL FLANCO DE SUBIDA
            wait for 0.1 ns;
            var_inst := inst;
            var_pc_out := PCout(9 downto 0);
            var_sp_out := SPout;

            --- SP PC OPCODE 19..16 15..12 11..8 7..4 3..0
            Hwrite(linea_res, var_sp_out, left, 7);--ESCRIBE EL CAMPO SP
            Hwrite(linea_res, var_pc_out, left, 8);--ESCRIBE EL CAMPO PC
            write(linea_res, var_inst(24 downto 20), left, 8);--ESCRIBE EL
             ↪  CAMPO OPCODE
            write(linea_res, var_inst(19 downto 16), left, 7);--ESCRIBE EL
             ↪  CAMPO 19..16
            write(linea_res, var_inst(15 downto 12), left, 7);--ESCRIBE EL
             ↪  CAMPO 15..12
            write(linea_res, var_inst(11 downto 8), left, 7);--ESCRIBE EL
             ↪  CAMPO 11..8
            write(linea_res, var_inst(7 downto 4), left, 7);--ESCRIBE EL
             ↪  CAMPO 7 .. 4
            write(linea_res, var_inst(3 downto 0), left, 7);--ESCRIBE EL
             ↪  CAMPO 3 .. 0

            writeline(arch_res, linea_res);-- escribe la linea en el archivo
        end loop;
        file_close(arch_en);  -- cierra el archivo
        file_close(arch_res);  -- cierra el archivo
        wait;
    end process;
end Behavioral;
```

3. Simulación

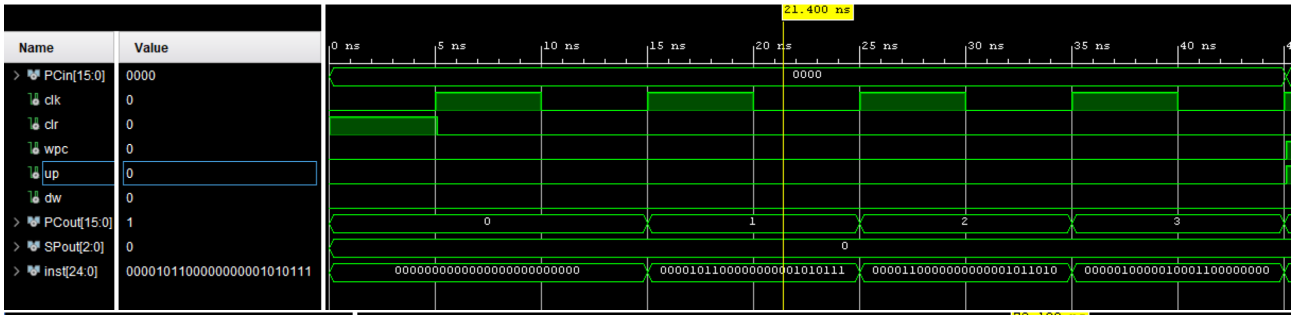
3.1. Programa a "ejecutar"

```
LI R6, #87
LI R8, #90
ADD R8, R2, R3
SUB R1, R2, R3
CALL 0x09
LI R6, #87
LI R8, #90
CALL 13
ADD R8, R2, R3
SUB R1, R2, R3
LI R6, #87
RET
SUB R1, R2, R3
LI R6, #87
RET
B 18
NOP
NOP
B 17
```

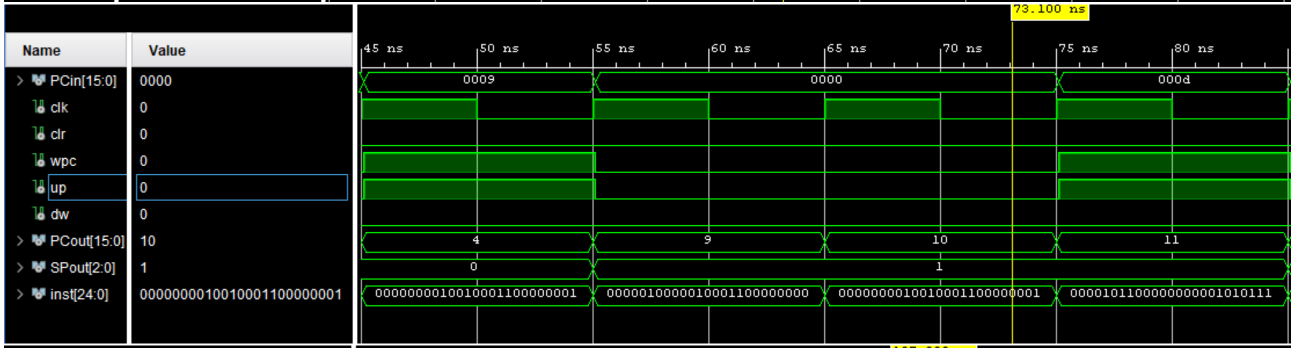
3.2. Archivo entrada: Estimulos.txt

```
--- PCIN CLR WPC UP DW
0000 1 0 0 0
0000 0 0 0 0
0000 0 0 0 0
0000 0 0 0 0
0000 0 0 0 0
0009 0 1 1 0
0000 0 0 0 0
0000 0 0 0 0
000D 0 1 1 0
0000 0 0 0 0
0000 0 0 0 0
0000 0 0 0 0
0000 0 0 0 1
0000 0 0 0 0
0000 0 0 0 0
0000 0 0 0 1
0012 0 1 0 0
0000 0 0 0 0
0000 0 0 0 0
0011 0 1 0 0
```

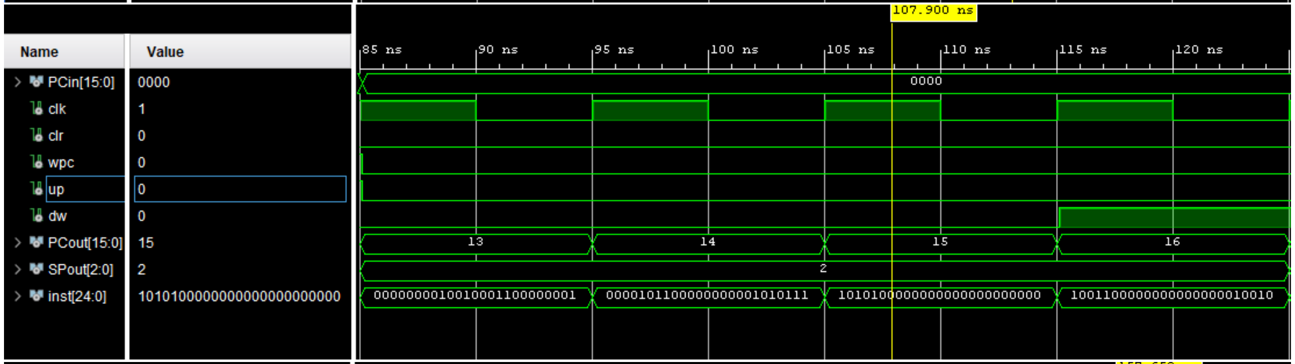
3.3. Forma de onda de simulación



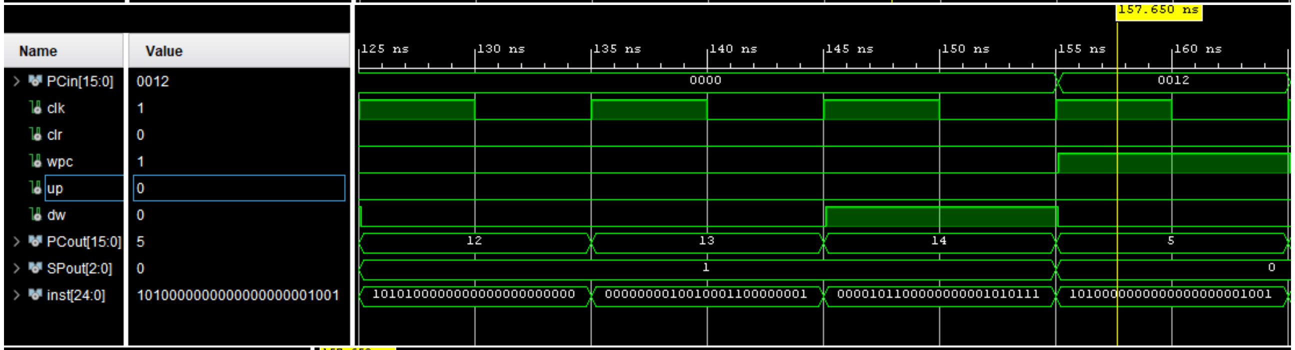
Name	Value
PCin[15:0]	0000
clk	0
clr	0
wpc	0
up	0
dw	0
PCout[15:0]	1
SPout[2:0]	0
inst[24:0]	0000101100000000001010111



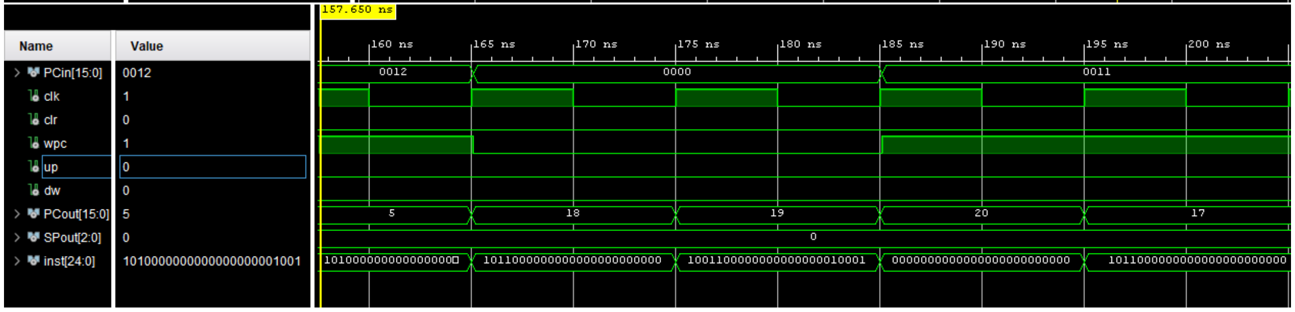
Name	Value
PCin[15:0]	0000
clk	0
clr	0
wpc	0
up	0
dw	0
PCout[15:0]	10
SPout[2:0]	1
inst[24:0]	0000000010010001100000001



Name	Value
PCin[15:0]	0000
clk	1
clr	0
wpc	0
up	0
dw	0
PCout[15:0]	15
SPout[2:0]	2
inst[24:0]	1010100000000000000000000



Name	Value
PCin[15:0]	0012
clk	1
clr	0
wpc	1
up	0
dw	0
PCout[15:0]	5
SPout[2:0]	0
inst[24:0]	1010000000000000000001001



Name	Value
PCin[15:0]	0012
clk	1
clr	0
wpc	1
up	0
dw	0
PCout[15:0]	5
SPout[2:0]	0
inst[24:0]	1010000000000000000001001

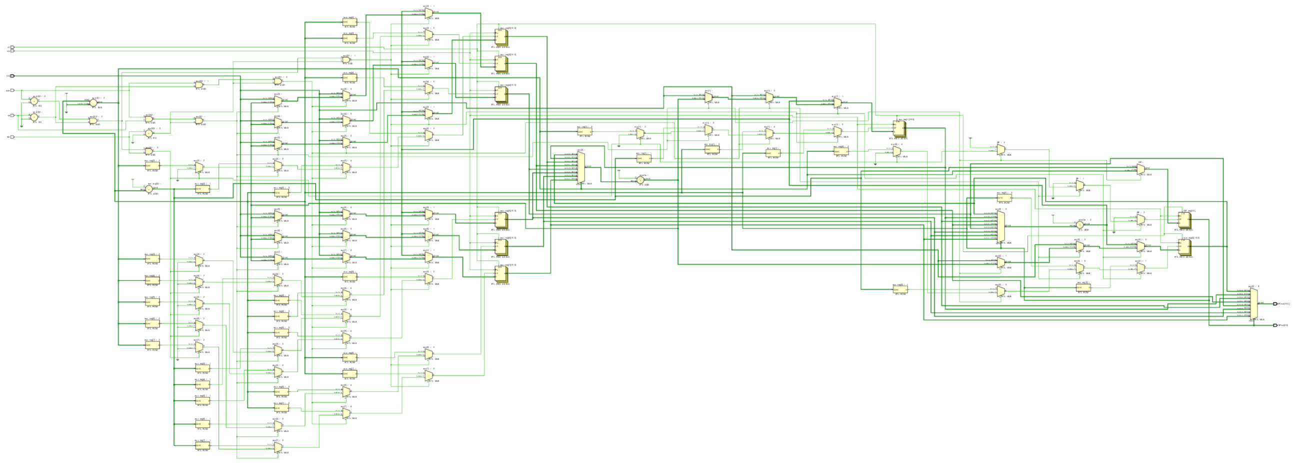
3.4. Archivo salida: Resultado.txt

```
SP      PC      OPCODE 19..16 15..12  11..8   7..4    3..0
0       000      00000   0000   0000   0000   0000   0000
0       001      00001   0110   0000   0000   0101   0111
0       002      00001   1000   0000   0000   0101   1010
0       003      00000   1000   0010   0011   0000   0000
0       004      00000   0001   0010   0011   0000   0001
1       009      00000   1000   0010   0011   0000   0000
1       00A      00000   0001   0010   0011   0000   0001
1       00B      00001   0110   0000   0000   0101   0111
2       00D      00000   0001   0010   0011   0000   0001
2       00E      00001   0110   0000   0000   0101   0111
2       00F      10101   0000   0000   0000   0000   0000
2       010      10011   0000   0000   0000   0001   0010
1       00C      10101   0000   0000   0000   0000   0000
1       00D      00000   0001   0010   0011   0000   0001
1       00E      00001   0110   0000   0000   0101   0111
0       005      10100   0000   0000   0000   0000   1001
0       012      10110   0000   0000   0000   0000   0000
0       013      10011   0000   0000   0000   0001   0001
0       014      00000   0000   0000   0000   0000   0000
0       011      10110   0000   0000   0000   0000   0000
```

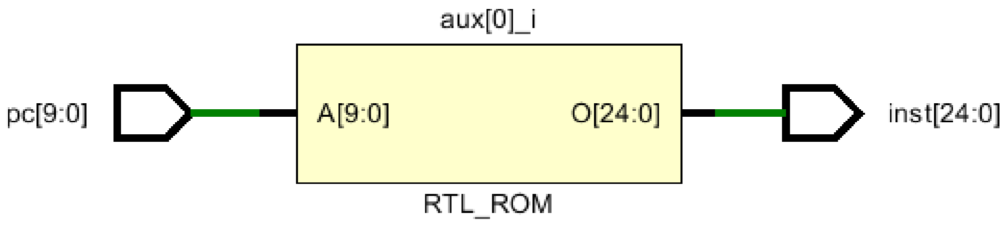
4. Diagramas RTL

4.1. Análisis RTL

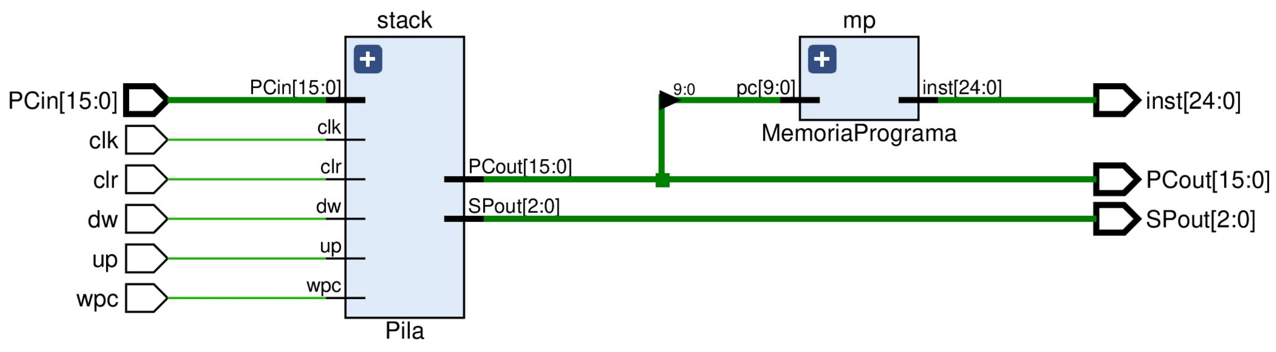
4.1.1. Pila de Hardware



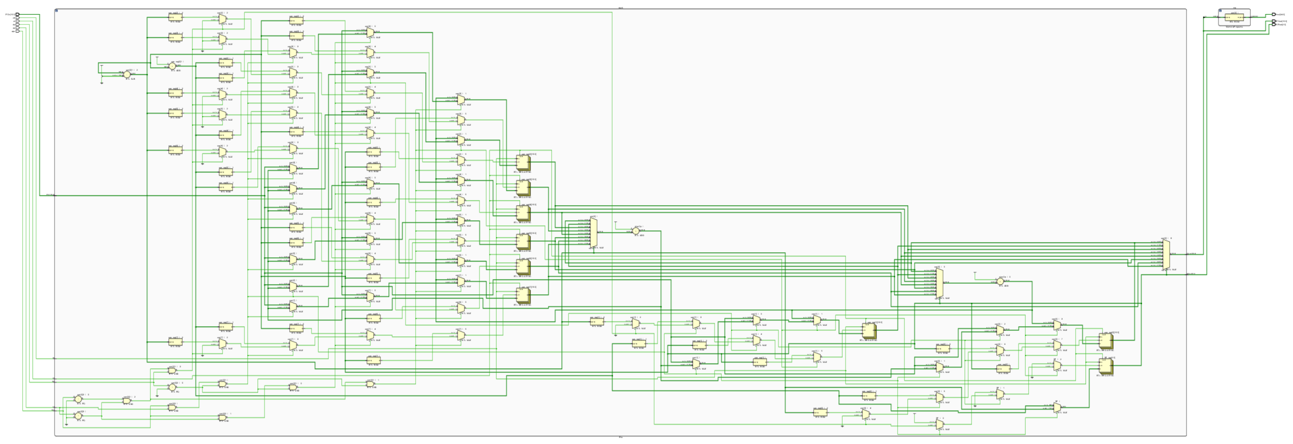
4.1.2. Memoria de Programa



4.1.3. Pila de Hardware y Memoria de Programa Comprimido

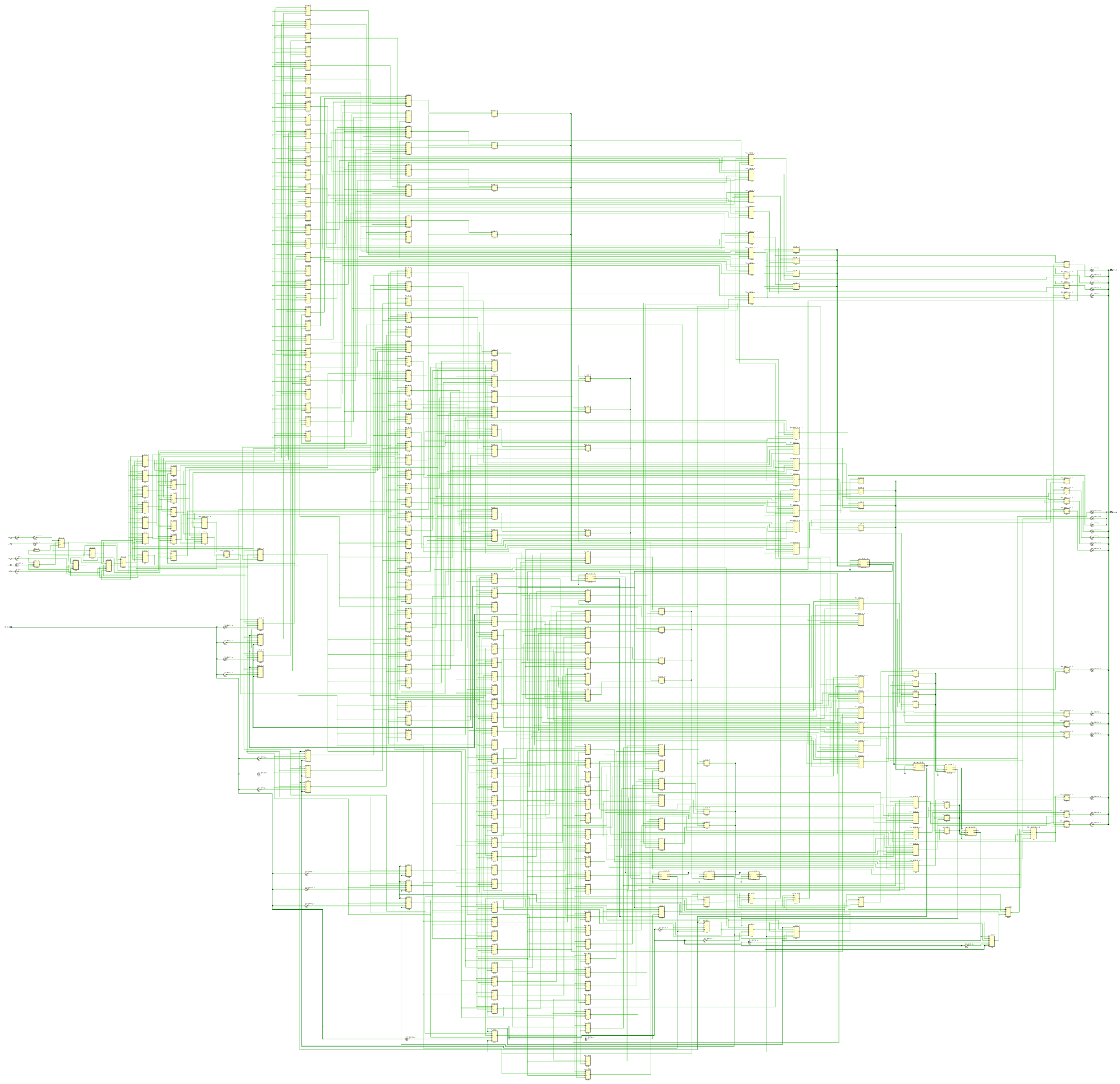


4.1.4. Pila de Hardware y Memoria de Programa Expandido

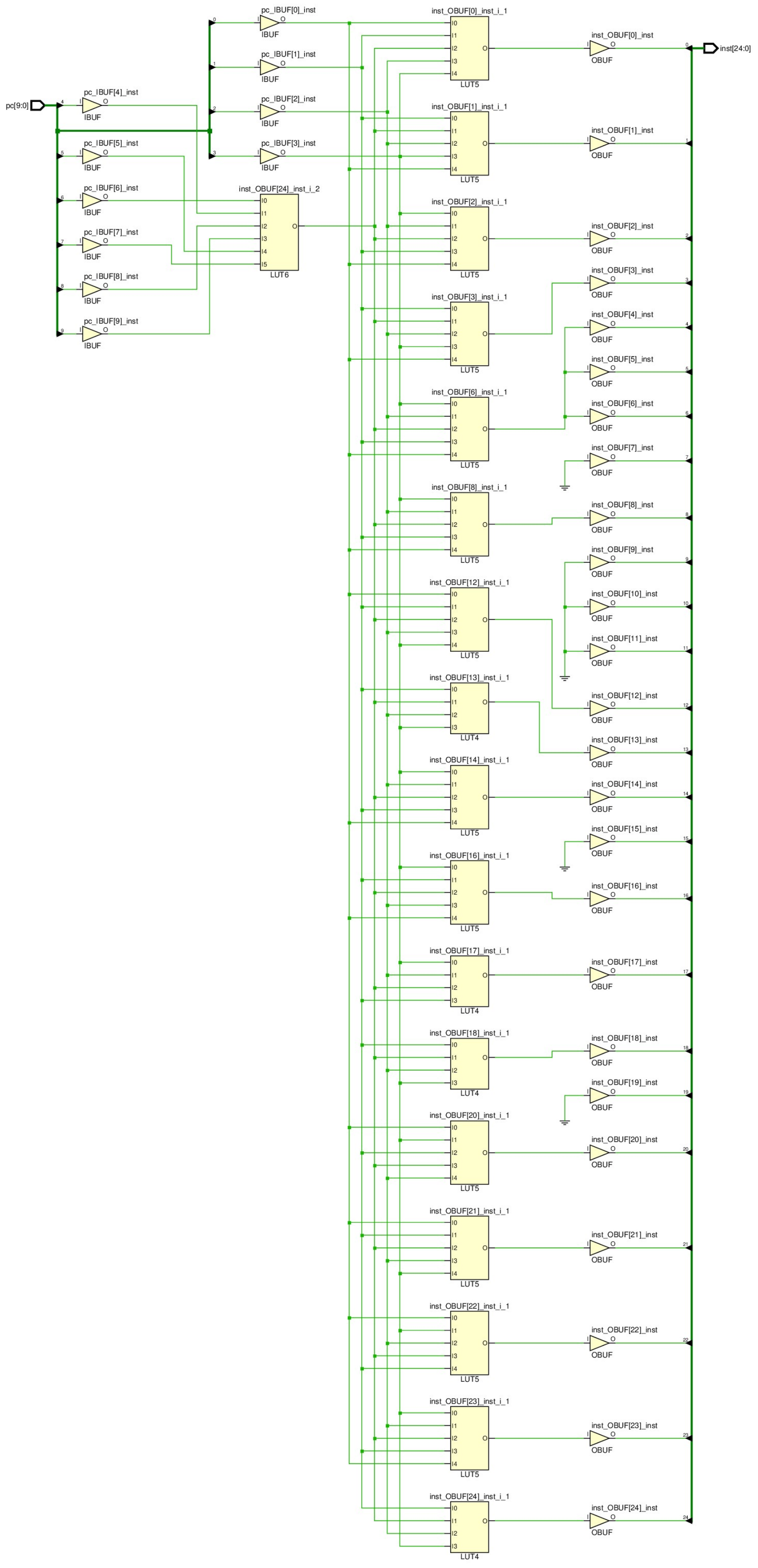


4.2. Synthesis

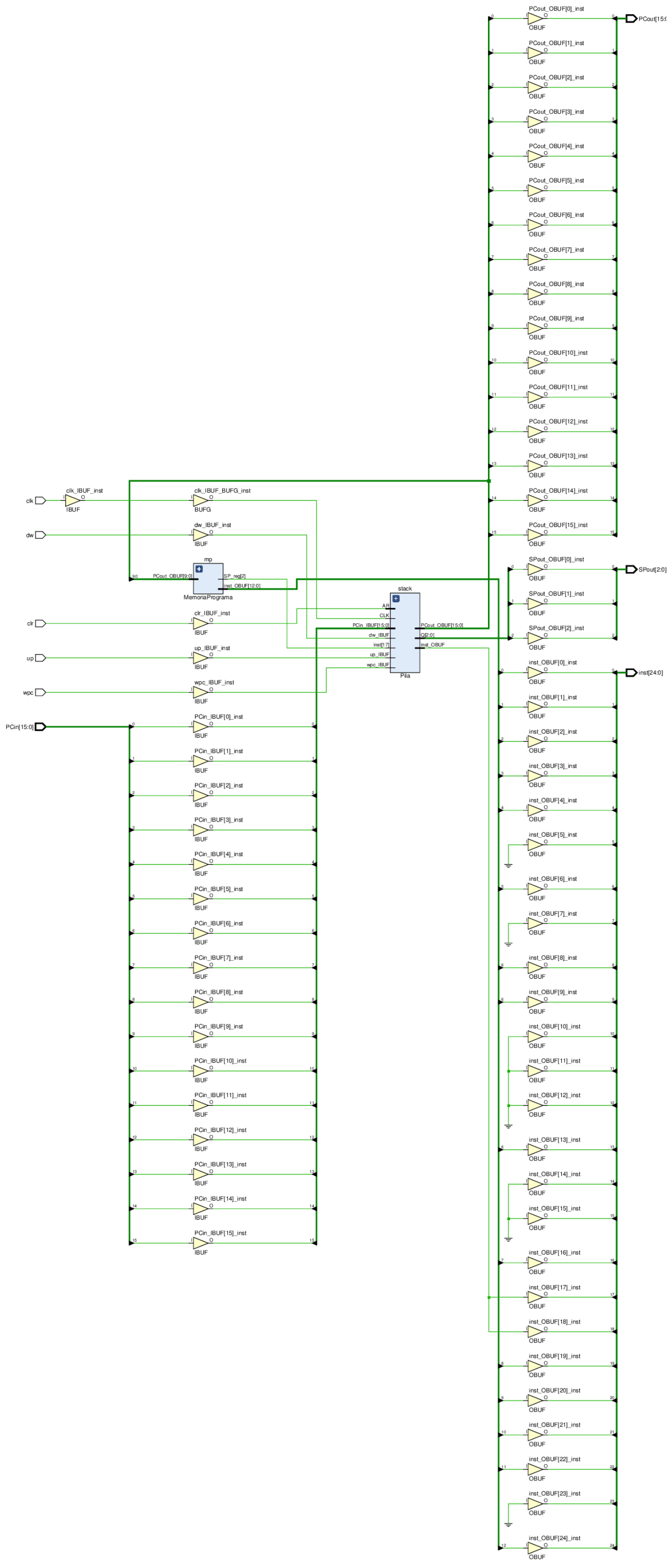
4.2.1. Pila de Hardware



4.2.2. Memoria de Programa



4.2.3. Pila de Hardware y Memoria de Programa Comprimido



4.2.4. Pila de Hardware y Memoria de Programa Expandido