

Instituto Politécnico Nacional
Escuela Superior de Computo



Práctica 13.
Unidad de Control.

Nombre: Flores Castro Luis Antonio.

Arquitectura de Computadoras.

Profesora: Vega García Nayeli.

Código de Implementación de la Carta ASM.

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

entity CartaASM is
Port(clk, clr, EQ, NE, LT, LE, GT, GE, nivel, BGETI, BGTI, BLETI, BLTI,
BNEQI, BEQI, TIPOR: in STD_LOGIC;
SDOPC, SM: out STD_LOGIC);
end CartaASM;

architecture Behavioral of CartaASM is
type estados is (E0);
signal act, sig: estados;

begin
process(clr, clk)
begin
if(clr='1')then
act<=E0;
elsif(rising_edge(clk))then
act<=sig;
end if;
end process;

--comportamiento de carta ASM
process(act, EQ, NE, LT, LE, GT, GE, nivel, BGETI, BGTI, BLETI,
BLTI, BNEQI, BEQI, TIPOR)
begin
SDOPC<='0';
SM<='0';
case act is
when E0=>
if(TIPOR='1')then
SM<='0';
else
if(BEQI='0')then
if(BNEQI='0')then
if(BLTI='0')then
if(BLETI='0')then
if(BGTI='0')then
if(BGETI='0')then
SM<='1';
SDOPC<='1';
else
if(nivel='1')then
SM<='1';

SDOPC<='0';
else
if(GE='1')then
SM<='1';
```

```

SDOPC<='1';
else
SM<='1';
SDOPC<='0';
end if;
end if;
else
if(nivel='1') then
SM<='1';
SDOPC<='0';
else
if(GT='1') then
SM<='1';
SDOPC<='1';
else
SM<='1';
SDOPC<='0';
end if;
end if;
else
if(nivel='1') then
SM<='1';
SDOPC<='0';
else
if(LE='1') then
SM<='1';
SDOPC<='1';
else
SM<='1';
SDOPC<='0';
end if;
end if;
end if;
else
if(nivel='1') then
SM<='1';
SDOPC<='0';
else
if(LT='1') then
SM<='1';
SDOPC<='1';
else
SM<='1';
SDOPC<='0';
end if;
end if;
end if;
else
if(nivel='1') then
SM<='1';

```

```

SDOPC<='0';
else
    if (NE='1') then
        SM<='1';
        SDOPC<='1';
    else
        SM<='1';
        SDOPC<='0';
    end if;
end if;
end if;
else
    if (nivel='1') then
        SM<='1';
        SDOPC<='0';
    else
        if (EQ='1') then
            SM<='1';
            SDOPC<='1';
        else
            SM<='1';
            SDOPC<='0';
        end if;
    end if;
end if;
end if;
sig<=E0;
end case;
end process;
end Behavioral;

```

Código de Implementación del Decodificador de Instrucción.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Decodificador is
    Port (opCode: in STD_LOGIC_VECTOR(4 downto 0);
          TIPOR, BEQI, BNEQI, BLTI, BLETI, BGTI, BGETI: out STD_LOGIC);
end Decodificador;

architecture Behavioral of Decodificador is
begin
    process (opCode)
    begin
        case opCode is
            when "00000" =>
                TIPOR<='1';
                BEQI<='0';
                BNEQI<='0';
                BLTI<='0';
                BLETI<='0';
                BGTI<='0';
                BGETI<='0';
            when "01101" =>
                TIPOR<='0';

```

```

        BEQI<='1';
        BNEQI<='0';
        BLTI<='0';
        BLETI<='0';
        BGTI<='0';
        BGETI<='0';
    when "01110"=>
        TIPOR<='0';
        BEQI<='0';
        BNEQI<='1';
        BLTI<='0';
        BLETI<='0';
        BGTI<='0';
        BGETI<='0';
    when "01111"=>
        TIPOR<='0';
        BEQI<='0';
        BNEQI<='0';
        BLTI<='1';
        BLETI<='0';
        BGTI<='0';
        BGETI<='0';
    when "10000"=>
        TIPOR<='0';
        BEQI<='0';
        BNEQI<='0';
        BLTI<='0';
        BLETI<='1';
        BGTI<='0';
        BGETI<='0';
    when "10001"=>
        TIPOR<='0';
        BEQI<='0';
        BNEQI<='0';
        BLTI<='0';
        BLETI<='0';
        BGTI<='1';
        BGETI<='0';
    when "10010"=>
        TIPOR<='0';
        BEQI<='0';
        BNEQI<='0';
        BLTI<='0';
        BLETI<='0';
        BGTI<='0';
        BGETI<='1';
    when others=>
        TIPOR<='0';
        BEQI<='0';
        BNEQI<='0';
        BLTI<='0';
        BLETI<='0';
        BGTI<='0';
        BGETI<='0';
    end case;
end process;
end Behavioral;

```

Código de Implementación de MfunCode.

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

entity MfunCode is
    Port(funCode: in STD_LOGIC_VECTOR(3 downto 0);
          microfun: out STD_LOGIC_VECTOR(19 downto 0));
end MfunCode;

architecture Behavioral of MfunCode is
    type memo is array(0 to (2**4)-1) of STD_LOGIC_VECTOR(19 downto 0);
    constant aux: memo:=
        "00000100010000110011", --ADD
        "00000100010001110011", --SUB
        "00000100010000000011", --AND
        "00000100010000010011", --OR
        "00000100010000100011", --XOR
        "00000100010011010011", --NAND
        "00000100010011000011", --NOR
        "00000100010010100011", --XNOR
        "00000100010011010011", --NOT
        "00000001110000000000", --SLL
        "00000001010000000000", --SRL
        others=>(others=>'0'));

    begin
        microfun<=aux(conv_integer(funCode));
    end Behavioral;
```

Código de Implementación de MopCode.

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

entity MopCode is
    Port(opCode: in STD_LOGIC_VECTOR(4 downto 0);
          microOp: out STD_LOGIC_VECTOR(19 downto 0));
end MopCode;

architecture Behavioral of MopCode is
    type memoc is array (0 to (2**5)-1) of STD_LOGIC_VECTOR(19 downto 0);
    constant aux: memoc:=
        "00001000000001110001", -- VERIF
        "00000000010000000000", -- LI
        "00000100010000001000", -- LWI
        "000010000000000001100", -- SWI
        "00001010000100110101", -- SW
        "00000100010100110011", -- ADDI
        "00000100010101110011", -- SUBI
```

```

"00000100010100000011", -- ANDI
"00000100010100010011", -- ORI
"00000100010100100011", -- XORI
"00000100010111010011", -- NANDI
"00000100010111000011", -- NORI
"00000100010110100011", -- XNORI
"10010000001100110011", -- BEQI
"10010000001100110011", -- BNEI
"10010000001100110011", -- BLTI
"10010000001100110011", -- BLETI
"10010000001100110011", -- BGTI
"10010000001100110011", -- BGETI
"00010000000000000000", -- B
"01010000000000000000", -- CALL
"00100000000000000000", -- RET
"00000000000000000000", -- NOP
"00000110010100110001", -- LW

```

```

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

    microOp<=aux(conv_integer(opCode));
end Behavioral;

```

Código de Implementación de Multiplexor MopCode.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Multiplexorop is
Port(opCode: in STD_LOGIC_VECTOR(4 downto 0);
      SDOPC: in STD_LOGIC;
      sout: out STD_LOGIC_VECTOR(4 downto 0));
end Multiplexorop;

architecture Behavioral of Multiplexorop is
    constant inicial: STD_LOGIC_VECTOR(4 downto 0):="00000";
    begin

        with SDOPC select
            sout<=
                opCode when '1',
                inicial when others;
    end Behavioral;

```

Código de Implementación de Multiplexor Microinstrucción.

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

entity Multiplexorpr is
Port(microfun, microop: in STD_LOGIC_VECTOR(19 downto 0);
      SM: in STD_LOGIC;
      salida: out STD_LOGIC_VECTOR(19 downto 0));
end Multiplexorpr;

architecture Behavioral of Multiplexorpr is
begin
    with SM select
        salida<=
            microop when '1',
            microfun when others;
end Behavioral;
```

Código de Implementación de Bloque de Condición.

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

entity Condicion is
Port(Q: in STD_LOGIC_VECTOR(3 downto 0);
      EQ, NE, LT, LE, GT, GE: out STD_LOGIC);
end Condicion;

architecture Behavioral of Condicion is
begin

    EQ<=Q(1);
    NE<=not Q(1);
    LT<=not Q(0);
    LE<=not Q(0) or Q(1);
    GT<=Q(0) and not Q(1);
    GE<=Q(0);
end Behavioral;
```

Código de Implementación de Bloque de Nivel.

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

entity Nivel is
Port(clk, clr: in STD_LOGIC;
      nivel: out STD_LOGIC);
end Nivel;

architecture Behavioral of Nivel is
    signal pclk, nclk: STD_LOGIC;
begin
    process(clk,clr)
```



```

begin
    if(clr='1')then
        pclk<='0';
    elsif(rising_edge(clk))then
        pclk<=not pclk;
    end if;
end process;

process(clk, clr)
begin
    if(clr='1')then
        nclk<='0';
    elsif(falling_edge(clk))then
        nclk<=not nclk;
    end if;
end process;

nivel<=pclk xor nclk;
end Behavioral;

```

Código de Implementación de Bloque de Registro de Banderas.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Regf is
Port(bandera: in STD_LOGIC_VECTOR(3 downto 0);
      clk, clr, LF: in STD_LOGIC;
      Q: out STD_LOGIC_VECTOR(3 downto 0));
end Regf;

architecture Behavioral of Regf is
begin
    process(clk, clr)
    begin
        if(clr='1')then
            Q<=(others=>'0');
        elsif(falling_edge(clk))then
            if(LF='1')then
                Q<=bandera;
            end if;
        end if;
    end process;
end Behavioral;

```

Código de Test Bench de la Carta ASM.

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

entity TbCarta is
end TbCarta;

architecture Behavioral of TbCarta is

    component CartaASM is
    Port(clk, clr, EQ, NE, LT, LE, GT, GE, nivel, BGETI, BGTI, BLETI, BLTI,
    BNEQI, BEQI, TIPOR: in STD_LOGIC;
        SDOPC, SM: out STD_LOGIC);
    end component;

    signal clk, clr, EQ, NE, LT, LE, GT, GE, nivel, BGETI, BGTI, BLETI,
    BLTI, BNEQI, BEQI, TIPOR, SDOPC, SM: STD_LOGIC:= '0';

begin
    carta: CartaASM
    Port map(
        clk=>clk,
        clr=>clr,
        EQ=>EQ,
        NE=>NE,
        LT=>LT,
        LE=>LE,
        GT=>GT,
        GE=>GE,
        nivel=>nivel,
        BGETI=>BGETI,
        BGTI=>BGTI,
        BLETI=>BLETI,
        BLTI=>BLTI,
        BNEQI=>BNEQI,
        BEQI=>BEQI,
        TIPOR=>TIPOR,
        SDOPC=>SDOPC,
        SM=>SM
    );

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

    process
    begin
        clr<='1';
        wait until rising_edge(clk);

        clr<='0';
```

```

    TIPOR<='1';
    wait until rising_edge(clk);

    TIPOR<='0';
    LT<='1';
    BLTI<='1';
    wait until rising_edge(clk);

    LT<='0';
    wait until rising_edge(clk);

    BLTI<='0';
    LE<='1';
    BLETI<='1';
    wait until rising_edge(clk);

    LE<='0';
    wait until rising_edge(clk);

    clr<='1';
    wait;
    end process;

end Behavioral;

```

Código de Test Bench del Decodificador de Instrucción.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TbDecodificador is
end TbDecodificador;

architecture Behavioral of TbDecodificador is
    component Decodificador is
        Port(opCode: in STD_LOGIC_VECTOR(4 downto 0);
            TIPOR, BEQI, BNEQI, BLTI, BLETI, BGTI, BGETI: out STD_LOGIC);
    end component;

    signal opCode: STD_LOGIC_VECTOR(4 downto 0);
    signal TIPOR, BEQI, BNEQI, BLTI, BLETI, BGTI, BGETI: STD_LOGIC;

begin
    tbdeco: Decodificador
        Port map(
            opCode=>opCode,
            TIPOR=>TIPOR,
            BEQI=>BEQI,
            BNEQI=>BNEQI,
            BLTI=>BLTI,
            BLETI=>BLETI,
            BGTI=>BGTI,
            BGETI=>BGETI
        );

    process

```

```

begin
  opCode<="00000";
  wait for 10 ns;
  opCode<="01101";
  wait for 10 ns;
  opCode<="01110";
  wait for 10 ns;
  opCode<="01111";
  wait for 10 ns;
  opCode<="10000";
  wait for 10 ns;
  opCode<="10001";
  wait for 10 ns;
  opCode<="10010";
  wait;
end process;
end Behavioral;

```

Código de Test Bench de MfunCode.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TbMfunCode is
end TbMfunCode;

architecture Behavioral of TbMfunCode is

  component MfunCode is
    Port(funCode: in STD_LOGIC_VECTOR(3 downto 0);
          microfun: out STD_LOGIC_VECTOR(19 downto 0));
  end component;

  signal funCode: STD_LOGIC_VECTOR(3 downto 0);
  signal microfun: STD_LOGIC_VECTOR(19 downto 0);

begin

  fun: MfunCode
    Port Map(
      funCode=>funCode,
      microfun=>microfun
    );

  process
  begin
    funCode<="0000";
    wait for 10 ns;

    funCode<="0001";
    wait for 10 ns;

    funCode<="0010";
    wait for 10 ns;

    funCode<="0011";

```

```

        wait for 10 ns;

        funCode<="0100";
        wait for 10 ns;

        funCode<="0101";
        wait for 10 ns;

        funCode<="0110";
        wait for 10 ns;

        funCode<="0111";
        wait for 10 ns;

        funCode<="1000";
        wait for 10 ns;

        wait;
    end process;
end Behavioral;

```

Código de Test Bench de MopCode.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TbMopCode is
end TbMopCode;

architecture Behavioral of TbMopCode is

    component MopCode is
        Port (opCode: in STD_LOGIC_VECTOR(4 downto 0);
              microOp: out STD_LOGIC_VECTOR(19 downto 0));
    end component;

    signal opCode:STD_LOGIC_VECTOR(4 downto 0);
    signal microOp:STD_LOGIC_VECTOR(19 downto 0);

begin

    opco: MopCode
        Port map(
            opCode=>opCode,
            microOp=>microOp
        );

    process
    begin
        opCode<="00000";
        wait for 10 ns;
        opCode<="00001";
        wait for 10 ns;
        opCode<="00010";
        wait for 10 ns;
    end process;

```

```

        opCode<="00011"
        wait for 10 ns;
        opCode<="00100"
        wait for 10 ns;
        opCode<="00101"
        wait for 10 ns;
        opCode<="00110"
        wait for 10 ns;
        opCode<="00111"
        wait for 10 ns;
        opCode<="01000"
        wait for 10 ns;
        wait;
    end Process;
end Behavioral;

```

Código de Test Bench del Bloque de Condición.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TbCondicion is
end TbCondicion;

architecture Behavioral of TbCondicion is

    component Condicion is
        Port(Q: in STD_LOGIC_VECTOR(3 downto 0);
            EQ, NE, LT, LE, GT, GE: out STD_LOGIC);
    end component;

    signal EQ, NE, LT, LE, GT, GE: STD_LOGIC;
    signal Q: STD_LOGIC_VECTOR(3 downto 0);

begin

    condi: Condicion
    Port map(
        Q=>Q,
        EQ=>EQ,
        NE=>NE,
        LT=>LT,
        LE=>LE,
        GT=>GT,
        GE=>GE
    );

    process
    begin
        Q<="0010";
        wait for 10 ns;

        Q<="1101";
        wait for 10 ns;
    end process;
end Behavioral;

```

```

        Q<="0001";
        wait for 10 ns;

        Q<="0000";
        wait for 10 ns;

        Q<="1111";
        wait;
    end process;
end Behavioral;

```

Código de Test Bench del Bloque de Nivel.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Tbnivel is
end Tbnivel;

architecture Behavioral of Tbnivel is

    component Nivel is
    Port(clk, clr: in STD_LOGIC;
          nivel: out STD_LOGIC);
    end component;

    signal clr,clk, niv: STD_LOGIC;

begin

    level: Nivel
        Port map(
            clk=>clk,
            clr=>clr,
            nivel=>niv
        );

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

    process
    begin
        clr<='1';
        wait for 3 ns;
        clr<='0';
        wait;
    end process;
end Behavioral;

```

Código de Test Bench del Registro de Banderas.

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

entity TbBanderas is
end TbBanderas;

architecture Behavioral of TbBanderas is

    component Regf is
    Port (bandera: in STD_LOGIC_VECTOR(3 downto 0);
          clk, clr, LF: in STD_LOGIC;
          Q: out STD_LOGIC_VECTOR(3 downto 0));
    end component;

    signal clr, clk, LF: STD_LOGIC;
    signal bandera, Q: STD_LOGIC_VECTOR(3 downto 0);

begin

    regb: Regf
    Port map(
        bandera=>bandera,
        clr=>clr,
        clk=>clk,
        LF=>LF,
        Q=>Q
    );

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

    process
    begin
        clr<='1';
        wait until falling_edge(clk);
        clr<='0';
        LF<='1';
        bandera<="1111";
        wait until falling_edge(clk);
        bandera<="0000";
        wait until falling_edge(clk);
        bandera<="0001";
        wait until falling_edge(clk);
        bandera<="0010";
        wait until falling_edge(clk);
        bandera<="0011";
        wait until falling_edge(clk);
        LF<='0';
        bandera<="0000";
        wait until falling_edge(clk);
```



```

    bandera<="0001";
    wait;
    end process;
end Behavioral;

```

Código de Paquete de la Unidad de Control.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

package PaqueteUnidad is

    --CartaASM
    component CartaASM is
        Port(clk, clr, EQ, NE, LT, LE, GT, GE, nivel, BGETI, BGTI, BLETI,
        BLTI, BNEQI, BEQI, TIPOR: in STD_LOGIC;
        SDOPC, SM: out STD_LOGIC);
    end component;

    --BLOQUE DE CONDICION
    component Condicion is
        Port(Q: in STD_LOGIC_VECTOR(3 downto 0);
        EQ, NE, LT, LE, GT, GE: out STD_LOGIC);
    end component;

    --DECODIFICADOR
    component Decodificador is
        Port(opCode: in STD_LOGIC_VECTOR(4 downto 0);
        TIPOR, BEQI, BNEQI, BLTI, BLETI, BGTI, BGETI: out STD_LOGIC);
    end component;

    --MFUNCODE
    component MfunCode is
        Port(funCode: in STD_LOGIC_VECTOR(3 downto 0);
        microfun: out STD_LOGIC_VECTOR(19 downto 0));
    end component;

    --MOPCODE
    component MopCode is
        Port (opCode: in STD_LOGIC_VECTOR(4 downto 0);
        microOp: out STD_LOGIC_VECTOR(19 downto 0));
    end component;

    --MULTIPLEXORES
    component Multiplexorop is
        Port(opCode: in STD_LOGIC_VECTOR(4 downto 0);
        SDOPC: in STD_LOGIC;
        sout: out STD_LOGIC_VECTOR(4 downto 0));
    end component;

    component Multiplexorpr is
        Port(microfun, microop: in STD_LOGIC_VECTOR(19 downto 0);
        SM: in STD_LOGIC;
        salida: out STD_LOGIC_VECTOR(19 downto 0));
    end component;
end PaqueteUnidad;

```

```

end component;

--NIVEL
component Nivel is
Port(clk, clr: in STD_LOGIC;
nivel: out STD_LOGIC);
end component;

--BANDERAS
component Regf is
Port(bandera: in STD_LOGIC_VECTOR(3 downto 0);
clk, clr, LF: in STD_LOGIC;
Q: out STD_LOGIC_VECTOR(3 downto 0));
end component;

end package;

```

Código de Implementación de la Unidad de Control.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.PaqueteUnidad.all;

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

architecture Behavioral of UnidadControl is

signal auxCondicion: STD_LOGIC_VECTOR(5 downto 0);
signal auxDecodificador: STD_LOGIC_VECTOR(6 downto 0);
signal auxNivel, auxSM, auxSDOPC: STD_LOGIC;
signal auxQ: STD_LOGIC_VECTOR(3 downto 0);
signal auxopCode: STD_LOGIC_VECTOR(4 downto 0);
signal auxMicrofun, auxMicroop, auxS: STD_LOGIC_VECTOR(19 downto 0);

begin

--para cartaASM
UC: CartaASM
Port map(
    clk=>clk,
    clr=>clr,
    EQ=>auxCondicion(5),
    NE=>auxCondicion(4),
    LT=>auxCondicion(3),
    LE=>auxCondicion(2),
    GT=>auxCondicion(1),
    GE=>auxCondicion(0),
    nivel=>auxNivel,
    BGETI=>auxDecodificador(0),
    BGTI=>auxDecodificador(1),

```

```

        BLETI=>auxDecodificador(2),
        BLTI=>auxDecodificador(3),
        BNEQI=>auxDecodificador(4),
        BEQI=>auxDecodificador(5),
        TIPOR=>auxDecodificador(6),
        SDOPC=>auxSDOPC,
        SM=>auxSM
    );

condi: Condicion
Port map(
    Q=>auxQ,
    EQ=>auxCondicion(5),
    NE=>auxCondicion(4),
    LT=>auxCondicion(3),
    LE=>auxCondicion(2),
    GT=>auxCondicion(1),
    GE=>auxCondicion(0)
);

decodi: Decodificador
Port map(
    opCode=>opCode,
    TIPOR=>auxDecodificador(6),
    BEQI=>auxDecodificador(5),
    BNEQI=>auxDecodificador(4),
    BLTI=>auxDecodificador(3),
    BLETI=>auxDecodificador(2),
    BGTI=>auxDecodificador(1),
    BGETI=>auxDecodificador(0)
);

mfun: MfunCode
Port map(
    funCode=>funCode,
    microfun=>auxMicrofun
);

mop: MopCode
Port map(
    opCode=>auxopCode,
    microOp=>auxMicroop
);

muxu: Multiplexorop
Port map(
    opCode=>opCode,
    SDOPC=>auxSDOPC,
    sout=>auxopCode
);

muxd: Multiplexorpr
Port map(
    microfun=>auxMicrofun,
    microop=>auxMicroop,
    SM=>auxSM,
    salida=>auxS

```

```

);

niv: Nivel
Port map(
clk=>clk,
clr=>clr,
nivel=>auxNivel
);

microInstruccion<=auxS;
n<=auxNivel;

ban: Regf
Port map(
bandera=>banderas,
clk=>clk,
clr=>clr,
LF=>LF,
Q=>auxQ
);

end Behavioral;

```

Código de Test Bench de la Unidad de Control.

```

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;
use work.PaqueteUnidad.ALL;

entity TbUnidadControl is
end TbUnidadControl;

architecture Behavioral of TbUnidadControl is

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

signal clk, clr, LF, n: STD_LOGIC:='0';
signal opCode: STD_LOGIC_VECTOR (4 downto 0):="00000";
signal funCode, banderas: STD_LOGIC_VECTOR (3 downto 0):="0000";
signal microInstruccion: STD_LOGIC_VECTOR(19 downto 0):=(others=>'0');

begin

UC: UnidadControl

```

```

Port map(
opCode=>opCode,
funCode=>funCode,
banderas=>banderas,
clk=>clk,
clr=>clr,
LF=>LF,
n=>n,
microInstruccion=>microInstruccion
);

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

process

    file RES: TEXT;
    variable L_RE: line;
    variable vmicroInstruccion: STD_LOGIC_VECTOR(19 downto 0);
    variable vnivel: string(1 to 4);

    file STIMU: TEXT;
    variable L_E: line;
    variable vopCode: STD_LOGIC_VECTOR(4 downto 0);
    variable vfunCode, vbanderas: STD_LOGIC_VECTOR(3 downto 0);
    variable vclr, vLF: STD_LOGIC;
    variable CADENA: string(1 to 8);
    variable CADENA2: string(1 to 19);
    variable CADENASL: string(1 to 2);

begin
    CADENASL:= "  ";
    file_open(STIMU, "C:\Users\Luis FC\Documents\Semestre 21-
2\Arquitectura de Computadoras\Unidad de Control\Unidad\ESTIMULOS.TXT",
READ_MODE);
    file_open(RES, "C:\Users\Luis FC\Documents\Semestre 21-
2\Arquitectura de Computadoras\Unidad de Control\Unidad\RESULTADOS.TXT",
WRITE_MODE);

    CADENA:="OPCODE ";
    write(L_RE, CADENA, left, 7);

    CADENA:="FUNCODE ";
    write(L_RE, CADENA, left, 7);

    CADENA:="BANDERAS";
    write(L_RE, CADENA, left, 7);

    CADENA:="      CLR ";
    write(L_RE, CADENA, left, 7);

    CADENA:="      LF  ";

```

```

write(L_RE, CADENA, left, 4);

CADENA:="MICROINS";
write(L_RE, CADENA, left, 20);

CADENA2:="          NIVEL          ";
write(L_RE, CADENA2, left, 7);

writeline(RES, L_RE);

while not endfile(STIMU) loop

    readline(STIMU,L_E);

    read(L_E,vopCode);
    opCode<=vopCode;

    read(L_E, vfunCode);
    funCode<=vfunCode;

    read(L_E, vbanderas);
    banderas<=vbanderas;

    read(L_E, vclr);
    clr<=vclr;

    read(L_E, vLF);
    LF<=vLF;

    wait for 10 ns;
    vmicroInstruccion:=microInstruccion;

    if(n='1') then
        vnivel:="ALTO";
    else
        vnivel:="BAJO";
    end if;

    write(L_RE, vopCode, left, 10);
    write(L_RE, vfunCode, left, 9);
    write(L_RE, vbanderas, left, 10);
    write(L_RE, vclr, left, 8);
    write(L_RE, vLF, left, 9);
    write(L_RE, vmicroInstruccion, left, 22);
    write(L_RE, vnivel, left, 5);
    writeline(RES,L_RE);

    wait for 5 ns;
    vmicroInstruccion:=microInstruccion;

    if(n='1') then
        vnivel:="ALTO";
    else
        vnivel:="BAJO";
    end if;

    write(L_RE, vopCode, left, 10);

```

```

        write(L_RE, vfunCode, left, 9);
        write(L_RE, vbanderas, left, 10);
        write(L_RE, vclr, left, 8);
        write(L_RE, vLF, left, 9);
        write(L_RE, vmicroInstruccion, left, 22);
        write(L_RE, vnivel, left, 5);
        writeline(RES,L_RE);

        write(L_RE, CADENASL, left, 7);
        writeline(RES,L_RE);

    end loop;

    file_close(STIMU);
    file_close(RES);
    wait;
    end process;
end Behavioral;

```

Diagrama RTL de la Carta ASM.

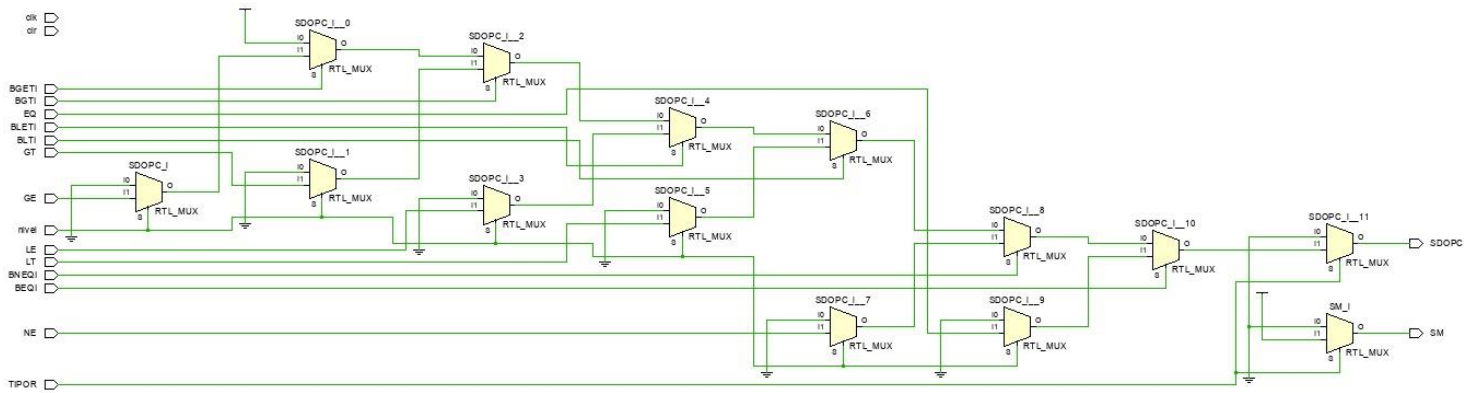


Figura 1. Diagrama RTL de la Carta ASM.

Diagrama RTL del Decodificador de Instrucción.

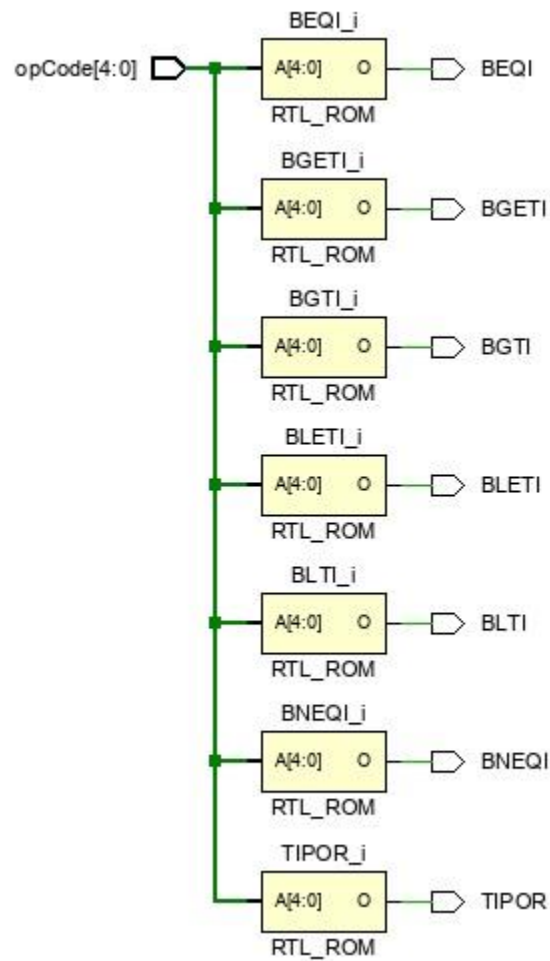


Figura 1.1 Diagrama RTL del Decodificador de Instrucción.

Diagrama RTL de MfunCode

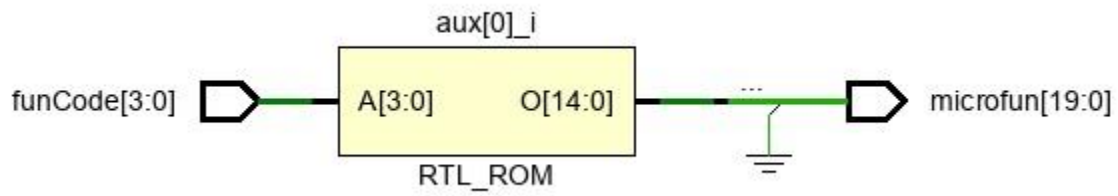


Figura 1.2 Diagrama RTL de MfunCode.

Diagrama RTL del MopCode.

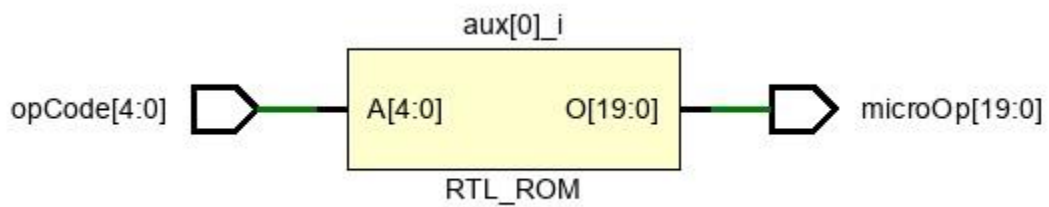


Figura 1.3 Diagrama RTL del MopCode.

Diagrama RTL del Multiplexor MopCode.

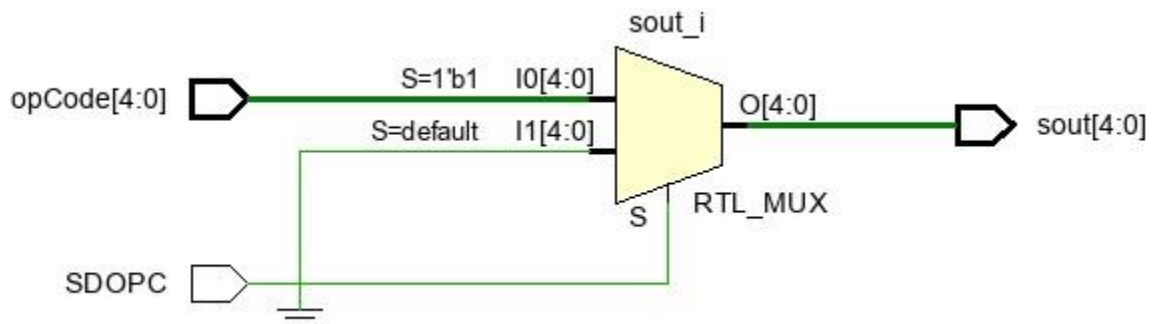


Figura 1.4 Diagrama RTL del Multiplexor MopCode.

Diagrama RTL del Multiplexor Microinstrucción.

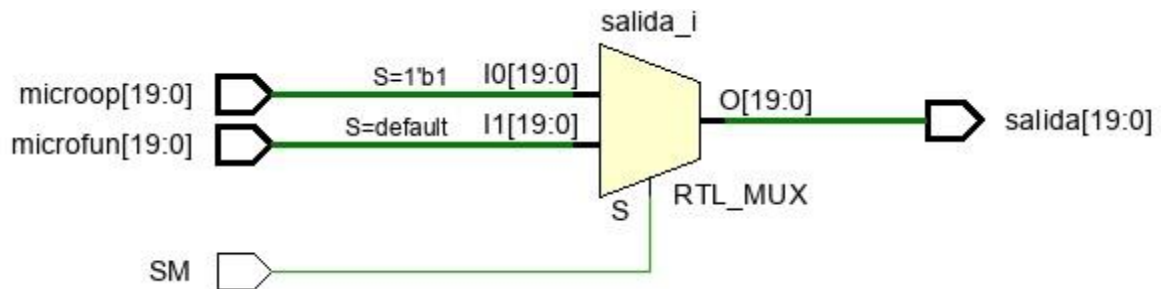


Figura 1.5 Diagrama RTL del Multiplexor Microinstrucción

Diagrama RTL del Bloque de Condición.

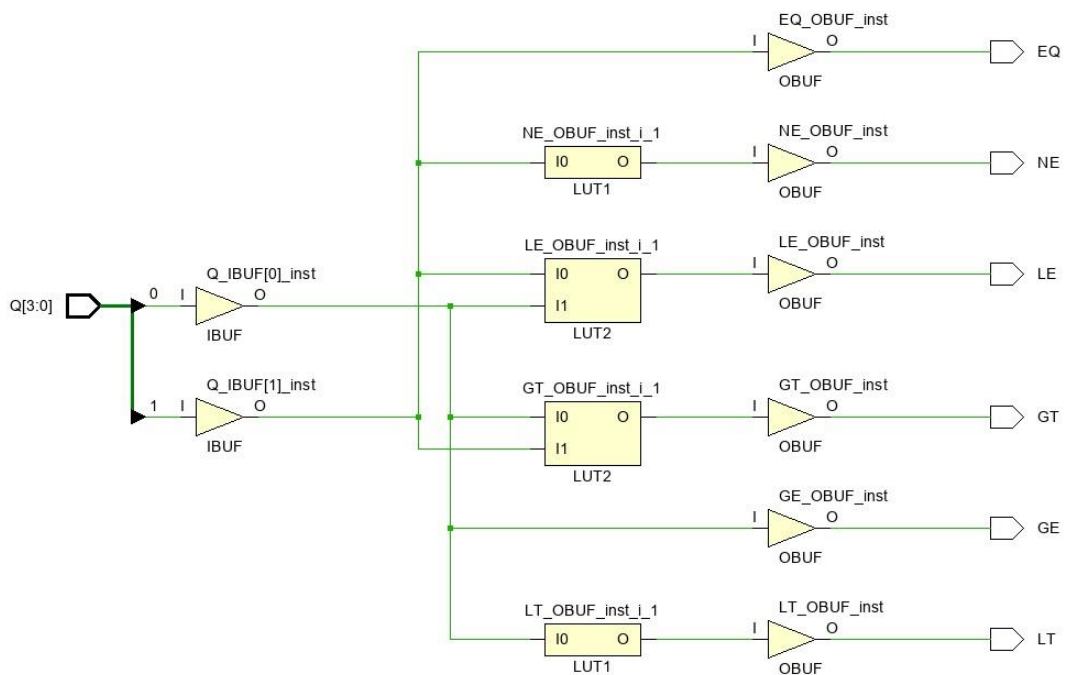


Figura 1.6 Diagrama RTL del Bloque de Condición.

Diagrama RTL del Bloque de Nivel.

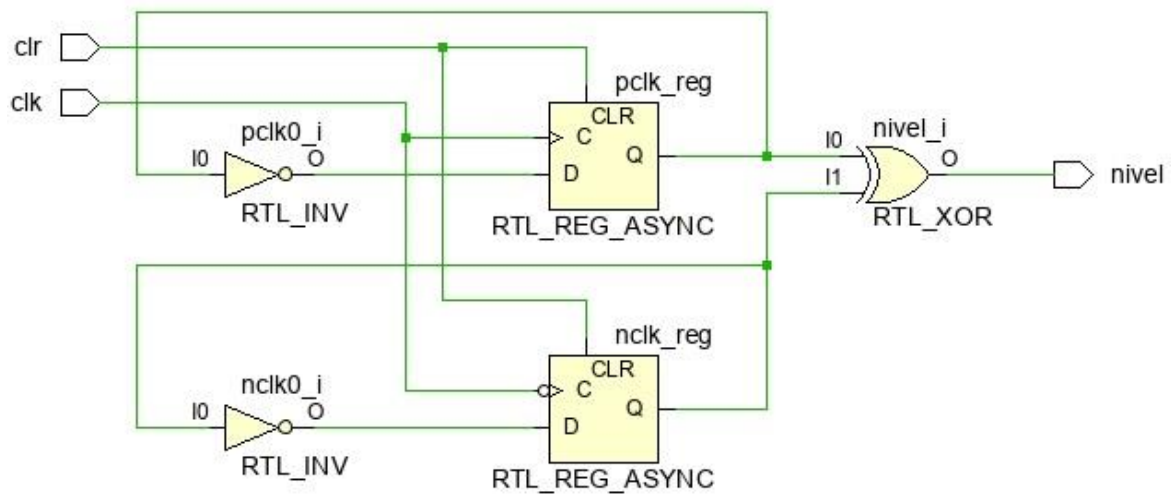


Figura 1.7 Diagrama RTL del Bloque de Nivel.

Diagrama RTL del Registro de Banderas.

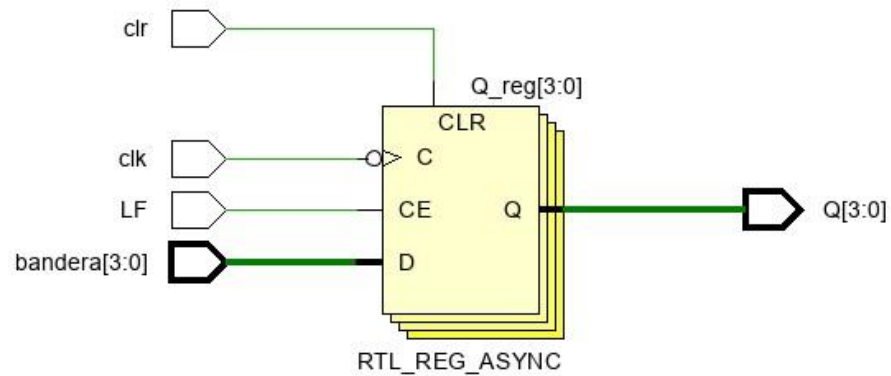


Figura 1.8 Diagrama RTL del Registro de Banderas.

Diagrama RTL de la Unidad de Control.

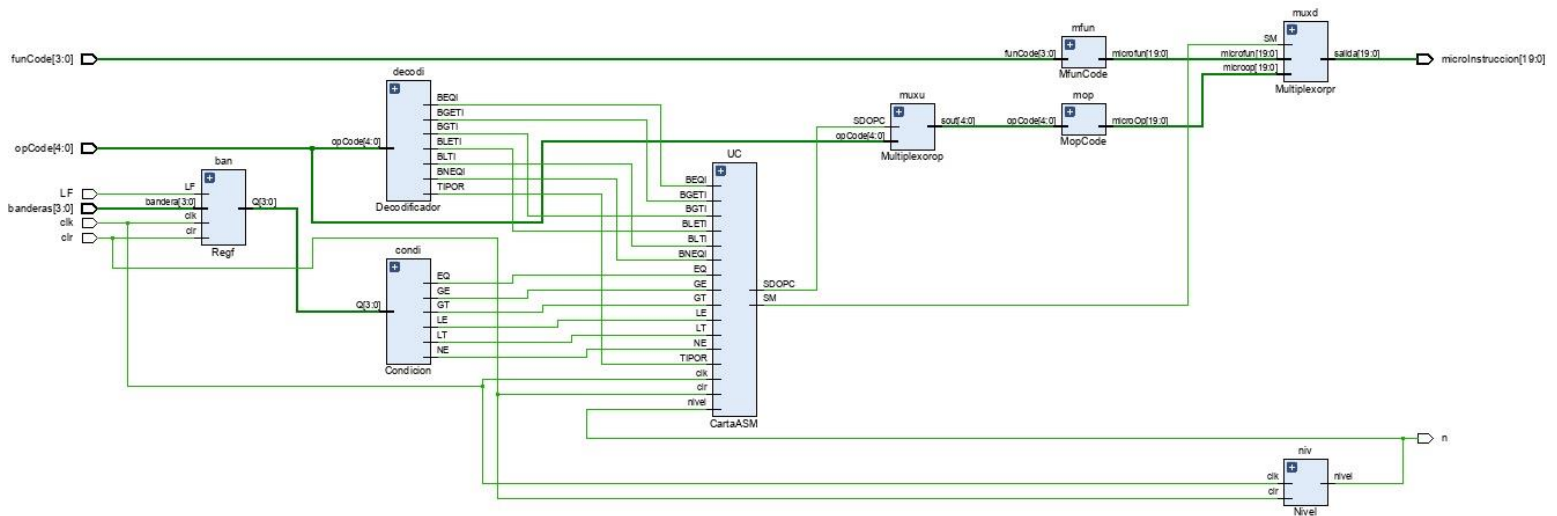


Figura 1.9 Diagrama RTL de la Unidad de Control.

Diagrama lógico de la Carta ASM.

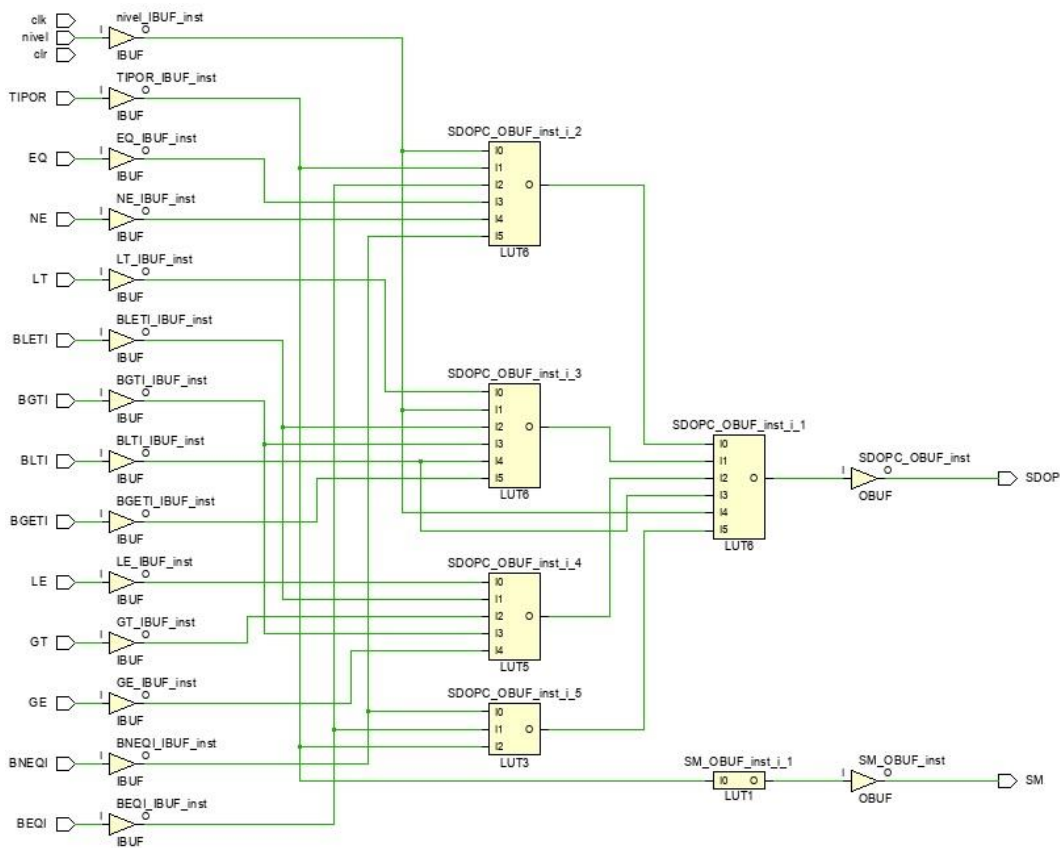


Figura 2 Diagrama lógico de la Carta ASM.

Diagrama lógico del Decodificador de Instrucción.

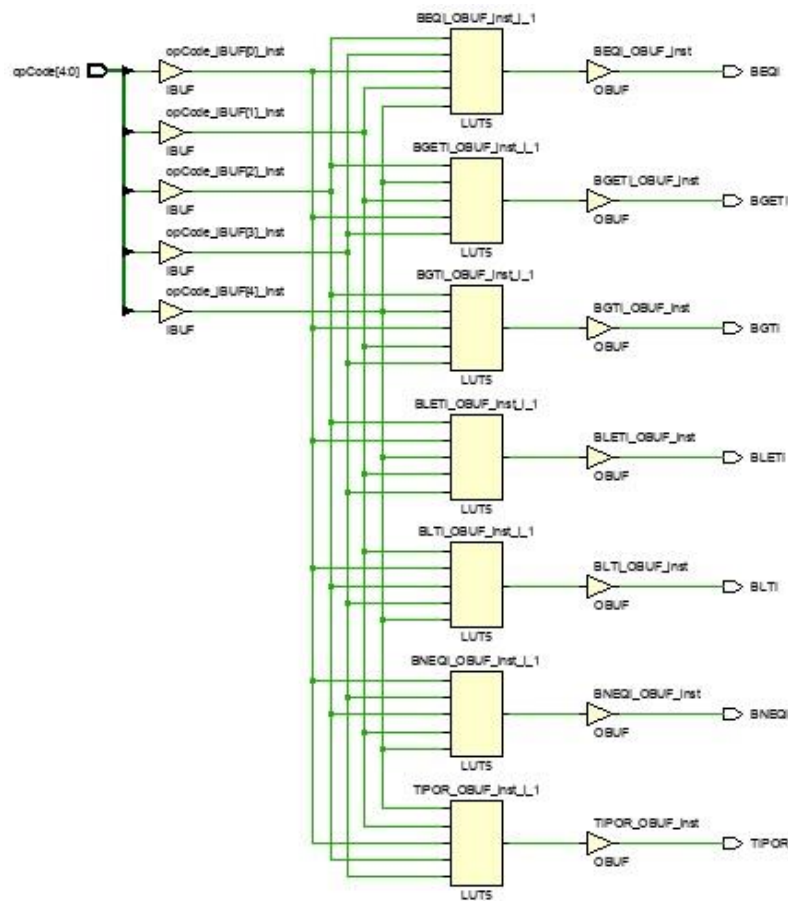


Figura 2.1 Diagrama lógico del Decodificador de Instrucción.

Diagrama lógico de MfunCode.

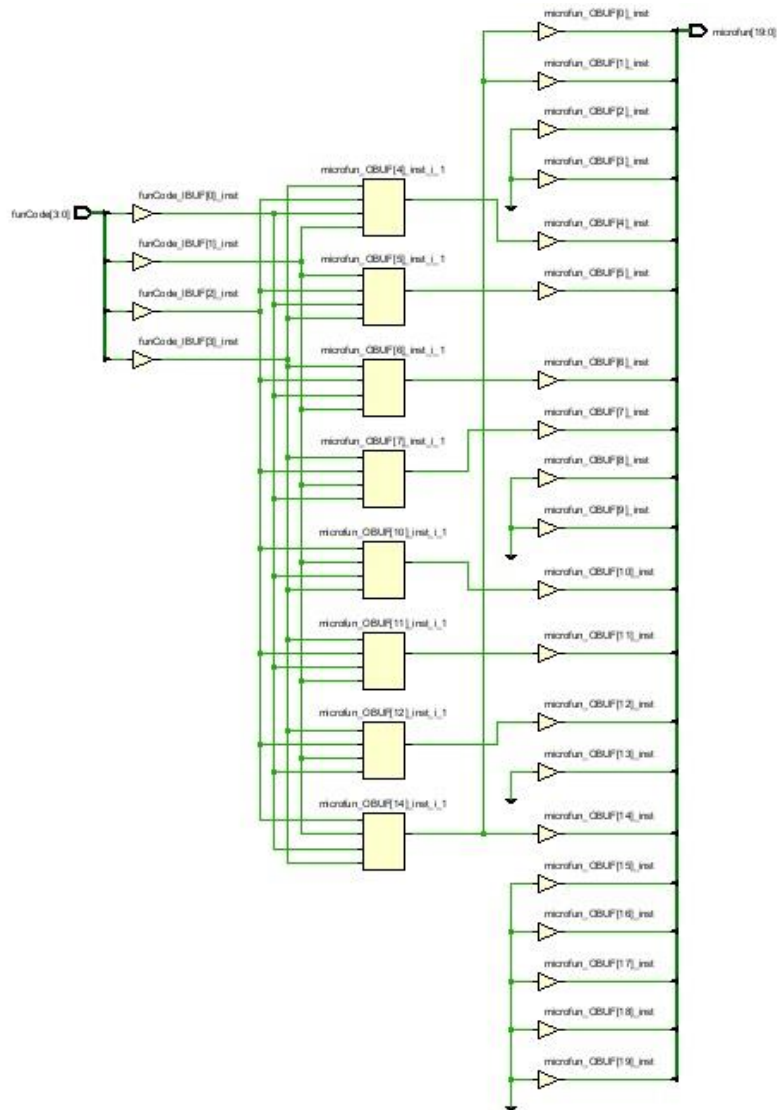


Figura 2.2 Diagrama lógico de MfunCode.

Diagrama lógico de MopCode.

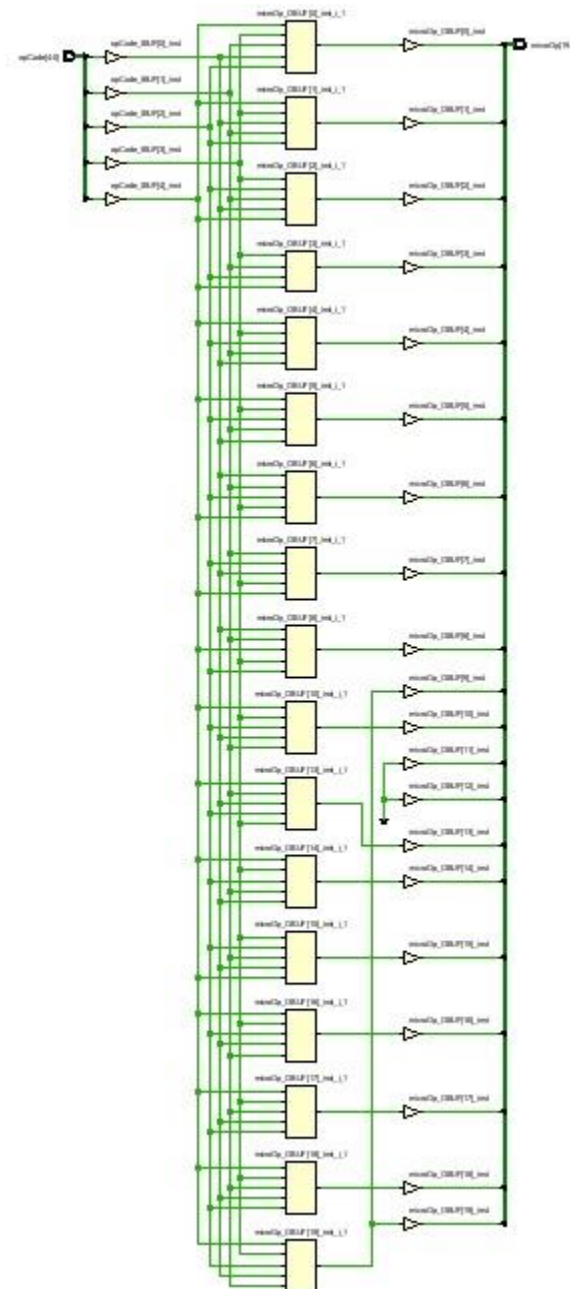


Figura 2.3 Diagrama lógico de MopCode.

Diagrama lógico del Multiplexor MopCode.

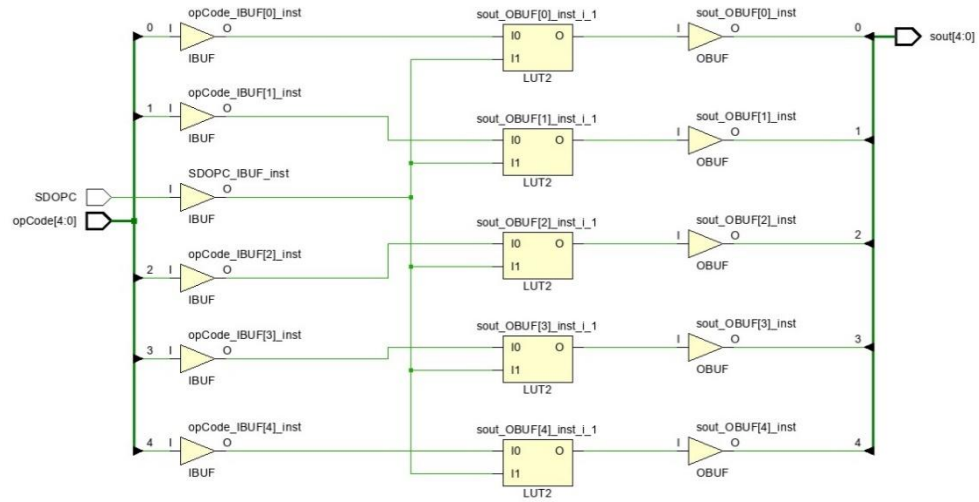


Figura 2.4 Diagrama lógico del Multiplexor MopCode.

Diagrama lógico del Multiplexor Microinstrucción.

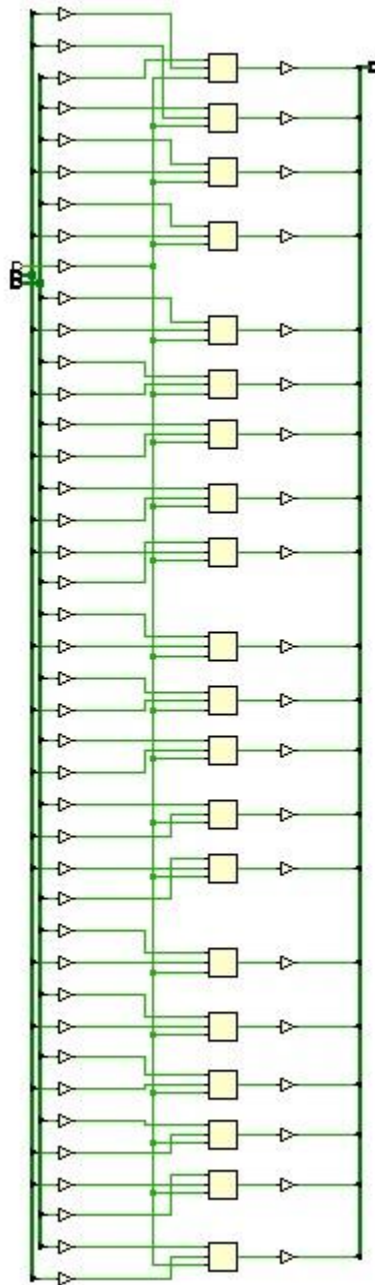


Figura 2.5 Diagrama lógico del Multiplexor Microinstrucción.

Diagrama lógico del Bloque de Condición

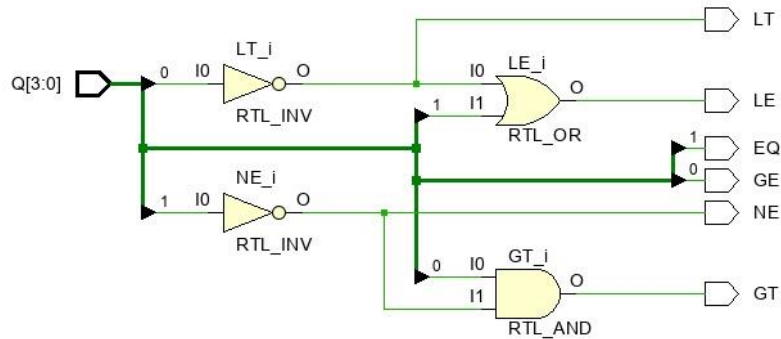


Figura 2.6 Diagrama lógico del Bloque de Condición.

Diagrama lógico del Bloque de Nivel.

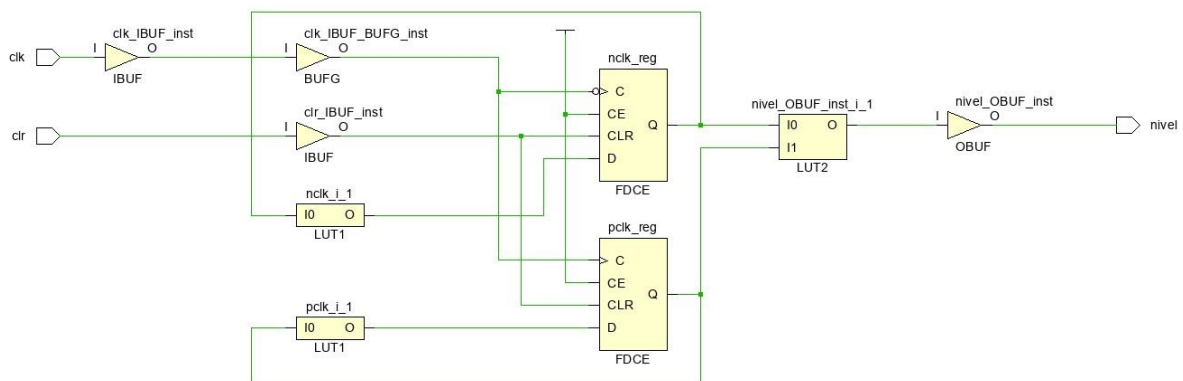


Figura 2.7 Diagrama lógico del Bloque de Nivel.

Diagrama lógico del Registro de Banderas.

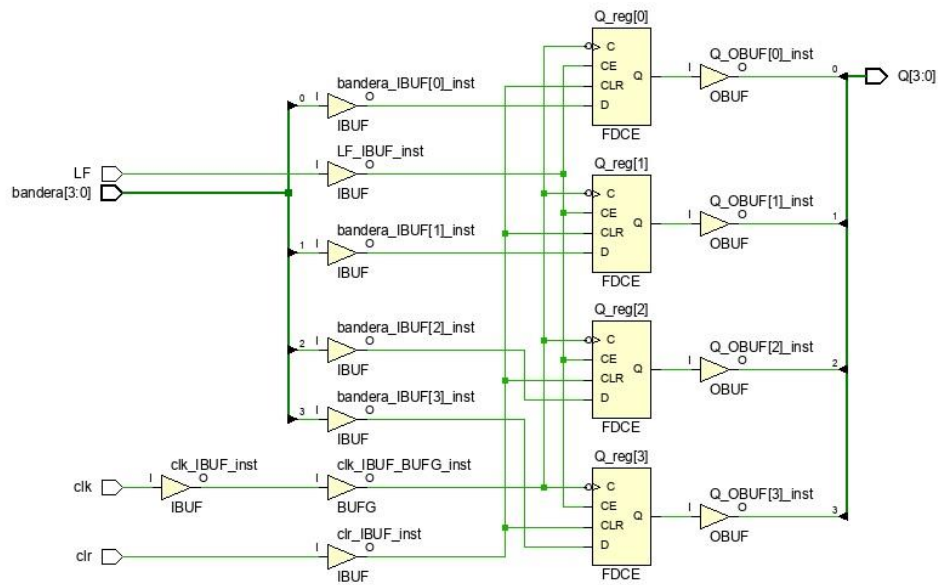


Figura 2.8 Diagrama lógico del Registro de Banderas.

Diagrama lógico de la Unidad de Control.

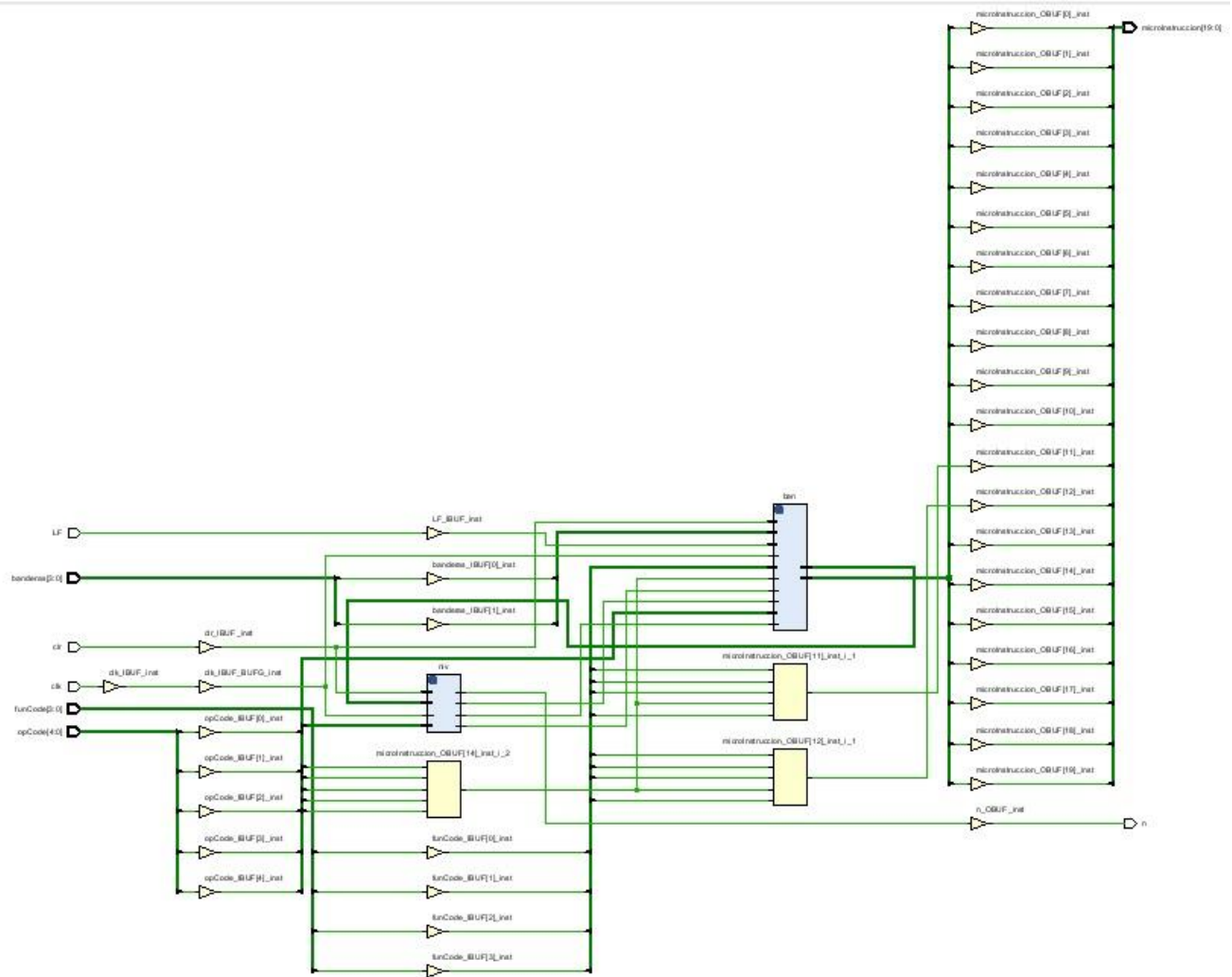


Figura 2.9 Diagrama lógico de la Unidad de Control.

Diagrama de Onda de la Carta ASM.



Figura 3. Diagrama de Onda de la Carta ASM.

Diagrama de Onda del Decodificador de Instrucción.

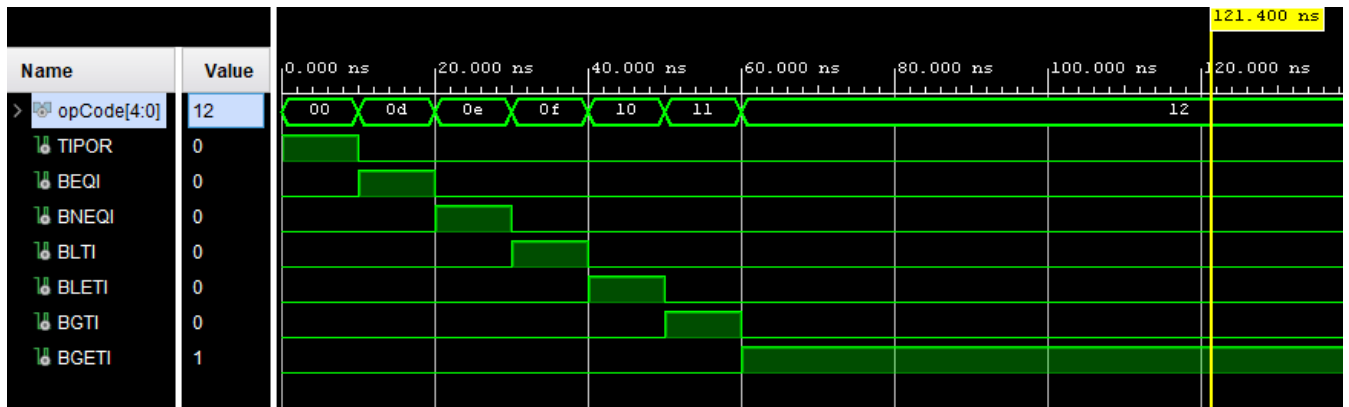


Figura 3.1 Diagrama de Onda del Decodificador de Instrucción.

Diagrama de Onda de MfunCode.

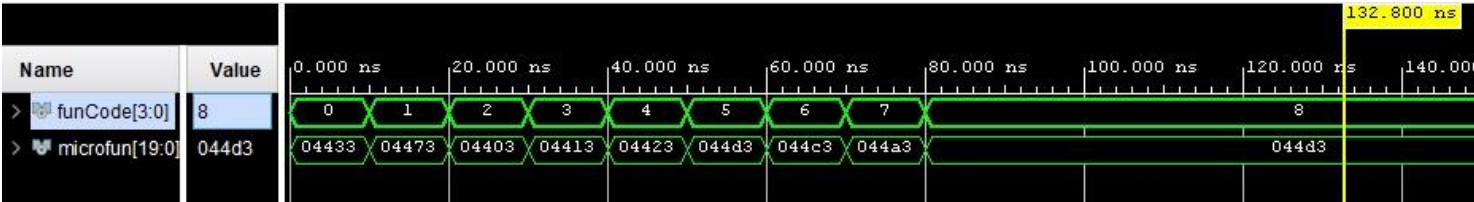


Figura 3.2 Diagrama de Onda de MfunCode.

Diagrama de Onda de MopCode.

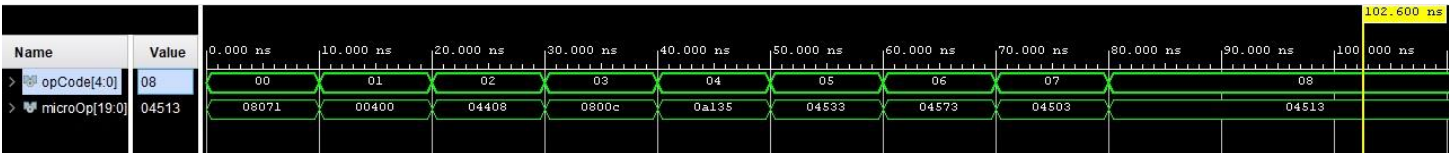


Figura 3.3 Diagrama de Onda de MopCode.

Diagrama de Onda de Bloque de Condición.

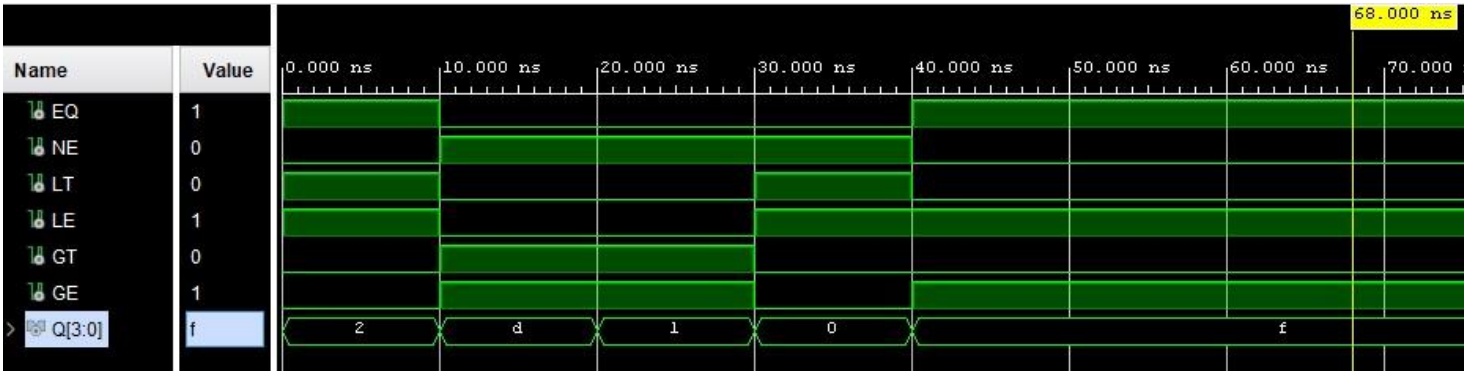


Figura 3.4 Diagrama de Onda del Bloque de Condición.

Diagrama de Onda de Bloque de Nivel.

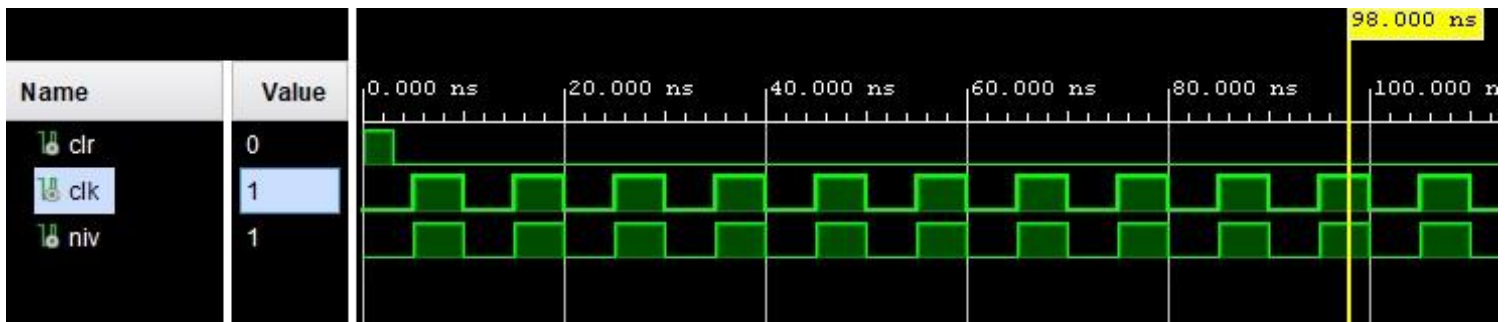


Figura 3.5 Diagrama de Onda del Bloque de Nivel.

Diagrama de Onda de Registro de Banderas.

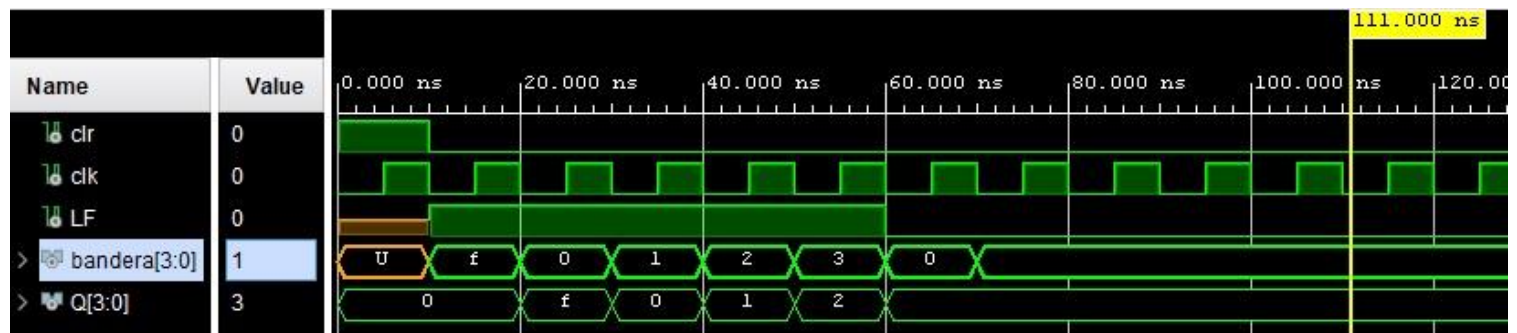


Figura 3.6 Diagrama de Onda del Registro de Banderas.

Diagrama de Onda de la Unidad de Control.

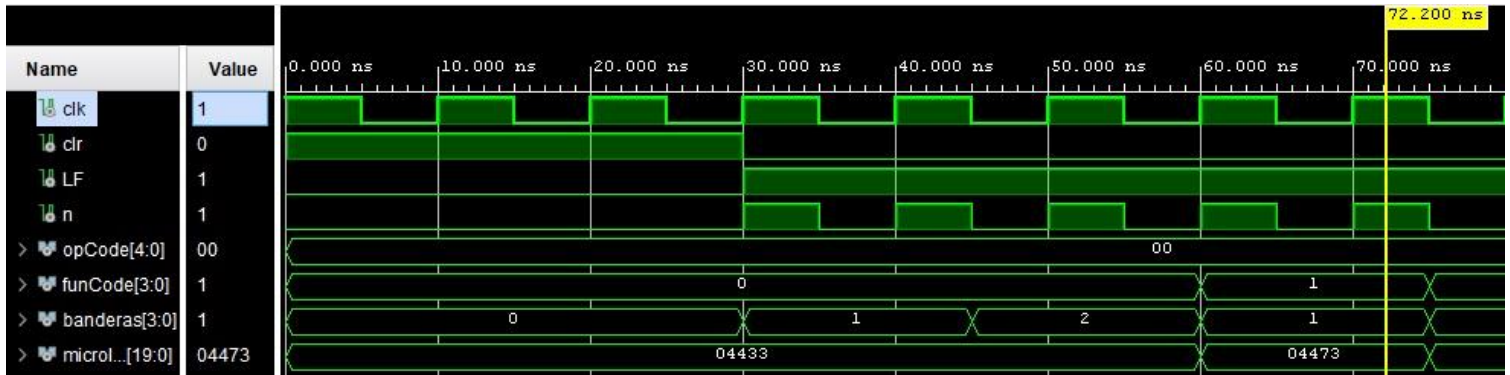


Figura 3.7 Diagrama de Onda de la Unidad Control.

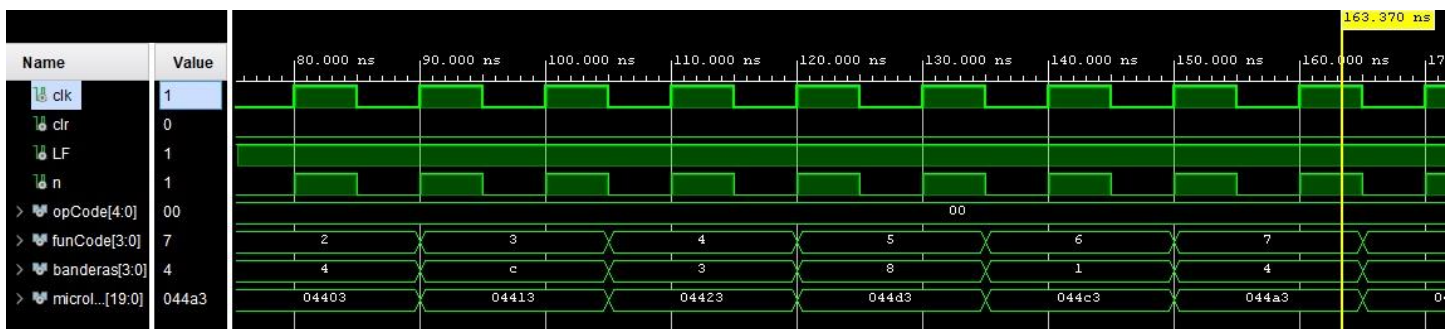


Figura 3.8 Diagrama de Onda de la Unidad Control.

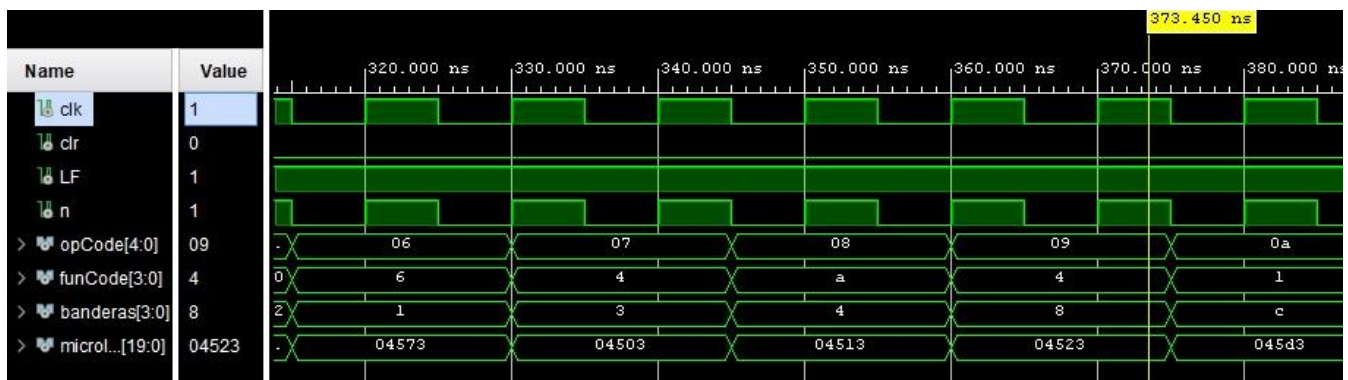



Figura 3.9 Diagrama de Onda de la Unidad Control.

Estímulos ingresados por Archivo.

 ESTIMULOS: Bloc de notas

Archivo	Edición	Formato
00000	0000	0000 1 0
00000	0000	0000 1 0
00000	0000	0001 0 1
00000	0000	0010 0 1
00000	0001	0001 0 1
00000	0010	0100 0 1
00000	0011	1100 0 1
00000	0100	0011 0 1
00000	0101	1000 0 1
00000	0110	0001 0 1
00000	0111	0100 0 1
00000	1000	0010 0 1
00000	1001	0000 0 0
00000	1010	0000 0 0
00000	1011	0000 0 0
00000	1100	0000 0 0
00001	0111	0000 0 0
00010	0100	0000 0 0
00011	1000	0000 0 0
00100	0110	0000 0 0
00101	0000	0010 0 1
00110	0110	0001 0 1
00111	0100	0011 0 1
01000	1010	0100 0 1
01001	0100	1000 0 1
01010	0001	1100 0 1
01011	0011	0101 0 1
01100	1111	1010 0 1
10111	0000	0000 0 1
01101	1111	0000 0 1
01101	1011	0010 0 1
01101	1101	0010 0 1
01110	1110	0010 0 1
01110	1100	0000 0 1
01110	0011	0000 0 1
01111	0001	1100 0 1

Figura 4. Estímulos en archivo .txt

```
01111 0000 1000 0 1
01111 0010 0100 0 1
10000 0100 0000 0 1
10000 0110 1110 0 1
10000 0101 1000 0 1
10001 0111 1010 0 1
10001 1010 1100 0 1
10001 1000 0000 0 1
10010 1111 1000 0 1
10010 1001 1010 0 1
10010 1101 1100 0 1
10011 1001 1100 0 0
10100 1111 0000 0 0
10101 0000 0000 0 0
10110 0000 0000 0 0
11000 0000 0000 0 0
```

Figura 4.1. Estímulos en archivo .txt

Resultados escritos en el Archivo.

*RESULTADOS: Bloc de notas

Archivo	Edición	Formato	Ver	Ayuda		
OPCODE	FUNCODE	BANDERAS	CLR	LF	MICROINS	NIVEL
00000	0000	0000	1	0	00000100010000110011	BAJO
00000	0000	0000	1	0	00000100010000110011	BAJO
00000	0000	0000	1	0	00000100010000110011	BAJO
00000	0000	0000	1	0	00000100010000110011	BAJO
00000	0000	0001	0	1	00000100010000110011	BAJO
00000	0000	0001	0	1	00000100010000110011	ALTO
00000	0000	0010	0	1	00000100010000110011	ALTO
00000	0000	0010	0	1	00000100010000110011	BAJO
00000	0001	0001	0	1	00000100010001110011	BAJO
00000	0001	0001	0	1	00000100010001110011	ALTO
00000	0010	0100	0	1	00000100010000000011	ALTO
00000	0010	0100	0	1	00000100010000000011	BAJO
00000	0011	1100	0	1	00000100010000010011	BAJO
00000	0011	1100	0	1	00000100010000010011	ALTO
00000	0100	0011	0	1	00000100010000100011	ALTO
00000	0100	0011	0	1	00000100010000100011	BAJO
00000	0101	1000	0	1	00000100010011010011	BAJO
00000	0101	1000	0	1	00000100010011010011	ALTO
00000	0110	0001	0	1	00000100010011000011	ALTO
00000	0110	0001	0	1	00000100010011000011	BAJO
00000	0111	0100	0	1	00000100010010100011	BAJO
00000	0111	0100	0	1	00000100010010100011	ALTO
00000	1000	0010	0	1	00000100010011010011	ALTO
00000	1000	0010	0	1	00000100010011010011	BAJO

Figura 4.2. Resultado de Test en Bench en archivo .txt

*RESULTADOS: Bloc de notas						
Archivo	Edición	Formato	Ver	Ayuda		
00000	1001	0000	0	0	00000001110000000000	BAJO
00000	1001	0000	0	0	00000001110000000000	ALTO
00000	1010	0000	0	0	00000001010000000000	ALTO
00000	1010	0000	0	0	00000001010000000000	BAJO
00000	1011	0000	0	0	00000000000000000000	BAJO
00000	1011	0000	0	0	00000000000000000000	ALTO
00000	1100	0000	0	0	00000000000000000000	ALTO
00000	1100	0000	0	0	00000000000000000000	BAJO
00001	0111	0000	0	0	00000000010000000000	BAJO
00001	0111	0000	0	0	00000000010000000000	ALTO
00010	0100	0000	0	0	00000100010000001000	ALTO
00010	0100	0000	0	0	00000100010000001000	BAJO
00011	1000	0000	0	0	00001000000000001100	BAJO
00011	1000	0000	0	0	00001000000000001100	ALTO
00100	0110	0000	0	0	00001010000100110101	ALTO
00100	0110	0000	0	0	00001010000100110101	BAJO
00101	0000	0010	0	1	00000100010100110011	BAJO
00101	0000	0010	0	1	00000100010100110011	ALTO
00110	0110	0001	0	1	00000100010101110011	ALTO
00110	0110	0001	0	1	00000100010101110011	BAJO
00111	0100	0011	0	1	00000100010100000011	BAJO
00111	0100	0011	0	1	00000100010100000011	ALTO
01000	1010	0100	0	1	00000100010100010011	ALTO
01000	1010	0100	0	1	00000100010100010011	BAJO

Figura 4.3. Resultado de Test en Bench en archivo .txt

*RESULTADOS: Bloc de notas						
Archivo	Edición	Formato	Ver	Ayuda		
10010	1001	1010	0	1	00001000000001110001	ALTO
10010	1001	1010	0	1	00001000000001110001	BAJO
10010	1101	1100	0	1	00001000000001110001	BAJO
10010	1101	1100	0	1	00001000000001110001	ALTO
10011	1001	1100	0	0	00010000000000000000	ALTO
10011	1001	1100	0	0	00010000000000000000	BAJO
10100	1111	0000	0	0	01010000000000000000	BAJO
10100	1111	0000	0	0	01010000000000000000	ALTO
10101	0000	0000	0	0	00100000000000000000	ALTO
10101	0000	0000	0	0	00100000000000000000	BAJO
10110	0000	0000	0	0	00000000000000000000	BAJO
10110	0000	0000	0	0	00000000000000000000	ALTO
11000	0000	0000	0	0	00000000000000000000	ALTO
11000	0000	0000	0	0	00000000000000000000	BAJO
UUUUU	UUUU	UUUU	U	U	00001000000001110001	BAJO
UUUUU	UUUU	UUUU	U	U	00001000000001110001	ALTO
UUUUU	UUUU	UUUU	U	U	00001000000001110001	ALTO
UUUUU	UUUU	UUUU	U	U	00001000000001110001	BAJO
UUUUU	UUUU	UUUU	U	U	00001000000001110001	BAJO
UUUUU	UUUU	UUUU	U	U	00001000000001110001	ALTO
UUUUU	UUUU	UUUU	U	U	00001000000001110001	BAJO
UUUUU	UUUU	UUUU	U	U	00001000000001110001	BAJO
UUUUU	UUUU	UUUU	U	U	00001000000001110001	ALTO

Figura 4.4. Resultado de Test en Bench en archivo .txt