Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica 14 - Unidad de Control

Unidad de aprendizaje: Arquitectura de Computadoras

Grupo: 3CV1

Alumno(a):
Ramos Diaz Enrique

Profesor(a):
Vega García Nayeli

28 de junio 2020

Índice

1. Código de implementación

1.1. Decodificador de Instrucción

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

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

architecture Behavioral of DecodificadorInstruccion 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';
```

```
41                  BNEQI <= '0';
42                  BLTI <= '1';
43                  BLETI <= '0';
44                  BGTI <= '0';
45                  BGETI <= '0';
46              when "10000" =>
47                  TIPOR <= '0';
48                  BEQI <= '0';
49                  BNEQI <= '0';
50                  BLTI <= '0';
51                  BLETI <= '1';
52                  BGTI <= '0';
53                  BGETI <= '0';
54              when "10001" =>
55                  TIPOR <= '0';
56                  BEQI <= '0';
57                  BNEQI <= '0';
58                  BLTI <= '0';
59                  BLETI <= '0';
60                  BGTI <= '1';
61                  BGETI <= '0';
62              when "10010" =>
63                  TIPOR <= '0';
64                  BEQI <= '0';
65                  BNEQI <= '0';
66                  BLTI <= '0';
67                  BLETI <= '0';
68                  BGTI <= '0';
69                  BGETI <= '1';
70            when others =>
71                  TIPOR <= '0';
72                  BEQI <= '0';
73                  BNEQI <= '0';
74                  BLTI <= '0';
75                  BLETI <= '0';
76                  BGTI <= '0';
77                  BGETI <= '0';
78          end case;
79      end process;
80  end Behavioral;
```

1.2. Memoria de Código de Función

```
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);
           microFuncion : out STD_LOGIC_VECTOR (19 downto 0));
end MfunCode;

architecture Behavioral of MfunCode is
    type arreglo is array (0 to (2**4)-1) of STD_LOGIC_VECTOR(19 downto 0);
    constant aux : arreglo := (
        "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
    microFuncion <= aux(conv_integer(funCode));
end Behavioral;
```

1.3. Memoria de Código de Operación

```
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);
           microOpCode : out  STD_LOGIC_VECTOR (19 downto 0));
end MopCode;
```

```
10
11  architecture Behavioral of MopCode is
12      type arreglo is array ( 0 to (2**5)-1 ) of std_logic_vector(19 downto
        ↪  0);
13      constant aux : arreglo := (
14          "00001000000001110001", -- VERIF
15          "00000000010000000000", -- LI
16          "00000100010000001000", -- LWI
17          "00001000000000001100", -- SWI
18          "00001010000100110101", -- SW
19          "00000100010100110011", -- ADDI
20          "00000100010101110011", -- SUBI
21          "00000100010100000011", -- ANDI
22          "00000100010100010011", -- ORI
23          "00000100010100100011", -- XORI
24          "00000100010111010011", -- NANDI
25          "00000100010111000011", -- NORI
26          "00000100010110100011", -- XNORI
27          "10010000001100110011", -- BEQI
28          "10010000001100110011", -- BNEI
29          "10010000001100110011", -- BLTI
30          "10010000001100110011", -- BLETI
31          "10010000001100110011", -- BGTI
32          "10010000001100110011", -- BGETI
33          "00010000000000000000", -- B
34          "01010000000000000000", -- CALL
35          "00100000000000000000", -- RET
36          "00000000000000000000", -- NOP
37          "00000110010100110001", -- LW
38
39      others => (others => '0')
40      );
41  begin
42      microOpCode <= aux(conv_integer(opCode));
43  end Behavioral;
```

1.4. 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;
```

1.5. 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;
```

```
8
9  architecture Behavioral of Condicion is
10 begin
11     --Q[OV, N, Z, C]
12     EQ <= Q(1);
13     NE <= not Q(1);
14     LT <= not Q(0);
15     LE <= not Q(0) or Q(1);
16     GT <= Q(0) and not Q(1);
17     GE <= Q(0);
18 end Behavioral;
```

1.6. Registro de banderas

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity Registro is
5      Port ( banderas : in STD_LOGIC_VECTOR (3 downto 0);
6             clr, clk, LF : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR (3 downto 0));
8  end Registro;
9
10 architecture Behavioral of Registro is
11 begin
12     process(clk, clr)
13         begin
14         if (clr = '1') then
15             Q <= (others => '0');
16         elsif (falling_edge (clk)) then
17             if (LF = '1') then
18                 Q <= banderas;
19             end if;
20         end if;
21     end process;
22 end Behavioral;
```

1.7. Multiplexor: MopCode

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity Mux2C is
```

```
    Port ( opCode : in STD_LOGIC_VECTOR (4 downto 0);
           SDOPC : in STD_LOGIC;
           salida : out STD_LOGIC_VECTOR (4 downto 0));
end Mux2C;

architecture Behavorial of Mux2C is
    constant ceros : STD_LOGIC_VECTOR (4 downto 0) := "00000";
begin
    with SDOPC select
        salida <=
            opCode when '1',
            ceros when others;
end Behavorial;
```

1.8. Multiplexor: MopCode - MfunCode

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

entity Multiplexor is
    Port (
        microFuncion, microOpcode : in STD_LOGIC_VECTOR (19 downto 0);
        SM : in STD_LOGIC;
        salida : out STD_LOGIC_VECTOR (19 downto 0));
end Multiplexor;

architecture Behavioral of Multiplexor is
begin
    with SM select
        salida <=
            microOpcode when '1' ,
            microFuncion when others;
end Behavioral;
```

1.9. Carta ASM

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity UnidadC is
5      Port ( clk, clr, EQ, NE, LT, LE, GT, GE, nivel, BGETI, BGTI, BLETI,
       ↪  BLTI, BNEQI, BEQI, TIPOR : in STD_LOGIC;
6             SDOPC, SM : out STD_LOGIC);
7  end UnidadC;
8
9  architecture Behavioral of UnidadC is
10    type estados is (e0);
11    signal actual, siguiente : estados;
12 begin
13    --Control de estados
14    process(clr, clk)
15    begin
16       if(clr = '1') then
17          actual <= e0;
18       elsif (rising_edge(clk)) then
19          actual <= siguiente;
20       end if;
21    end process;
22
23    --Carta ASM
24    process(actual, EQ, NE, LT, LE, GT, GE, nivel, BGETI, BGTI, BLETI, BLTI,
      ↪  BNEQI, BEQI, TIPOR)
25    begin
26       SDOPC <= '0';
27       SM <= '0';
28       case actual is
29             when e0 =>
30                 if (TIPOR = '1') then
31                     SM <= '0';
32                 else
33                     if (BEQI = '0') then
34                         if (BNEQI = '0') then
35                             if (BLTI = '0') then
36                                 if (BLETI = '0') then
37                                     if (BGTI = '0') then
38                                        if (BGETI = '0') then
39                                             SM <= '1';
40                                             SDOPC <= '1';
```

```
41                                  else --BGETI 1
42                                      if (nivel = '1') then
43                                          SM <= '1';
44                                          SDOPC <= '0';
45                                      else --NA 0
46                                          if (GE = '1') then
47                                              SM <= '1';
48                                              SDOPC <= '1';
49                                          else --GE 0
50                                              SM <= '1';
51                                              SDOPC <= '0';
52                                          end if;
53                                      end if;
54                                  end if;
55                              else -- BGTI 1
56                                  if (nivel = '1') then
57                                      SM <= '1';
58                                      SDOPC <= '0';
59                                  else --NA 0
60                                      if (GT = '1') then
61                                          SM <= '1';
62                                          SDOPC <= '1';
63                                      else --GT 0
64                                          SM <= '1';
65                                          SDOPC <= '0';
66                                      end if;
67                                  end if;
68                              end if;
69                          else --BLETI 1
70                              if (nivel = '1') then
71                                  SM <= '1';
72                                  SDOPC <= '0';
73                              else --NA 0
74                                  if (LE = '1') then
75                                      SM <= '1';
76                                      SDOPC <= '1';
77                                  else --LE 0
78                                      SM <= '1';
79                                      SDOPC <= '0';
80                                  end if;
81                              end if;
82                          end if;
83                      else --BLTI 1
84                          if (nivel = '1') then
```

```
85                                      SM <= '1';
86                                      SDOPC <= '0';
87                                  else --NA 0
88                                      if (LT = '1') then
89                                          SM <= '1';
90                                          SDOPC <= '1';
91                                      else --LT 0
92                                          SM <= '1';
93                                          SDOPC <= '0';
94                                      end if;
95                                  end if;
96                              end if;
97                          else --BNEQI 1
98                              if (nivel = '1') then
99                                  SM <= '1';
100                                 SDOPC <= '0';
101                             else --NA 0
102                                 if (NE = '1') then
103                                     SM <= '1';
104                                     SDOPC <= '1';
105                                 else --NE 0
106                                     SM <= '1';
107                                     SDOPC <= '0';
108                                 end if;
109                             end if;
110                         end if;
111                     else --BEQI 1
112                         if (nivel = '1') then
113                             SM <= '1';
114                             SDOPC <= '0';
115                         else --NA 0
116                             if (EQ = '1') then
117                                 SM <= '1';
118                                 SDOPC <= '1';
119                             else --EQ 0
120                                 SM <= '1';
121                                 SDOPC <= '0';
122                             end if;
123                         end if;
124                     end if;
125                 end if;
126             siguiente <= e0;
127     end case;
128 end process;
```

```
129 end Behavioral;
```

1.10. Unidad de Control

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5  entity UnidadControl is
6      Port ( clk, clr, LF : in STD_LOGIC;
7             opCode : in STD_LOGIC_VECTOR (4 downto 0);
8             funCode, banderas : in STD_LOGIC_VECTOR (3 downto 0);
9             na: out STD_LOGIC;
10            microInstruccion : out STD_LOGIC_VECTOR (19 downto 0));
11 end UnidadControl;
12
13 architecture Behavioral of UnidadControl is
14     component UnidadC is
15         Port ( clk, clr, EQ, NE, LT, LE, GT, GE, nivel, BGETI, BGTI, BLETI,
           ↪  BLTI, BNEQI, BEQI, TIPOR : in STD_LOGIC;
16                SDOPC, SM : out STD_LOGIC);
17     end component;
18
19     component Condicion is
20         Port ( Q : in STD_LOGIC_VECTOR (3 downto 0);
21                EQ, NE, LT, LE, GT, GE : out STD_LOGIC);
22     end component;
23
24     component DecodificadorInstruccion is
25         Port ( opCode : in STD_LOGIC_VECTOR (4 downto 0);
26                TIPOR, BEQI, BNEQI, BLTI, BLETI, BGTI, BGETI : out
                   ↪  STD_LOGIC);
27     end component;
28
29     component MfunCode is
30         Port ( funCode : in STD_LOGIC_VECTOR (3 downto 0);
31                microFuncion : out STD_LOGIC_VECTOR (19 downto 0));
32     end component;
33
34     component MopCode is
35         Port ( opCode : in  STD_LOGIC_VECTOR (4 downto 0);
36                microOpCode : out  STD_LOGIC_VECTOR (19 downto 0));
37     end component;
```

```
38
39      component Mux2C is
40          Port ( opCode : in STD_LOGIC_VECTOR (4 downto 0);
41                 SDOPC : in STD_LOGIC;
42                 salida : out STD_LOGIC_VECTOR (4 downto 0));
43      end component;
44
45      component Multiplexor is
46          Port (
47              microFuncion, microOpcode : in STD_LOGIC_VECTOR (19 downto 0);
48              SM : in STD_LOGIC;
49              salida : out STD_LOGIC_VECTOR (19 downto 0));
50      end component;
51
52      component Nivel is
53          Port ( clk, clr : in STD_LOGIC;
54                 nivel : out STD_LOGIC);
55      end component;
56
57      component Registro is
58          Port ( banderas : in STD_LOGIC_VECTOR (3 downto 0);
59                 clr, clk, LF : in STD_LOGIC;
60                 Q : out STD_LOGIC_VECTOR (3 downto 0));
61      end component;
62
63      --[EQ, NE, LT, LE, GT, GE]
64      signal auxCondicion : STD_LOGIC_VECTOR (5 downto 0);
65      --[TIPOR, BEQI, BNEQI, BLTI, BLETI, BGTI, BGETI]
66      signal auxDecodificador : STD_LOGIC_VECTOR (6 downto 0);
67      signal auxNivel, auxSM, auxSDOPC : STD_LOGIC;
68      signal auxQ : STD_LOGIC_VECTOR(3 downto 0);
69      signal auxOpCode : STD_LOGIC_VECTOR(4 downto 0);
70      signal auxMicroFun, auxMicroOp, auxSalida : STD_LOGIC_VECTOR(19 downto
        ↪  0);
71  begin
72       uc : UnidadC Port map (
73          clk => clk,
74          clr => clr,
75          EQ => auxCondicion(5),
76          NE => auxCondicion(4),
77          LT => auxCondicion(3),
78          LE => auxCondicion(2),
79          GT => auxCondicion(1),
80          GE => auxCondicion(0),
```

```
81          nivel => auxNivel,
82          BGETI => auxDecodificador(0),
83          BGTI => auxDecodificador(1),
84          BLETI => auxDecodificador(2),
85          BLTI => auxDecodificador(3),
86          BNEQI => auxDecodificador(4),
87          BEQI => auxDecodificador(5),
88          TIPOR => auxDecodificador(6),
89          SDOPC => auxSDOPC,
90          SM => auxSM
91      );
92
93
94      con : Condicion Port map (
95          Q => auxQ,
96          EQ => auxCondicion(5),
97          NE => auxCondicion(4),
98          LT => auxCondicion(3),
99          LE => auxCondicion(2),
100         GT => auxCondicion(1),
101         GE => auxCondicion(0)
102     );
103
104     deco : DecodificadorInstruccion Port map (
105         opCode => opCode,
106         TIPOR => auxDecodificador(6),
107         BEQI => auxDecodificador(5),
108         BNEQI => auxDecodificador(4),
109         BLTI => auxDecodificador(3),
110         BLETI => auxDecodificador(2),
111         BGTI => auxDecodificador(1),
112         BGETI => auxDecodificador(0)
113     );
114
115     fun : MfunCode Port map (
116         funCode => funCode,
117         microFuncion => auxMicroFun
118     );
119
120     op : MopCode Port map (
121         opCode => auxOpCode,
122         microOpCode => auxMicroOp
123     );
124
```

```
125     mux2 : Mux2C Port map (
126         opCode => opCode,
127         SDOPC => auxSDOPC,
128         salida => auxOpCode
129     );
130 
131     mux : Multiplexor Port map (
132         microFuncion => auxMicroFun,
133         microOpCode => auxMicroOp,
134         SM => auxSM,
135         salida => auxSalida
136     );
137 
138     niv : Nivel Port map (
139         clk => clk,
140         clr => clr,
141         nivel => auxNivel
142     );
143 
144     microInstruccion <= auxSalida;
145     na <= auxNivel;
146 
147     reg : Registro port map(
148         banderas => banderas,
149         clr => clr,
150         clk => clk,
151         LF => LF,
152         Q => auxQ
153     );
154 
155 end Behavioral;
```

2. Código de simulación

2.1. Decodificador de Instrucción

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

entity tbDecodificadorInstruccion is
end tbDecodificadorInstruccion;

architecture Behavioral of tbDecodificadorInstruccion is
    component DecodificadorInstruccion 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
    deco : DecodificadorInstruccion 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;
```

2.2. Memoria de Código de Función

```
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);
               microFuncion : out STD_LOGIC_VECTOR (19 downto 0));
    end component;

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

begin
    fun : MfunCode Port map (
        funCode => funCode,
        microFuncion => microFuncion
    );

    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;
    end process;
end Behavioral;
```

2.3. Memoria de Código de Operación

```
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);
               microOpCode : out  STD_LOGIC_VECTOR (19 downto 0));
    end component;

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

begin
    op : MopCode Port map (
        opCode => OpCode,
        microOpCode => microOpCode
    );

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

2.4. 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, na : STD_LOGIC;

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

    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;
```

2.5. 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
    con : Condicion Port map (
        Q => Q,
        EQ => EQ,
        NE => NE,
        LT => LT,
        LE => LE,
        GT => GT,
        GE => GE
    );

    --Q[OV, N, Z, C]
    process
    begin
        Q <= "0010";
```

```
        wait for 10 ns;
        Q <= "1101";
        wait for 10 ns;
        Q <= "0001";
        wait for 10 ns;
        Q <= "0000";
        wait for 10 ns;
        Q <= "1111";
        wait;
    end process;

end Behavioral;
```

2.6. Registro de banderas

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

entity tbRegistro is
end tbRegistro;

architecture Behavioral of tbRegistro is
    component Registro is
        Port ( banderas : in STD_LOGIC_VECTOR (3 downto 0);
               clr, clk, LF : in STD_LOGIC;
               Q : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

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

begin
    reg : Registro port map(
        banderas => banderas,
        clr => clr,
        clk => clk,
        LF => LF,
        Q => Q
    );

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

```
29          clk <= '1';
30          wait for 5 ns;
31      end process;
32
33      process
34      begin
35          clr <= '1';
36          wait until falling_edge(clk);
37          clr <= '0';
38          LF <= '1';
39          banderas <= "1111";
40          wait until falling_edge(clk);
41          banderas <= "0000";
42          wait until falling_edge(clk);
43          banderas <= "0001";
44          wait until falling_edge(clk);
45          banderas <= "0010";
46          wait until falling_edge(clk);
47          banderas <= "0011";
48          wait until falling_edge(clk);
49          LF <= '0';
50          banderas <= "0000";
51          wait until falling_edge(clk);
52          banderas <= "0001";
53          wait;
54      end process;
55  end Behavioral;
```

2.7. Carta ASM

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity tbUnidadC is
5   end tbUnidadC;
6
7   architecture Behavioral of tbUnidadC is
8       component UnidadC is
9           Port ( clk, clr, EQ, NE, LT, LE, GT, GE, nivel, BGETI, BGTI, BLETI,
            ↪  BLTI, BNEQI, BEQI, TIPOR : in STD_LOGIC;
10                 SDOPC, SM : out STD_LOGIC);
11      end component;
12
```

```
13      signal clk, clr, EQ, NE, LT, LE, GT, GE, nivel, BGETI, BGTI, BLETI,
        ↪  BLTI, BNEQI, BEQI, TIPOR, SDOPC, SM : STD_LOGIC := '0';
14
15  begin
16       uc : UnidadC Port map (
17          clk => clk,
18          clr => clr,
19          EQ => EQ,
20          NE => NE,
21          LT => LT,
22          LE => LE,
23          GT => GT,
24          GE => GE,
25          nivel => nivel,
26          BGETI => BGETI,
27          BGTI => BGTI,
28          BLETI => BLETI,
29          BLTI => BLTI,
30          BNEQI => BNEQI,
31          BEQI => BEQI,
32          TIPOR => TIPOR,
33          SDOPC => SDOPC,
34          SM => SM
35      );
36
37      reloj : process begin
38          clk <= '0';
39          wait for 5 ns;
40          clk <= '1';
41          wait for 5 ns;
42      end process;
43
44      process
45      begin
46          clr <= '1';
47          wait until rising_edge(clk);
48
49          clr <= '0';
50          TIPOR <= '1';
51          wait until rising_edge(clk);
52
53          TIPOR <= '0';
54          LT  <= '1';
55          BLTI <= '1';
```

```
56          wait until rising_edge(clk);
57
58          LT <= '0';
59          wait until rising_edge(clk);
60
61          BLTI <= '0';
62          LE <= '1';
63          BLETI <= '1';
64          wait until rising_edge(clk);
65
66          LE <= '0';
67          wait until rising_edge(clk);
68
69          clr <= '1';
70          wait;
71      end process;
72
73  end Behavioral;
```

2.8. Unidad de Control

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_arith.all;
4   use IEEE.STD_LOGIC_unsigned.ALL;
5   use IEEE.STD_LOGIC_TEXTIO.ALL;
6   use STD.TEXTIO.ALL;
7
8   entity test_bench is
9   end test_bench;
10
11  architecture Behavioral of test_bench is
12      component UnidadControl is
13          Port ( clk, clr, LF : in STD_LOGIC;
14                 opCode : in STD_LOGIC_VECTOR (4 downto 0);
15                 funCode, banderas : in STD_LOGIC_VECTOR (3 downto 0);
16                 na : out STD_LOGIC;
17                 microInstruccion : out STD_LOGIC_VECTOR (19 downto 0));
18      end component;
19
20      signal na, clk, clr, LF : STD_LOGIC := '0';
21      signal opCode : STD_LOGIC_VECTOR (4 downto 0) := "00000";
22      signal funCode, banderas : STD_LOGIC_VECTOR (3 downto 0) := "0000";
```

```
23      signal microInstruccion : STD_LOGIC_VECTOR (19 downto 0) := (others =>
        ↪  '0');
24
25  begin
26      U1 : UnidadControl port map(
27          opCode => opCode,
28          funCode => funCode,
29          banderas => banderas,
30          clk => clk,
31          clr => clr,
32          LF => LF,
33          na => na,
34          microInstruccion => microInstruccion
35      );
36
37      reloj : process
38      begin
39          clk <= '1';
40          wait for 5 ns;
41          clk <= '0';
42          wait for 5 ns;
43      end process;
44
45       process
46          file arch_res : text;   --Apuntadores tipo
            ↪  txt
47          variable linea_res : line;
48          variable var_microinstruccion : STD_LOGIC_VECTOR (19 downto 0);
49          variable var_nivel : string (1 to 4);
50
51          file arch_en : text; --Apuntadores tipo txt
52          variable linea_en: line;
53          variable var_op_code : STD_LOGIC_VECTOR (4 downto 0);
54          variable var_fun_code, var_banderas : STD_LOGIC_VECTOR (3 downto 0);
55          variable var_clr, var_lf : STD_LOGIC;
56          variable cadena : string (1 to 8);
57          variable cadena2 : string (1 to 19);
58          variable cadenaSaltoLinea : string (1 to 2);
59
60      begin
61          cadenaSaltoLinea := "  ";
62          --- OPCODE FUNCODE BANDERAS CLR LF
63          file_open(arch_en, "Estimulos.txt", READ_MODE);
64
```

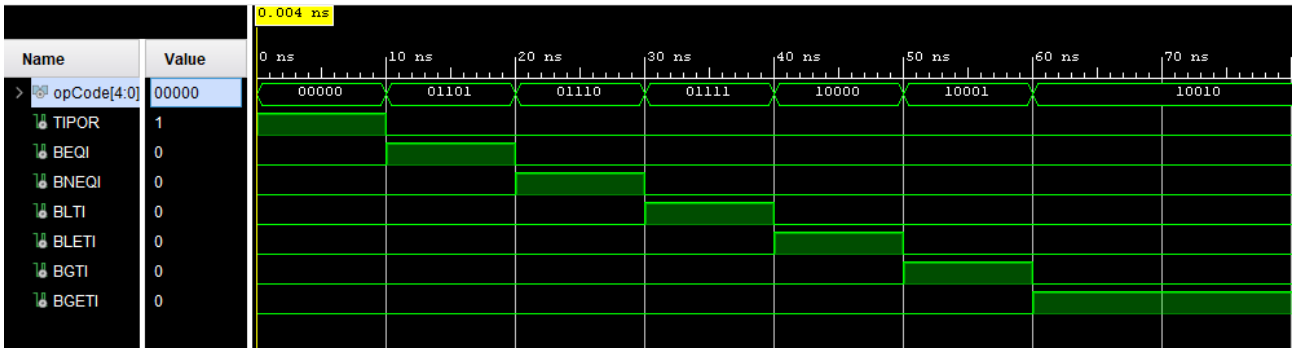
```
65          --- OPCODE FUNCODE BANDERAS CLR LF MICROINS NIVEL
66          file_open(arch_res, "Resultado.txt", WRITE_MODE);
67
68          cadena := "OPCODE  ";
69          write(linea_res, cadena, right, cadena'LENGTH+1);--ESCRIBE LA cadena
             ↪  "OPCODE"
70          cadena := "FUNCODE ";
71          write(linea_res, cadena, right, cadena'LENGTH+1);--ESCRIBE LA cadena
             ↪  "FUNCODE"
72          cadena := "BANDERAS";
73          write(linea_res, cadena, right, cadena'LENGTH+1);--ESCRIBE LA cadena
             ↪  "BANDERAS"
74          cadena := "CLR     ";
75          write(linea_res, cadena, right, cadena'LENGTH+1);--ESCRIBE LA cadena
             ↪  "CLR"
76          cadena := "LF      ";
77          write(linea_res, cadena, right, cadena'LENGTH+1);--ESCRIBE LA cadena
             ↪  "LF"
78          cadena := "MICROINS";
79          write(linea_res, cadena, right, cadena'LENGTH+1);--ESCRIBE LA cadena
             ↪  "MICROINS"
80          cadena2 := "              NIVEL";
81          write(linea_res, cadena2, right, cadena'LENGTH+1);--ESCRIBE LA
             ↪  cadena "NIVEL"
82
83          writeline(arch_res, linea_res);-- escribe la linea en el archivo
84
85          while not endfile(arch_en) loop
86              readline(arch_en, linea_en); -- lee una linea completa
87              --- OPCODE FUNCODE BANDERAS CLR LF
88
89              --Lee OPCODE
90              read(linea_en, var_op_code);
91              opCode <= var_op_code;
92
93              --Lee FUNCODE
94              read(linea_en, var_fun_code);
95              funCode <= var_fun_code;
96
97              --Lee BANDERAS
98              read(linea_en, var_banderas);
99              banderas <= var_banderas;
100
101             --Lee CLR
```

```
102             read(linea_en, var_clr);
103             clr <= var_clr;
104 
105             --Lee LF
106             read(linea_en, var_lf);
107             LF <= var_lf;
108 
109             wait for 5 ns; --ESPERA AL FLANCO DE SUBIDA
110             var_microinstruccion := microInstruccion;
111 
112             if(na = '1') then
113                 var_nivel := "ALTO";
114             else
115                 var_nivel := "BAJO";
116             end if;
117 
118             --- OPCODE FUNCODE BANDERAS CLR LF MICROINS NIVEL
119             write(linea_res, var_op_code, left, 10);--ESCRIBE EL CAMPO
                ↪  OPCODE
120             write(linea_res, var_fun_code, left, 9);--ESCRIBE EL CAMPO
                ↪  FUNCODE
121             write(linea_res, var_banderas, left, 10);--ESCRIBE EL CAMPO
                ↪  BANDERAS
122             write(linea_res, var_clr, left, 8);--ESCRIBE EL CAMPO CLR
123             write(linea_res, var_lf, left, 9);--ESCRIBE EL CAMPO LF
124             write(linea_res, var_microinstruccion, left, 22);--ESCRIBE EL
                ↪  CAMPO MICROINS
125             write(linea_res, var_nivel, left, 5);--ESCRIBE EL CAMPO NIVEL
126 
127             writeline(arch_res, linea_res);-- escribe la linea en el archivo
128 
129             wait for 5 ns; --ESPERA AL FLANCO DE BAJADA
130             var_microinstruccion := microInstruccion;
131 
132             if(na = '1') then
133                 var_nivel := "ALTO";
134             else
135                 var_nivel := "BAJO";
136             end if;
137 
138             --- OPCODE FUNCODE BANDERAS CLR LF MICROINS NIVEL
139             write(linea_res, var_op_code, left, 10);--ESCRIBE EL CAMPO
                ↪  OPCODE
```

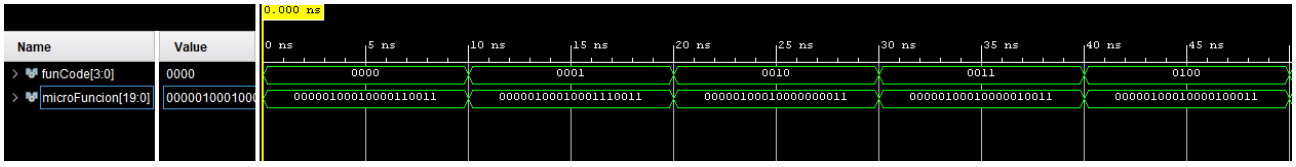
```
140             write(linea_res, var_fun_code, left, 9);--ESCRIBE EL CAMPO
                ↪  FUNCODE
141             write(linea_res, var_banderas, left, 10);--ESCRIBE EL CAMPO
                ↪  BANDERAS
142             write(linea_res, var_clr, left, 8);--ESCRIBE EL CAMPO CLR
143             write(linea_res, var_lf, left, 9);--ESCRIBE EL CAMPO LF
144             write(linea_res, var_microinstruccion, left, 22);--ESCRIBE EL
                ↪  CAMPO MICROINS
145             write(linea_res, var_nivel, left, 5);--ESCRIBE EL CAMPO NIVEL
                ↪
146             writeline(arch_res, linea_res);-- escribe la linea en el archivo
147
148             write(linea_res, cadenaSaltoLinea, right,
                ↪  cadena'LENGTH+1);--ESCRIBE cadena vacia
149             writeline(arch_res, linea_res);-- escribe la linea en el archivo
150         end loop;
151         file_close(arch_en);  -- cierra el archivo
152         file_close(arch_res);  -- cierra el archivo
153         wait;
154     end process;
155 end Behavioral;
```

3. Simulación

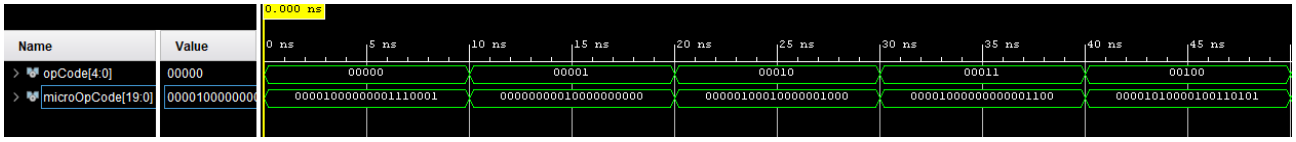
3.1. Decodificador de Instrucción



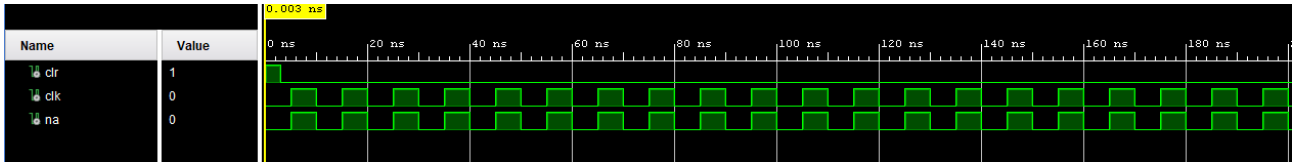
3.2. Memoria de Código de Función



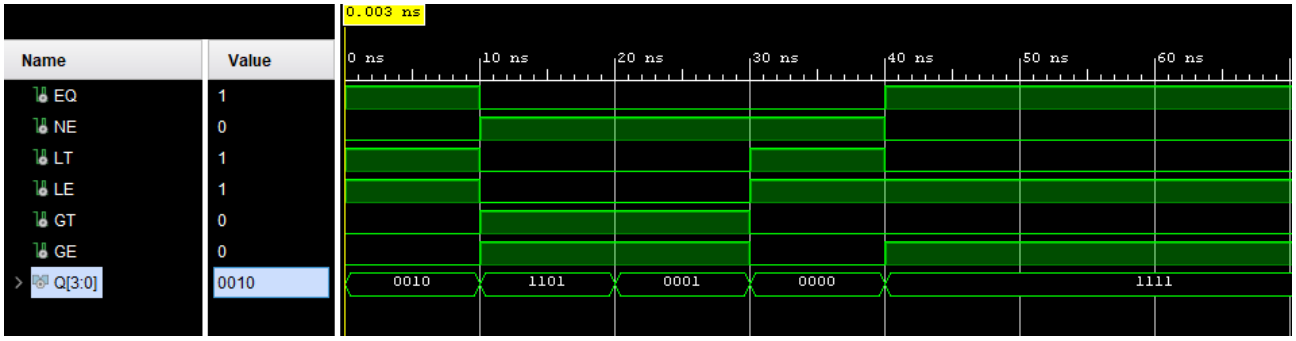
3.3. Memoria de Código de Operación



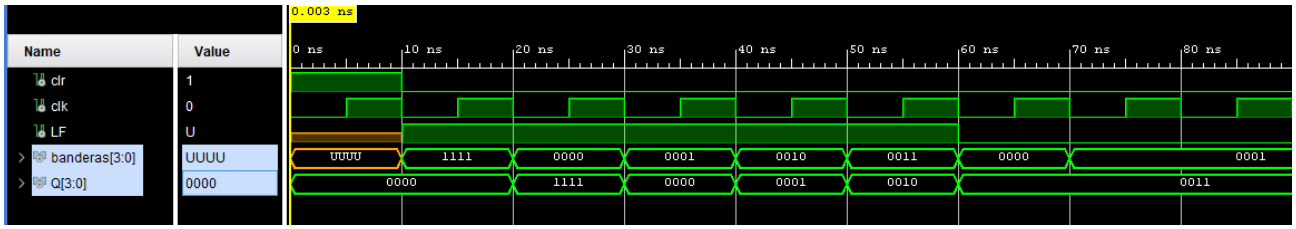
3.4. Nivel



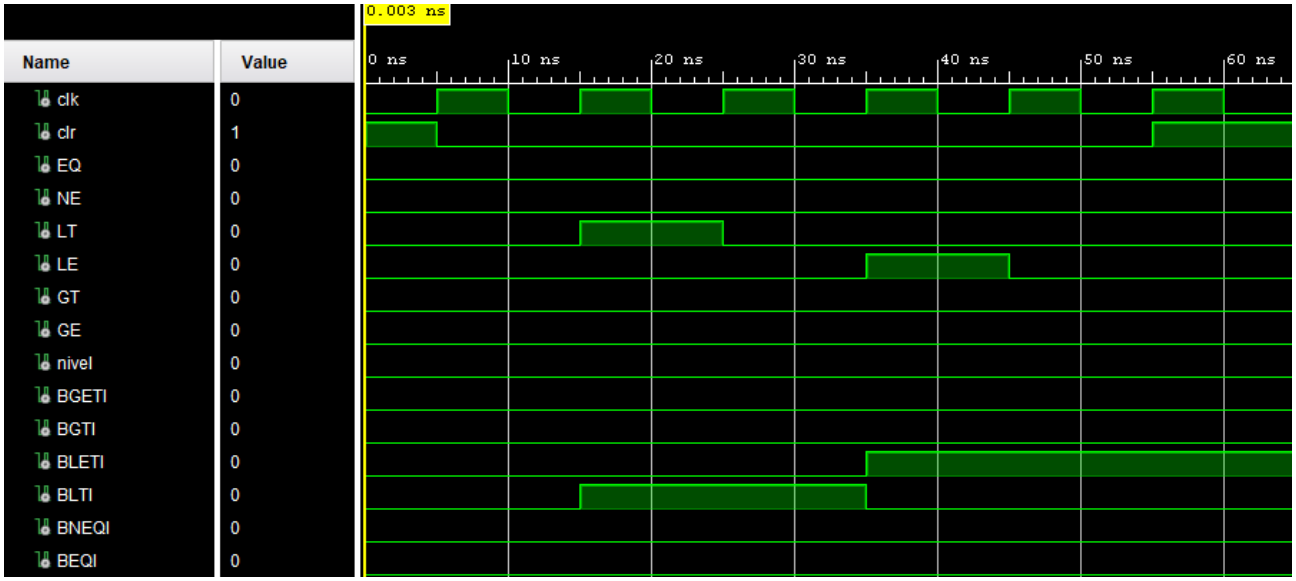
3.5. Condición

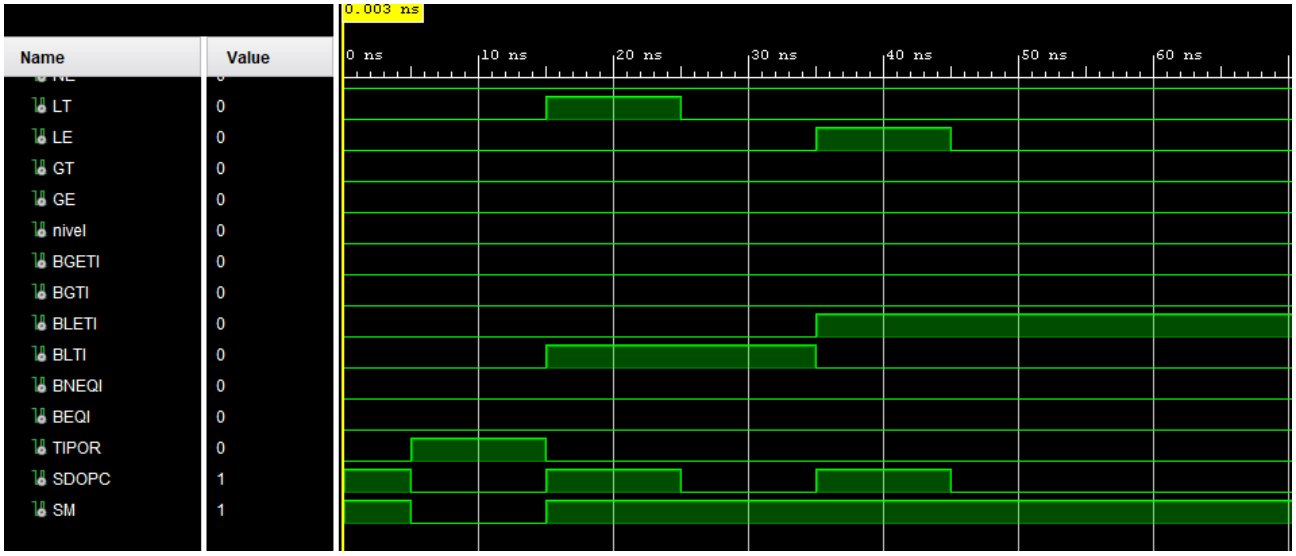


3.6. Registro de banderas



3.7. Carta ASM


0.003 ns
Name
Value
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
clk 0
clr 1
EQ 0
NE 0
LT 0
LE 0
GT 0
GE 0
nivel 0
BGETI 0
BGTI 0
BLETI 0
BLTI 0
BNEQI 0
BEQI 0

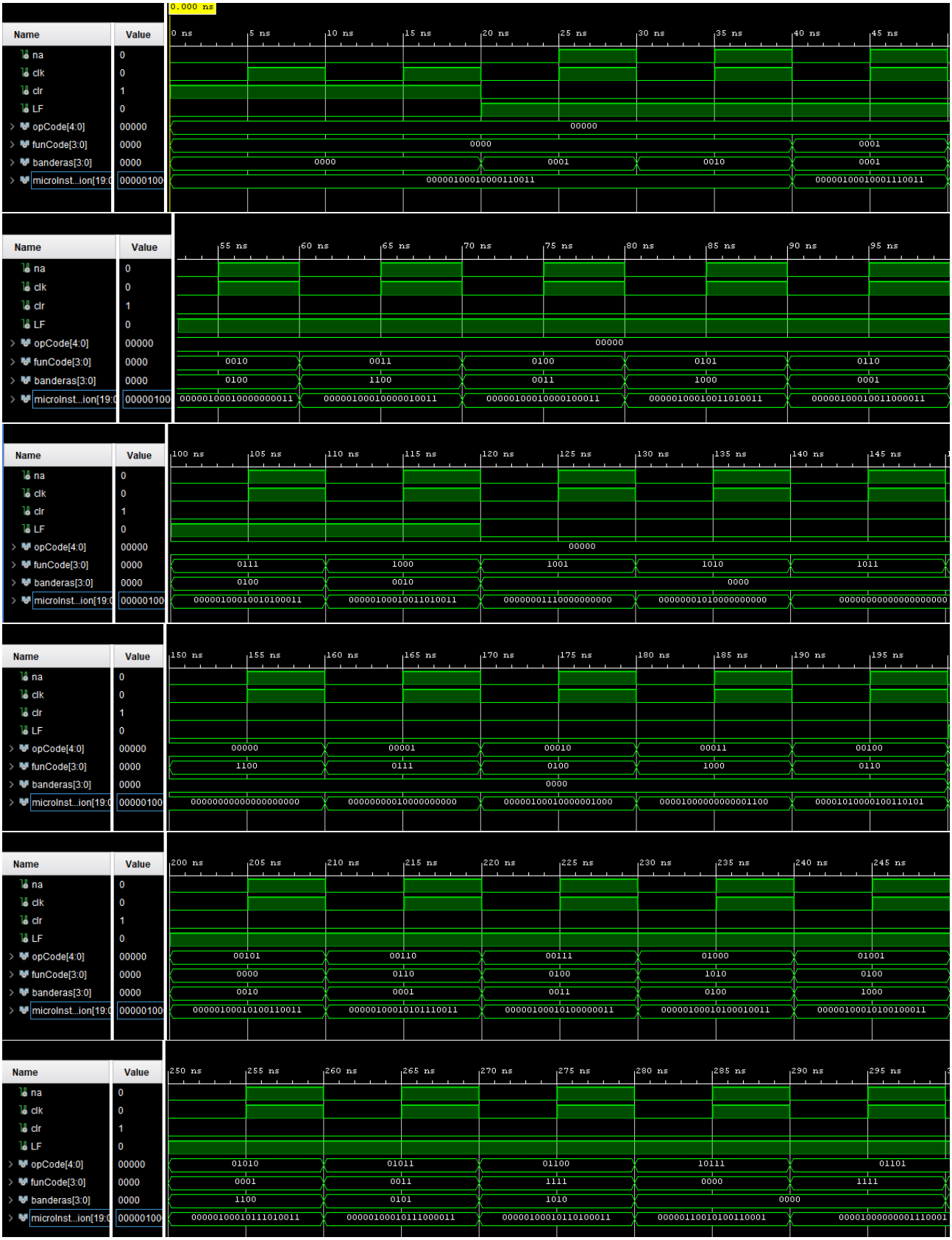

0.003 ns
Name
Value
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
LT 0
LE 0
GT 0
GE 0
nivel 0
BGETI 0
BGTI 0
BLETI 0
BLTI 0
BNEQI 0
BEQI 0
TIPOR 0
SDOPC 1
SM 1

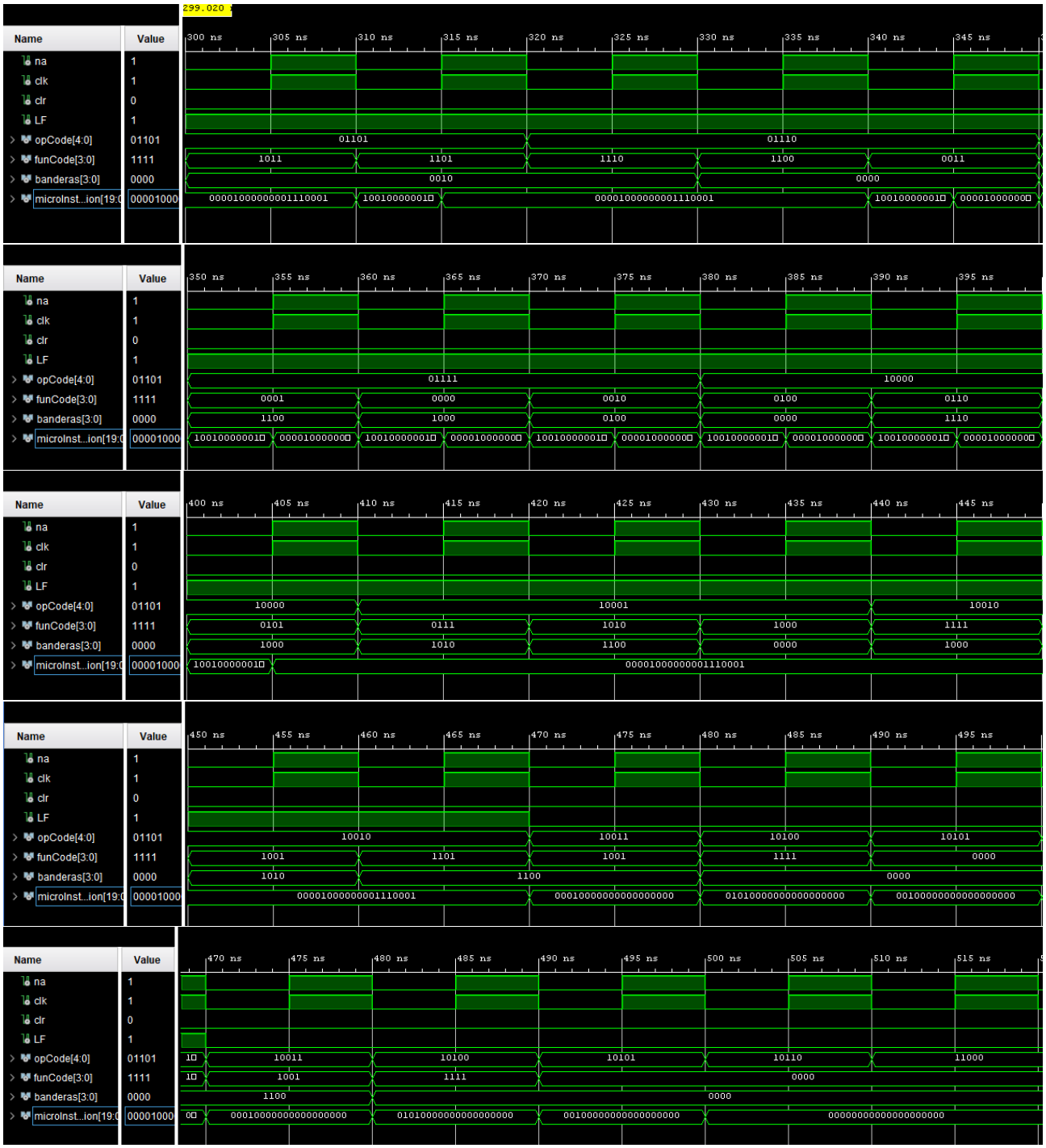
3.8. Unidad de Control

3.8.1. Archivo de Entrada: Estimulos.txt

```
--- OPCODE FUNCODE BANDERAS CLR LF
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
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
```

3.8.2. Forma de onda




Name
Value
na
clk
clr
LF
opCode[4:0]
funCode[3:0]
banderas[3:0]
microInst...ion[19:0]
300 ns
305 ns
310 ns
315 ns
320 ns
325 ns
330 ns
335 ns
340 ns
345 ns
01101
01110
1011
1101
1110
1100
0011
0010
0000
00001000
350 ns
355 ns
360 ns
365 ns
370 ns
375 ns
380 ns
385 ns
390 ns
395 ns
01111
10000
0001
0000
0010
0100
0110
1100
1000
0100
0000
1110
400 ns
405 ns
410 ns
415 ns
420 ns
425 ns
430 ns
435 ns
440 ns
445 ns
10000
10001
10010
0101
0111
1010
1000
1111
1000
1010
1100
0000
1000
450 ns
455 ns
460 ns
465 ns
470 ns
475 ns
480 ns
485 ns
490 ns
495 ns
10010
10011
10100
10101
1001
1101
1001
1111
0000
1010
1100
0000
470 ns
475 ns
480 ns
485 ns
490 ns
495 ns
500 ns
505 ns
510 ns
515 ns
10011
10100
10101
10110
11000
1001
1111
0000
1100
0000

3.8.3. Archivo de Salida: Resultado.txt

```
 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  ALTO
00000     0000     0001      0       1        00000100010000110011  BAJO

00000     0000     0010      0       1        00000100010000110011  ALTO
00000     0000     0010      0       1        00000100010000110011  BAJO

00000     0001     0001      0       1        00000100010001110011  ALTO
00000     0001     0001      0       1        00000100010001110011  BAJO

00000     0010     0100      0       1        00000100010000000011  ALTO
00000     0010     0100      0       1        00000100010000000011  BAJO

00000     0011     1100      0       1        00000100010000010011  ALTO
00000     0011     1100      0       1        00000100010000010011  BAJO

00000     0100     0011      0       1        00000100010000100011  ALTO
00000     0100     0011      0       1        00000100010000100011  BAJO

00000     0101     1000      0       1        00000100010011010011  ALTO
00000     0101     1000      0       1        00000100010011010011  BAJO

00000     0110     0001      0       1        00000100010011000011  ALTO
00000     0110     0001      0       1        00000100010011000011  BAJO

00000     0111     0100      0       1        00000100010010100011  ALTO
00000     0111     0100      0       1        00000100010010100011  BAJO

00000     1000     0010      0       1        00000100010011010011  ALTO
00000     1000     0010      0       1        00000100010011010011  BAJO

00000     1001     0000      0       0        00000001110000000000  ALTO
00000     1001     0000      0       0        00000001110000000000  BAJO

00000     1010     0000      0       0        00000001010000000000  ALTO
00000     1010     0000      0       0        00000001010000000000  BAJO
```

```
00000   1011   0000   0   0   00000000000000000000   ALTO
00000   1011   0000   0   0   00000000000000000000   BAJO

00000   1100   0000   0   0   00000000000000000000   ALTO
00000   1100   0000   0   0   00000000000000000000   BAJO

00001   0111   0000   0   0   00000000010000000000   ALTO
00001   0111   0000   0   0   00000000010000000000   BAJO

00010   0100   0000   0   0   00000100010000001000   ALTO
00010   0100   0000   0   0   00000100010000001000   BAJO

00011   1000   0000   0   0   00001000000000001100   ALTO
00011   1000   0000   0   0   00001000000000001100   BAJO

00100   0110   0000   0   0   00001010000100110101   ALTO
00100   0110   0000   0   0   00001010000100110101   BAJO

00101   0000   0010   0   1   00000100010100110011   ALTO
00101   0000   0010   0   1   00000100010100110011   BAJO

00110   0110   0001   0   1   00000100010101110011   ALTO
00110   0110   0001   0   1   00000100010101110011   BAJO

00111   0100   0011   0   1   00000100010100000011   ALTO
00111   0100   0011   0   1   00000100010100000011   BAJO

01000   1010   0100   0   1   00000100010100010011   ALTO
01000   1010   0100   0   1   00000100010100010011   BAJO

01001   0100   1000   0   1   00000100010100100011   ALTO
01001   0100   1000   0   1   00000100010100100011   BAJO

01010   0001   1100   0   1   00000100010111010011   ALTO
01010   0001   1100   0   1   00000100010111010011   BAJO

01011   0011   0101   0   1   00000100010111000011   ALTO
01011   0011   0101   0   1   00000100010111000011   BAJO

01100   1111   1010   0   1   00000100010110100011   ALTO
01100   1111   1010   0   1   00000100010110100011   BAJO

10111   0000   0000   0   1   00000110010100110001   ALTO
```

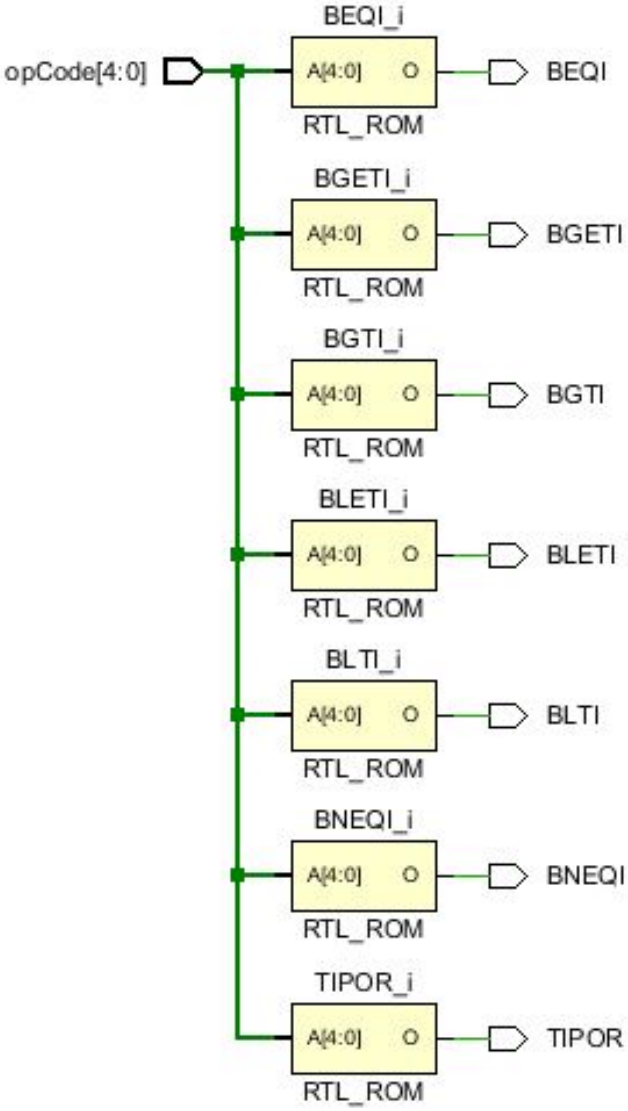
```
10111     0000     0000     0     1     00000110010100110001  BAJO

01101     1111     0000     0     1     00001000000001110001  ALTO
01101     1111     0000     0     1     00001000000001110001  BAJO

01101     1011     0010     0     1     00001000000001110001  ALTO
01101     1011     0010     0     1     10010000001100110011  BAJO

01101     1101     0010     0     1     00001000000001110001  ALTO
01101     1101     0010     0     1     10010000001100110011  BAJO

01110     1110     0010     0     1     00001000000001110001  ALTO
01110     1110     0010     0     1     00001000000001110001  BAJO

01110     1100     0000     0     1     00001000000001110001  ALTO
01110     1100     0000     0     1     10010000001100110011  BAJO

01110     0011     0000     0     1     00001000000001110001  ALTO
01110     0011     0000     0     1     10010000001100110011  BAJO

01111     0001     1100     0     1     00001000000001110001  ALTO
01111     0001     1100     0     1     10010000001100110011  BAJO

01111     0000     1000     0     1     00001000000001110001  ALTO
01111     0000     1000     0     1     10010000001100110011  BAJO

01111     0010     0100     0     1     00001000000001110001  ALTO
01111     0010     0100     0     1     10010000001100110011  BAJO

10000     0100     0000     0     1     00001000000001110001  ALTO
10000     0100     0000     0     1     10010000001100110011  BAJO

10000     0110     1110     0     1     00001000000001110001  ALTO
10000     0110     1110     0     1     10010000001100110011  BAJO

10000     0101     1000     0     1     00001000000001110001  ALTO
10000     0101     1000     0     1     10010000001100110011  BAJO

10001     0111     1010     0     1     00001000000001110001  ALTO
10001     0111     1010     0     1     00001000000001110001  BAJO

10001     1010     1100     0     1     00001000000001110001  ALTO
10001     1010     1100     0     1     00001000000001110001  BAJO
```

```
10001     1000     0000     0     1     00001000000001110001  ALTO
10001     1000     0000     0     1     00001000000001110001  BAJO

10010     1111     1000     0     1     00001000000001110001  ALTO
10010     1111     1000     0     1     00001000000001110001  BAJO

10010     1001     1010     0     1     00001000000001110001  ALTO
10010     1001     1010     0     1     00001000000001110001  BAJO

10010     1101     1100     0     1     00001000000001110001  ALTO
10010     1101     1100     0     1     00001000000001110001  BAJO

10011     1001     1100     0     0     00010000000000000000  ALTO
10011     1001     1100     0     0     00010000000000000000  BAJO

10100     1111     0000     0     0     01010000000000000000  ALTO
10100     1111     0000     0     0     01010000000000000000  BAJO

10101     0000     0000     0     0     00100000000000000000  ALTO
10101     0000     0000     0     0     00100000000000000000  BAJO

10110     0000     0000     0     0     00000000000000000000  ALTO
10110     0000     0000     0     0     00000000000000000000  BAJO

11000     0000     0000     0     0     00000000000000000000  ALTO
11000     0000     0000     0     0     00000000000000000000  BAJO
```

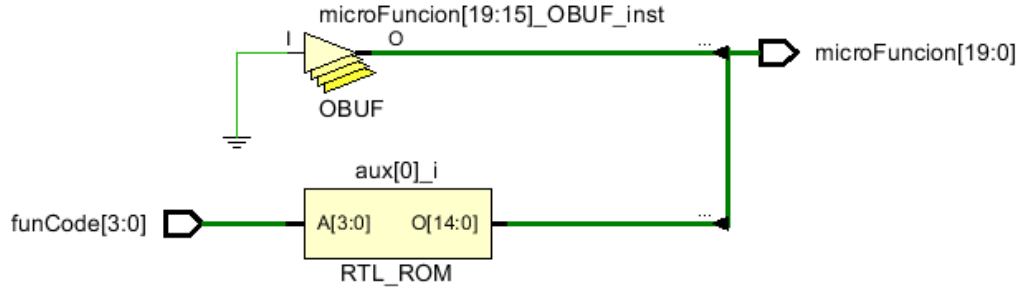
4. Diagramas RTL

4.1. Análisis RTL

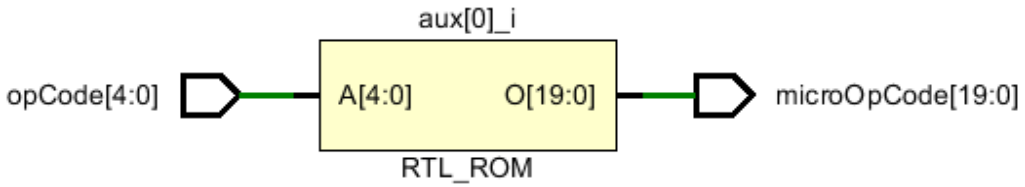
4.1.1. Decodificador de Instrucción



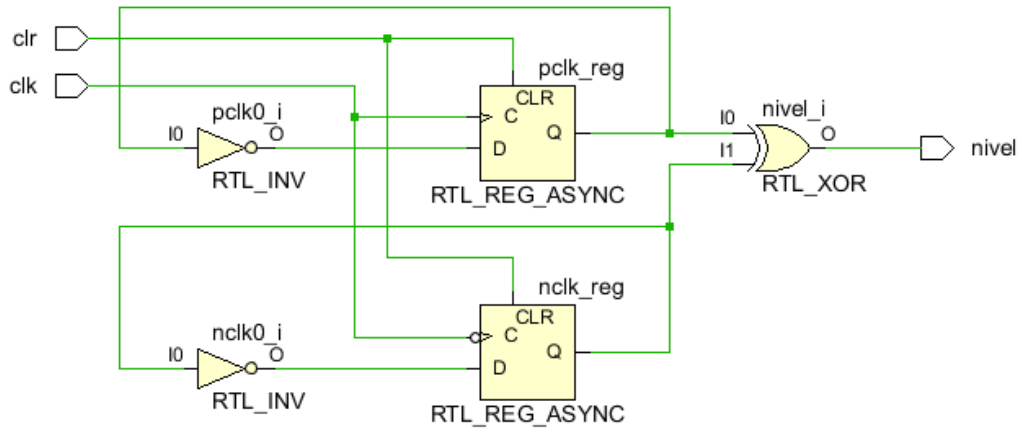
4.1.2. Memoria de Código de Función



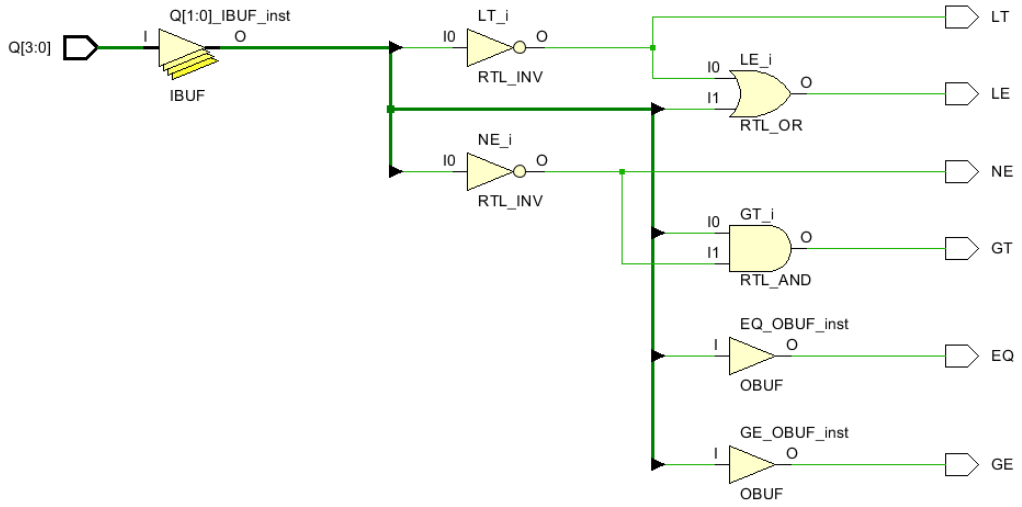
4.1.3. Memoria de Código de Operación



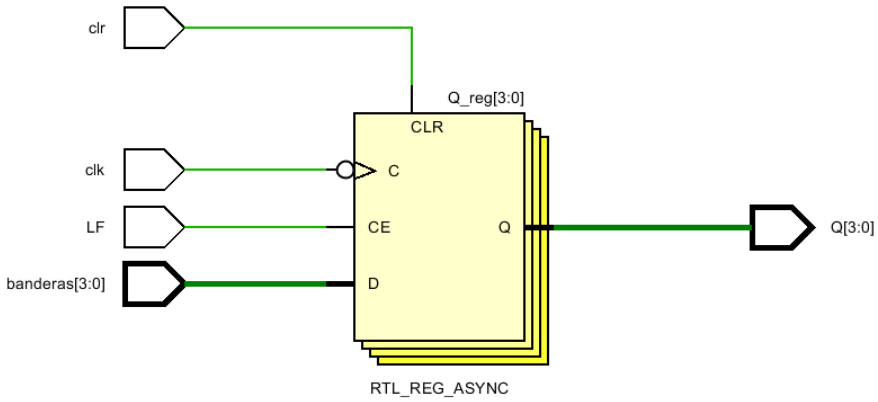
4.1.4. Nivel



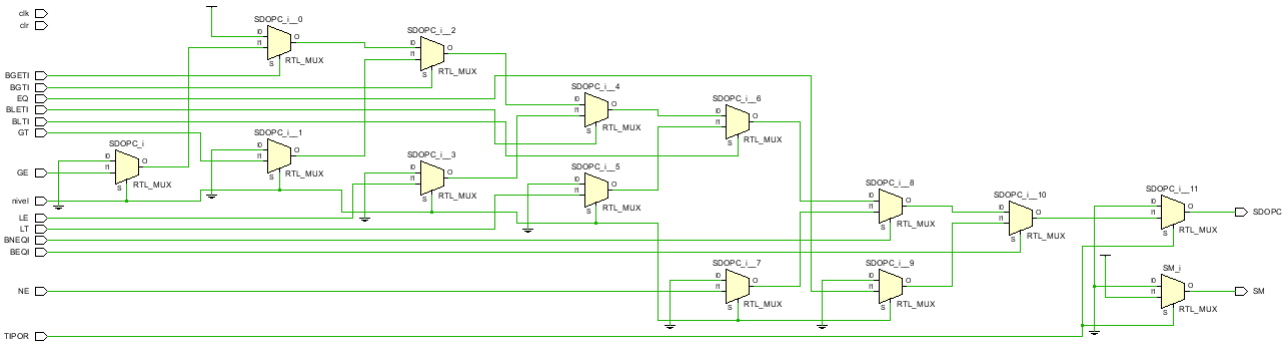
4.1.5. Condición



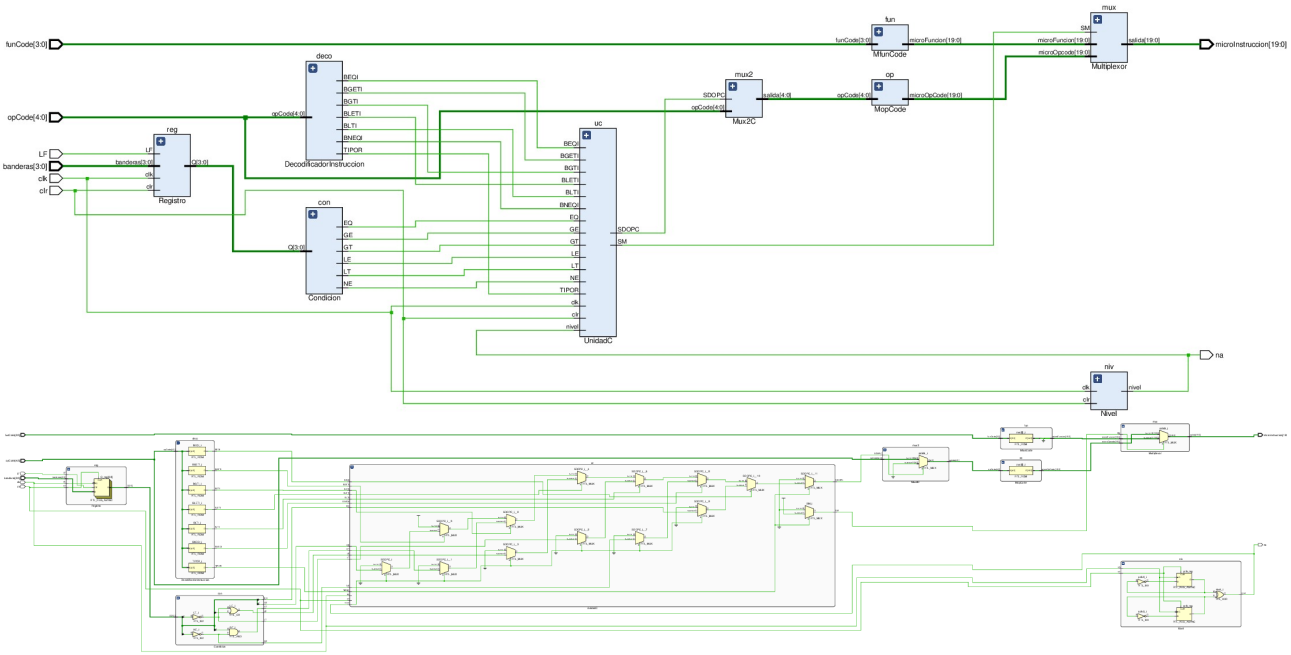
4.1.6. Registro de banderas



4.1.7. Carta ASM

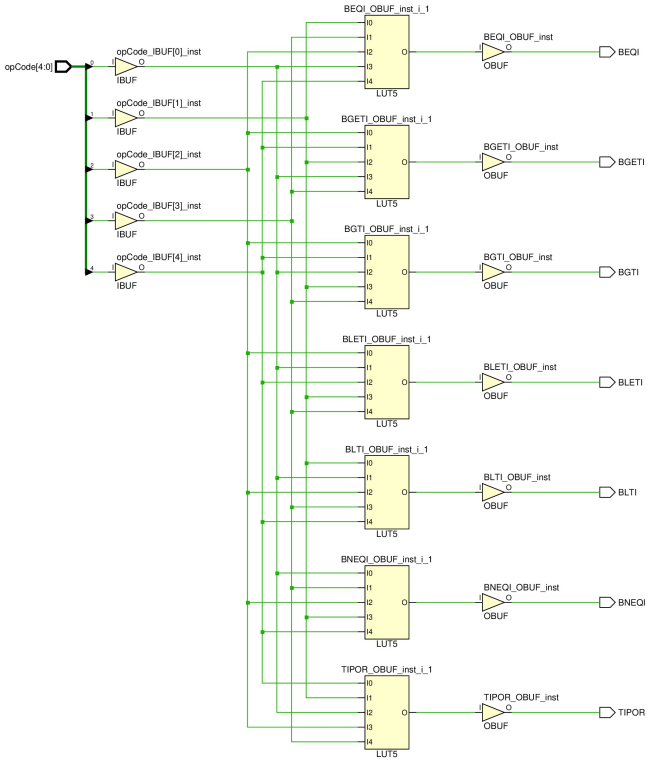


4.1.8. Unidad de Control

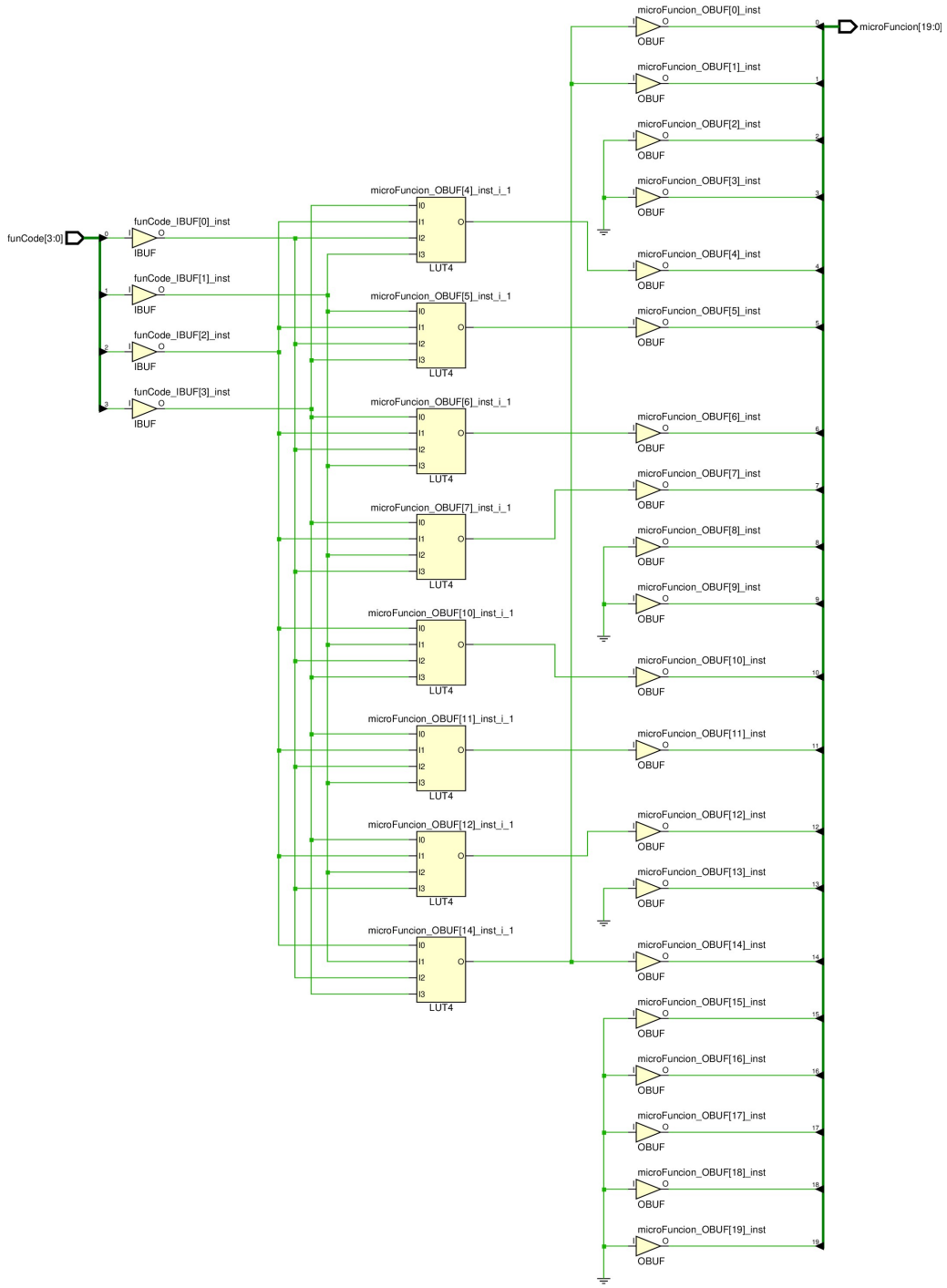


4.2. Synthesis

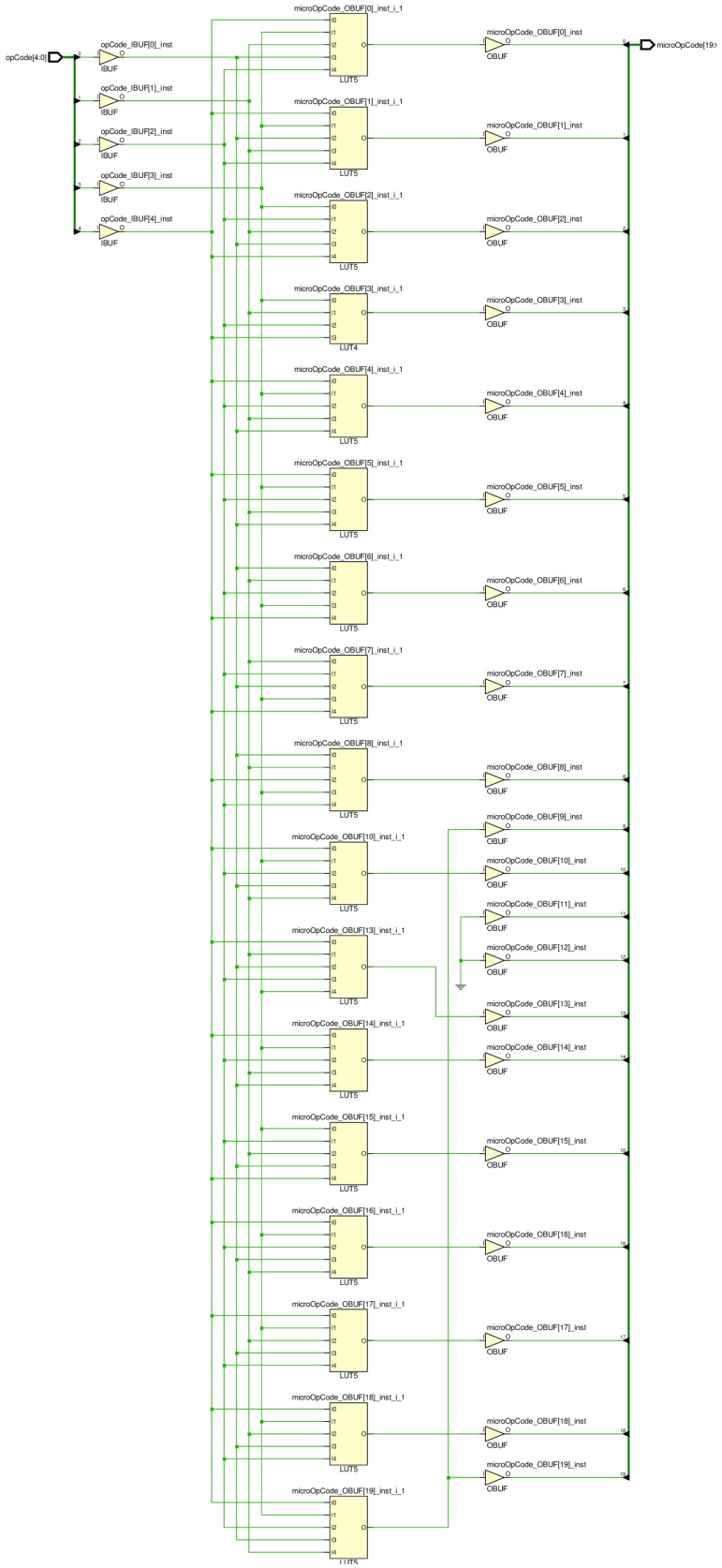
4.2.1. Decodificador de Instrucción



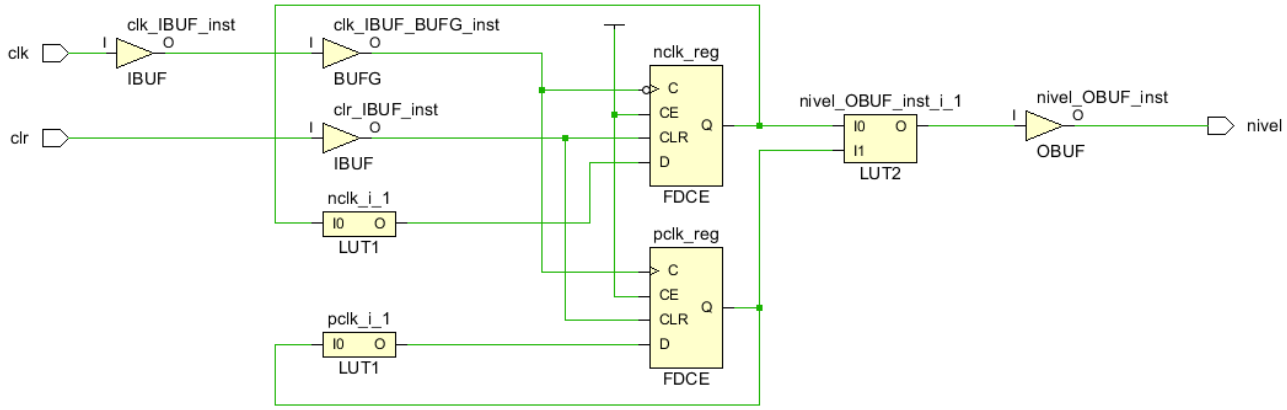
4.2.2. Memoria de Código de Función



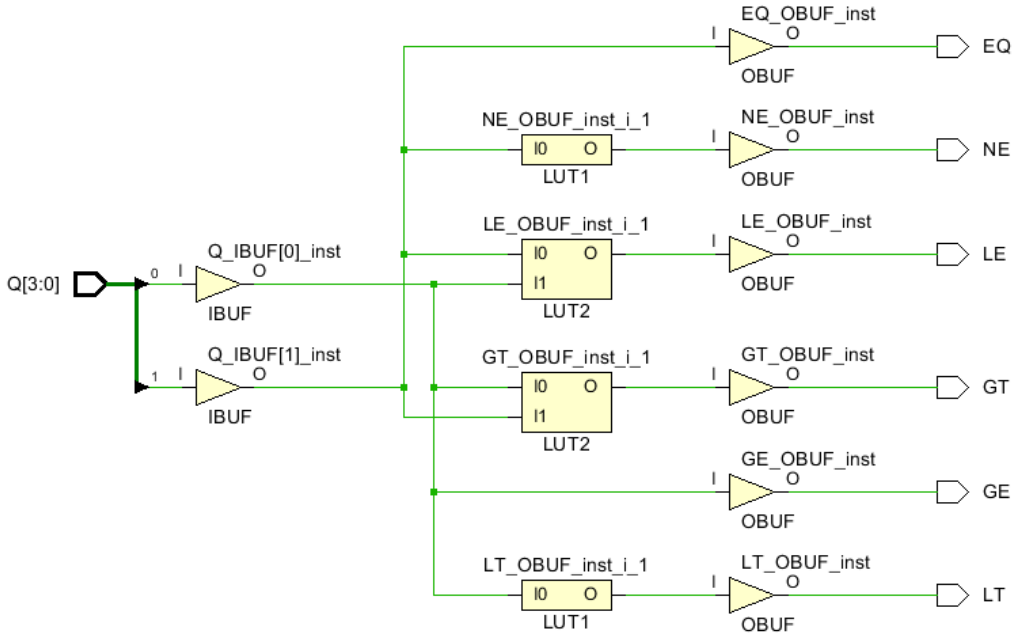
4.2.3. Memoria de Código de Operación



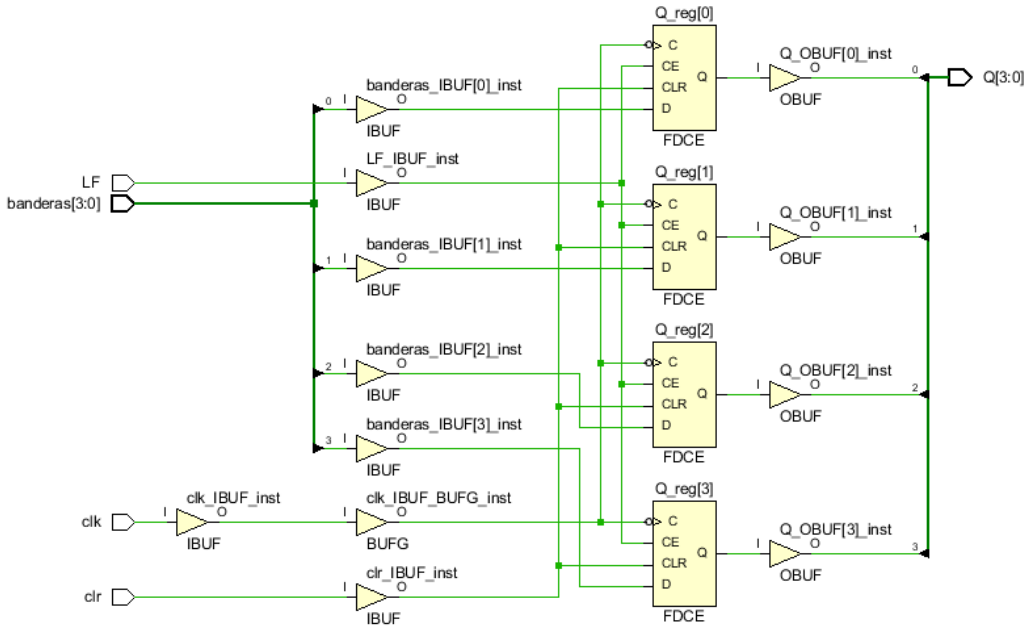
4.2.4. Nivel



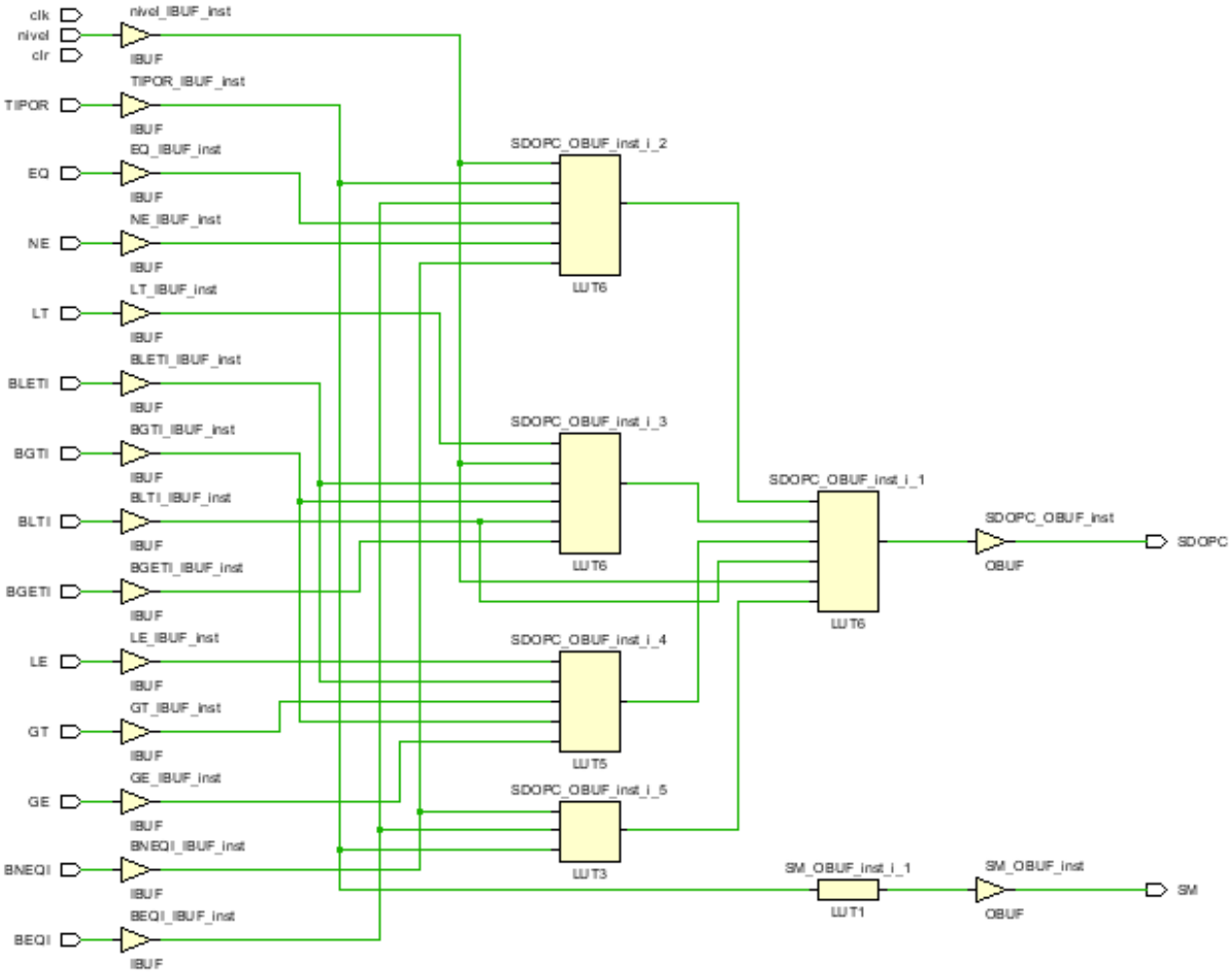
4.2.5. Condición



4.2.6. Registro de banderas



4.2.7. Carta ASM



4.2.8. Unidad de Control