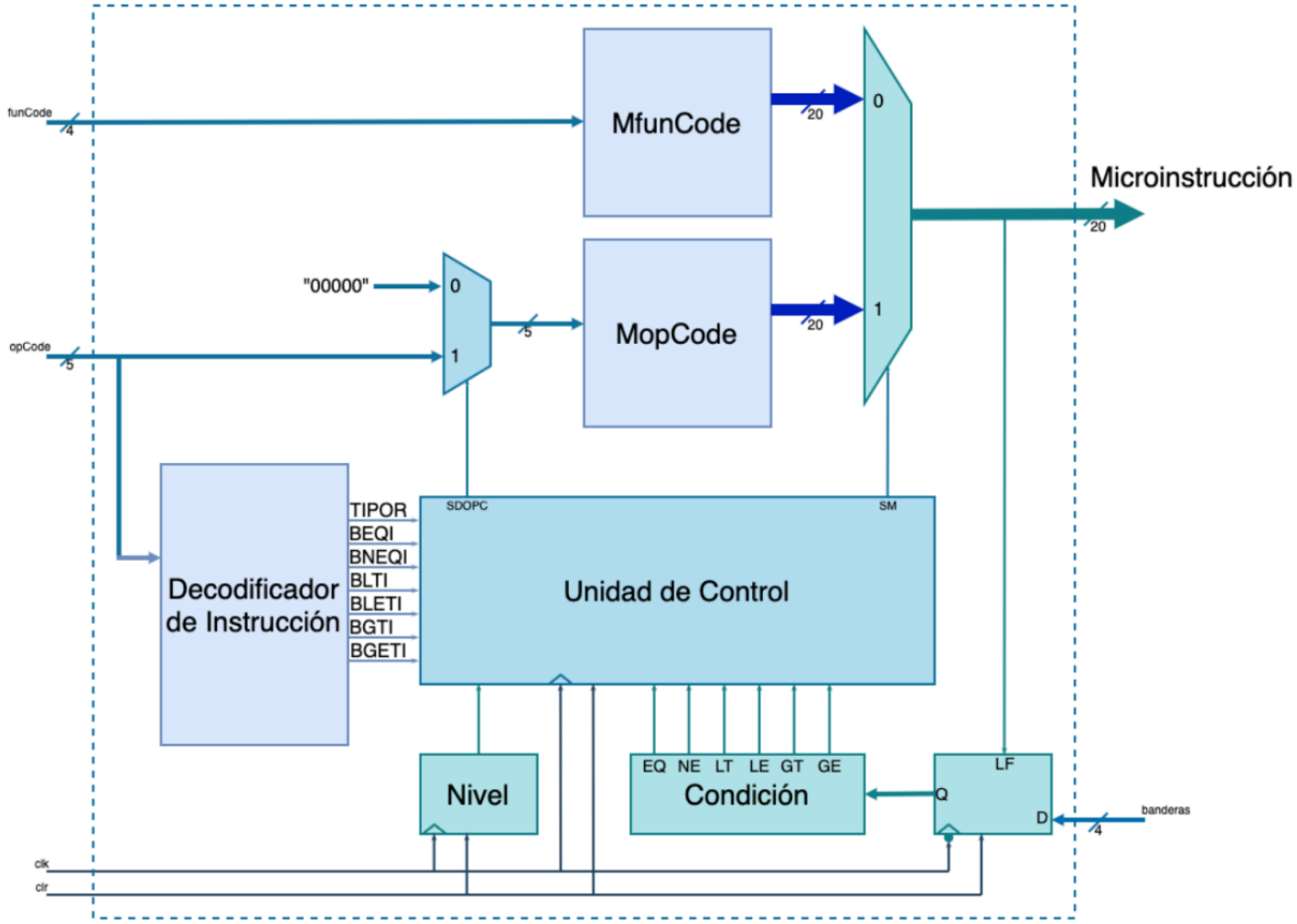
Alumno:	García González Aarón Antonio	11
Grupo:	3CV2	
Unidad de Aprendizaje:	Arquitectura de computadoras	
Profesora:	Vega García Nayeli	
Practica #14:	Unidad de control	
Fecha:	Viernes 26 de Junio de 2020	

Implementar la unidad de control, por bloques, diseñada en clase.



Índice

1. Código de implementación

1.1. Memoria de código de función

```
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.
6.  entity MFunCode is
7.      generic ( n : integer := 4 );
8.      Port (
9.          codigo_funcion : in STD_LOGIC_VECTOR(n-1 downto 0);
10.         microinstruccion_fcode : out STD_LOGIC_VECTOR (19 downto 0)
11.     );
12. end MFunCode;
13.
14. architecture Behavioral of MFunCode is
15.     -- Declaracion de microinstrucciones (Solo tipo R)
16.     -- SDMP UP DW WPC SR2 SWD SEXT SHE DIR WR SOP1 SOP2 ALUOP3 ALUOP2 ALUOP1 ALUOP0 SDMD WD SR LF
17.
18.     constant R_ADD : STD_LOGIC_VECTOR (19 downto 0) := "00000100010000110011"; -- ADD 0
19.     constant R_SUB : STD_LOGIC_VECTOR (19 downto 0) := "00000100010001110011"; -- SUB 1
20.
21.     constant R_AND : STD_LOGIC_VECTOR (19 downto 0) := "00000100010000000011"; -- AND 2
22.     constant R_OR : STD_LOGIC_VECTOR (19 downto 0) :=  "00000100010000010011"; -- OR 3
23.     constant R_XOR : STD_LOGIC_VECTOR (19 downto 0) := "00000100010000100011"; -- XOR 4
24.     constant R_NAND : STD_LOGIC_VECTOR (19 downto 0) :="00000100010011010011"; -- NAND 5
25.     constant R_NOR : STD_LOGIC_VECTOR (19 downto 0) := "00000100010011000011"; -- NOR 6
26.     constant R_XNOR : STD_LOGIC_VECTOR (19 downto 0) :="00000100010001100011"; -- XNOR 7
27.     constant R_NOT : STD_LOGIC_VECTOR (19 downto 0) := "00000100010011010011"; -- NOT 8
28.
29.     constant R_SLL : STD_LOGIC_VECTOR (19 downto 0) := "00000001110000000000"; -- SLL 9
30.     constant R_SRL : STD_LOGIC_VECTOR (19 downto 0) := "00000001010000000000"; -- SRL 10
31.
32.     type memoria is array (0 to (2**n)-1) of std_logic_vector(19 downto 0);
33.
34.     constant funCode : memoria := (
35.         R_ADD,
36.         R_SUB,
37.         R_AND,
38.         R_OR,
39.         R_XOR,
40.         R_NAND,
41.         R_NOR,
42.         R_XNOR,
43.         R_NOT,
44.         R_SLL,
45.         R_SRL,
46.         others => (others => '0')
47.     );
48.
49.     begin
50.         microinstruccion_fcode <= funCode(conv_integer(codigo_funcion));
51.
52. end Behavioral;
```

1.2. Memoria de código de operación

```
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.
6.  entity MOpCode is
7.      generic ( n : integer := 5 );
8.      Port (
9.          codigo_operacion : in STD_LOGIC_VECTOR(n-1 downto 0);
10.         microinstruccion : out STD_LOGIC_VECTOR (19 downto 0)
11.     );
12. end MOpCode;
13.
14. architecture Behavioral of MOpCode is
15.     -- Declaracion de microinstrucciones (Todas menos tipo R)
16.     -- SDMP UP DW WPC SR2 SWD SEXT SHE DIR WR SOP1 SOP2 ALUOP3 ALUOP2 ALUOP1 ALUOP0 SDMD WD SR LF
17.
18.     constant VERIFICACION : STD_LOGIC_VECTOR(19 downto 0) := "00001000000001110001"; -
    - VERIFICACION 0
19.     constant LI : STD_LOGIC_VECTOR(19 downto 0) := "00000000010000000000"; -- LI 1
20.     constant LWI : STD_LOGIC_VECTOR(19 downto 0) := "00001100010000001000"; -- LWI 2
21.     constant SWI : STD_LOGIC_VECTOR(19 downto 0) := "00001000000000001100"; -- SWI 3
22.     constant SW : STD_LOGIC_VECTOR(19 downto 0) := "00001010000100110101"; -- SW 4
23.
24.     constant ADDI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010100110011"; -- ADDI 5
25.     constant SUBI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010101110011"; -- SUBI 6
26.
27.     constant ANDI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010100000011"; -- ANDI 7
28.     constant ORI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010100010011"; -- ORI 8
29.     constant XORI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010100100011"; -- XORI 9
30.     constant NANDI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010111010011"; -- NANDI 10
31.     constant NORI : STD_LOGIC_VECTOR(19 downto 0) :=  "00000100010111000011"; -- NORI 11
32.     constant XNORI : STD_LOGIC_VECTOR(19 downto 0) := "00000100010101100011"; -- XNORI 12
33.
34.     constant SALTO : STD_LOGIC_VECTOR(19 downto 0) := "10010000001100110011"; -- SALTO 13-18
35.     constant B : STD_LOGIC_VECTOR(19 downto 0) := "00010000000000000000"; -- B 19
36.
37.     constant CALL : STD_LOGIC_VECTOR(19 downto 0) := "01010000000000000000"; -- CALL 20
38.     constant RET : STD_LOGIC_VECTOR(19 downto 0) := "00100000000000000000"; -- RET 21
39.     constant NOP : STD_LOGIC_VECTOR(19 downto 0) := "00000000000000000000"; -- NOP 22
40.
41.     constant LW : STD_LOGIC_VECTOR(19 downto 0) := "00000110010100110001"; -- LW 23
42.
43.     type memoria is array (0 to (2**n)-1) of std_logic_vector(19 downto 0);
44.     constant opCode : memoria := (
45.         VERIFICACION,
46.         LI,
47.         LWI,
48.         SWI,
49.         SW,
50.         ADDI,
51.         SUBI,
52.         ANDI,
53.         ORI,
54.         XORI,
55.         NANDI,
56.         NORI,
57.         XNORI,
58.         SALTO, -- BEQI
59.         SALTO, -- BNEI
60.         SALTO, -- BLTI
61.         SALTO, -- BLETI
```

```
62.         SALTO, -- BGTI
63.         SALTO, -- BGETI
64.         B,
65.         CALL,
66.         RET,
67.         NOP,
68.         LW,
69.         others => (others => '0')
70.     );
71. 
72.     begin
73.     microinstruccion <= opCode(conv_integer(codigo_operacion));
74. 
75. 
76. end Behavioral;
```

1.3. Decodificador

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.
5.  entity decodificador is
6.      Port (
7.          codigo_operacion : in STD_LOGIC_VECTOR(4 downto 0);
8.          tipo_R : out STD_LOGIC;
9.          BEQI : out STD_LOGIC;
10.         BNEI : out STD_LOGIC;
11.         BLTI : out STD_LOGIC;
12.         BLETI : out STD_LOGIC;
13.         BGTI : out STD_LOGIC;
14.         BGETI : out STD_LOGIC
15.     );
16. end decodificador;
17.
18. architecture Behavioral of decodificador is
19.     begin
20.
21.     process(codigo_operacion)
22.         begin
23.         if(codigo_operacion = "00000") then -- TIPO R
24.             tipo_R <= '1';
25.             BEQI <= '0';
26.             BNEI <= '0';
27.             BLTI <= '0';
28.             BLETI <= '0';
29.             BGTI <= '0';
30.             BGETI <= '0';
31.
32.         elsif(codigo_operacion = "01101") then -- BEQI
33.             tipo_R <= '0';
34.             BEQI <= '1';
35.             BNEI <= '0';
36.             BLTI <= '0';
37.             BLETI <= '0';
38.             BGTI <= '0';
39.             BGETI <= '0';
40.         elsif(codigo_operacion = "01110") then -- BNEI
41.             tipo_R <= '0';
42.             BEQI <= '0';
43.             BNEI <= '1';
44.             BLTI <= '0';
45.             BLETI <= '0';
46.             BGTI <= '0';
47.             BGETI <= '0';
48.         elsif(codigo_operacion = "01111") then -- BLTI
49.             tipo_R <= '0';
50.             BEQI <= '0';
51.             BNEI <= '0';
52.             BLTI <= '1';
53.             BLETI <= '0';
54.             BGTI <= '0';
55.             BGETI <= '0';
56.         elsif(codigo_operacion = "10000") then -- BLTEI
57.             tipo_R <= '0';
58.             BEQI <= '0';
59.             BNEI <= '0';
60.             BLTI <= '0';
61.             BLETI <= '1';
62.             BGTI <= '0';
```

```
63.             BGETI <= '0';
64.
65.         elsif(codigo_operacion = "10001") then -- BGTI
66.             tipo_R <= '0';
67.             BEQI <= '0';
68.             BNEI <= '0';
69.             BLTI <= '0';
70.             BLETI <= '0';
71.             BGTI <= '1';
72.             BGETI <= '0';
73.         elsif(codigo_operacion = "10010") then -- BGETI
74.             tipo_R <= '0';
75.             BEQI <= '0';
76.             BNEI <= '0';
77.             BLTI <= '0';
78.             BLETI <= '0';
79.             BGTI <= '0';
80.             BGETI <= '1';
81.         else -- NO ES TIPO R NI SALTO CONDICIONAL
82.             tipo_R <= '0';
83.             BEQI <= '0';
84.             BNEI <= '0';
85.             BLTI <= '0';
86.             BLETI <= '0';
87.             BGTI <= '0';
88.             BGETI <= '0';
89.         end if;
90.     end process;
91.
92. end Behavioral;
```

1.4. Multiplexores

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity mux5bits is
5.      Port (
6.          codigo_op: in STD_LOGIC_VECTOR(4 downto 0);
7.          sdopc : in STD_LOGIC;
8.          salida : out STD_LOGIC_VECTOR(4 downto 0)
9.      );
10. end mux5bits;
11.
12. architecture Behavioral of mux5bits is
13.     constant cero : STD_LOGIC_VECTOR(4 downto 0) := "00000";
14.     begin
15.         with sdopc select
16.             salida <=
17.                 codigo_op when '1',
18.                 cero when others;
19.
20. end Behavioral;
```

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity mux20bits is
5.      Port (
6.          codigo_fu: in STD_LOGIC_VECTOR(19 downto 0);
7.          codigo_op: in STD_LOGIC_VECTOR(19 downto 0);
8.          sm : in STD_LOGIC;
9.          salida : out STD_LOGIC_VECTOR(19 downto 0)
10.     );
11. end mux20bits;
12.
13. architecture Behavioral of mux20bits is
14.     begin
15.
16.     with sm select
17.         salida <= codigo_op when '1',
18.         codigo_fu when others;
19. end Behavioral;
```

1.5. Nivel

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity nivel is
5.      Port ( clk : in STD_LOGIC;
6.             clr : in STD_LOGIC;
7.             na : out STD_LOGIC);
8.  end nivel;
9.
10. architecture Behavioral of nivel is
11.     signal pclk, nclk : STD_LOGIC;
12.     begin
13.
14.     ALTO : process(clr, clk)
15.         begin
16.         if(clr = '1') then
17.             pclk <= '0';
18.         elsif(rising_edge(clk)) then
19.             pclk <= not pclk;
20.         end if;
21.     end process;
22.
23.     BAJO : process(clr, clk)
24.         begin
25.         if(clr = '1') then
26.             nclk <= '0';
27.         elsif(falling_edge(clk)) then
28.             nclk <= not nclk;
29.         end if;
30.     end process;
31.
32.     na <= nclk xor pclk; -- LOGICA COMBINATORIA
33.
34. end Behavioral;
```

1.6. Condición

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity condicion is
5.      Port(
6.          banderas : in STD_LOGIC_VECTOR(3 downto 0);
7.          EQ : out STD_LOGIC;
8.          NE : out STD_LOGIC;
9.          LT : out STD_LOGIC;
10.         LE : out STD_LOGIC;
11.         GT : out STD_LOGIC;
12.         GE : out STD_LOGIC
13.     );
14. end condicion;
15.
16. architecture Behavioral of condicion is
17.     -- BANDERAS 0 - C, 1 - Z, 2- N, 3 - OV
18.     signal C, Z, N, OV : STD_LOGIC;
19.
20.     begin
21.
22.     -- fetch de banderas
23.     C <= banderas(0);
24.     Z <= banderas(1);
25.     N <= banderas(2);
26.     OV <= banderas(3);
27.
28.     EQ <= Z;
29.     NE <= not Z;
30.     LT <= not C;
31.     LE <= Z or (not C);
32.     GT <= (not Z) and C;
33.     GE <= C;
34.
35. end Behavioral;
```

1.7. Registro

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity registro is
5.      Port (
6.          banderas_entrada : in STD_LOGIC_VECTOR(3 downto 0);
7.          lf : in STD_LOGIC;
8.          clk : in STD_LOGIC;
9.          clr : in STD_LOGIC;
10.         banderas_salida : out STD_LOGIC_VECTOR(3 downto 0)
11.     );
12. end registro;
13.
14. architecture Behavioral of registro is
15.     begin
16.
17.     process
18.         begin
19.         if(clr = '1') then
20.             banderas_salida <= "0000";
21.         elsif(falling_edge(clk)) then
22.             if(lf = '1')then
23.                 banderas_salida <= banderas_entrada;
24.             end if;
25.         end if;
26.     end process;
27. end Behavioral;
```

1.8. Unidad de control

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity control is -- ASM
5.      Port (
6.          TIPOR, BEQI, BNEI, BLTI, BLETI, BGTI, BGETI : in STD_LOGIC;
7.          EQ, NE, LT, LE, GT, GE : in STD_LOGIC;
8.          clk, clr, NA: in STD_LOGIC;
9.          SDOPC, SM : out STD_LOGIC
10.     );
11. end control;
12.
13. architecture Behavioral of control is
14.     type estados is (A);
15.     signal estado_actual, estado_siguiente : estados;
16.
17.     begin
18.
19.     trnasicion : process(clr, clk)-- establece el cambio de estado actual a estado siguinete
20.         begin
21.         if(clr = '1') then
22.             estado_actual <= A;
23.         elsif(rising_edge(clk)) then
24.             estado_actual <= estado_siguiente;
25.         end if;
26.     end process;
27.
28.     asm : process(TIPOR, BEQI, BNEI, BLTI, BLETI, BGTI, BGETI, NA, EQ, NE, LT, LE, GT, GE, NA, estado
    _actual)
29.         begin
30.         SM <= '0';
31.         SDOPC <= '0';
32.
33.         case estado_actual is
34.             when A => estado_siguiente <= A;
35.             if(TIPOR = '1') then
36.                  SM <= '0'; -- No es necesaria por la inicializacion en 0's
37.             else
38.                 if (BEQI = '1') then
39.                     if (NA = '1') then
40.                         SM <= '1';
41.                     else
42.                         if (EQ = '1') then
43.                             SM <= '1';
44.                             SDOPC <= '1';
45.                         else
46.                             SM <= '1';
47.                         end if;
48.                     end if;
49.                 elsif (BNEI = '1') then
50.                     if (NA = '1') then
51.                         sm <= '1';
52.                     else
53.                         if (NE = '1') then
54.                             SDOPC <= '1';
55.                             SM <= '1';
56.                         else
57.                             SM <= '1';
58.                         end if;
59.                     end if;
```

```
60.                 elsif (BLTI = '1') then
61.                     if (NA = '1') then
62.                         SM <= '1';
63.                     else
64.                         if (LT = '1') then
65.                             SDOPC <= '1';
66.                             SM <= '1';
67.                         else
68.                             SM <= '1';
69.                         end if;
70.                     end if;
71.                 elsif (BLETI = '1') then
72.                     if (NA = '1') then
73.                         SM <= '1';
74.                     else
75.                         if (LE = '1') then
76.                             SDOPC <= '1';
77.                             sm <= '1';
78.                         else
79.                             SM <= '1';
80.                         end if;
81.                     end if;
82.                 elsif (BGTI = '1') then
83.                     if (NA = '1') then
84.                         SM <= '1';
85.                     else
86.                         if (GT = '1') then
87.                             SDOPC <= '1';
88.                             SM <= '1';
89.                         else
90.                             SM <= '1';
91.                         end if;
92.                     end if;
93.                 elsif (BGETI = '1') then
94.                     if (NA = '1') then
95.                         SM <= '1';
96.                     else
97.                         if (GE = '1') then
98.                             SDOPC <= '1';
99.                             SM <= '1';
100.                               else
101.                                   SM <= '1';
102.                               end if;
103.                           end if;
104.                       else
105.                           SDOPC <= '1';
106.                           SM <= '1';
107.                       end if;
108.                   end if;
109.               end case;
110.           end process;
111.
112.       end Behavioral;
```

1.9. Arquitectura completa

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity uniddad_control is
5.      Port (
6.          funCode : in STD_LOGIC_VECTOR(3 downto 0);
7.          opCode : in STD_LOGIC_VECTOR(4 downto 0);
8.          clk, clr, lf : in STD_LOGIC;
9.          banderas : in STD_LOGIC_VECTOR(3 downto 0);
10.         microInstruccion : out STD_LOGIC_VECTOR(19 downto 0)
11.     );
12. end uniddad_control;
13.
14. architecture Behavioral of uniddad_control is
15.
16.     -- memoria de codigo de funcion
17.     component MFunCode is
18.         Port (
19.             codigo_funcion : in STD_LOGIC_VECTOR(3 downto 0);
20.             microinstruccion_fcode : out STD_LOGIC_VECTOR (19 downto 0));
21.     end component;
22.
23.     -- memoria de codigo de operacion
24.     component MOpCode is
25.         Port (
26.             codigo_operacion : in STD_LOGIC_VECTOR(4 downto 0);
27.             microinstruccion : out STD_LOGIC_VECTOR (19 downto 0));
28.     end component;
29.
30.     -- condicion
31.     component condicion is
32.         Port(
33.             banderas : in STD_LOGIC_VECTOR(3 downto 0);
34.             EQ, NE, LT, LE, GT, GE : out STD_LOGIC);
35.     end component;
36.
37.     -- control (ASM)
38.     component control is
39.         Port (
40.             TIPOR, BEQI, BNEI, BLTI, BLETI, BGTI, BGETI : in STD_LOGIC;
41.             EQ, NE, LT, LE, GT, GE : in STD_LOGIC;
42.             clk, clr, NA: in STD_LOGIC;
43.             SDOPC, SM : out STD_LOGIC);
44.     end component;
45.
46.     -- decodificador
47.     component decodificador is
48.         Port (
49.             codigo_operacion : in STD_LOGIC_VECTOR(4 downto 0);
50.             tipo_R : out STD_LOGIC;
51.             BEQI ,BNEI , BLTI, BLETI, BGTI, BGETI : out STD_LOGIC);
52.     end component;
53.
54.     -- multiplexor de 5 bits
55.     component mux5bits is
56.         Port (
57.             codigo_op: in STD_LOGIC_VECTOR(4 downto 0);
58.             sdopc : in STD_LOGIC;
59.             salida : out STD_LOGIC_VECTOR(4 downto 0));
60.     end component;
61.
62.     -- multiplexor de 20 bits
```

```
63.     component mux20bits is
64.         Port (
65.             codigo_fu: in STD_LOGIC_VECTOR(19 downto 0);
66.             codigo_op: in STD_LOGIC_VECTOR(19 downto 0);
67.             sm : in STD_LOGIC;
68.             salida : out STD_LOGIC_VECTOR(19 downto 0));
69.     end component;
70.
71.     -- nivel
72.     component nivel is
73.         Port ( clk : in STD_LOGIC;
74.                clr : in STD_LOGIC;
75.                    na : out STD_LOGIC);
76.     end component;
77.
78.     -- registro
79.     component registro is
80.         Port (
81.             banderas_entrada : in STD_LOGIC_VECTOR(3 downto 0);
82.             lf, clk, clr : in STD_LOGIC;
83.             banderas_salida : out STD_LOGIC_VECTOR(3 downto 0));
84.     end component;
85.
86.     -- declaracion de señales de transporte (BUSES)
87.     signal EQ, NE, LT, LE, GT, GE : STD_LOGIC;
88.     signal NA, SDOPC, SM : STD_LOGIC;
89.     signal TIPOR, BEQI, BNEI, BLTI, BLETI, BGTI, BGETI: STD_LOGIC;
90.     signal auxUFCode, auxUOpCode, auxSalida : STD_LOGIC_VECTOR(19 downto 0);
91.     signal auxOpCode : STD_LOGIC_VECTOR(4 downto 0);
92.     signal auxBanderas : STD_LOGIC_VECTOR(3 downto 0);
93.
94.     begin
95.
96.     -- instanciar y mapear modulo de memoria de codigo de funcion
97.     MFC : MFunCode
98.         Port map (
99.             codigo_funcion => funCode,
100.                microinstruccion_fcode => auxUFCode
101.             );
102.
103.         -- instanciar y mapear modulo de memoria de codigo de operacion
104.         MOC : MOpCode
105.             Port map(
106.                 codigo_operacion => auxOpCode,
107.                 microinstruccion => auxUOpCode
108.             );
109.
110.         -- instanciar y mapear modulo de condicion
111.         COND : condicion
112.             Port map(
113.                 banderas => auxBanderas,
114.                 EQ => EQ,
115.                 NE => NE,
116.                 LT => LT,
117.                 LE => LE,
118.                 GT => GT,
119.                 GE => GE
120.             );
121.
122.         -- Instanciar y mapear modulo de control
123.         CONTR : component control
124.             Port map(
125.                 TIPOR => TIPOR,
126.                 BEQI => BEQI,
```

```
127.                    BNEI => BNEI,
128.                    BLTI => BLTI,
129.                    BLETI => BLETI,
130.                    BGTI => BGTI,
131.                    BGETI => BGETI,
132.                    EQ => EQ,
133.                    NE => NE,
134.                    LT => LT,
135.                    LE => LE,
136.                    GT => GT,
137.                    GE => GE,
138.                    clk => clk,
139.                    clr => clr,
140.                    NA => NA,
141.                    SDOPC => SDOPC,
142.                    SM => SM
143.                    );
144.
145.            -- Instanciar y mapear modulo de decodificacion
146.            DECO : decodificador
147.                Port map(
148.                    codigo_operacion => opCode,
149.                    tipo_R => TIPOR,
150.                    BEQI => BEQI,
151.                    BNEI => BNEI,
152.                    BLTI => BLTI,
153.                    BLETI => BLETI,
154.                    BGTI => BGTI,
155.                    BGETI => BGETI
156.                );
157.
158.            -- Instanciar y mapear mux de 5 bits
159.            MUX5 : Mux5bits
160.                Port map(
161.                    codigo_op => opCode,
162.                    sdopc => SDOPC,
163.                    salida => auxOpCode
164.                );
165.
166.            -- Instanciar y mapear mux de 20 bits
167.            MUX20 : mux20bits
168.                Port map(
169.                    codigo_fu => auxUFCode,
170.                    codigo_op => auxUOpCode,
171.                    salida => auxSalida,
172.                    sm => SM
173.                );
174.
175.            -- Instanciar y mapear el modulo nivel
176.            NIV : nivel
177.                Port map(
178.                    clk => clk,
179.                    clr => clk,
180.                    na => NA
181.                );
182.
183.            -- Instanciar y mapear el modulo de registro
184.            REG : registro
185.                Port map(
186.                    banderas_entrada => banderas,
187.                    lf => lf,
188.                    clk => clk,
189.                    clr => clr,
190.                    banderas_salida => auxBanderas
```

```
191.             );
192.
193.         microInstruccion <= auxSalida;
194.
195.     end Behavioral;
```

2. Código de simulación

2.1. Memoria de código de función

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity tb_MFunCode is
5.  end tb_MFunCode;
6.
7.  architecture Behavioral of tb_MFunCode is
8.      component MFunCode is
9.          Port (
10.             codigo_funcion : in STD_LOGIC_VECTOR(3 downto 0);
11.             microinstruccion_fcode : out STD_LOGIC_VECTOR (19 downto 0)
12.         );
13.     end component;
14.
15.     signal codigo_funcion : STD_LOGIC_VECTOR(3 downto 0);
16.     signal microinstruccion_fcode : STD_LOGIC_VECTOR (19 downto 0);
17.
18.     begin
19.
20.     u1 : MFunCode port map (
21.         codigo_funcion => codigo_funcion,
22.         microinstruccion_fcode => microinstruccion_fcode
23.     );
24.
25.     -- Stimulus process
26.     SP : process
27.         begin
28.         -- Vamos a probar con cada numero
29.
30.         codigo_funcion <= "0000"; -- ADD
31.         wait for 10 ns;
32.         codigo_funcion <= "0001"; -- SUB
33.         wait for 10 ns;
34.         codigo_funcion <= "0010"; -- AND
35.         wait for 10 ns;
36.         codigo_funcion <= "0011"; -- OR
37.         wait for 10 ns;
38.         codigo_funcion <= "0100"; -- XOR
39.         wait for 10 ns;
40.         codigo_funcion <= "0101"; -- NAND
41.         wait for 10 ns;
42.         codigo_funcion <= "0110"; -- NOR
43.         wait for 10 ns;
44.         codigo_funcion <= "0111"; -- XNOR
45.         wait for 10 ns;
46.         codigo_funcion <= "1000"; -- NOT
47.         wait for 10 ns;
48.         codigo_funcion <= "1001"; -- SLL
49.         wait for 10 ns;
50.         codigo_funcion <= "1010"; -- SRL
51.         wait for 10 ns;
52.
53.         wait;
54.     end process;
55.
56. end Behavioral;
```

2.2. Memoria de código de operación

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity tb_MOpCode is
5.  end tb_MOpCode;
6.
7.  architecture Behavioral of tb_MOpCode is
8.      component MOpCode is
9.          Port (
10.             codigo_operacion : in STD_LOGIC_VECTOR(4 downto 0);
11.             microinstruccion : out STD_LOGIC_VECTOR (19 downto 0));
12.     end component;
13.
14.     signal codigo_operacion : STD_LOGIC_VECTOR(4 downto 0);
15.     signal microinstruccion : STD_LOGIC_VECTOR (19 downto 0);
16.
17.     begin
18.
19.     u1 : MOpCode port map (
20.         codigo_operacion => codigo_operacion,
21.         microinstruccion => microinstruccion
22.     );
23.
24.     -- Stimulus process
25.     SP : process
26.         begin
27.         -- Vamos a probar con cada numero
28.
29.         codigo_operacion <= "00000"; -- VERIFICACION
30.         wait for 10 ns;
31.         codigo_operacion <= "00001"; -- LI
32.         wait for 10 ns;
33.         codigo_operacion <= "00010"; -- LWI
34.         wait for 10 ns;
35.         codigo_operacion <= "00011"; -- SWI
36.         wait for 10 ns;
37.         codigo_operacion <= "00100"; -- SW
38.         wait for 10 ns;
39.         codigo_operacion <= "00101"; -- ADDI
40.         wait for 10 ns;
41.         codigo_operacion <= "00110"; -- SUBI
42.         wait for 10 ns;
43.         codigo_operacion <= "00111"; -- ANDI
44.         wait for 10 ns;
45.         codigo_operacion <= "01000"; -- ORI
46.         wait for 10 ns;
47.         codigo_operacion <= "01001"; -- XORI
48.         wait for 10 ns;
49.         codigo_operacion <= "01010"; -- NANDI
50.         wait for 10 ns;
51.         codigo_operacion <= "01011"; -- NORI
52.         wait for 10 ns;
53.         codigo_operacion <= "01100"; -- XNORI
54.         wait for 10 ns;
55.         codigo_operacion <= "01101"; -- BEQI
56.         wait for 10 ns;
57.         codigo_operacion <= "01110"; -- BNEI
58.         wait for 10 ns;
59.         codigo_operacion <= "01111"; -- BLTI
60.         wait for 10 ns;
61.         codigo_operacion <= "10000"; -- BLETI
62.         wait for 10 ns;
```

```
63.         codigo_operacion <= "10001"; -- BGTI
64.         wait for 10 ns;
65.         codigo_operacion <= "10010"; -- BGETI
66.         wait for 10 ns;
67.         codigo_operacion <= "10011"; -- B
68.         wait for 10 ns;
69.         codigo_operacion <= "10100"; -- CALL
70.         wait for 10 ns;
71.         codigo_operacion <= "10101"; -- RET
72.         wait for 10 ns;
73.         codigo_operacion <= "10110"; -- NOP
74.         wait for 10 ns;
75.         codigo_operacion <= "10111"; -- LW
76.         wait for 10 ns;
77.
78.         wait;
79.     end process;
80.
81.
82. end Behavioral;
```

2.3. Decodificador

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity tb_decodificador is
5.  end tb_decodificador;
6.
7.  architecture Behavioral of tb_decodificador is
8.      component decodificador is
9.          Port (
10.             codigo_operacion : in STD_LOGIC_VECTOR(4 downto 0);
11.             tipo_R : out STD_LOGIC;
12.             BEQI : out STD_LOGIC;
13.             BNEI : out STD_LOGIC;
14.             BLTI : out STD_LOGIC;
15.             BLETI : out STD_LOGIC;
16.             BGTI : out STD_LOGIC;
17.             BGETI : out STD_LOGIC
18.         );
19.     end component;
20.
21.     -- input signal
22.     signal codigo_operacion : STD_LOGIC_VECTOR (4 downto 0);
23.
24.     -- output signal
25.     signal tipo_R : STD_LOGIC;
26.     signal BEQI : STD_LOGIC;
27.     signal BNEI : STD_LOGIC;
28.     signal BLTI : STD_LOGIC;
29.     signal BLETI : STD_LOGIC;
30.     signal BGTI : STD_LOGIC;
31.     signal BGETI : STD_LOGIC;
32.
33.     begin
34.
35.     -- Insatnciate compornent
36.     DECO : decodificador Port map(
37.         codigo_operacion => codigo_operacion,
38.         tipo_R => tipo_R,
39.         BEQI => BEQI,
40.         BNEI => BNEI,
41.         BLTI => BLTI,
42.         BLETI => BLETI,
43.         BGTI => BGTI,
44.         BGETI => BGETI
45.     );
46.
47.     -- Stimulus process
48.     SP : process
49.         begin
50.
51.         -- Vamos a probar con cada numero
52.         codigo_operacion <= "00000"; -- TIPO R
53.         wait for 10 ns;
54.
55.         codigo_operacion <= "01101"; -- BEQI
56.         wait for 10 ns;
57.
58.         codigo_operacion <= "01110"; -- BNEI
59.         wait for 10 ns;
60.
61.         codigo_operacion <= "01111"; -- BLTI
62.         wait for 10 ns;
```

```
63.
64.         codigo_operacion <= "10000"; -- BLETI
65.         wait for 10 ns;
66.
67.         codigo_operacion <= "10001"; -- BGTI
68.         wait for 10 ns;
69.
70.         codigo_operacion <= "10010"; -- BGETI
71.         wait for 10 ns;
72.
73.         codigo_operacion <= "00001"; -- OTRO LI
74.         wait for 10 ns;
75.
76.         wait;
77.     end process;
78.
79.
80. end Behavioral;
```

2.4. Multiplexores

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.
5.  entity TB_mux5bits is
6.  end TB_mux5bits;
7.
8.  architecture Behavioral of TB_mux5bits is
9.      component mux5bits is
10.         Port (
11.             codigo_op : in STD_LOGIC_VECTOR(4 downto 0);
12.             sdopc : in STD_LOGIC;
13.             salida : out STD_LOGIC_VECTOR(4 downto 0)
14.         );
15.     end component;
16.
17.     -- input signs
18.     signal codigo_op : STD_LOGIC_VECTOR(4 downto 0);
19.     signal sdopc : STD_LOGIC;
20.
21.     -- output sign
22.     signal salida : STD_LOGIC_VECTOR(4 downto 0);
23.
24.     begin
25.
26.     -- Instanciate component
27.     MUX : mux5bits Port map(
28.             codigo_op => codigo_op,
29.             sdopc => sdopc,
30.             salida => salida
31.     );
32.
33.         -- Stimulus process
34.     SP : process
35.         begin
36.         codigo_op <= "00011";
37.         sdopc <= '1';
38.         wait for 20 ns;
39.         codigo_op <= "00111";
40.         sdopc <= '1';
41.         wait for 20 ns;
42.         codigo_op <= "00111";
43.         sdopc <= '0';
44.         wait for 20 ns;
45.
46.         wait;
47.     end process;
48.
49. end Behavioral;
```

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity tb_mux20bits is
5.  end tb_mux20bits;
6.
7.  architecture Behavioral of tb_mux20bits is
8.      component mux20bits is
9.          Port (
```

```
            codigo_fu: in STD_LOGIC_VECTOR(19 downto 0);
            codigo_op: in STD_LOGIC_VECTOR(19 downto 0);
            sm : in STD_LOGIC;
            salida : out STD_LOGIC_VECTOR(19 downto 0)
        );
    end component;

    signal codigo_fu : STD_LOGIC_VECTOR(19 downto 0);
    signal codigo_op :  STD_LOGIC_VECTOR(19 downto 0);
    signal sm : STD_LOGIC;
    signal salida :  STD_LOGIC_VECTOR(19 downto 0);

    begin

    -- Instanciate component
    MUX : mux20bits Port map(
           codigo_fu => codigo_fu,
           codigo_op => codigo_op,
           sm => sm,
           salida => salida
    );


    -- Stimulus process
    SP : process
        begin

        codigo_fu <= "00000100010000110011"; -- ADD
        codigo_op <= "00000000000000000000"; -- NOP
        sm <= '0';
        wait for 20 ns;
        sm <= '1';

        wait;
    end process;

end Behavioral;
```

2.5. Nivel

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.
5.  entity tb_nivel is
6.  end tb_nivel;
7.
8.  architecture Behavioral of tb_nivel is
9.      component nivel is
10.         Port (
11.             clk : in STD_LOGIC;
12.             clr : in STD_LOGIC;
13.             na : out STD_LOGIC
14.         );
15.     end component;
16.
17.     -- input signs
18.     signal clr, clk : STD_LOGIC;
19.
20.     -- output signs
21.     signal na : STD_LOGIC;
22.
23.     begin
24.
25.     -- Mapear las seÒales
26.     LEVEL : nivel Port map(
27.             clk => clk,
28.             clr => clr,
29.             na => na
30.     );
31.     -- Proceso de reloj
32.     CLOCK : process
33.         begin
34.             clk <= '0';
35.             wait for 10ns;
36.             clk <= '1';
37.             wait for 10ns;
38.     end process;
39.
40.     -- Estimulos de proceso
41.     SP : process
42.         begin
43.         clr <= '1';
44.         wait for 30 ns;
45.
46.         clr <= '0';
47.         wait for 100 ns;
48.
49.         wait;
50.     end process;
51.
52. end Behavioral;
```

2.6. Condición

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity tb_condicion is
5.  end tb_condicion;
6.
7.  architecture Behavioral of tb_condicion is
8.      component condicion is
9.          Port(
10.             banderas : in STD_LOGIC_VECTOR(3 downto 0);
11.             EQ : out STD_LOGIC;
12.             NE : out STD_LOGIC;
13.             LT : out STD_LOGIC;
14.             LE : out STD_LOGIC;
15.             GT : out STD_LOGIC;
16.             GE : out STD_LOGIC
17.         );
18.     end component;
19.
20.     -- seÒales
21.     signal banderas : STD_LOGIC_VECTOR(3 downto 0);
22.     signal EQ : STD_LOGIC;
23.     signal NE : STD_LOGIC;
24.     signal LT : STD_LOGIC;
25.     signal LE : STD_LOGIC;
26.     signal GT : STD_LOGIC;
27.     signal GE : STD_LOGIC;
28.
29.     begin
30.
31.     -- Hacer el mapeo de seÒales y componente
32.     COND : condicion Port map(
33.            banderas => banderas,
34.            EQ => EQ,
35.            NE => NE,
36.            LT => LT,
37.            LE => LE,
38.            GT => GT,
39.            GE => GE
40.     );
41.
42.
43.         -- Stimulus process
44.     SP : process
45.         begin
46.         banderas <= "0000";
47.         wait for 150 ns;
48.         banderas <= "0110";
49.         wait for 150 ns;
50.         banderas <= "0010";
51.         wait for 150 ns;
52.         banderas <= "0100";
53.         wait for 150 ns;
54.         wait;
55.     end process;
56.
57. end Behavioral;
```

2.7. Registro

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity tb_registro is
5.  end tb_registro;
6.
7.  architecture Behavioral of tb_registro is
8.      component registro is
9.          Port (
10.             banderas_entrada : in STD_LOGIC_VECTOR(3 downto 0);
11.             lf : in STD_LOGIC;
12.             clk : in STD_LOGIC;
13.             clr : in STD_LOGIC;
14.             banderas_salida : out STD_LOGIC_VECTOR(3 downto 0)
15.         );
16.     end component;
17.
18.     signal banderas_entrada : STD_LOGIC_VECTOR(3 downto 0);
19.     signal lf : STD_LOGIC;
20.     signal clk : STD_LOGIC;
21.     signal clr : STD_LOGIC;
22.     signal banderas_salida : STD_LOGIC_VECTOR(3 downto 0);
23.
24.     begin
25.
26.     -- Instaciate component for test
27.     REG : registro Port map(
28.            banderas_entrada => banderas_entrada,
29.            lf => lf,
30.            clr => clr,
31.            clk => clk,
32.            banderas_salida => banderas_salida
33.     );
34.
35.     -- Clock process
36.     CLOCK : process
37.         begin
38.         clk <= '0';
39.         wait for 10ns;
40.         clk <= '1';
41.         wait for 10ns;
42.     end process;
43.
44.     -- Stimulus process
45.     SP : process
46.         begin
47.         clr <= '1';
48.         lf <= '1';
49.         banderas_entrada <= "0000";
50.         wait for 40 ns;
51.
52.         lf <= '0';
53.         banderas_entrada <= "0000";
54.         wait for 40 ns;
55.
56.         clr <= '0';
57.         banderas_entrada <= "0010";
58.         wait for 40 ns;
59.
60.         lf <= '1';
61.         banderas_entrada <= "0010";
62.         wait for 40 ns;
```

```
63.
64.         lf <= '0';
65.         banderas_entrada <= "1110";
66.         wait for 40 ns;
67.         wait;
68.     end process;
69. end Behavioral;
```

2.8. Unidad de control

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity tb_control is
5.  end tb_control;
6.
7.  architecture Behavioral of tb_control is
8.      component control is -- ASM
9.          Port (
10.             TIPOR, BEQI, BNEI, BLTI, BLETI, BGTI, BGETI : in STD_LOGIC;
11.             EQ, NE, LT, LE, GT, GE : in STD_LOGIC;
12.             clk, clr, NA: in STD_LOGIC;
13.             SDOPC, SM : out STD_LOGIC
14.         );
15.     end component;
16.
17.     -- seÒales para comunicacion
18.     signal TIPOR, BEQI, BNEI, BLTI, BLETI, BGTI, BGETI :  STD_LOGIC;
19.     signal EQ, NE, LT, LE, GT, GE : STD_LOGIC;
20.     signal clk, clr, NA: STD_LOGIC;
21.     signal SDOPC, SM : STD_LOGIC;
22.
23.     begin
24.
25.     -- mapeo de seÒales
26.     ASM : control Port map(
27.             TIPOR => TIPOR,
28.             BEQI => BEQI,
29.             BNEI => BNEI,
30.             BLTI => BLTI,
31.             BLETI => BLETI,
32.             BGTI => BGTI,
33.             BGETI => BGETI,
34.             EQ => EQ,
35.             NE => NE,
36.             LT => LT,
37.             LE => LE,
38.             GT => GT,
39.             GE => GE,
40.             clk => clk,
41.             clr => clr,
42.             NA => NA,
43.             SDOPC => SDOPC,
44.             SM => SM
45.     );
46.
47.     -- Clock process
48.     CLOCK : process
49.         begin
50.         clk <= '0';
51.         wait for 10ns;
52.         clk <= '1';
53.         wait for 10ns;
54.     end process;
55.
56.     -- Stimulus process
57.     SP : process
58.         begin
59.
60.         clr <= '1';
61.         wait for 30 ns;
62.
```

```
63.         TIPOR <= '0';
64.         BEQI <= '1';
65.         BNEI <= '0';
66.         BLTI <= '0';
67.         BLETI <= '0';
68.         BGTI <= '0';
69.         BGETI <= '0';
70.
71.         NA <= '1';
72.
73.         EQ <= '1';
74.         NE <= '0';
75.         LT <= '0';
76.         LE <= '0';
77.         GT <= '0';
78.         GE <= '0';
79.
80.         wait for 30 ns;
81.
82.         clr <= '0';
83.         wait for 30 ns;
84.
85.         NA <= '0';
86.         wait for 30 ns;
87.
88.         clr <= '1';
89.
90.         TIPOR <= '1';
91.         BEQI <= '0';
92.         BNEI <= '0';
93.         BLTI <= '0';
94.         BLETI <= '0';
95.         BGTI <= '0';
96.         BGETI <= '0';
97.         wait for 30 ns;
98.
99.         wait;
100.           end process;
101.
102.       end Behavioral;
```

2.9. Arquitectura completa

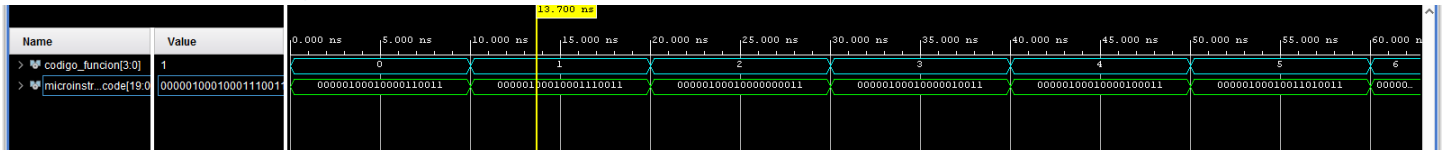
```
1.  LIBRARY ieee;
2.  USE ieee.std_logic_1164.ALL;
3.  USE ieee.std_logic_arith.all;
4.  USE IEEE.STD_LOGIC_unsigned.ALL;
5.  LIBRARY STD;
6.  USE STD.TEXTIO.ALL;
7.  USE ieee.std_logic_TEXTIO.ALL;
8.
9.  entity tb_uniddad_control is
10. end tb_uniddad_control;
11.
12. architecture Behavioral of tb_uniddad_control is
13.     component uniddad_control is
14.         Port (
15.             funCode : in STD_LOGIC_VECTOR(3 downto 0);
16.             opCode : in STD_LOGIC_VECTOR(4 downto 0);
17.             clk, clr, lf : in STD_LOGIC;
18.             banderas : in STD_LOGIC_VECTOR(3 downto 0);
19.             microInstruccion : out STD_LOGIC_VECTOR(19 downto 0)
20.         );
21.     end component;
22.
23.     -- señales de comunicacion (Buses)
24.     signal funCode : STD_LOGIC_VECTOR(3 downto 0);
25.     signal opCode : STD_LOGIC_VECTOR(4 downto 0);
26.     signal clk, clr, lf : STD_LOGIC;
27.     signal banderas : STD_LOGIC_VECTOR(3 downto 0);
28.     signal microInstruccion : STD_LOGIC_VECTOR(19 downto 0);
29.
30.     begin
31.
32.     uut : uniddad_control Port map(
33.         funCode => funCode,
34.         opCode => opCode,
35.         clk => clk,
36.         clr => clr,
37.         lf => lf,
38.         banderas => banderas,
39.         microInstruccion => microInstruccion
40.     );
41.
42.     -- proceso de reloj
43.     CLOCK : process
44.         begin
45.         clk <= '0';
46.         wait for 5 ns;
47.         clk <= '1';
48.         wait for 5 ns;
49.     end process;
50.
51.     -- Estimulos de proceso
52.     SIM : process
53.         -- ENTRADAS
54.         file ENTRADAS : TEXT;
55.         variable LINEA_E : line;
56.
57.         variable V_OP_CODE : STD_LOGIC_VECTOR(4 downto 0);
58.         variable V_FUN_CODE : STD_LOGIC_VECTOR(3 downto 0);
59.         variable V_BANDERAS : STD_LOGIC_VECTOR(3 downto 0);
60.         variable V_CLR : STD_LOGIC;
61.         variable V_LF : STD_LOGIC;
62.
```

```
63.         variable  CADENA : STRING(1 TO 9);
64.         variable CADENA2 : STRING(1 TO 21);
65.
66.         -- SALIDAS
67.         file SALIDAS : TEXT;
68.         variable LINEA_RES : line;
69.
70.         variable V_MICROINSTRUCCION : STD_LOGIC_VECTOR(19 downto 0);
71.
72.         begin
73.             file_open(SALIDAS, "C:\Users\Aaron\Desktop\P14\SALIDAS.txt", WRITE_MODE);
74.             file_open(ENTRADAS, "C:\Users\Aaron\Desktop\P14\ENTRADAS.txt", READ_MODE);
75.
76.             --Encabezados
77.             CADENA := "OP_CODE  ";
78.             write(LINEA_RES, CADENA, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "OP_CODE"
79.             CADENA := "FUN_CODE ";
80.             write(LINEA_RES, CADENA, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "FUN_CODE"
81.             CADENA := "BANDERAS ";
82.             write(LINEA_RES, CADENA, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "BANDERAS"
83.             CADENA := "CLR      ";
84.             write(LINEA_RES, CADENA, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "CLR"
85.             CADENA := "LF       ";
86.             write(LINEA_RES, CADENA, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "LF"
87.             CADENA2 := "MICROINSTRUCCION     ";
88.             write(LINEA_RES, CADENA2, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "MICROINSTRUCCION"
89.             CADENA := "NIVEL    ";
90.             write(LINEA_RES, CADENA, left, CADENA'LENGTH+1);--ESCRIBE LA CADENA "NIVEL"
91.
92.             writeline(SALIDAS, LINEA_RES);-- escribe la linea en el archivo
93.
94.             -- Leer y escribir estimulos
95.             clr <= '1';
96.             wait for 10 ns;
97.             clr <= '0';
98.             wait for 10 ns;
99.
100.                for i in 1 to 52 loop
101.
102.                    -- lee la linea completa
103.                    readline(ENTRADAS, LINEA_E);
104.                    -- leer operacion code
105.                    read(LINEA_E, V_OP_CODE);
106.                    -- leer funcion code
107.                    read(LINEA_E, V_FUN_CODE);
108.                    -- BANDERAS
109.                    read(LINEA_E, V_BANDERAS);
110.                    -- CLR
111.                    read(LINEA_E, V_CLR);
112.                    -- LF
113.                    read(LINEA_E, V_LF);
114.
115.                    opCode <= V_OP_CODE;
116.                    funCode <= V_FUN_CODE;
117.                    banderas <= V_BANDERAS;
118.                    lf <= V_LF;
119.
120.                    wait until falling_edge(clk);
121.
122.                    clr <= V_CLR;
123.                    wait for 2 ns;
124.
125.                    CADENA := "ALTO     ";
126.                    V_MICROINSTRUCCION := microInstruccion;
```

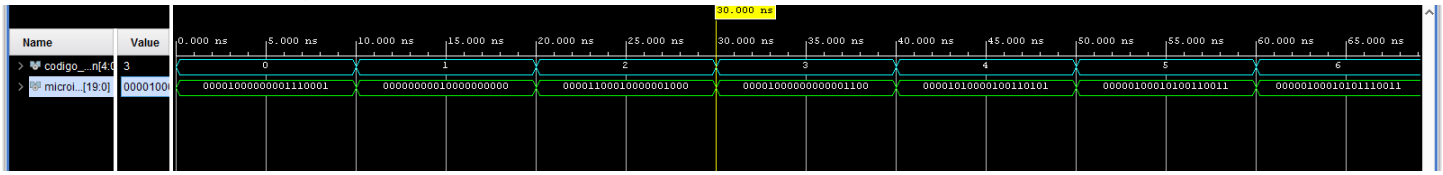
```
127.
128.                         write(LINEA_RES, V_OP_CODE, LEFT, 10);
129.                         write(LINEA_RES, V_FUN_CODE, LEFT, 10);
130.                         write(LINEA_RES, V_BANDERAS, LEFT, 10);
131.                         write(LINEA_RES, V_CLR, LEFT, 10);
132.                         write(LINEA_RES, V_LF, LEFT, 10);
133.                         write(LINEA_RES, V_MICROINSTRUCCION, LEFT, 21);
134.                         write(LINEA_RES, CADENA, LEFT, 10);
135.                         writeline(SALIDAS, LINEA_RES);
136.
137.                         wait until rising_edge(clk);
138.
139.                         CADENA := "BAJO     ";
140.                         wait for 2 ns;
141.
142.                         V_MICROINSTRUCCION := microInstruccion;
143.
144.                         write(LINEA_RES, V_OP_CODE, LEFT, 10);
145.                         write(LINEA_RES, V_FUN_CODE, LEFT, 10);
146.                         write(LINEA_RES, V_BANDERAS, LEFT, 10);
147.                         write(LINEA_RES, V_CLR, LEFT, 10);
148.                         write(LINEA_RES, V_LF, LEFT, 10);
149.                         write(LINEA_RES, V_MICROINSTRUCCION, LEFT, 21);
150.                         write(LINEA_RES, CADENA, LEFT, 10);
151.                         writeline(SALIDAS, LINEA_RES);
152.
153.                         -- CADENA := "         ";
154.                         -- write(LINEA_RES, CADENA, LEFT, 10);
155.                         -- writeline(SALIDAS, LINEA_RES);
156.
157.                     end loop;
158.
159.                     -- cierra el archivos
160.                     file_close(SALIDAS);
161.                     file_close(ENTRADAS);
162.
163.                     wait;
164.             end process;
165.         end Behavioral;
```

3. Simulación

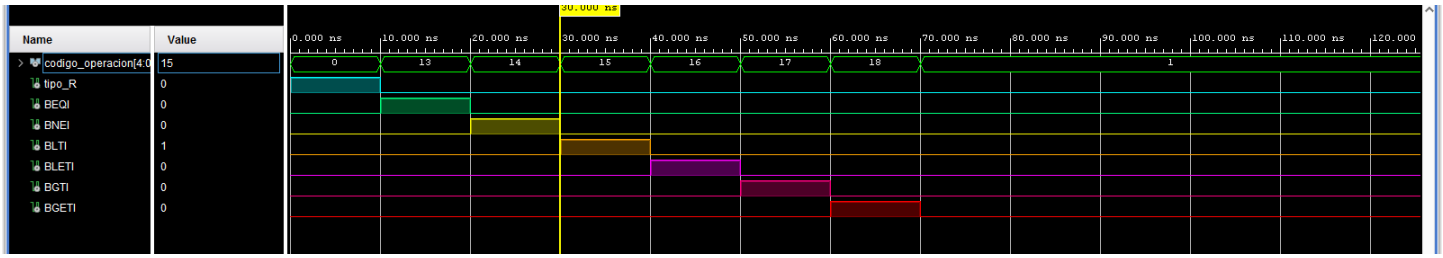
3.1. Memoria de código de función



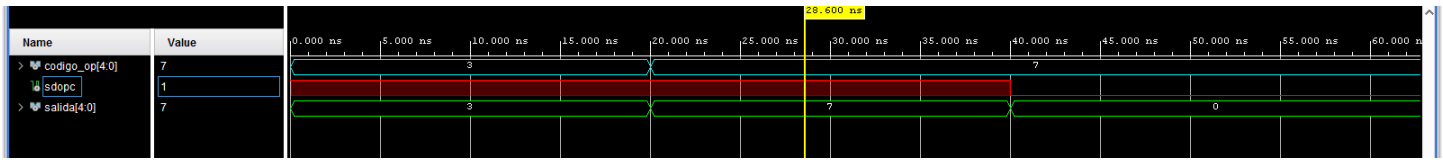
3.2. Memoria de código de operación



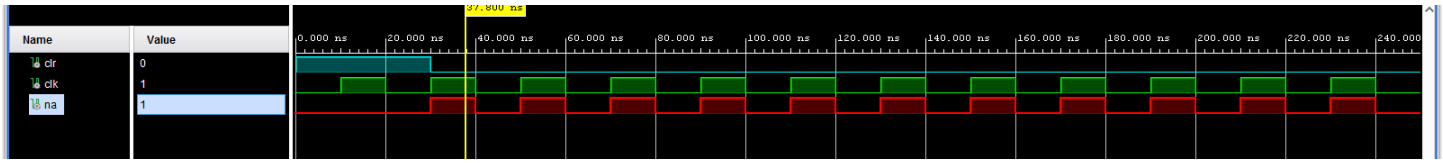
3.3. Decodificador



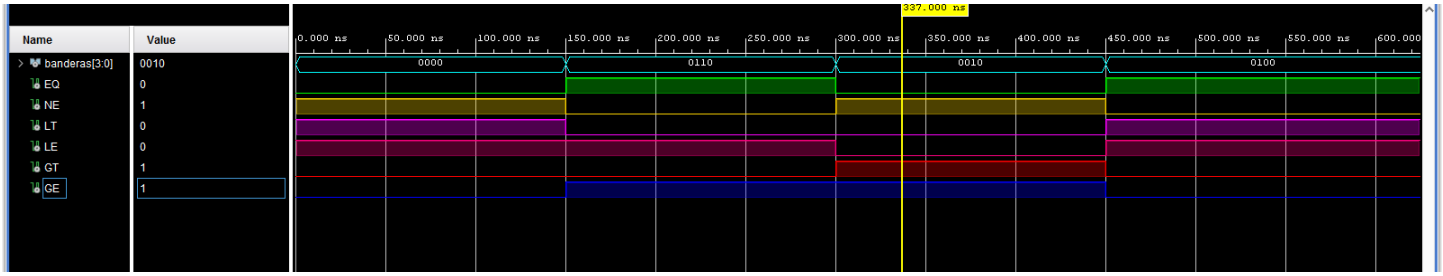
3.4. Multiplexores



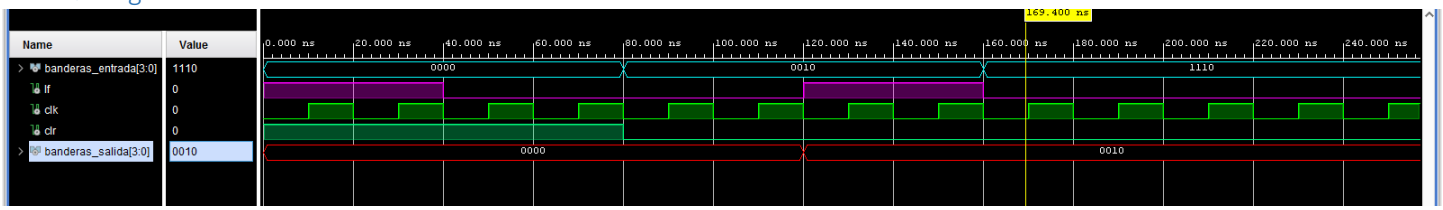
3.5. Nivel



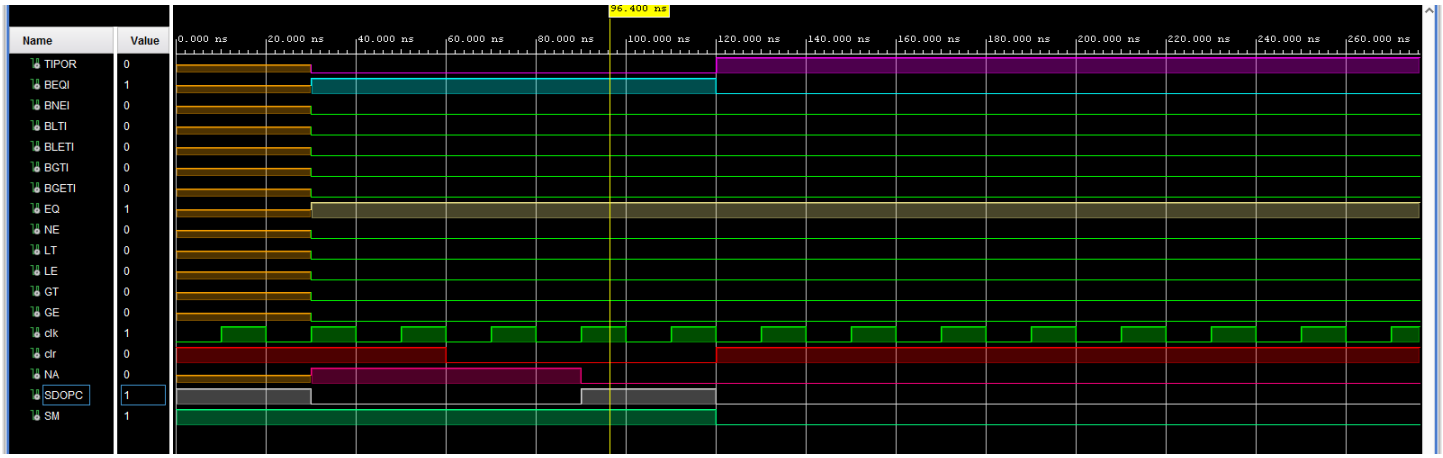
3.6. Condición



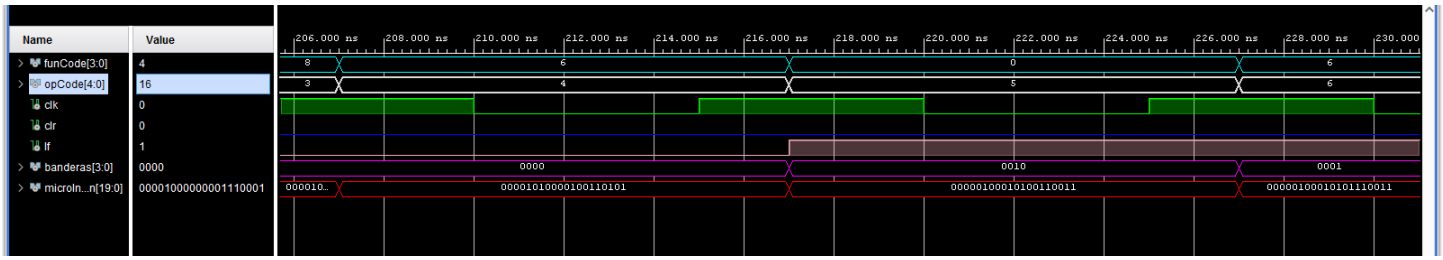
3.7. Registro

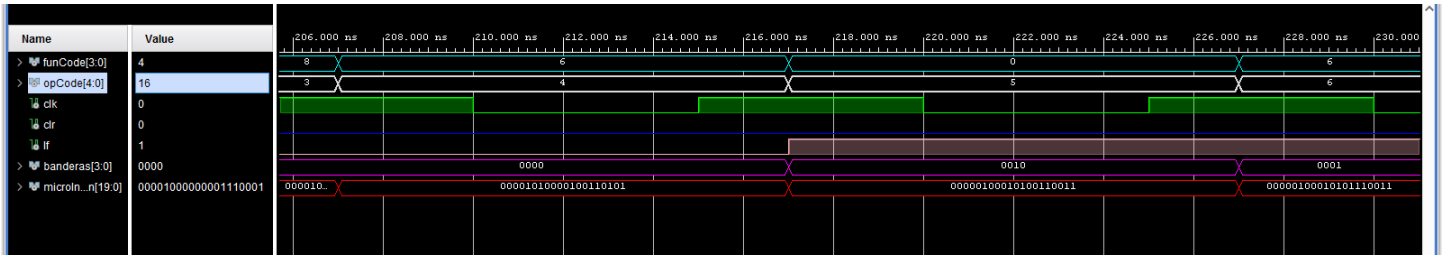


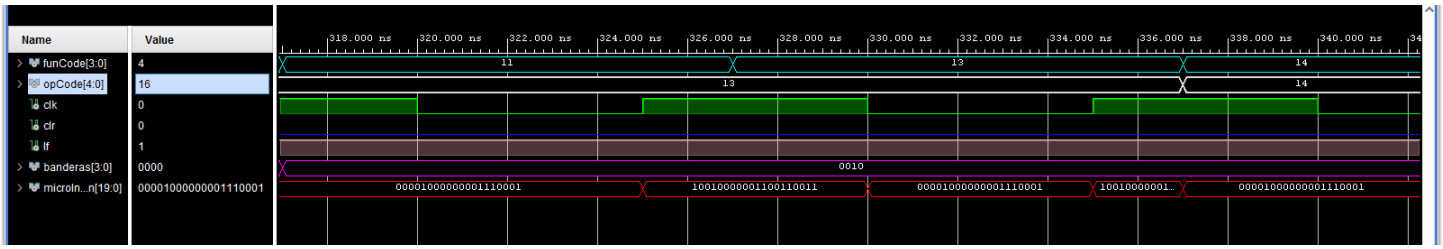
3.8. Unidad de control



3.9. Arquitectura completa




Name
Value
funCode[3:0]
4
opCode[4:0]
16
clk
0
clr
0
lf
1
banderas[3:0]
0000
microIn...n[19:0]
00001000000001110001
206.000 ns
208.000 ns
210.000 ns
212.000 ns
214.000 ns
216.000 ns
218.000 ns
220.000 ns
222.000 ns
224.000 ns
226.000 ns
228.000 ns
230.000
8
6
0
6
3
4
5
6
0000
0010
0001
000010..
000010100001001101O1
000001000100110011
000001000101011110011


Name
Value
funCode[3:0]
4
opCode[4:0]
16
clk
0
clr
0
lf
1
banderas[3:0]
0000
microIn...n[19:0]
00001000000001110001
318.000 ns
320.000 ns
322.000 ns
324.000 ns
326.000 ns
328.000 ns
330.000 ns
332.000 ns
334.000 ns
336.000 ns
338.000 ns
340.000 ns
11
13
14
13
14
0010
00001000000001110001
10010000001100110011
00001000000001110001
10010000001...
00001000000001110001

3.9.1. Estímulos de simulación

```
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
```

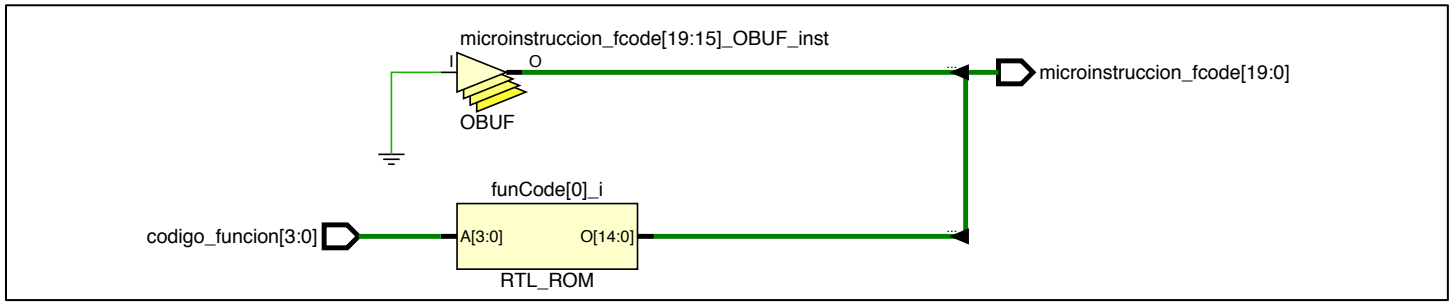
3.9.2. Salida de simulación

```
OP_CODE   FUN_CODE  BANDERAS  CLR       LF        MICROINSTRUCCION     NIVEL
00000     0000      0000      1         0         00000100010000110011 ALTO
00000     0000      0000      1         0         00000100010000110011 BAJO
00000     0000      0000      1         0         00000100010000110011 ALTO
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         00000100010001100011 ALTO
00000     0111      0100      0         1         00000100010001100011 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         00001100010000001000 ALTO
00010     0100      0000      0         0         00001100010000001000 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         00000100010101100011 ALTO
01100     1111      1010      0         1         00000100010101100011 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
```

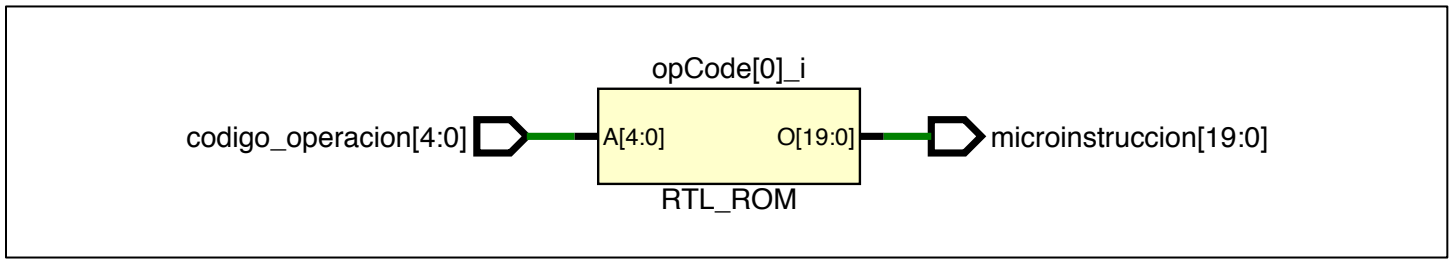
```
63. 01101     1011      0010      0         1         1001000000110011 BAJO
64. 01101     1101      0010      0         1         0000100000001110001 ALTO
65. 01101     1101      0010      0         1         1001000000110011 BAJO
66. 01110     1110      0010      0         1         0000100000001110001 ALTO
67. 01110     1110      0010      0         1         0000100000001110001 BAJO
68. 01110     1100      0000      0         1         0000100000001110001 ALTO
69. 01110     1100      0000      0         1         1001000000110011 BAJO
70. 01110     0011      0000      0         1         0000100000001110001 ALTO
71. 01110     0011      0000      0         1         1001000000110011 BAJO
72. 01111     0001      1100      0         1         0000100000001110001 ALTO
73. 01111     0001      1100      0         1         1001000000110011 BAJO
74. 01111     0000      1000      0         1         0000100000001110001 ALTO
75. 01111     0000      1000      0         1         1001000000110011 BAJO
76. 01111     0010      0100      0         1         0000100000001110001 ALTO
77. 01111     0010      0100      0         1         1001000000110011 BAJO
78. 10000     0100      0000      0         1         0000100000001110001 ALTO
79. 10000     0100      0000      0         1         1001000000110011 BAJO
80. 10000     0110      1110      0         1         0000100000001110001 ALTO
81. 10000     0110      1110      0         1         1001000000110011 BAJO
82. 10000     0101      1000      0         1         0000100000001110001 ALTO
83. 10000     0101      1000      0         1         1001000000110011 BAJO
84. 10001     0111      1010      0         1         0000100000001110001 ALTO
85. 10001     0111      1010      0         1         0000100000001110001 BAJO
86. 10001     1010      1100      0         1         0000100000001110001 ALTO
87. 10001     1010      1100      0         1         0000100000001110001 BAJO
88. 10001     1000      0000      0         1         0000100000001110001 ALTO
89. 10001     1000      0000      0         1         0000100000001110001 BAJO
90. 10010     1111      1000      0         1         0000100000001110001 ALTO
91. 10010     1111      1000      0         1         0000100000001110001 BAJO
92. 10010     1001      1010      0         1         0000100000001110001 ALTO
93. 10010     1001      1010      0         1         0000100000001110001 BAJO
94. 10010     1101      1100      0         1         0000100000001110001 ALTO
95. 10010     1101      1100      0         1         0000100000001110001 BAJO
96. 10011     1001      1100      0         0         0001000000000000000 ALTO
97. 10011     1001      1100      0         0         0001000000000000000 BAJO
98. 10100     1111      0000      0         0         0101000000000000000 ALTO
99. 10100     1111      0000      0         0         0101000000000000000 BAJO
100.       10101     0000      0000      0         0         0010000000000000000 ALTO
101.       10101     0000      0000      0         0         0010000000000000000 BAJO
102.       10110     0000      0000      0         0         0000000000000000000 ALTO
103.       10110     0000      0000      0         0         0000000000000000000 BAJO
104.       11000     0000      0000      0         0         0000000000000000000 ALTO
105.       11000     0000      0000      0         0         0000000000000000000 BAJO
```

4. Diagrama RTL

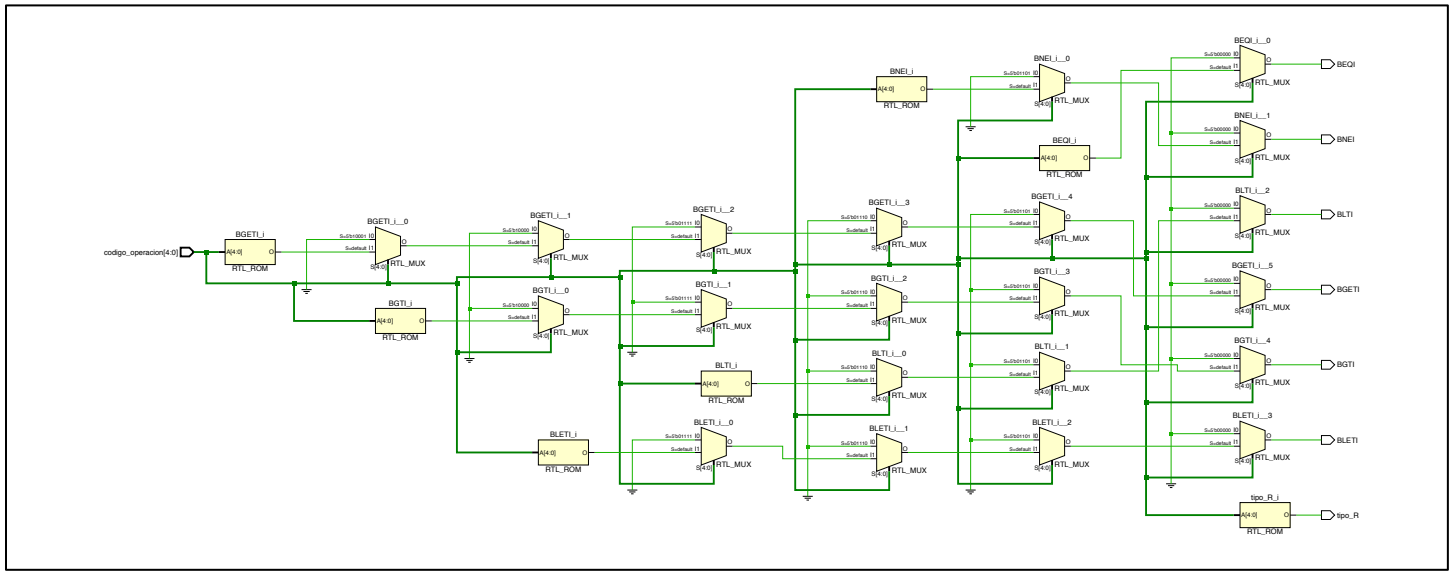
4.1. Memoria de código de función



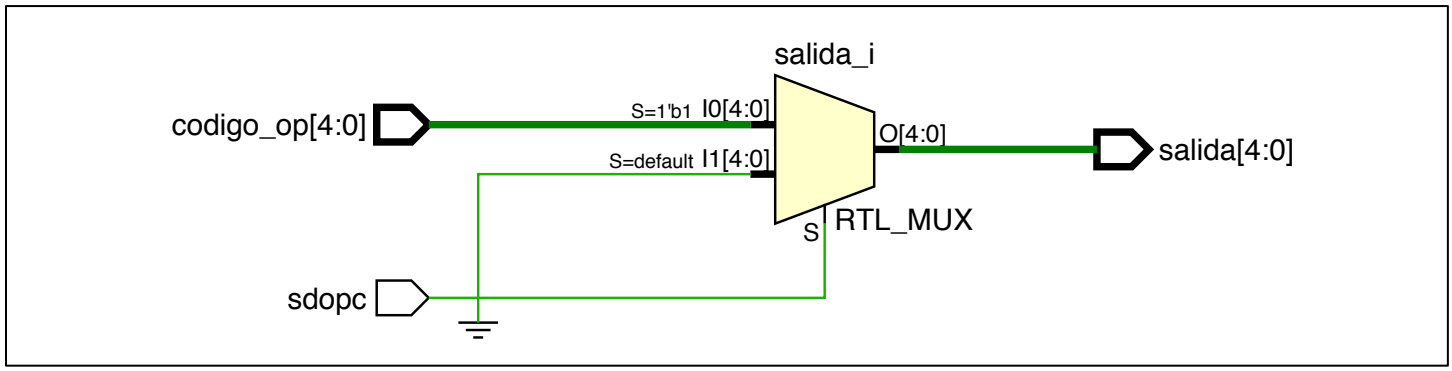
4.2. Memoria de código de operación

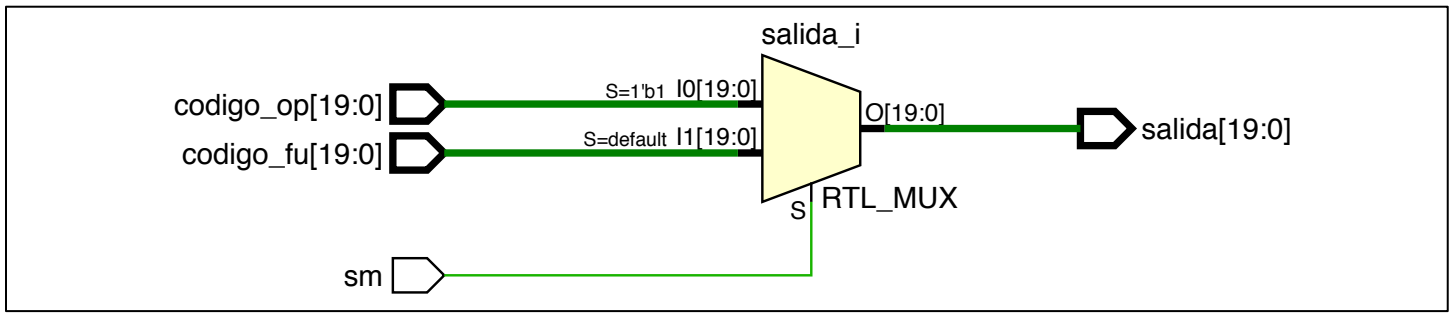


4.3. Decodificador

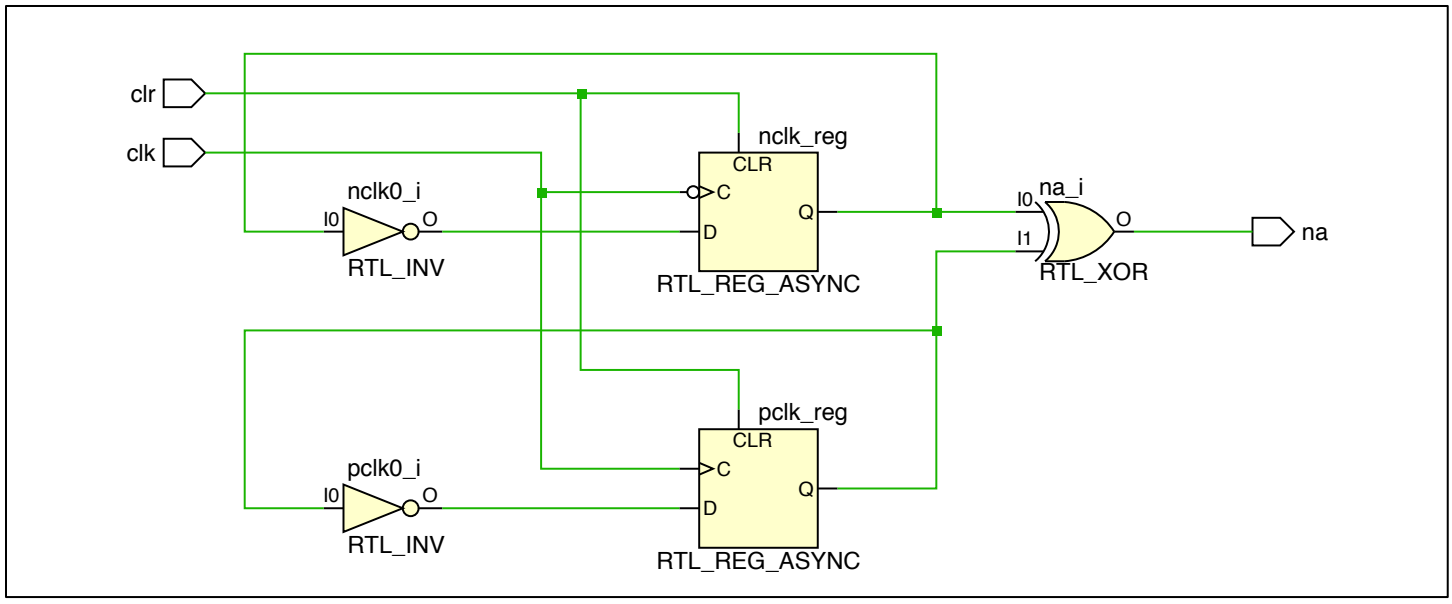


4.4. Multiplexores

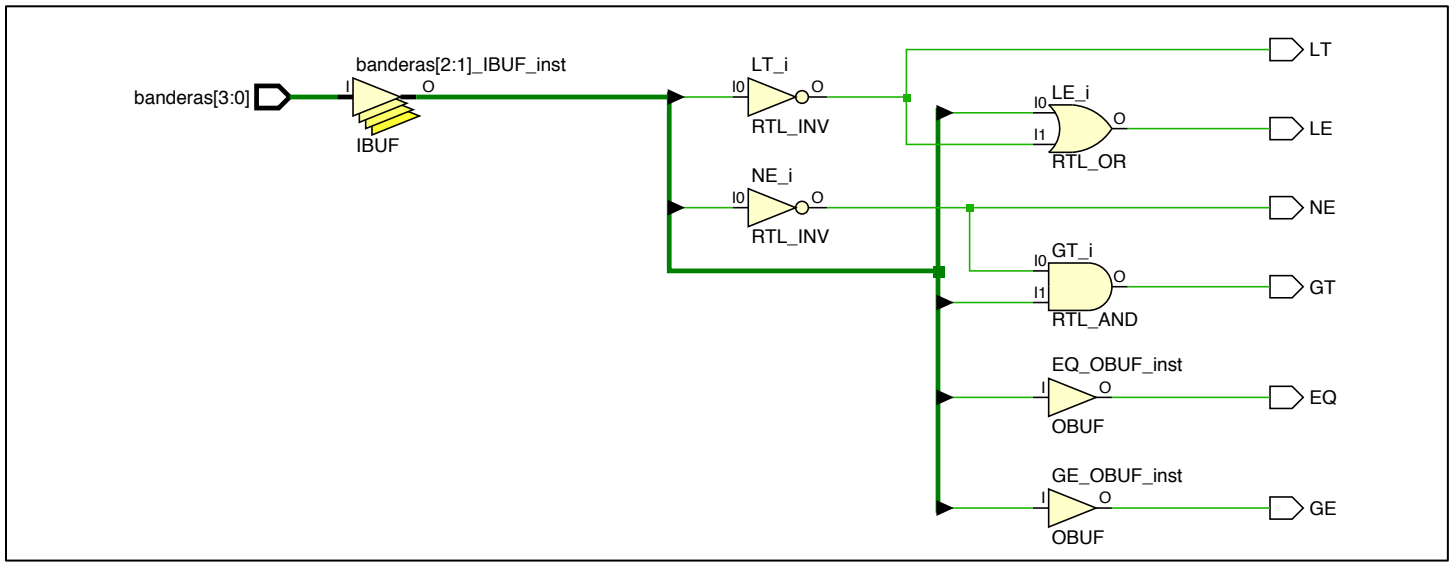




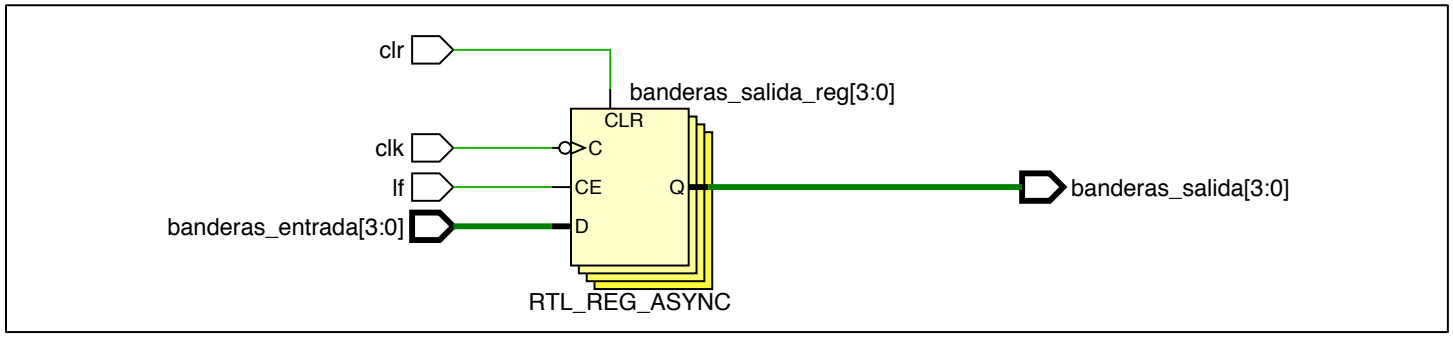
4.5. Nivel



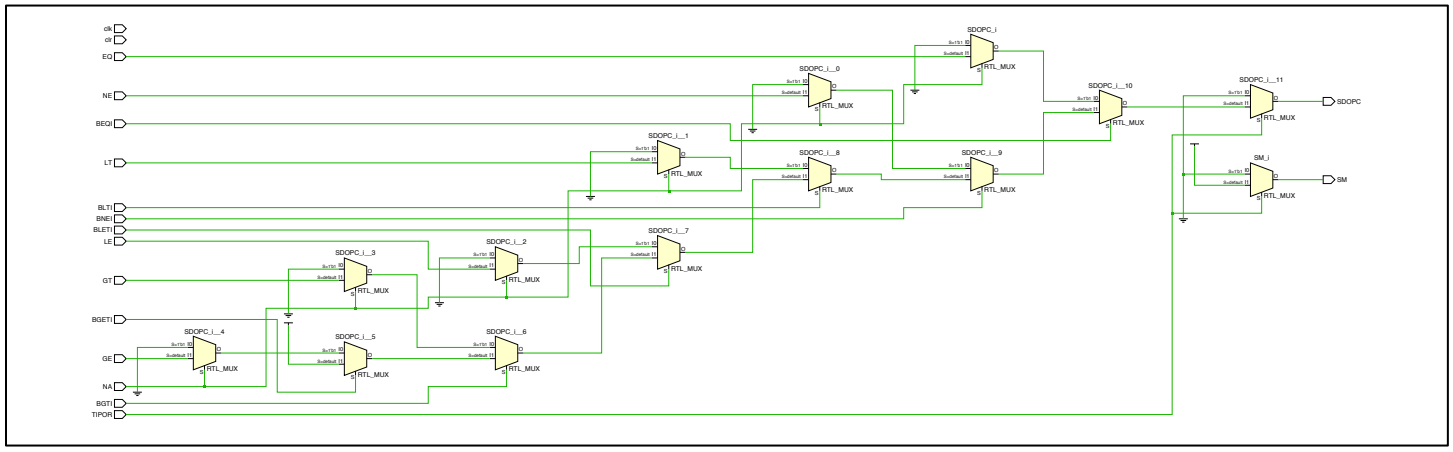
4.6. Condición



4.7. Registro



4.8. Unidad de control



4.9. Arquitectura completa