



Instituto Politécnico Nacional
Escuela Superior de Computo



Práctica 8.

Memorias del Procesador.

Nombre: Flores Castro Luis Antonio.

Arquitectura de Computadoras.

Profesora: Vega García Nayeli.

Calculo del tamaño de buses y direcciones de la Memoria de datos.

Se tiene

m x n = 4096 bytes.

Donde n se obtiene del set de instrucciones (literal de 16 bits)

n = tamaño de palabra = 16 bits

Ahora m = número de palabras

1 byte = 8 bits

(4096 bytes)(8 bits) = 32768

Despejando m, se tiene:

$$2^m \times n = 4096 \text{ bytes}$$

$$m = \log_2\left(\frac{32768}{n}\right)$$

$$m = \log_2\left(\frac{32768}{16}\right)$$

$$m = \log_2(2048)$$

m = 11 bits

Código VHDL de Memoria de Datos.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. use IEEE.STD_LOGIC_unsigned.ALL;
4. use IEEE.STD_LOGIC_arith.ALL;
5.
6. entity datamemory is
7.     generic(n: integer:=16;
8.             m: integer:=11);
9.     Port(dir:in STD_LOGIC_VECTOR(m-1 downto 0);
10.          dataIn: in STD_LOGIC_VECTOR(n-1 downto 0);
11.          WD, clk: STD_LOGIC;
12.          dataOut: out STD_LOGIC_VECTOR(n-1 downto 0));
13. end datamemory;
14.
15. architecture Behavioral of datamemory is
16.     type mem is array (0 to (2**m)-1) of std_logic_vector(n-
17. 1 downto 0);
18.     signal aux :mem;
19.
20.     begin
21.     process(clk)
22.     begin
23.         if(rising_edge(clk)) then
24.             if(WD='1') then
25.                 aux(conv_integer(dir))<=dataIn;
26.             end if;
27.         end if;
28.     end process;
29.     dataOut<=aux(conv_integer(dir));--lectura asincrona
30. end Behavioral;
```

Código VHDL Test-Bench de Memoria de Datos.

```
1. library IEEE;
2. library STD;
3. use STD.TEXTIO.ALL;
4. use IEEE.STD_LOGIC_TEXTIO.ALL;
5. use IEEE.STD_LOGIC_1164.ALL;
6. use IEEE.STD_LOGIC_UNSIGNED.ALL;
7. use IEEE.STD_LOGIC_ARITH.ALL;
8.
9. entity Tbdatemem is
10.     end tbdatemem;
11.
12.     architecture TB of Tbdatemem is
13.         component datamemory is
14.             Port(dir:in STD_LOGIC_VECTOR(10 downto 0);
15.                 dataIn: in STD_LOGIC_VECTOR(15 downto 0);
16.                 WD, clk: STD_LOGIC;
17.                 dataOut: out STD_LOGIC_VECTOR(15 downto 0));
18.         end component;
19.
20.         --signals
21.         signal dir: STD_LOGIC_VECTOR(10 downto 0);
22.         signal dataIn: STD_LOGIC_VECTOR(15 downto 0);
23.         signal WD: STD_LOGIC;
24.         signal clk: STD_LOGIC;
25.         signal dataOut: STD_LOGIC_VECTOR(15 downto 0);
26.
27.         begin
28.
29.         clock : process
30.             begin
31.                 clk<='0';
32.                 wait for 5 ns;
33.                 clk<='1';
34.                 wait for 5 ns;
35.             end process;
36.
37.         TestMD: datamemory
38.             Port map(
39.                 dir=>dir,
40.                 dataIn=>dataIn,
41.                 WD=>WD,
42.                 clk=>clk,
43.                 dataOut=>dataOut
44.             );
45.
46.
47.         estimulos: process
48.             file RES: TEXT;
```

```

49.         variable L_RE: line;
50.         variable vdo: STD_LOGIC_VECTOR(15 downto 0);--
    salida
51.
52.         file STIMU: TEXT;
53.         variable L_E: line;
54.         variable vdir: STD_LOGIC_VECTOR(10 downto 0);
55.         variable vdataIn: STD_LOGIC_VECTOR(15 downto 0);
56.         variable vWD: STD_LOGIC;
57.
58.         variable CADENA: STRING(1 to 7);
59.         begin
60.
61.             file_open(STIMU, "C:\Users\Luis
FC\Documents\Semestre 21-2\Arquitectura de
Computadoras\Memoria de Datos\ESTIMULOS.TXT", READ_MODE);
62.             file_open(RES, "C:\Users\Luis
FC\Documents\Semestre 21-2\Arquitectura de
Computadoras\Memoria de Datos\RESULTADO.TXT", WRITE_MODE);
63.
64.             CADENA:="add      ";
65.             write(L_RE,CADENA,left,1);
66.             CADENA:="WD      ";
67.             write(L_RE,CADENA,left,1);
68.             CADENA:="dataIn ";
69.             write(L_RE,CADENA,left,3);
70.             CADENA:="dataOut";
71.             write(L_RE,CADENA,left,3);
72.             writeline(RES,L_RE);
73.
74.             FOR i in 0 to 11 loop
75.                 readline(STIMU,L_E);
76.
77.                 hread(L_E,vdir);
78.                 dir<=vdir;
79.
80.                 read(L_E,vWD);
81.                 WD<=vWD;
82.
83.                 hread(L_E,vdataIn);
84.                 dataIn<=vdataIn;
85.
86.                 wait until rising_edge(clk);
87.
88.                 vdo:=dataOut;
89.
90.                 --escritura
91.                 hwrite(L_RE,vdir,left,7);
92.                 write(L_RE,vWD,left,7);
93.                 hwrite(L_RE,vdataIn,left,7);
94.                 hwrite(L_RE,vdo,left,7);
95.

```

```
96.          writeline(RES,L_RE);
97.
98.          end loop;
99.          file_close(STIMU);
100.         file_close(RES);
101.         wait;
102.         end process;
103.     end TB;
```

Diagrama lógico de la Memoria de Datos.

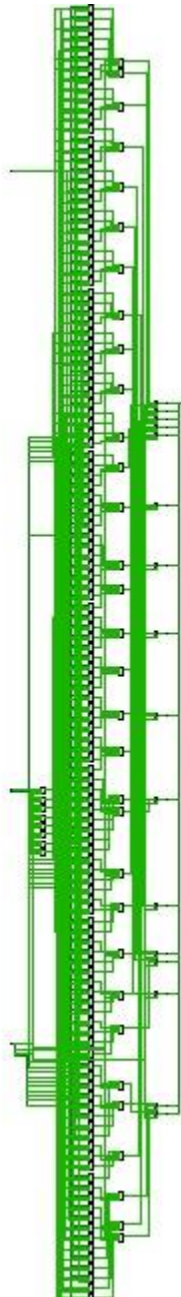


Figura 1. Diagrama lógico de la Memoria de Datos.

Diagrama RTL de la Memoria de Datos.

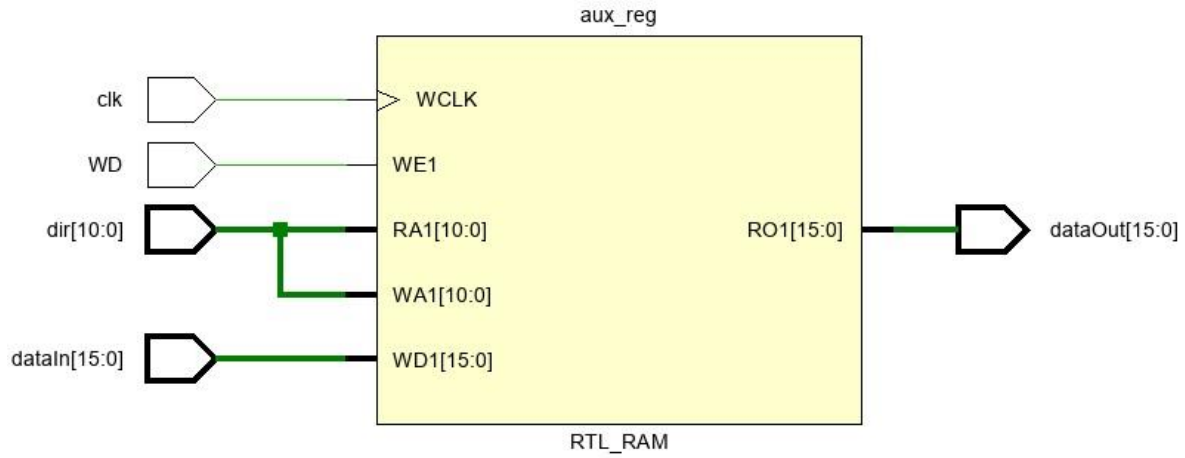


Figura 1.1 Diagrama RTL de la Memoria de Datos.

Simulación de Onda de la Memoria de Datos.

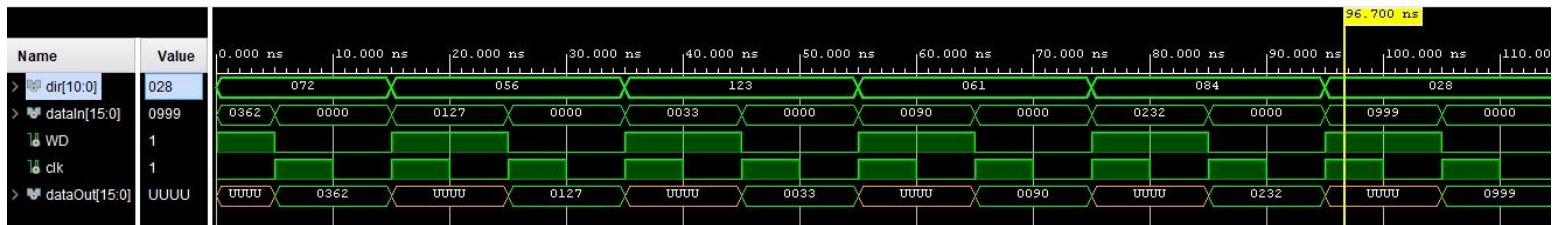


Figura 1.2 Diagrama de Onda de la Memoria de Datos.

Estímulos ingresados a la Memoria de Datos.

1. Escritura en la localidad x72 con el valor x362
2. Lectura de la localidad x72
3. Escritura en la localidad x56 con el valor x127
4. Lectura de la localidad x56
5. Escritura en la localidad x123 con el valor x33
6. Lectura de la localidad x123
7. Escritura en la localidad x61 con el valor x90
8. Lectura de la localidad x61
9. Escritura en la localidad x84 con el valor x232
10. Lectura de la localidad x84
11. Escritura en la localidad x28 con el valor x999
12. Lectura de la localidad x28


Archivo de estímulos de entrada de la Memoria de Datos.



Archivo	Edición	Formato	Ver	Ayuda
072	1	0362		
072	0	0000		
056	1	0127		
056	0	0000		
123	1	0033		
123	0	0000		
061	1	0090		
061	0	0000		
084	1	0232		
084	0	0000		
028	1	0999		
028	0	0000		

Figura 1.3 Archivo de Texto de estímulos de entrada de la Memoria de Datos.

Archivo de salida de la Memoria de Datos.

 *RESULTADO: Bloc de notas

Archivo	Edición	Formato	Ver	Ayuda
add	WD	dataIn	dataOut	
072	1	0362	XXXX	
072	0	0000	0362	
056	1	0127	XXXX	
056	0	0000	0127	
123	1	0033	XXXX	
123	0	0000	0033	
061	1	0090	XXXX	
061	0	0000	0090	
084	1	0232	XXXX	
084	0	0000	0232	
028	1	0999	XXXX	
028	0	0000	0999	

Figura 1.4 Archivo de resultado de la Memoria de Datos generado al realizar el Test Bench.

Calculo del tamaño de buses y direcciones de la Memoria de Programa.

Se tiene

m x n = 3200 bytes.

Donde n se obtiene del set de instrucciones

n = tamaño de palabra = 25 bits

Ahora m = número de palabras

1 byte = 8 bits

(3200 bytes)(8 bits) = 25600

Despejando m, se tiene:

$$2^m \times n = 3200 \text{ bytes}$$

$$m = \log_2\left(\frac{25600}{n}\right)$$

$$m = \log_2\left(\frac{25600}{25}\right)$$

$$m = \log_2(1024)$$

m = 10 bits

Código VHDL de Memoria de Programa.

```
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 dataprogram is
7.     generic ( m : integer := 10;
8.               n : integer := 25);
9.     Port ( PC : in STD_LOGIC_VECTOR (m-1 downto 0);
10.           ints : out STD_LOGIC_VECTOR (n-1 downto 0));
11. end dataprogram;
12.
13. architecture Behavioral of dataprogram is
14.     --Instrucciones tipo R
15.     constant TR : STD_LOGIC_VECTOR(4 downto 0) := "00000";
16.     --Instrucciones de Carga y Almacenamiento
17.     constant LI:STD_LOGIC_VECTOR(4 downto 0) := "00001";
18.     constant LWI:STD_LOGIC_VECTOR(4 downto 0) := "00010";
19.     constant LW:STD_LOGIC_VECTOR(4 downto 0) := "10111";
20.     constant SWI:STD_LOGIC_VECTOR(4 downto 0) := "00011";
21.     constant SW:STD_LOGIC_VECTOR(4 downto 0) := "00100";
22.
23.     --Instrucciones Aritmeticas
24.     constant ADD:STD_LOGIC_VECTOR(3 downto 0) := "0000";--
25.     constant SUB:STD_LOGIC_VECTOR(3 downto 0) := "0001";--
26.     constant ADDI:STD_LOGIC_VECTOR(4 downto 0) := "00101";
27.     constant SUBI:STD_LOGIC_VECTOR(4 downto 0) := "00110";
28.
29.     --Instrucciones Lógicas
30.     constant OPAND:STD_LOGIC_VECTOR(3 downto 0) := "0010";
31.     constant OPOR:STD_LOGIC_VECTOR(3 downto 0) := "0011";
32.     constant OPXOR:STD_LOGIC_VECTOR(3 downto 0) := "0100";
33.     constant OPNAND:STD_LOGIC_VECTOR(3 downto 0) := "0101"
34.     ;
35.     constant OPNOR:STD_LOGIC_VECTOR(3 downto 0) := "0110";
36.     constant OPXNOR:STD_LOGIC_VECTOR(3 downto 0) := "0111"
37.     ;
38.     constant OPNOT:STD_LOGIC_VECTOR(3 downto 0) := "1000";
39.     constant ANDI:STD_LOGIC_VECTOR(4 downto 0) := "00111";
40.     constant ORI:STD_LOGIC_VECTOR(4 downto 0) := "01000";
41.     constant XORI:STD_LOGIC_VECTOR(4 downto 0) := "01001";
42.     constant NANDI:STD_LOGIC_VECTOR(4 downto 0) := "01010"
43.     ;
44.     constant NORI:STD_LOGIC_VECTOR(4 downto 0) := "01011";
45.     constant XNORI:STD_LOGIC_VECTOR(4 downto 0) := "01100"
46.     ;
47. end architecture;
```

```

44.      --Instrucciones de Corrimiento
45.      constant OPSLL:STD_LOGIC_VECTOR(3 downto 0):="1001";
46.      constant OPSRL:STD_LOGIC_VECTOR(3 downto 0):="1010";
47.      --constant OPR:STD_LOGIC_VECTOR(4 downto
    0):="00000";
48.
49.      --Instrucciones de Saltos Condicionales e
    Incondicionales
50.      constant BEQI:STD_LOGIC_VECTOR(4 downto 0):="01101";
51.      constant BNEI:STD_LOGIC_VECTOR(4 downto 0):="01110";
52.      constant BLTI:STD_LOGIC_VECTOR(4 downto 0):="01111";
53.      constant BLETI:STD_LOGIC_VECTOR(4 downto 0):="10000"
    ;
54.      constant BGTI:STD_LOGIC_VECTOR(4 downto 0):="10001";
55.      constant BGETI:STD_LOGIC_VECTOR(4 downto 0):="10010"
    ;
56.      constant B:STD_LOGIC_VECTOR(4 downto 0):="10011";
57.
58.      --Instrucciones de Manejo de Subrutinas
59.      constant CALL:STD_LOGIC_VECTOR(4 downto 0):="10100";
60.      constant RET:STD_LOGIC_VECTOR(4 downto 0):="10101";
61.
62.      --Otras Instrucciones
63.      constant NOP:STD_LOGIC_VECTOR(4 downto 0):="10110";
64.      constant SU:STD_LOGIC_VECTOR(3 downto 0):="0000"; --
    Sin usar 4 bits
65.
66.      --Registros
67.      constant R0 : STD_LOGIC_VECTOR (3 downto 0) := "0000
    ";
68.      constant R1 : STD_LOGIC_VECTOR (3 downto 0) := "0001
    ";
69.      constant R2 : STD_LOGIC_VECTOR (3 downto 0) := "0010
    ";
70.      constant R3 : STD_LOGIC_VECTOR (3 downto 0) := "0011
    ";
71.      constant R4 : STD_LOGIC_VECTOR (3 downto 0) := "0100
    ";
72.      constant R5 : STD_LOGIC_VECTOR (3 downto 0) := "0101
    ";
73.      constant R6 : STD_LOGIC_VECTOR (3 downto 0) := "0110
    ";
74.      constant R7 : STD_LOGIC_VECTOR (3 downto 0) := "0111
    ";
75.      constant R8 : STD_LOGIC_VECTOR (3 downto 0) := "1000
    ";
76.
77.      type mem is array (0 to (2**m)-
    1) of STD_LOGIC_VECTOR(n-1 downto 0);
78.      constant aux : mem := (--aquí nace la memoria
79.          LI & R0 & x"0000",--0
80.          LI & R1 & x"0001",--1

```

```

81.          LI & R2 & x"0000",--2
82.          LI & R3 & x"000c",--3
83.          TR & R4 & R0 & R1 & SU & ADD,--4
84.          SWI & R4 & x"0048",--5
85.          ADDI & R0 & R1 & x"000",--6
86.          ADDI & R1 & R4 & x"000",--7
87.          ADDI & R2 & R2 & x"001",--8
88.          BNEI & R3 & R2 & x"ffb",--9    CONDICION SALTO A
LINEA 4  "11111111011"    PC=9+(-5)=4
89.          NOP & SU & SU & SU & SU & SU,--10
90.          B & SU & x"000a",--11    SALTO A FIN
91.          others => (others => '0')
92.          );
93.  begin
94.      ints <= aux(conv_integer(PC));
95.  end Behavioral;

```

Código VHDL Test-Bench de Memoria de Programa.

```
1. library ieee;
2. library STD;
3. use ieee.STD_LOGIC_1164.ALL;
4. use ieee.STD_LOGIC_arith.all;
5. use ieee.STD_LOGIC_unsigned.ALL;
6. use ieee.STD_LOGIC_TEXTIO.ALL;
7. use STD.TEXTIO.ALL;
8.
9. entity Tbprogramem is
10.     end Tbprogramem;
11.
12.     architecture Behavioral of Tbprogramem is
13.         component dataprogram is
14.             Port ( PC : in STD_LOGIC_VECTOR (9 downto 0);
15.                 ints : out STD_LOGIC_VECTOR (24 downto 0)
16.             );
17.         end component;
18.         signal PC : STD_LOGIC_VECTOR (9 downto 0) := "000000
19.         0000";
20.         signal ints : STD_LOGIC_VECTOR (24 downto 0);
21.     begin
22.         mp: dataprogram Port map (
23.             PC => PC,
24.             ints => ints
25.         );
26.         process
27.             file RES: TEXT;
28.
29.             variable L_RE : line;
30.
31.             variable vints : STD_LOGIC_VECTOR (24 downto 0);
32.             variable CADENA : STRING(1 to 6);
33.         begin
34.             file_open(RES, "C:\Users\Luis
35.             FC\Documents\Semestre 21-2\Arquitectura de
36.             Computadoras\Memoria de Programa\RESULTADO.txt",
37.             WRITE_MODE);
38.
39.             CADENA:= "PC      ";
40.             write(L_RE, CADENA, left, 9);
41.             CADENA:= "OPCODE";
42.             write(L_RE, CADENA, left, 7);
43.             CADENA:= "19..16";
44.             write(L_RE, CADENA, left, 8);
45.             CADENA:= "15..12";
46.             write(L_RE, CADENA, left, 9);
47.             CADENA:= "11..8  ";
```

```

44.         write(L_RE, CADENA, left, 8);
45.         CADENA:= "7..4 ";
46.         write(L_RE, CADENA, left, 7);
47.         CADENA:= "3..0 ";
48.         write(L_RE, CADENA, left, 8);
49.         writeline(RES, L_RE);
50.
51.         for i in 0 to 11 loop
52.             wait for 10 ns;
53.             vints := ints;
54.
55.             Hwrite(L_RE, PC, left, 9);
56.             write(L_RE, vints(24 downto 20), left, 8);
57.             write(L_RE, vints(19 downto 16), left, 8);
58.             write(L_RE, vints(15 downto 12), left, 8);
59.             write(L_RE, vints(11 downto 8), left, 8);
60.             write(L_RE, vints(7 downto 4), left, 8);
61.             write(L_RE, vints(3 downto 0), left, 8);
62.             writeline(RES, L_RE);
63.             PC <= PC + 1;
64.         end loop;
65.         file_close(RES);
66.         wait;
67.     end process;
68. end Behavioral;

```


Diagrama lógico de la Memoria de Programa.

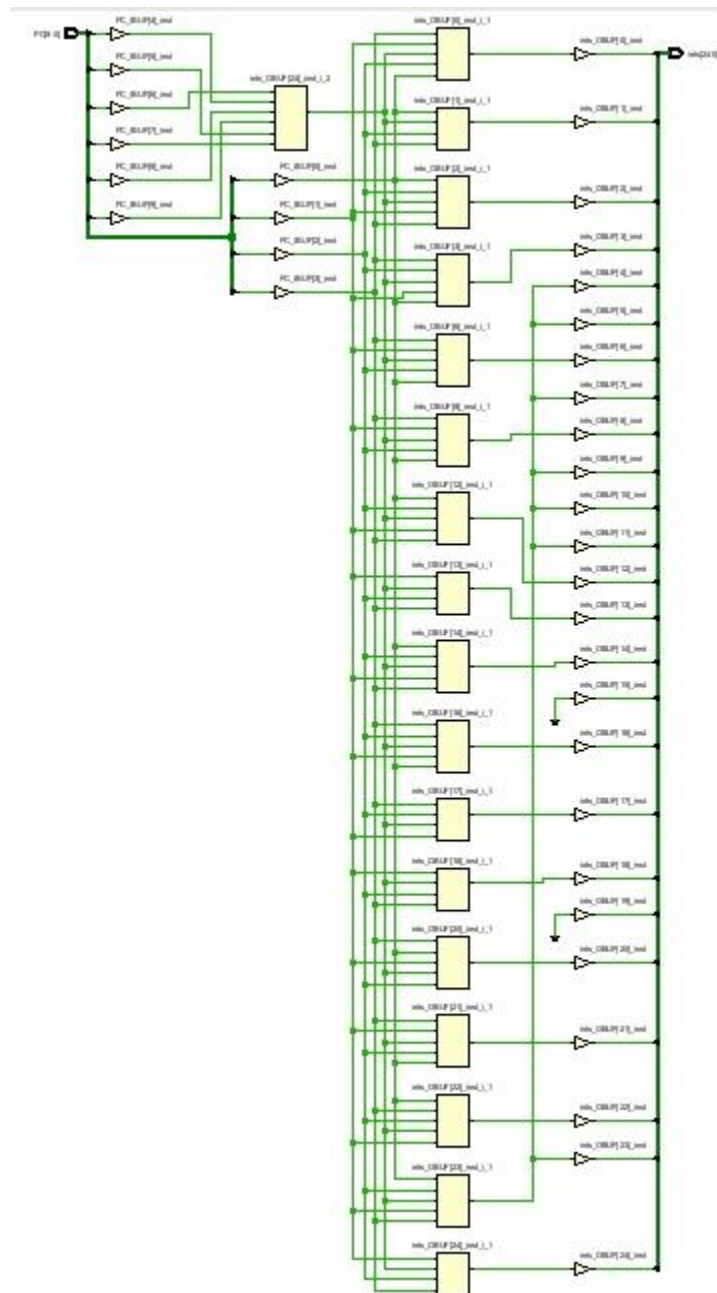


Figura 2 Diagrama lógico de la Memoria de Programa.

Diagrama RTL de la Memoria de Programa.

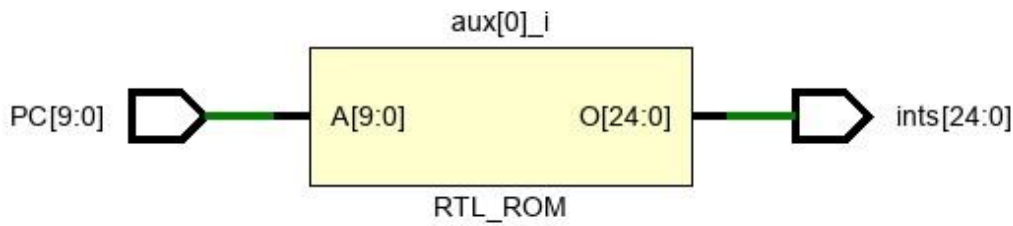


Figura 2.1 Diagrama RTL de la Memoria de Programa.

Simulación de Onda de la Memoria de Programa.

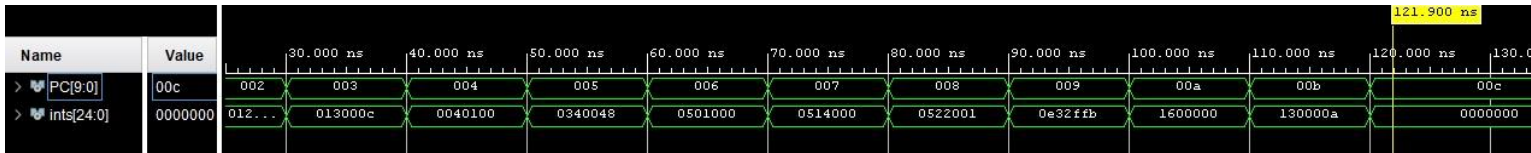


Figura 2.2 Diagrama de Onda de la Memoria de Programa.

Inicialización de la Memoria de Programa.

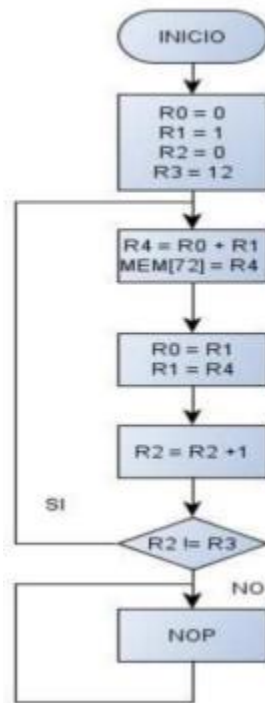



Figura 2.3 Programa que nace con la Memoria de Programa.

Implementación del programa dentro del código de implementación de la Memoria de Programa.

```
1. LI & R0 & x"0000",--0
2. LI & R1 & x"0001",--1
3. LI & R2 & x"0000",--2
4. LI & R3 & x"000c",--3
5. TR & R4 & R0 & R1 & SU & ADD,--4
6. SWI & R4 & x"0048",--5
7. ADDI & R0 & R1 & x"000",--6
8. ADDI & R1 & R4 & x"000",--7
9. ADDI & R2 & R2 & x"001",--8
10. BNEI & R3 & R2 & x"ffb",--9    CONDICION SALTO A LINEA
    4  "11111111011"    PC=9+(-5)=4
11. NOP & SU & SU & SU & SU,--10
12. B & SU & x"000a",--11    SALTO A FIN
```

Figura 2.4 Instrucciones del programa.

Archivo de Salida de la Memoria de Programa.

 *RESULTADO: Bloc de notas

Archivo	Edición	Formato	Ver	Ayuda		
PC	OPCODE	19..16	15..12	11..8	7..4	3..0
000	00001	0000	0000	0000	0000	0000
001	00001	0001	0000	0000	0000	0001
002	00001	0010	0000	0000	0000	0000
003	00001	0011	0000	0000	0000	1100
004	00000	0100	0000	0001	0000	0000
005	00011	0100	0000	0000	0100	1000
006	00101	0000	0001	0000	0000	0000
007	00101	0001	0100	0000	0000	0000
008	00101	0010	0010	0000	0000	0001
009	01110	0011	0010	1111	1111	1011
00A	10110	0000	0000	0000	0000	0000
00B	10011	0000	0000	0000	0000	1010

Figura 2.4 Archivo de Salida de la Memoria de Programa generado al realizar el Test Bench.