Instituto Politécnico Nacional

Escuela Superior de Cómputo

Proyecto ESCOMIPS Parte 1

Unidad de aprendizaje: Arquitectura de Computadoras

Grupo: 3CV1

Alumno(a):
Ramos Diaz Enrique

Profesor(a):
Vega García Nayeli

3 de julio 2020

Índice

1 Código de implementación **2**
1.1 Extensor de Signo 2
1.2 Extensor de Dirección 2
1.3 Procesador ESCOMips 3

2 Diagrama RTL **9**

3 Programa Punto 2: Código Memoria de Programa **10**
3.1 Programa: Suma de dos numeros 10
3.2 Memoria de Programa 10

4 Simulación Programa Punto 2 . . . . . **12**

5 Tabla Ejecución Programa Punto 2 . . . . . **13**

6 Programa Punto 5: Código Memoria de Programa **14**
6.1 Programa: Multiplicación de dos números usando sucesión de sumas 14
6.2 Memoria de Programa 14

7 Simulación Programa Punto 5 . . . . . **16**

8 Tabla Ejecución Programa Punto 5 . . . . . **17**

1. Código de implementación

1.1. Extensor de Signo

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

entity ExtensorSigno is
     Port ( Slit_in : in STD_LOGIC_VECTOR (11 downto 0);
            Slit_out : out STD_LOGIC_VECTOR (15 downto 0));
end ExtensorSigno;

architecture Behavioral of ExtensorSigno is
    signal lit : STD_LOGIC_VECTOR (15 downto 0);
    signal signo : STD_LOGIC;
begin
    signo <= Slit_in(11);
    lit(15) <= signo;
    lit(14) <= signo;
    lit(13) <= signo;
    lit(12) <= signo;
    lit(11 downto 0) <= Slit_in;
    Slit_out <= lit;
end Behavioral;
```

1.2. Extensor de Dirección

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

entity ExtensorDireccion is
     Port ( lit_in : in STD_LOGIC_VECTOR (11 downto 0);
            lit_out : out STD_LOGIC_VECTOR (15 downto 0));
end ExtensorDireccion;

architecture Behavioral of ExtensorDireccion is
    signal lit : STD_LOGIC_VECTOR (15 downto 0);
    constant dir : STD_LOGIC_VECTOR (3 downto 0) := "0000";
begin
    lit(11 downto 0) <= lit_in;
    lit(15 downto 12) <= dir;
    lit_out <= lit;
end Behavioral;
```

1.3. Procesador ESCOMips

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity ESCOMips is
5      Port ( rclr, clk : in STD_LOGIC;
6             pc : out STD_LOGIC_VECTOR (9 downto 0);
7             instruccion : out STD_LOGIC_VECTOR (24 downto 0);
8             RD1, RD2, resALU, busSR : out STD_LOGIC_VECTOR (15 downto 0));
9  end ESCOMips;
10
11 architecture Behavioral of ESCOMips is
12     -- Pila
13     component Pila is
14         Port ( PCin : in STD_LOGIC_VECTOR (15 downto 0);
15                clk, clr, wpc, up, dw : in STD_LOGIC;
16                PCout : out STD_LOGIC_VECTOR (15 downto 0));
17     end component;
18
19     --MemPrograma
20     component MemoriaPrograma1 is
21         Port ( pc : in STD_LOGIC_VECTOR (9 downto 0);
22                inst : out STD_LOGIC_VECTOR (24 downto 0));
23     end component;
24
25     -- Unidad Control
26     component UnidadControl is
27         Port ( clk, clr : in STD_LOGIC;
28                opCode : in STD_LOGIC_VECTOR (4 downto 0);
29                funCode, banderas : in STD_LOGIC_VECTOR (3 downto 0);
30                microInstruccion : out STD_LOGIC_VECTOR (19 downto 0));
31     end component;
32
33     --ArchivoRegistro
34     component ArchivoRegistros is
35         Port ( wr, dir, she, clk, clr : in STD_LOGIC;
36                writeReg, readReg1, readReg2, shamt : in STD_LOGIC_VECTOR (3
                   ↪  downto 0);
37                writeData : in STD_LOGIC_VECTOR (15 downto 0);
38                readData1, readData2 : out STD_LOGIC_VECTOR (15 downto 0));
39     end component;
40
41     --ALU
```

```
42      component ALUNBits is
43          Port ( a, b : in STD_LOGIC_VECTOR (15 downto 0);
44                 aluop : in STD_LOGIC_VECTOR (3 downto 0);
45                 res : out STD_LOGIC_VECTOR (15 downto 0);
46                 banderas : out STD_LOGIC_VECTOR (3 downto 0));
47      end component;
48
49      --MemDatos
50      component MemoriaDatos is
51          Port ( add : in STD_LOGIC_VECTOR (9 downto 0);
52                 dataIn : in STD_LOGIC_VECTOR (15 downto 0);
53                 clk, wd : in STD_LOGIC;
54                 dataOut : out STD_LOGIC_VECTOR (15 downto 0));
55      end component;
56
57      --ExtensorDireccion
58      component ExtensorDireccion is
59           Port ( lit_in : in STD_LOGIC_VECTOR (11 downto 0);
60                  lit_out : out STD_LOGIC_VECTOR (15 downto 0));
61      end component;
62
63      --ExtensorSigno
64      component ExtensorSigno is
65           Port ( Slit_in : in STD_LOGIC_VECTOR (11 downto 0);
66                  Slit_out : out STD_LOGIC_VECTOR (15 downto 0));
67      end component;
68
69      --Mux16Bits
70      component Mux16Bits is
71          Port ( a, b : in STD_LOGIC_VECTOR (15 downto 0);
72                 control : in STD_LOGIC;
73                 salida : out STD_LOGIC_VECTOR (15 downto 0));
74      end component;
75
76      --SR2
77      component MuxSR2 is
78          Port ( a, b : in STD_LOGIC_VECTOR (3 downto 0);
79                 control : in STD_LOGIC;
80                 salida : out STD_LOGIC_VECTOR (3 downto 0));
81      end component;
82
83      signal auxSDMP, auxSWD, auxSEXT, auxSOP1, auxSOP2, auxSDMD, auxSR,
        ↪  auxPCout : STD_LOGIC_VECTOR (15 downto 0) := (others => '0');
84      signal auxInst : STD_LOGIC_VECTOR (24 downto 0) := (others => '0');
```

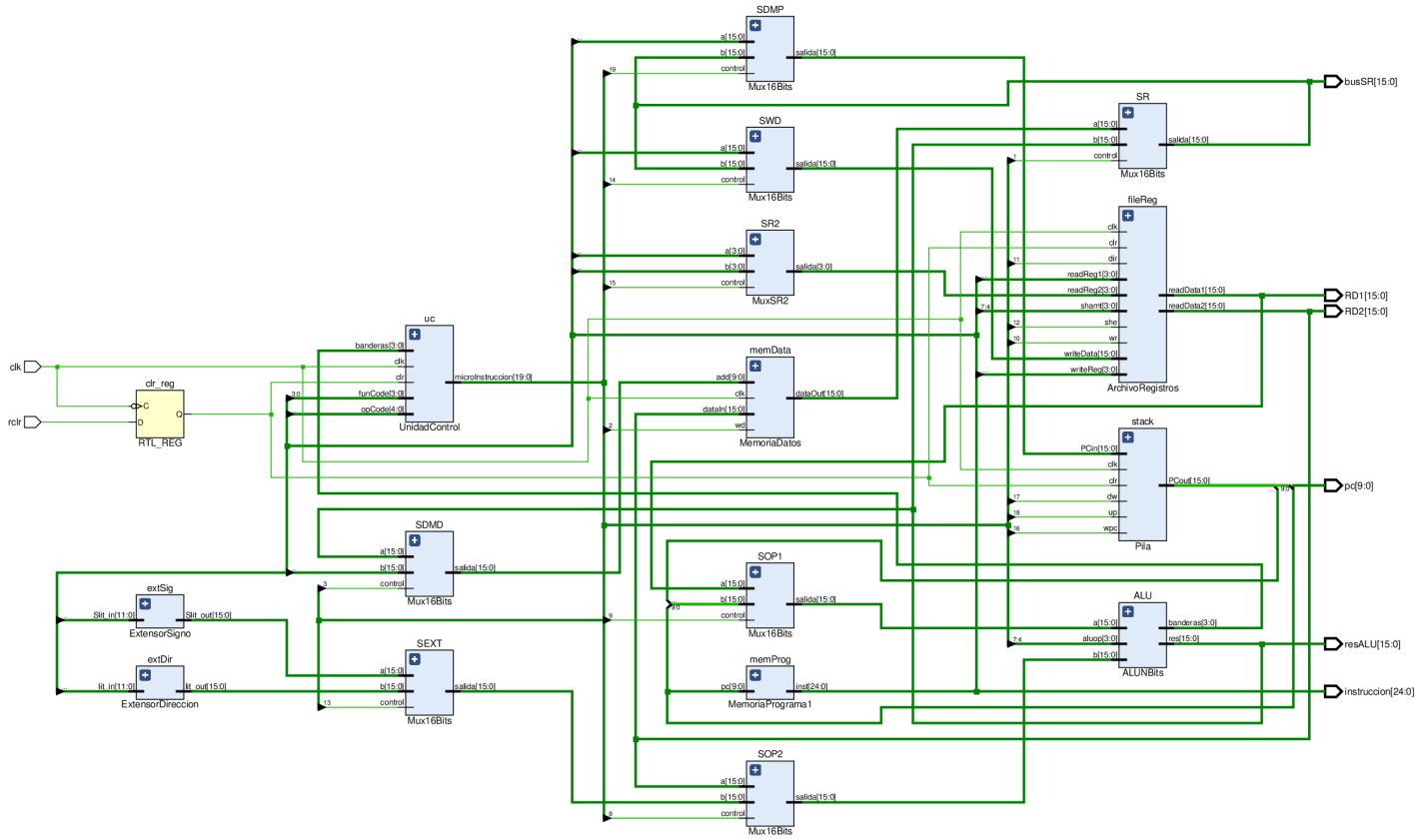
```
85      signal clr : STD_LOGIC := '0';
86      signal auxBanderas, auxSR2 : STD_LOGIC_VECTOR (3 downto 0) := (others =>
        ↪  '0');
87      signal auxMicroInstruccion : STD_LOGIC_VECTOR (19 downto 0) := (others
        ↪  => '0');
88      signal auxReadData1, auxReadData2 : STD_LOGIC_VECTOR (15 downto 0) :=
        ↪  (others => '0');
89      signal auxExtDir, auxExtSig, auxResALU, auxDataOut : STD_LOGIC_VECTOR
        ↪  (15 downto 0) := (others => '0');
90  begin
91      process(clk)
92      begin
93          if(falling_edge(clk)) then
94              clr <= rclr;
95          end if;
96      end process;
97
98      --MICROINSTRUCCION
99      --  19 18 17  16  15  14  13   12  11 10  9    8    7        6         5
        ↪  4      3   2  1  0
100     --SDMP UP DW WPC SR2 SWD SEXT SHE DIR WR SOP1 SOP2 ALUOP[3] ALUOP[2]
        ↪  ALUOP[1] ALOP[0] SDMD WD SR LF
101     stack : Pila Port map (
102         PCin => auxSDMP,
103         clk => clk,
104         clr => clr,
105         wpc => auxMicroInstruccion(16),
106         up => auxMicroInstruccion(18),
107         dw => auxMicroInstruccion(17),
108         PCout => auxPCout
109     );
110
111     memProg : MemoriaPrograma1 Port map (
112         pc => auxPCout(9 downto 0),
113         inst => auxInst
114     );
115
116     -- Unidad Control
117     uc : UnidadControl Port map (
118         clk => clk,
119         clr => clr,
120         opCode => auxInst(24 downto 20),
121         funCode => auxInst(3 downto 0),
122         banderas => auxBanderas,
```

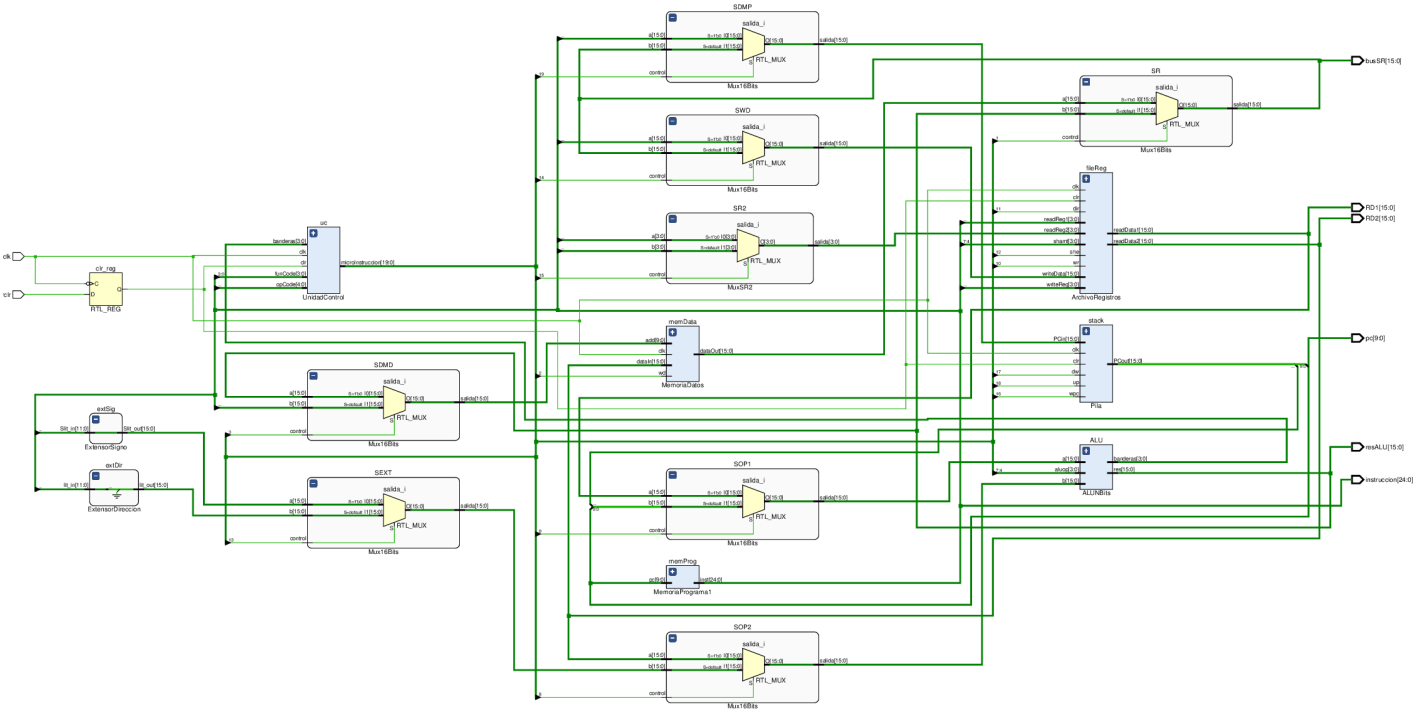
```
123         microInstruccion => auxMicroInstruccion
124     );
125 
126     --SR2
127     SR2 : MuxSR2 Port map (
128         a => auxInst(11 downto 8),
129         b => auxInst(19 downto 16),
130         control => auxMicroInstruccion(15),
131         salida => auxSR2
132     );
133 
134     --SWD
135     SWD : Mux16Bits Port map (
136         a => auxInst(15 downto 0),
137         b => auxSR,
138         control => auxMicroInstruccion(14),
139         salida => auxSWD
140     );
141 
142     --Archivo de Registros
143     fileReg : ArchivoRegistros Port map (
144         wr => auxMicroInstruccion(10),
145         dir => auxMicroInstruccion(11),
146         she => auxMicroInstruccion(12),
147         clk => clk,
148         clr => clr,
149         writeReg => auxInst(19 downto 16),
150         readReg1 => auxInst(15 downto 12),
151         readReg2 => auxSR2,
152         shamt => auxInst(7 downto 4),
153         writeData => auxSWD,
154         readData1 => auxReadData1,
155         readData2 => auxReadData2
156     );
157 
158     --ExtensorSigno
159     extSig : ExtensorSigno Port map (
160         Slit_in => auxInst(11 downto 0),
161         Slit_out => auxExtSig
162     );
163 
164     --ExtensorDireccion
165     extDir : ExtensorDireccion Port map (
166         lit_in => auxInst(11 downto 0),
```

```
167         lit_out => auxExtDir
168     );
169 
170     --SEXT
171     SEXT : Mux16Bits Port map (
172         a => auxExtSig,
173         b => auxExtDir,
174         control => auxMicroInstruccion(13),
175         salida => auxSEXT
176     );
177 
178     --SOP1
179     SOP1 : Mux16Bits Port map (
180         a => auxReadData1,
181         b => auxPCOut,
182         control => auxMicroInstruccion(9),
183         salida => auxSOP1
184     );
185 
186     --SOP2
187     SOP2 : Mux16Bits Port map (
188         a => auxReadData2,
189         b => auxSEXT,
190         control => auxMicroInstruccion(8),
191         salida => auxSOP2
192     );
193 
194     --ALU
195     ALU : ALUNBits Port map (
196         a => auxSOP1,
197         b => auxSOP2,
198         aluop => auxMicroInstruccion(7 downto 4),
199         res => auxResALU,
200         banderas => auxBanderas
201     );
202 
203     --SDMD
204     SDMD : Mux16Bits Port map (
205         a => auxResALU,
206         b => auxInst(15 downto 0),
207         control => auxMicroInstruccion(3),
208         salida => auxSDMD
209     );
210 
```

```
211     --MemDatos
212     memData : MemoriaDatos Port map (
213         add => auxSDMD(9 downto 0),
214         dataIn => auxReadData2,
215         clk => clk,
216         wd => auxMicroInstruccion(2),
217         dataOut => auxDataOut
218     );
219 
220     --SR
221     SR : Mux16Bits Port map (
222         a => auxDataOut,
223         b => auxResALU,
224         control => auxMicroInstruccion(1),
225         salida => auxSR
226     );
227 
228     --SDMP
229     SDMP : Mux16Bits Port map (
230         a => auxInst(15 downto 0),
231         b => auxSR,
232         control => auxMicroInstruccion(19),
233         salida => auxSDMP
234     );
235 
236     pc <= auxPCout(9 downto 0);
237     instruccion <= auxInst;
238     RD1 <= auxReadData1;
239     RD2 <= auxReadData2;
240     resALU <= auxResALU;
241     busSR <= auxSR;
242 end Behavioral;
```

2. Diagrama RTL





3. Programa Punto 2: Código Memoria de Programa

3.1. Programa: Suma de dos numeros

```
      LI  R0, #1
      LI  R1, #7
suma: ADD R1, R1, R0
      SWI R1, 0x05
      B suma
```

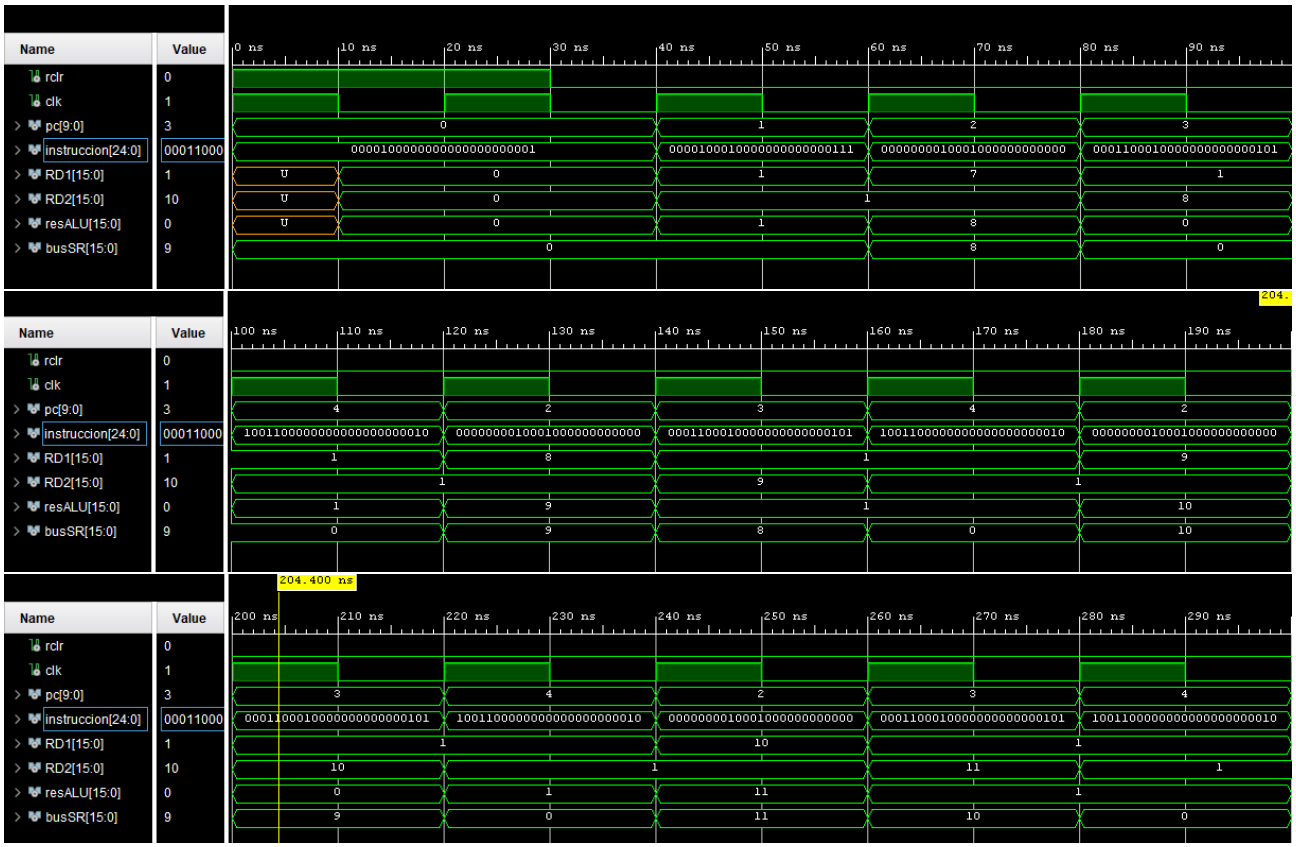
3.2. Memoria de Programa

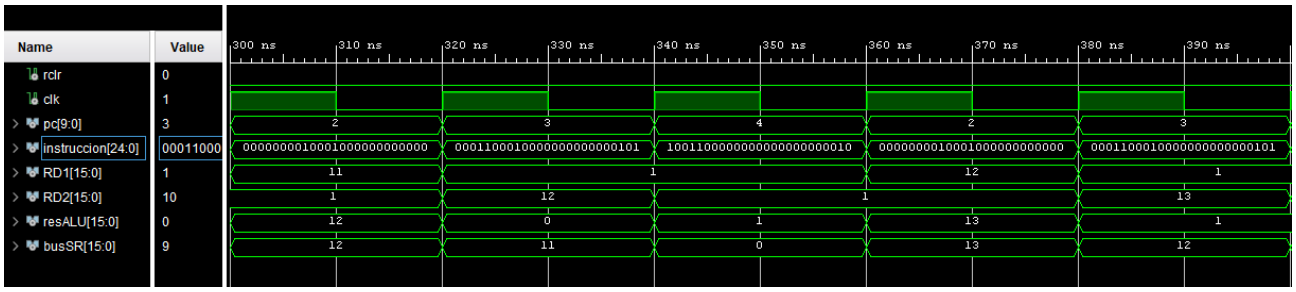
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity MemoriaPrograma1 is
    generic ( m : integer := 10;
              n : integer := 25);
    Port ( pc : in STD_LOGIC_VECTOR (m-1 downto 0);
           inst : out STD_LOGIC_VECTOR (n-1 downto 0));
end MemoriaPrograma1;

architecture Behavioral of MemoriaPrograma1 is
    --Instrucciones tipo R
    constant tipoR : STD_LOGIC_VECTOR (4 downto 0) := "00000";
    --Carga y Almacenamiento
    constant LI : STD_LOGIC_VECTOR (4 downto 0) := "00001";
    constant LWI : STD_LOGIC_VECTOR (4 downto 0) := "00010";
    constant LW : STD_LOGIC_VECTOR (4 downto 0) := "10111";
    constant SWI : STD_LOGIC_VECTOR (4 downto 0) := "00011";
    constant SW : STD_LOGIC_VECTOR (4 downto 0) := "00100";
    --Aritmticas
    constant ADDI : STD_LOGIC_VECTOR (4 downto 0) := "00101";
    constant SUBI : STD_LOGIC_VECTOR (4 downto 0) := "00110";
    --Identificador Aritmticas R
    constant ADD : STD_LOGIC_VECTOR (3 downto 0) := "0000";
    constant SUB : STD_LOGIC_VECTOR (3 downto 0) := "0001";
    --Logicas
    constant ANDI : STD_LOGIC_VECTOR (4 downto 0) := "00111";
    constant ORI : STD_LOGIC_VECTOR (4 downto 0) := "01000";
    constant XORI : STD_LOGIC_VECTOR (4 downto 0) := "01001";
```

```
32      constant NANDI : STD_LOGIC_VECTOR (4 downto 0) := "01010";
33      constant NORI : STD_LOGIC_VECTOR (4 downto 0) := "01011";
34      constant XNORI : STD_LOGIC_VECTOR (4 downto 0) := "01100";
35      --Identificador Logicas R
36      constant ANDR : STD_LOGIC_VECTOR (3 downto 0) := "0010";
37      constant ORR : STD_LOGIC_VECTOR (3 downto 0) := "0011";
38      constant XORR : STD_LOGIC_VECTOR (3 downto 0) := "0100";
39      constant NANDR : STD_LOGIC_VECTOR (3 downto 0) := "0101";
40      constant NORR : STD_LOGIC_VECTOR (3 downto 0) := "0110";
41      constant XNORR : STD_LOGIC_VECTOR (3 downto 0) := "0111";
42      constant NOTR : STD_LOGIC_VECTOR (3 downto 0) := "1000";
43      --Identificador Corrimiento R
44      constant SLLR : STD_LOGIC_VECTOR (3 downto 0) := "1001";
45      constant SRLR : STD_LOGIC_VECTOR (3 downto 0) := "1010";
46      constant OPR : STD_LOGIC_VECTOR (4 downto 0) := "00000";
47      --Saltos Condicionales e Incondicionales
48      constant BEQI : STD_LOGIC_VECTOR (4 downto 0) := "01101";
49      constant BNEI : STD_LOGIC_VECTOR (4 downto 0) := "01110";
50      constant BLTI : STD_LOGIC_VECTOR (4 downto 0) := "01111";
51      constant BLETI : STD_LOGIC_VECTOR (4 downto 0) := "10000";
52      constant BGTI : STD_LOGIC_VECTOR (4 downto 0) := "10001";
53      constant BGETI : STD_LOGIC_VECTOR (4 downto 0) := "10010";
54      constant B : STD_LOGIC_VECTOR (4 downto 0) := "10011";
55      --Manejo de Subrutinas
56      constant CALL : STD_LOGIC_VECTOR (4 downto 0) := "10100";
57      constant RET : STD_LOGIC_VECTOR (4 downto 0) := "10101";
58      --Otros
59      constant NOP : STD_LOGIC_VECTOR (4 downto 0) := "10110";
60      constant SU : STD_LOGIC_VECTOR (3 downto 0) := "0000"; -- sin usar
61      --Registros
62      constant R0 : STD_LOGIC_VECTOR (3 downto 0) := "0000";
63      constant R1 : STD_LOGIC_VECTOR (3 downto 0) := "0001";
64      constant R2 : STD_LOGIC_VECTOR (3 downto 0) := "0010";
65      constant R3 : STD_LOGIC_VECTOR (3 downto 0) := "0011";
66      constant R4 : STD_LOGIC_VECTOR (3 downto 0) := "0100";
67      constant R5 : STD_LOGIC_VECTOR (3 downto 0) := "0101";
68      constant R6 : STD_LOGIC_VECTOR (3 downto 0) := "0110";
69      constant R7 : STD_LOGIC_VECTOR (3 downto 0) := "0111";
70      constant R8 : STD_LOGIC_VECTOR (3 downto 0) := "1000";
71      constant R9 : STD_LOGIC_VECTOR (3 downto 0) := "1001";
72      constant R10 : STD_LOGIC_VECTOR (3 downto 0) := "1010";
73      constant R11 : STD_LOGIC_VECTOR (3 downto 0) := "1011";
74      constant R12 : STD_LOGIC_VECTOR (3 downto 0) := "1100";
75      constant R13 : STD_LOGIC_VECTOR (3 downto 0) := "1101";
```

```
76      constant R14 : STD_LOGIC_VECTOR (3 downto 0) := "1110";
77      constant R15 : STD_LOGIC_VECTOR (3 downto 0) := "1111";
78
79      type banco is array (0 to (2**m)-1) of STD_LOGIC_VECTOR(n-1 downto 0);
80      constant aux : banco := (
81          LI & R0 & x"0001",                  --LI  R0, #1
82          LI & R1 & x"0007",                  --LI  R1, #7
83          tipoR & R1 & R1 & R0 & SU & ADD,    --suma: ADD R1, R1, R0
84          SWI & R1 & x"0005",                 --SWI R1, 0x05
85          B & SU & x"0002",                   --B suma
86          others => (others => '0')
87      );
88  begin
89      inst <= aux(conv_integer(pc));
90  end Behavioral;
```

4. Simulación Programa Punto 2





5. Tabla Ejecución Programa Punto 2

Bus	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
PC	0	0	1	2	3	4	2	3	4	2	3
Instrucción	LI R0, #1	LI R0, #1	LI R1, #7	ADD R1, R1, R0	SWI R1, 0x05	B 0x02	ADD R1, R1, R0	SWI R1, 0x05	B 0x02	ADD R1, R1, R0	SWI R1, 0x05
ReadData1	0	0	1	7	1	1	8	1	1	9	1
ReadData2	0	0	1	1	8	1	1	9	1	1	10
ResALU	0	0	1	8	0	1	9	1	1	10	0
BusSR	0	0	0	8	0	0	9	8	0	10	9

6. Programa Punto 5: Código Memoria de Programa

6.1. Programa: Multiplicación de dos números usando sucesión de sumas

```
     LI R1, #10
     LI R2, #3
et1: ADD R3, R3, R1
     ADDI R0, R0, #1
     BLTI R2, R0, et1
fin: NOP
     B fin
```

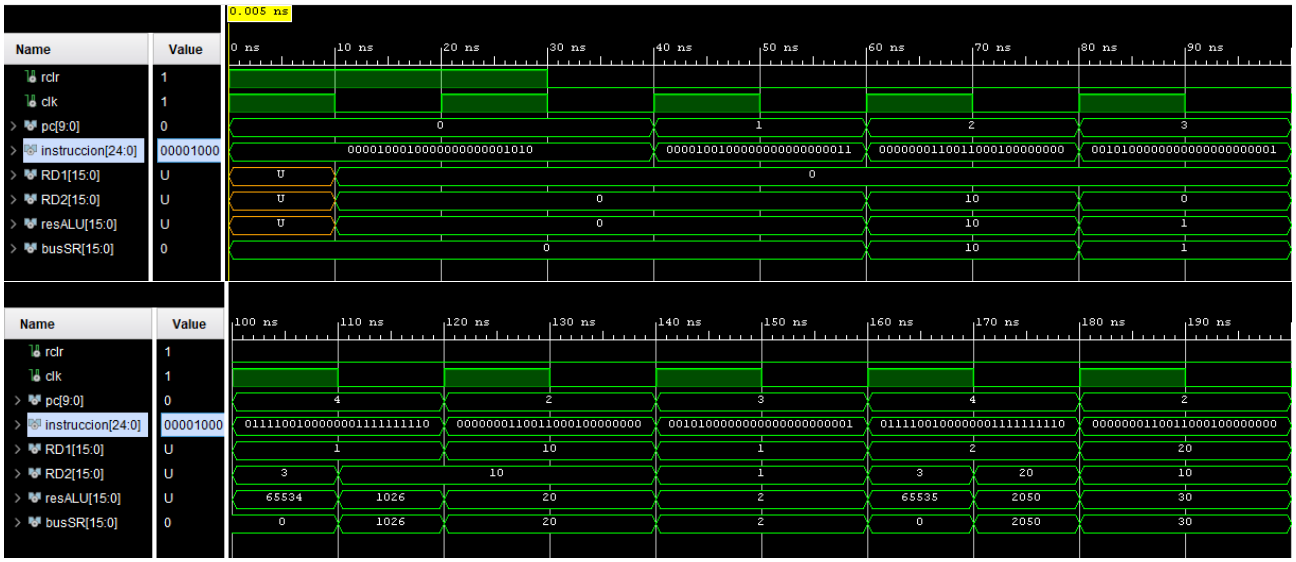
6.2. Memoria de Programa

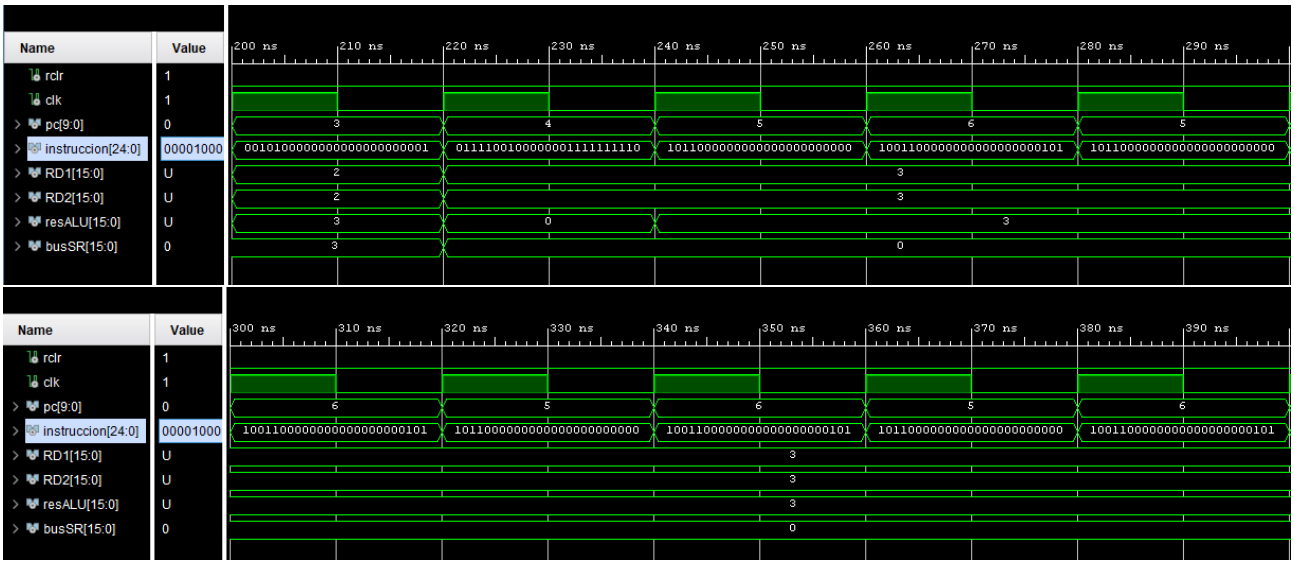
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity MemoriaPrograma2 is
    generic ( m : integer := 10;
              n : integer := 25);
    Port ( pc : in STD_LOGIC_VECTOR (m-1 downto 0);
           inst : out STD_LOGIC_VECTOR (n-1 downto 0));
end MemoriaPrograma2;

architecture Behavioral of MemoriaPrograma2 is
    --Instrucciones tipo R
    constant tipoR : STD_LOGIC_VECTOR (4 downto 0) := "00000";
    --Carga y Almacenamiento
    constant LI : STD_LOGIC_VECTOR (4 downto 0) := "00001";
    constant LWI : STD_LOGIC_VECTOR (4 downto 0) := "00010";
    constant LW : STD_LOGIC_VECTOR (4 downto 0) := "10111";
    constant SWI : STD_LOGIC_VECTOR (4 downto 0) := "00011";
    constant SW : STD_LOGIC_VECTOR (4 downto 0) := "00100";
    --Aritmticas
    constant ADDI : STD_LOGIC_VECTOR (4 downto 0) := "00101";
    constant SUBI : STD_LOGIC_VECTOR (4 downto 0) := "00110";
    --Identificador Aritmticas R
    constant ADD : STD_LOGIC_VECTOR (3 downto 0) := "0000";
    constant SUB : STD_LOGIC_VECTOR (3 downto 0) := "0001";
    --Logicas
    constant ANDI : STD_LOGIC_VECTOR (4 downto 0) := "00111";
```

```
30      constant ORI : STD_LOGIC_VECTOR (4 downto 0) := "01000";
31      constant XORI : STD_LOGIC_VECTOR (4 downto 0) := "01001";
32      constant NANDI : STD_LOGIC_VECTOR (4 downto 0) := "01010";
33      constant NORI : STD_LOGIC_VECTOR (4 downto 0) := "01011";
34      constant XNORI : STD_LOGIC_VECTOR (4 downto 0) := "01100";
35      --Identificador Logicas R
36      constant ANDR : STD_LOGIC_VECTOR (3 downto 0) := "0010";
37      constant ORR : STD_LOGIC_VECTOR (3 downto 0) := "0011";
38      constant XORR : STD_LOGIC_VECTOR (3 downto 0) := "0100";
39      constant NANDR : STD_LOGIC_VECTOR (3 downto 0) := "0101";
40      constant NORR : STD_LOGIC_VECTOR (3 downto 0) := "0110";
41      constant XNORR : STD_LOGIC_VECTOR (3 downto 0) := "0111";
42      constant NOTR : STD_LOGIC_VECTOR (3 downto 0) := "1000";
43      --Identificador Corrimiento R
44      constant SLLR : STD_LOGIC_VECTOR (3 downto 0) := "1001";
45      constant SRLR : STD_LOGIC_VECTOR (3 downto 0) := "1010";
46      constant OPR : STD_LOGIC_VECTOR (4 downto 0) := "00000";
47      --Saltos Condicionales e Incondicionales
48      constant BEQI : STD_LOGIC_VECTOR (4 downto 0) := "01101";
49      constant BNEI : STD_LOGIC_VECTOR (4 downto 0) := "01110";
50      constant BLTI : STD_LOGIC_VECTOR (4 downto 0) := "01111";
51      constant BLETI : STD_LOGIC_VECTOR (4 downto 0) := "10000";
52      constant BGTI : STD_LOGIC_VECTOR (4 downto 0) := "10001";
53      constant BGETI : STD_LOGIC_VECTOR (4 downto 0) := "10010";
54      constant B : STD_LOGIC_VECTOR (4 downto 0) := "10011";
55      --Manejo de Subrutinas
56      constant CALL : STD_LOGIC_VECTOR (4 downto 0) := "10100";
57      constant RET : STD_LOGIC_VECTOR (4 downto 0) := "10101";
58      --Otros
59      constant NOP : STD_LOGIC_VECTOR (4 downto 0) := "10110";
60      constant SU : STD_LOGIC_VECTOR (3 downto 0) := "0000"; -- sin usar
61      --Registros
62      constant R0 : STD_LOGIC_VECTOR (3 downto 0) := "0000";
63      constant R1 : STD_LOGIC_VECTOR (3 downto 0) := "0001";
64      constant R2 : STD_LOGIC_VECTOR (3 downto 0) := "0010";
65      constant R3 : STD_LOGIC_VECTOR (3 downto 0) := "0011";
66      constant R4 : STD_LOGIC_VECTOR (3 downto 0) := "0100";
67      constant R5 : STD_LOGIC_VECTOR (3 downto 0) := "0101";
68      constant R6 : STD_LOGIC_VECTOR (3 downto 0) := "0110";
69      constant R7 : STD_LOGIC_VECTOR (3 downto 0) := "0111";
70      constant R8 : STD_LOGIC_VECTOR (3 downto 0) := "1000";
71      constant R9 : STD_LOGIC_VECTOR (3 downto 0) := "1001";
72      constant R10 : STD_LOGIC_VECTOR (3 downto 0) := "1010";
73      constant R11 : STD_LOGIC_VECTOR (3 downto 0) := "1011";
```

```
74      constant R12 : STD_LOGIC_VECTOR (3 downto 0) := "1100";
75      constant R13 : STD_LOGIC_VECTOR (3 downto 0) := "1101";
76      constant R14 : STD_LOGIC_VECTOR (3 downto 0) := "1110";
77      constant R15 : STD_LOGIC_VECTOR (3 downto 0) := "1111";
78
79      type banco is array (0 to (2**m)-1) of STD_LOGIC_VECTOR(n-1 downto 0);
80      constant aux : banco := (
81          LI & R1 & x"000A",                  --LI  R1, #10
82          LI & R2 & x"0003",                  --LI  R2, #3
83          tipoR & R3 & R3 & R1 & SU & ADD,    --et1: ADD R3, R3, R1
84          ADDI & R0 & R0 & x"001",            --ADDI R0, R0, #1
85          BLTI & R2 & R0 & x"3FE",            --BLTI R2, R0, et1 (001111111110
             ↪  = -2)
86          NOP & SU & SU & SU & SU & SU,       --fin: NOP
87          B & SU & x"0005",                   --B fin
88          others => (others => '0')
89      );
90  begin
91      inst <= aux(conv_integer(pc));
92  end Behavioral;
```

7. Simulación Programa Punto 5





8. Tabla Ejecución Programa Punto 5

Bus	T1	T2	T3	T4	T5	T6-AL	T6-BA	T7	T8	T9-AL	T9-BA	T10	T11
PC	0	0	1	2	3	4		2	3	4		2	3
Instrucción	LI R1, #10	LI R1, #10	LI R2, #3	ADD R3, R3, R1	ADDI R0, R0, #1	BLTI R2, R0, 0x3FE		ADD R3, R3, R1	ADDI R0, R0, #1	BLTI R2, R0, 0x3FE		ADD R3, R3, R1	ADDI R0, R0, #1
ReadData1	0	0	0	0	0	1		10	1	2		20	2
ReadData2	0	0	0	10	0	3	10	10	1	3	20	10	2
ResALU	0	0	0	10	1	65534	1026	20	2	65534	2050	30	3
BusSR	0	0	0	10	1	0	1026	20	2	0	2050	30	3