Instituto Politécnico Nacional
Escuela Superior de Computo

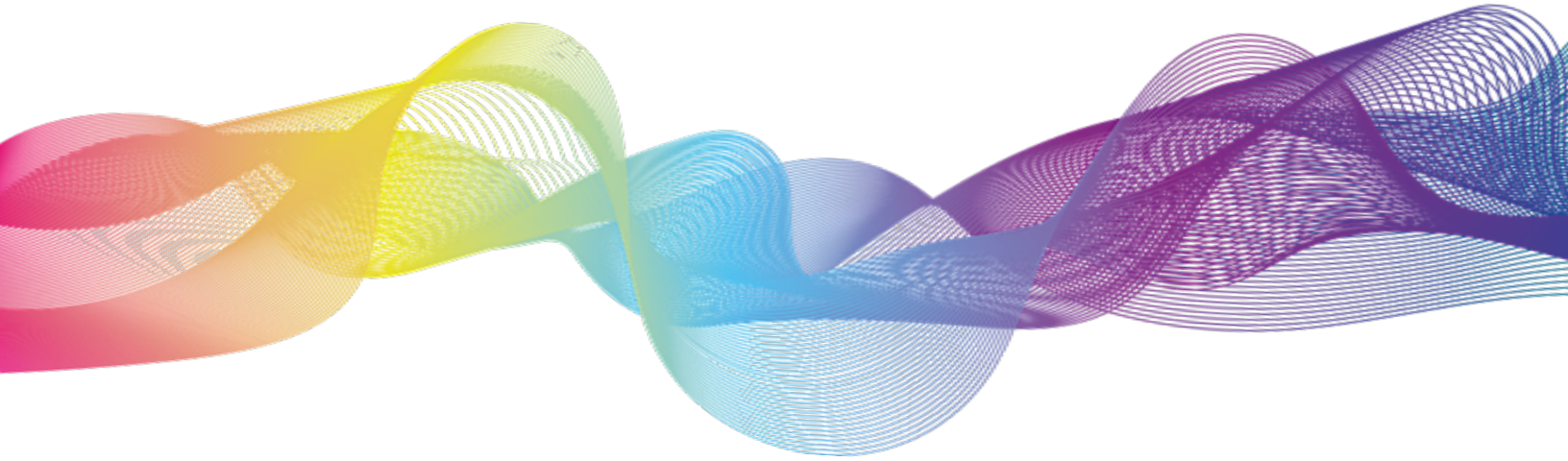
Unidad de aprendizaje: Arquitectura de computadoras [3CV2]

Profesor@: Nayeli Vega García

Proyecto : Implementación del ESCOMips

Alumno: García González Aarón Antonio [N.L. 11]

Julio, 2020

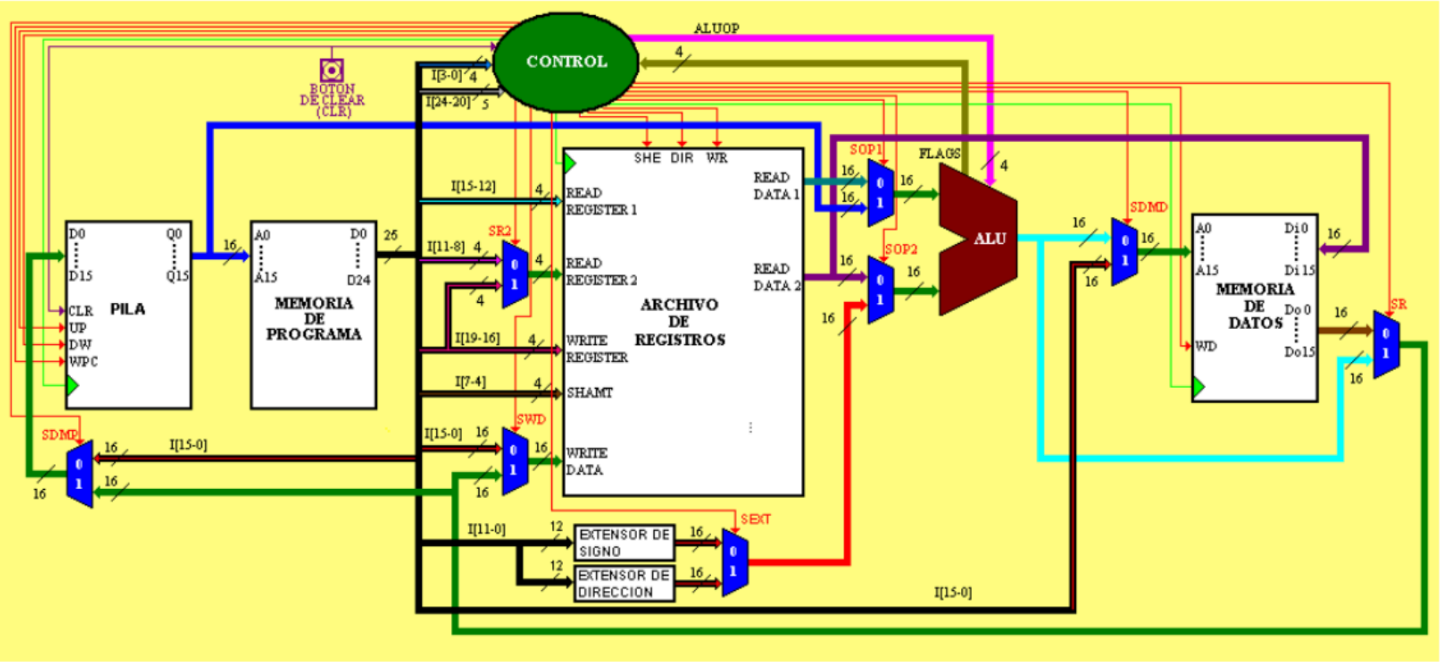


Índice

1. Especificación del proyecto

Implementar el procesador ESCOMips con las siguientes configuraciones:

- Debe estar implementado por componentes
- Tamaño de los buses de la ALU: 16 bits
- Organización de la memoria de datos: 1024 x 16
- Organización de la memoria de programa: 1024 x 25



Dadas las configuraciones anteriores, vamos a definir el tamaño de los buses de cada componente:

- Pila: Dado que no se especifica modificación alguna, se quedará de la misma densidad que el diagrama original, es decir un bus de entrada de hasta 8 niveles con direcciones de manera simultanea y cada una de estas palabras con un tamaño de 16 bits.
- Memoria de programa: Se solicita una organización de 1024 x 25 bits, es decir hay posibilidad de cargar la memoria ROM hasta con 1024 instrucciones de manera simultanea, por lo que el bus de entrada será de Log2(1024), es decir 10 bits, y tomara únicamente de la salida de la pila los valores entre 0 y 9 bits.
- Archivo de registros: Dado que no se especifica modificación alguna, se quedará de la misma densidad que el diagrama original, 16 registros y cada uno con datos o direcciones de 16 bits, son 16 registros debido a que cada registro en dirección es de 4 bits, por lo que 2 potencializado a la 4, es 16.
- ALU: Se solicita ser de 16 bits, que es idéntico al diagrama original, por los dos operadores de entrada y resultado será de 16 bits.
- Memoria de datos: Se solicita una organización de 1024 x 16 bits, es decir hay posibilidad de cargar tener almacenado hasta 1024 datos de manera simultanea, por lo que el bus de entrada será de Log (1024), es decir 10 bits, y sus salidas son los 16 bits de tamaño de palabra de datos que tiene.
- Unidad de control: Todo se queda igual, es decir nos regresa la microinstrucción adecuada de 20 bits.

1.1. Código de implementación

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity ESCOMIPS is
5.     Port (
6.         CLK, RCLR : IN STD_LOGIC;
7.         LECTURA_PC : OUT STD_LOGIC_VECTOR(15 downto 0);
8.         LECTURA_INSTRUCCION : OUT STD_LOGIC_VECTOR(24 downto 0);
9.         LECTURA_READDATA1, LECTURA_READDATA2 : OUT STD_LOGIC_VECTOR(15 downto 0);
10.             LECTURA_RES_ALU : OUT STD_LOGIC_VECTOR(15 downto 0);
11.             LECTURA_BUS_SR : OUT STD_LOGIC_VECTOR(15 downto 0);
12.             LECTURA_MICROINSTRUCCION : OUT STD_LOGIC_VECTOR(19 downto 0);
13.             LECTURA_NA : OUT STD_LOGIC
14.         );
15.     end ESCOMIPS;
16.
17.     architecture Behavioral of ESCOMIPS is
18.         -- pila
19.         component pila is
20.             Port (
21.                 clk, clr, up, dw, wpc : in  STD_LOGIC;
22.                 pcin : in  STD_LOGIC_VECTOR (15 downto 0);
23.                 pcout : out  STD_LOGIC_VECTOR (15 downto 0));
24.         end component;
25.
26.         -- memoria de programa
27.         component MemoriaPrograma is
28.         Port (
29.             dir : in  STD_LOGIC_VECTOR (9 downto 0);
30.             dout : out  STD_LOGIC_VECTOR (24 downto 0));
31.         end component;
32.
33.         -- archivo de registros
34.         component Archivo_Registro is
35.             Port (
36.                 wr,she,dir,clk,clr : in  STD_LOGIC;
37.                 write_reg,read_reg1,read_reg2,shamt : in  STD_LOGIC_VECTOR (3 down
    to 0);
38.                 write_data : in  STD_LOGIC_VECTOR (15 downto 0);
39.                 read_data1,read_data2 : out  STD_LOGIC_VECTOR (15 downto 0));
40.         end component;
41.
42.         -- alu
43.         component Alu_16bits is
44.             Port (
45.                 a,b : in  STD_LOGIC_VECTOR (15 downto 0);
46.                 aluop : in STD_LOGIC_VECTOR  (3 downto 0);
47.                 s : out  STD_LOGIC_VECTOR (15 downto 0);
48.                 flags : out STD_LOGIC_VECTOR (3 downto 0));
49.         end component;
50.
51.         -- memoria de datos
52.         component MemoriaDatos is
53.             Port (
54.                 dir : in  STD_LOGIC_VECTOR (9 downto 0);
55.                 data_in : in  STD_LOGIC_VECTOR (15 downto 0);
```

```
56.                 data_out : out  STD_LOGIC_VECTOR (15 downto 0);
57.                 WD, CLK : in  STD_LOGIC);
58.         end component;
59.
60.         -- unidad de control
61.         component UnidadControl is
62.             Port (
63.                 clk,clr: STD_LOGIC;
64.                 cOperacion : in  STD_LOGIC_VECTOR (4 downto 0);
65.                 cFuncion,flags : in  STD_LOGIC_VECTOR (3 downto 0);
66.                 s : out  STD_LOGIC_VECTOR (19 downto 0);
67.                 NA : OUT STD_LOGIC);
68.         end component;
69.
70.         -- delcaracion de buses de transporte de datos
71.
72.         -- SDMP UP DW WPC SR2 SWD SEXT SHE DIR WR SOP1 SOP2 ALUOP3 ALUOP2 ALUOP1
   ALUOP0 SDMD WD SR LF
73.         -- 19   18 17  16  15  14   13  12  11
   10   09   08     07     06     05     04   03 02 01 00
74.
75.         signal microInstruccion : STD_LOGIC_VECTOR(19 downto 0);
76.         signal salida_pila : STD_LOGIC_VECTOR(15 downto 0);
77.         signal Instruccion : STD_LOGIC_VECTOR(24 downto 0);
78.         signal extension_signo,
   extension_direccion : STD_LOGIC_VECTOR(15 downto 0);
79.         signal readData1, readData2 : STD_LOGIC_VECTOR(15 downto 0);
80.         signal banderas_alu_salida : STD_LOGIC_VECTOR(3 downto 0);
81.         signal resALU : STD_LOGIC_VECTOR(15 downto 0);
82.         signal salida_memoria_datos : STD_LOGIC_VECTOR(15 downto 0);
83.
84.         -- Salidas de muxes
85.         signal SDMP : STD_LOGIC_VECTOR(15 downto 0);
86.         signal SR2 : STD_LOGIC_VECTOR(3 downto 0);
87.         signal SWD : STD_LOGIC_VECTOR(15 downto 0);
88.         signal SEXT : STD_LOGIC_VECTOR(15 downto 0);
89.         signal SOP1, SOP2 : STD_LOGIC_VECTOR(15 downto 0);
90.         signal SDMD : STD_LOGIC_VECTOR(15 downto 0);
91.         signal SR : STD_LOGIC_VECTOR(15 downto 0);
92.
93.         signal CLR : STD_LOGIC;
94.         signal NA : STD_LOGIC;
95.
96.         begin
97.
98.         process(CLK)
99.             begin
100.            if(falling_edge(clk)) then
101.                CLR <= RCLR;
102.            end if;
103.        end process;
104.
105.        STACK : pila Port map (
106.            clk => CLK,
107.            clr => CLR,
108.            up => microInstruccion(18),
109.            dw => microInstruccion(17),
110.            wpc => microInstruccion(16),
111.            pcin => SDMP,
112.            pcout => salida_pila
```

```
113.         );
114.
115.         MEMPROG : MemoriaPrograma Port map(
116.             dir => salida_pila(9 downto 0),
117.             dout => Instruccion
118.         );
119.
120.         FILEREGISTER : Archivo_Registro Port map(
121.             wr => microInstruccion(10),
122.             she => microInstruccion(12),
123.             dir => microInstruccion(11),
124.             clk => CLK,
125.             clr => CLR,
126.             write_reg => Instruccion(19 downto 16),
127.             read_reg1 => Instruccion(15 downto 12),
128.             read_reg2 => SR2,
129.             shamt => Instruccion(7 downto 4),
130.             write_data => SWD,
131.             read_data1 => readData1,
132.             read_data2 => readData2
133.         );
134.
135.        ALUN : Alu_16bits Port map(
136.             a => SOP1,
137.             b => SOP2,
138.             aluop => microInstruccion(7 downto 4),
139.             s => resALU,
140.             flags => banderas_alu_salida
141.         );
142.
143.         MEMDATA : MemoriaDatos Port map(
144.             dir => SDMD(9 downto 0),
145.             data_in => readData2,
146.             data_out => salida_memoria_datos,
147.             WD => microInstruccion(2),
148.             CLK => CLK
149.         );
150.
151.         UNITCONTROL : UnidadControl Port map(
152.             clk => CLK,
153.             clr => CLR,
154.             cOperacion => Instruccion(24 downto 20),
155.             cFuncion => Instruccion(3 downto 0),
156.             flags => banderas_alu_salida,
157.             s => microInstruccion,
158.             NA => NA
159.         );
160.
161.         SR <= salida_memoria_datos when (microInstruccion(1) = '0') else resALU;
162.         SDMP <= Instruccion(15 downto 0) when (microInstruccion(19) = '0') else SR
    ;
163.         SR2 <= Instruccion(11 downto 8) when (microInstruccion(15) = '0') else Ins
   truccion(19 downto 16);
164.         SWD <= Instruccion(15 downto 0) when (microInstruccion(14) = '0') else SR;
165.         SEXT <= (Instruccion(11) & Instruccion(11) & Instruccion(11) & Instruccion
   (11) & Instruccion(11 downto 0)) when (microInstruccion(13) = '0') else ("0000" & I
   nstruccion(11 downto 0));
166.         SOP1 <= readData1 when (microinstruccion(9) = '0') else salida_pila;
167.         SOP2 <= readData2 when (microinstruccion(8) = '0') else SEXT;
```

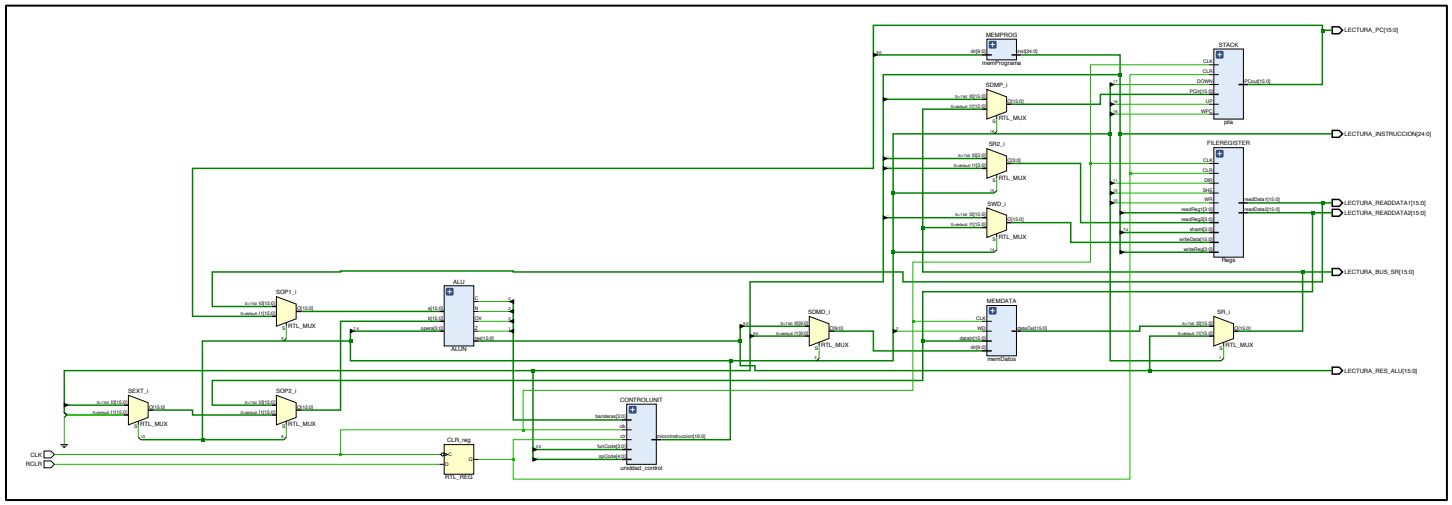
```
168.         SDMD <= resALU when (microinstruccion(3) = '0') else Instruccion(15 downto
    0);
169.
170.         LECTURA_PC <= salida_pila;
171.         LECTURA_INSTRUCCION <= Instruccion;
172.         LECTURA_READDATA1 <= readData1;
173.         LECTURA_READDATA2 <= readData2;
174.         LECTURA_RES_ALU <= resALU;
175.         LECTURA_BUS_SR <= SR;
176.         LECTURA_MICROINSTRUCCION <= microInstruccion;
177.         LECTURA_NA <= NA;
178.
179.     end Behavioral;
180.
```

1.2. Código de simulación

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity ESCOMIPS_TB is
5. end ESCOMIPS_TB;
6.
7. architecture Behavioral of ESCOMIPS_TB is
8.     component ESCOMips is
9.         Port (
10.                 CLK, RCLR : IN STD_LOGIC;
11.                 LECTURA_PC : OUT STD_LOGIC_VECTOR(15 downto 0);
12.                 LECTURA_INSTRUCCION : OUT STD_LOGIC_VECTOR(24 downto 0);
13.                 LECTURA_READDATA1,
   LECTURA_READDATA2 : OUT STD_LOGIC_VECTOR(15 downto 0);
14.                 LECTURA_RES_ALU : OUT STD_LOGIC_VECTOR(15 downto 0);
15.                 LECTURA_BUS_SR : OUT STD_LOGIC_VECTOR(15 downto 0);
16.                 LECTURA_MICROINSTRUCCION : OUT STD_LOGIC_VECTOR(19 downto 0);
17.                 LECTURA_NA : OUT STD_LOGIC
18.             );
19.         end component;
20.
21.         -- Se?ales de transporte
22.         signal CLK, RCLR : STD_LOGIC;
23.         signal LECTURA_PC : STD_LOGIC_VECTOR(15 downto 0);
24.         signal LECTURA_INSTRUCCION : STD_LOGIC_VECTOR(24 downto 0);
25.         signal LECTURA_READDATA1,
   LECTURA_READDATA2 : STD_LOGIC_VECTOR(15 downto 0);
26.         signal LECTURA_RES_ALU : STD_LOGIC_VECTOR(15 downto 0);
27.         signal LECTURA_BUS_SR : STD_LOGIC_VECTOR(15 downto 0);
28.         signal LECTURA_MICROINSTRUCCION : STD_LOGIC_VECTOR(19 downto 0);
29.         signal LECTURA_NA : STD_LOGIC;
30.
31.         begin
32.
33.         ESCOMipsMAP : ESCOMips  Port map(
34.             CLK => CLK,
35.             RCLR => RCLR,
36.             LECTURA_PC => LECTURA_PC ,
37.             LECTURA_INSTRUCCION => LECTURA_INSTRUCCION,
38.             LECTURA_READDATA1 => LECTURA_READDATA1,
39.             LECTURA_READDATA2 => LECTURA_READDATA2,
40.             LECTURA_RES_ALU => LECTURA_RES_ALU,
41.             LECTURA_BUS_SR => LECTURA_BUS_SR,
42.             LECTURA_MICROINSTRUCCION => LECTURA_MICROINSTRUCCION,
43.             LECTURA_NA => LECTURA_NA
44.         );
45.
46.         CLOCK : process
47.             begin
48.             CLK <= '0';
49.             wait for 5 ns;
50.             CLK <= '1';
51.             wait for 5 ns;
52.         end process;
53.
54.         RESET : process
```

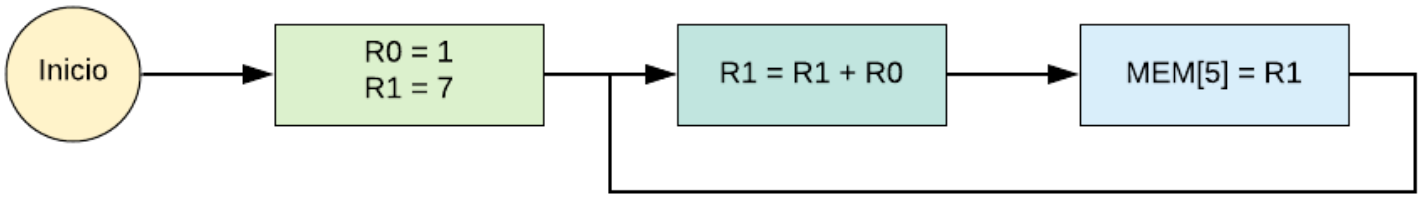
```
55.             begin
56.             RCLR <= '1';
57.             wait for 20 ns;
58.             RCLR <= '0';
59.             wait;
60.         end process;
61.
62.     end Behavioral;
```

1.3. Diagrama RTL



2. Programa 1 - Sumador unitario

Cargar en la memoria de programa el código del diagrama de flujo que se muestra a continuación. Las instrucciones deben cargarse en la memoria utilizando las constantes definidas en la práctica de Memoria de programa.



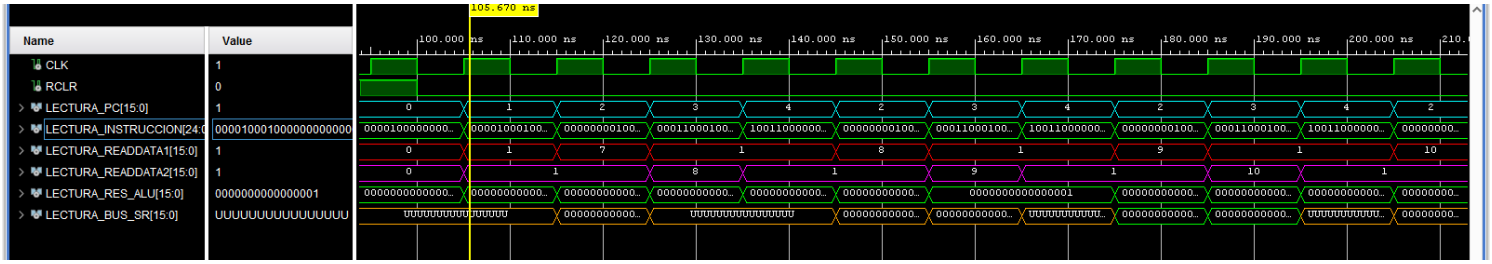
2.1. Programa en memoria de programa

```
1.  LI & R0 & x"0001",                -- 0
2.  LI & R1 & x"0007",                -- 1
3.  tipo_r & R1 & R1 & R0 & SU & add, -- 2
4.  SWI & R1 & x"0005",               -- 3
5.  B & SU & x"0002",                 -- 4
```

2.2. Análisis tabular

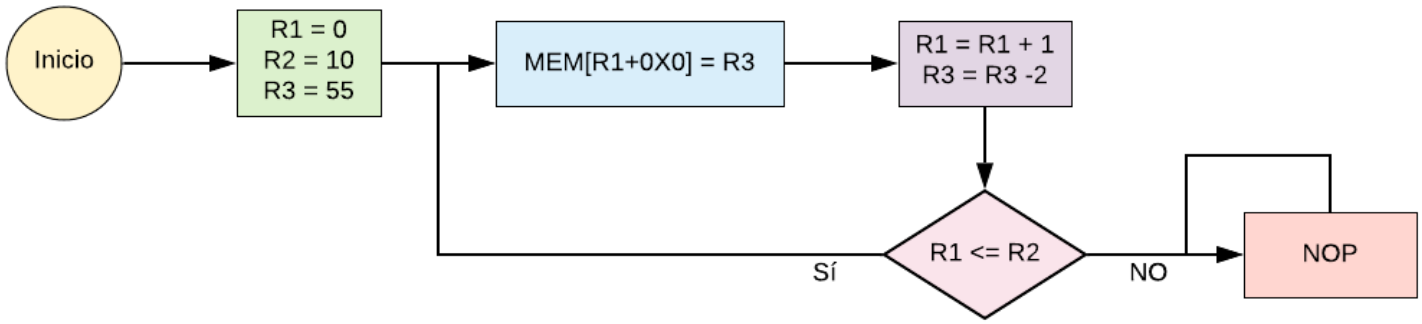
Bus	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
PC	0	1	2	3	4	2	3	4	2	3	4
Instrucción	LI R0, #1	LI R0, #1	ADD R1, R1, R0	SWI R1, #5	B 0x2	ADD R1, R1, R0	SWI R1, #5	B 0x2	ADD R1, R1, R0	SWI R1, #5	B 0x2
Read Data 1	0	1	7	1	1	8	1	1	9	1	1
Read Data 2	0	1	1	8	1	1	9	1	1	10	1
Res ALU	0	1	8	0	1	9	1	1	10	0	1
Bus SR	U	U	U	U	U	U	U	U	10	9	U

2.3. Simulación



3. Programa 2 – Llenar arreglo de 10 localidades en memoria con números impares descendentes comenzando en 55

Cargar en la memoria de programa el código del diagrama de flujo que se muestra a continuación. Las instrucciones deben cargarse en la memoria utilizando las constantes definidas en la práctica de Memoria de programa.



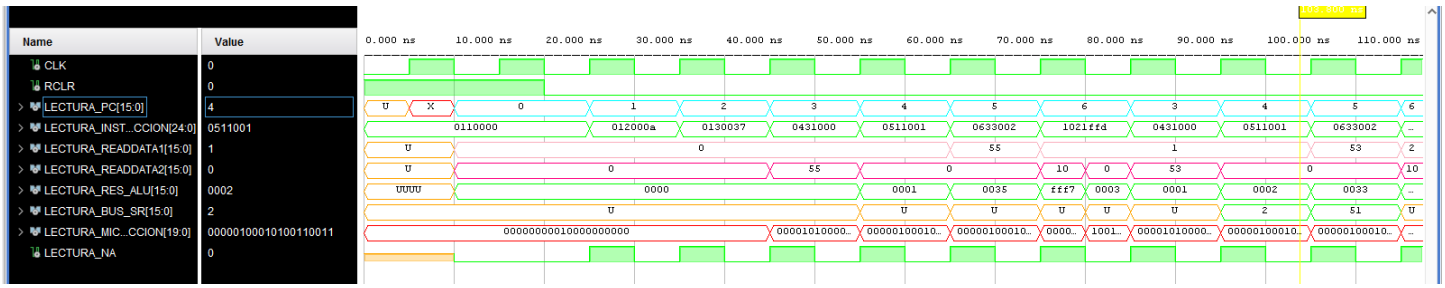
3.1. Programa en memoria de programa

```
1.  LI & R1 & x"0000",              -- 0
2.  LI & R2 & x"000A",              -- 1
3.  LI & R3 & x"0037",              -- 2
4.  SW & R3 & R1 & x"000",          -- 3
5.  ADDI & R1 & R1 & x"001",        -- 4
6.  SUBI & R3 & R3 & x"002",        -- 5
7.  BLETI & R2 & R1 & x"ffd",       -- 6
8.  NOP & SU & SU & SU & SU & SU,   -- 7
9.  B & SU & x"0007",               -- 8
```

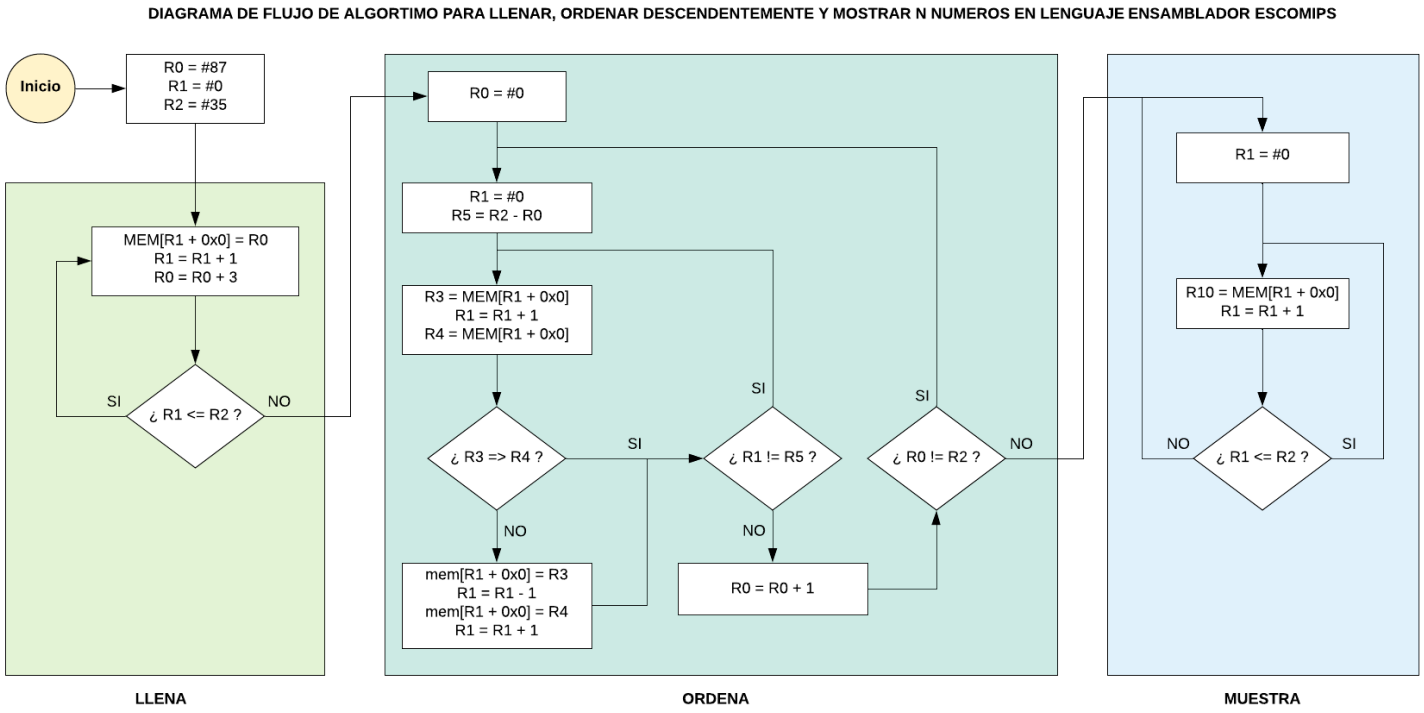
3.2. Análisis tabular

Bus	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
PC	0	1	2	3	4	5	6	3	4	5	6
Instrucción	LI R1, #0	LI R2, #10	LI R2, #55	SW R3, R1(0x0)	ADDI R1, R1, #1	SUBI R3, R3, #2	BLETI R2, R1, 0xFFD	SW R3, R1(0x0)	ADDI R1, R1, #1	SUBI R3, R3, #2	BLETI R2, R1, 0xFFD
Read Data 1	0	0	0	0	0	55	1	1	1	53	2
Read Data 2	0	0	0	55	0	0	10	0	53	0	0
Res ALU	0	0	0	0	1	53	3	1	2	51	3
Bus SR	U	U	U	U	U	U	U	U	2	51	U

3.3. Simulación



4. Programa 3. Llenar un arreglo de 35 localidades de memoria con números en sucesión de 3 en 3 a partir del 87, ordenarlos descendentemente y mostrar el arreglo ordenado.



4.1. Programa en memoria de programa

```
1.  LI & R0 & x"0057", --                              0
2.  LI & R1 & x"0000", -- Contador                     1
3.  LI & R2 & x"0008", -- N= 20                        2
4.  CALL & SU & x"0007",--Llenar                       3
5.  CALL & SU & x"000C",--Ordenar                      4
6.  CALL & SU & x"001B",--Mostrar                      5
7.  B & SU & x"0005",--                                6
8.
9.  -------------     SUBRUTINA LLENAR    -------------------
10.
11. SW & R0 & R1 & x"000", --                          7
12. ADDI & R1 & R1 & x"001",--                         8
13. ADDI & R0 & R0 & x"003",--                         9
14. BLETI & R2 & R1 & x"FFD",--                        10
15. RET & SU & SU & SU & SU & SU,--                    11
16.
17. -------------     SUBRUTINA ORDENAR    -------------------
18.
19. LI & R0 & x"0000",  --LI R0=0                      12
20. LI & R1 & x"0000",  --LI R1=0                      13
21. tipo_r & R5 & R2 & R0 & su & sub, --SUB R5= R2-R0  14
22. LW & R3 & R1 & x"000", --LW R3=MEM[R1+0]           15
23. ADDI & R1 & R1 & x"001", --R1=R1+1                 16
24. LW & R4 & R1 & x"000", --LW R4=MEM[R1+0]           17
25. BGETI & R4 & R3 & x"005", --R3 <= R4               18
```

```
26. SW & R3 & R1 & x"000", -- mem[R1+0]=R3              19
27. SUBI & R1 & R1 & x"001", --R1=R1-1                  20
28. SW & R4 & R1 & x"000", -- mem[R1+0]=R4              21
29. ADDI & R1 & R1 & x"001", --R1=R1+1                  22
30. BNEI & R5 & R1 & x"FF8", --R1 != R5                 23
31. ADDI & R0 & R0 & x"001", --R0=R0+1                  24
32. BNEI & R2 & R0 & x"FF4", --R0 != R2                 25
33. RET & SU & SU & SU & SU & SU,-- RET                 26
34.
35. -------------     SUBRUTINA MOSTRAR    -------------------
36.
37. LI & R1 & x"0000",  --LI R1=0                       27
38. LW & R10 & R1 & x"000",--   LW R10=MEM[R1+0 ]       28
39. ADDI & R1 & R1 & x"001", --R1=R1+1                  29
40. BLETI & R2 & R1 & x"FFE",-- R1<=R2                  30
41. RET & SU & SU & SU & SU & SU,--RET                  31
```