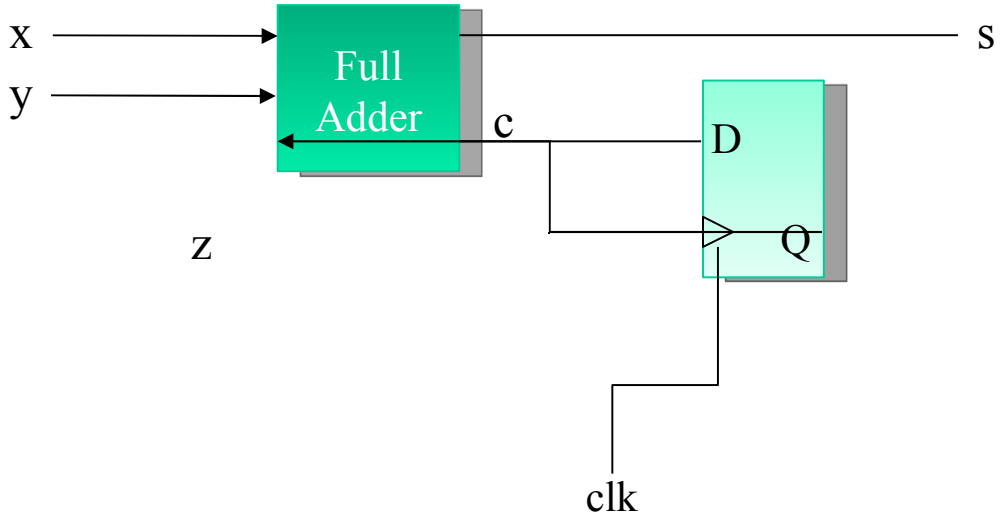
Digital System Design Course

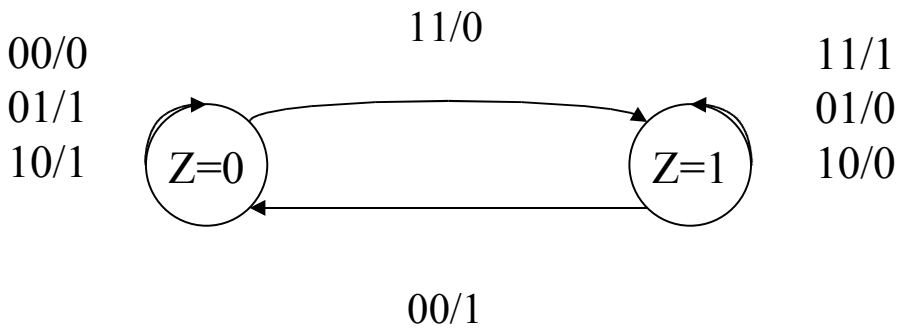
Ejercicio 1

- El sumador completo de la figura recibe dos entradas externas X y Y; la tercera entrada Z viene de la salida del Flip-flop D. El carry se transfiere al flip-flop en cada pulso de reloj. La salida externa S da la suma de X, Y, y Z. Obtenga la tabla de estados y el diagrama de estados del circuito secuencial.



Ejercicio 1

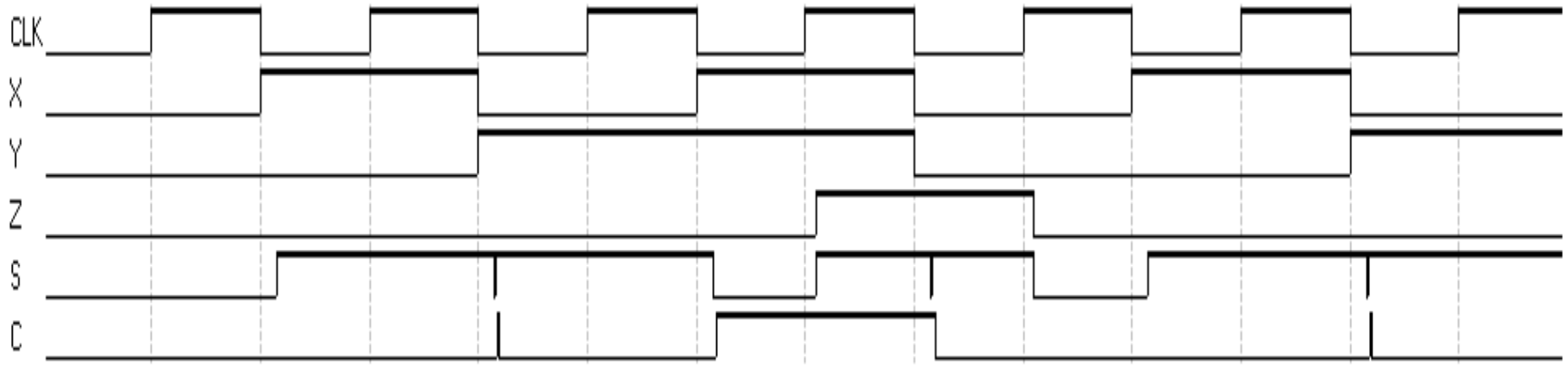
X	Y	Z	Z+	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Entradas: xy
Salida: s

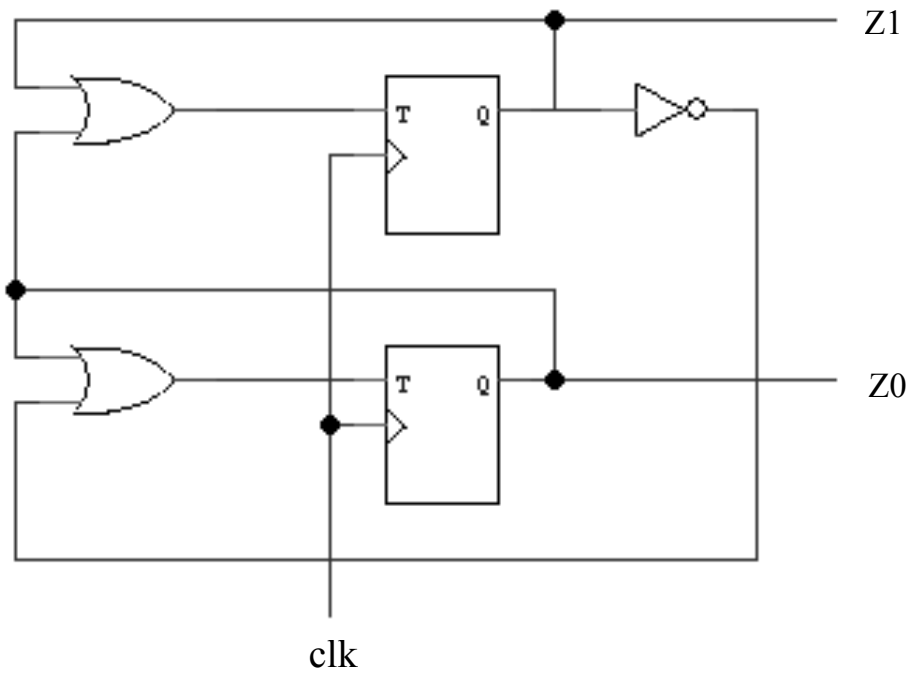
Ejercicio 1: Simulación

Universidad del Valle



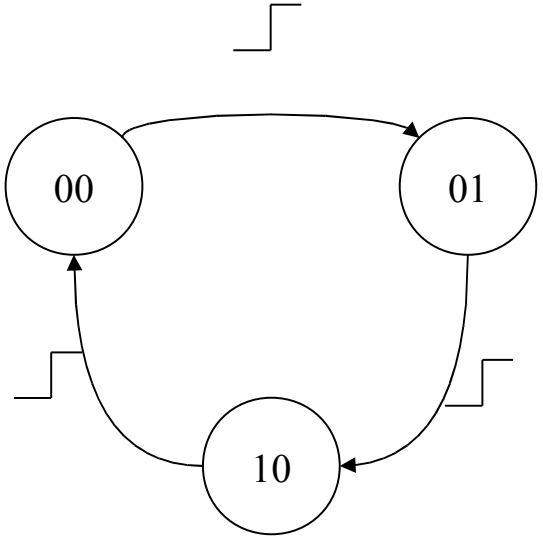
Ejercicio 2

- Deduzca la tabla de estado y diagrama de estado del circuito secuencial de la figura. Cual es la función del circuito?



Ejercicio 2

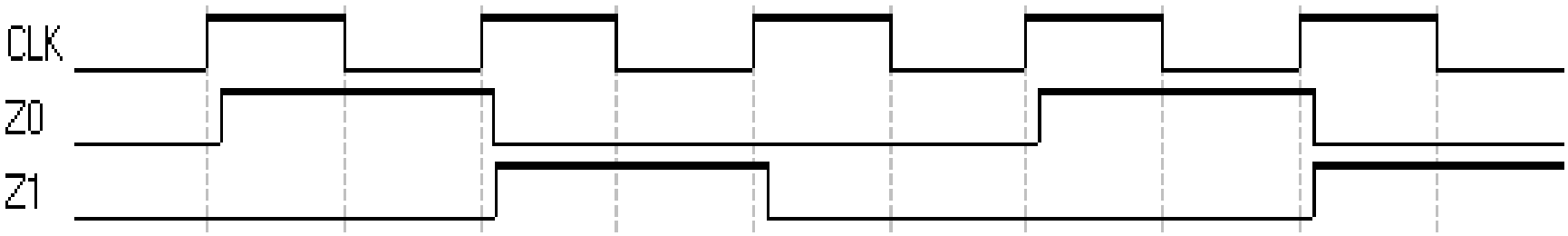
Estado Presente		Próximo Estado		Salida	
Q1	Q0	Q1+	Q0+	Z1	Z0
0	0	0	1	0	0
0	1	1	0	0	1
1	0	0	0	1	0
1	1	0	0	1	1



El circuito realiza la cuenta 00 – 01 – 10

Ejercicio 2: Simulación

Universidad del Valle



Ejercicio 3

Universidad del Valle

- Un circuito secuencial tiene dos flip-flops (A y B), dos entradas X y Y, y una salida Z. Las funciones de entrada de los flip-flops y la función de salida del circuito son las siguientes:

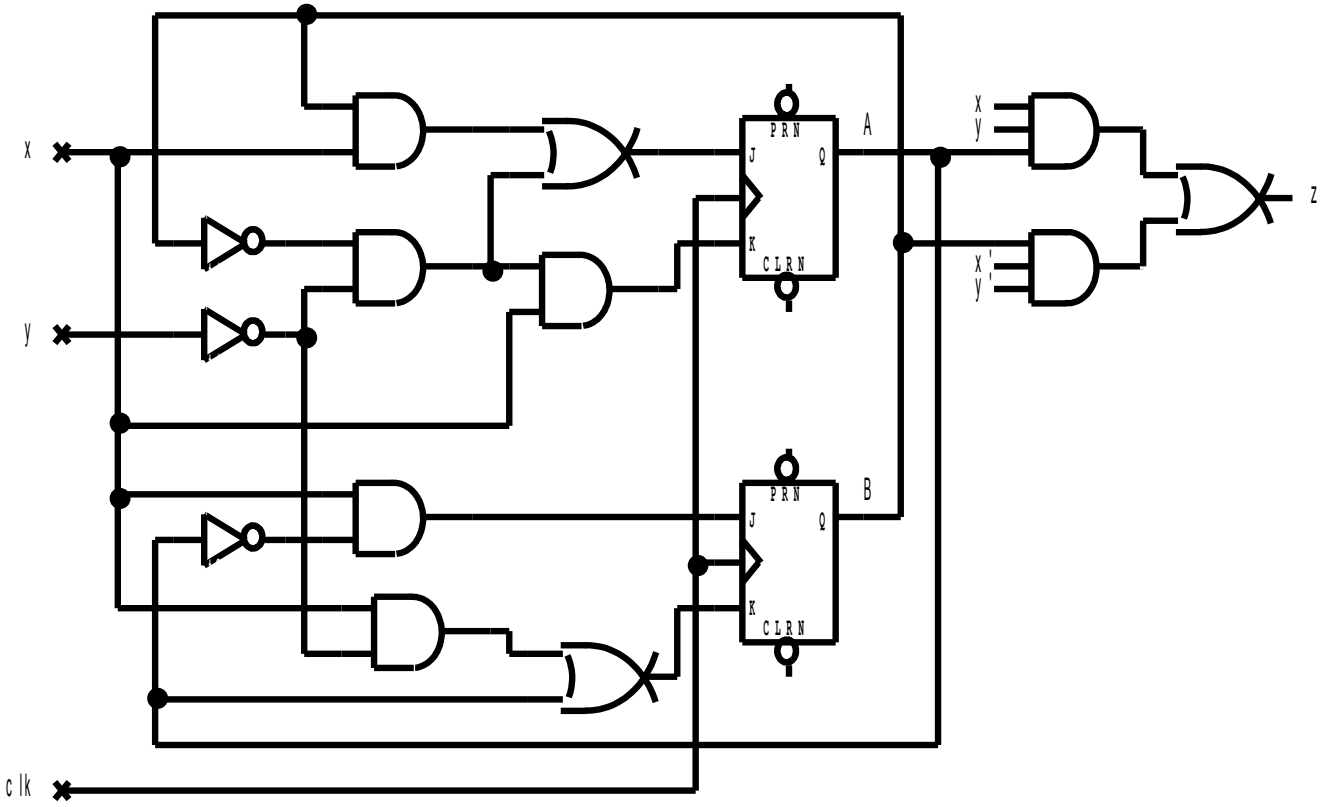
$$JA = xB + y'B' \qquad KA = xy'B'$$

$$JB = xA' \qquad KB = xy' + A$$

$$z = xyA + x'y'B$$

Obtenga el diagrama lógico, la tabla de estado y el diagrama de estados.

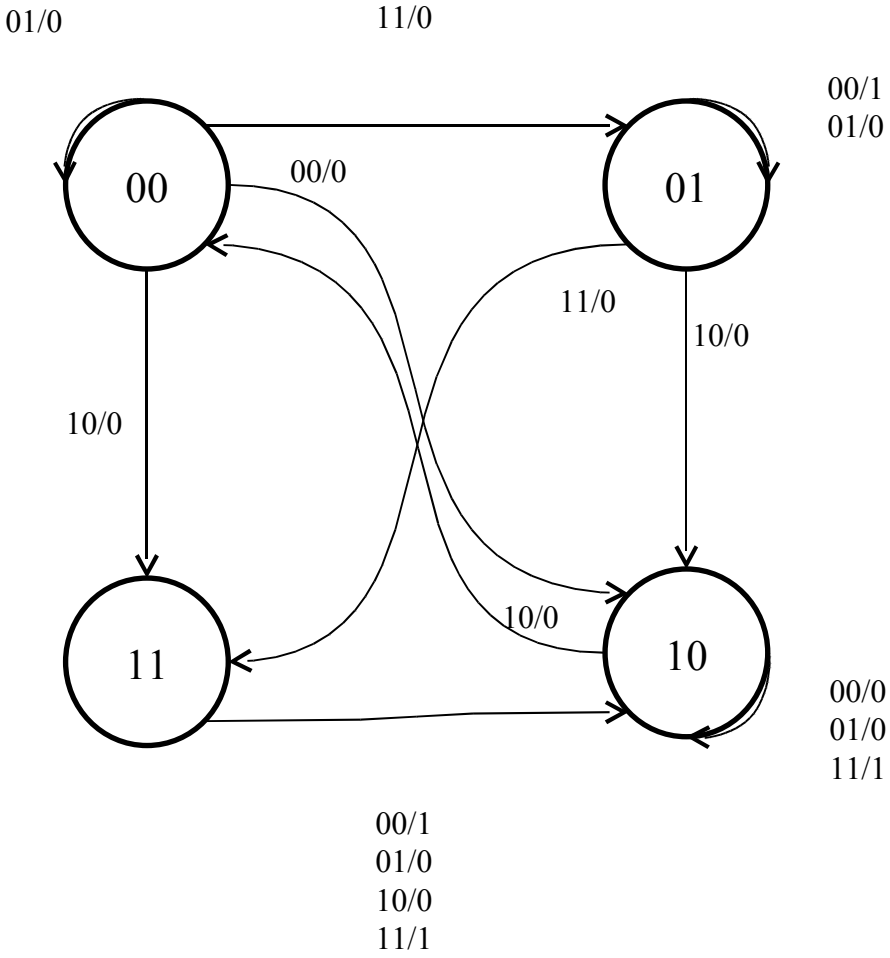
Ejercicio 3



Ejercicio 3

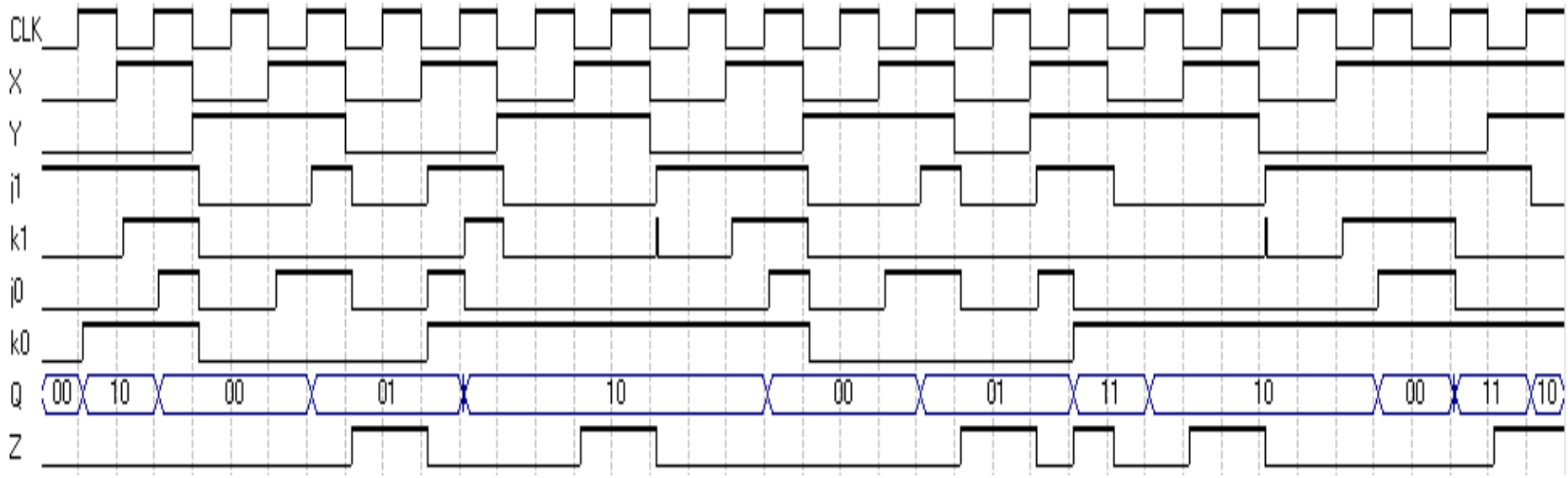
Estado Presente		Entradas		Entradas Flip-flops				Próximo Estado		Salida
A	B	x	y	JA	KA	JB	KB	A+	B+	z
0	0	0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	1	1	1	1	1	0
0	0	1	1	0	0	1	0	0	1	0
0	1	0	0	0	0	0	0	0	1	1
0	1	0	1	0	0	0	0	0	1	0
0	1	1	0	1	0	1	1	1	0	0
0	1	1	1	1	0	1	0	1	1	0
1	0	0	0	1	0	0	1	1	0	0
1	0	0	1	0	0	0	1	1	0	0
1	0	1	0	1	1	0	1	0	0	0
1	0	1	1	0	0	0	1	1	0	1
1	1	0	0	0	0	0	1	1	0	1
1	1	0	1	0	0	0	1	1	0	0
1	1	1	0	1	0	0	1	1	0	0
1	1	1	1	1	0	0	1	1	0	1

Ejercicio 3


01/0
11/0
00/1
01/0
00
01
00/0
11/0
10/0
10/0
11
10
10/0
00/0
01/0
11/1
00/1
01/0
10/0
11/1

Ejercicio 3: Simulación

Universidad del Valle



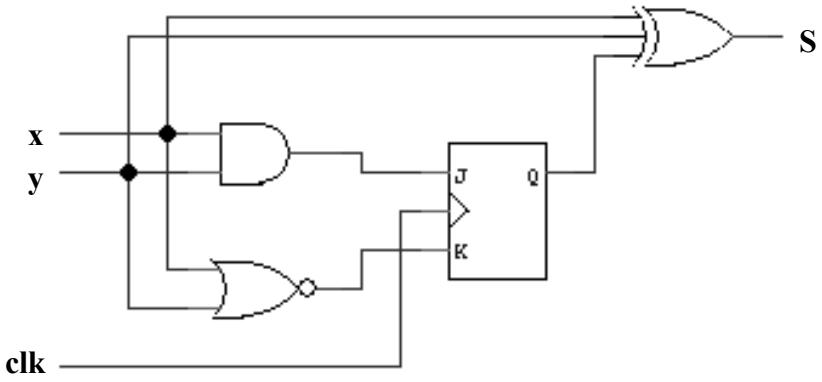
Ejercicio 4

Universidad del Valle

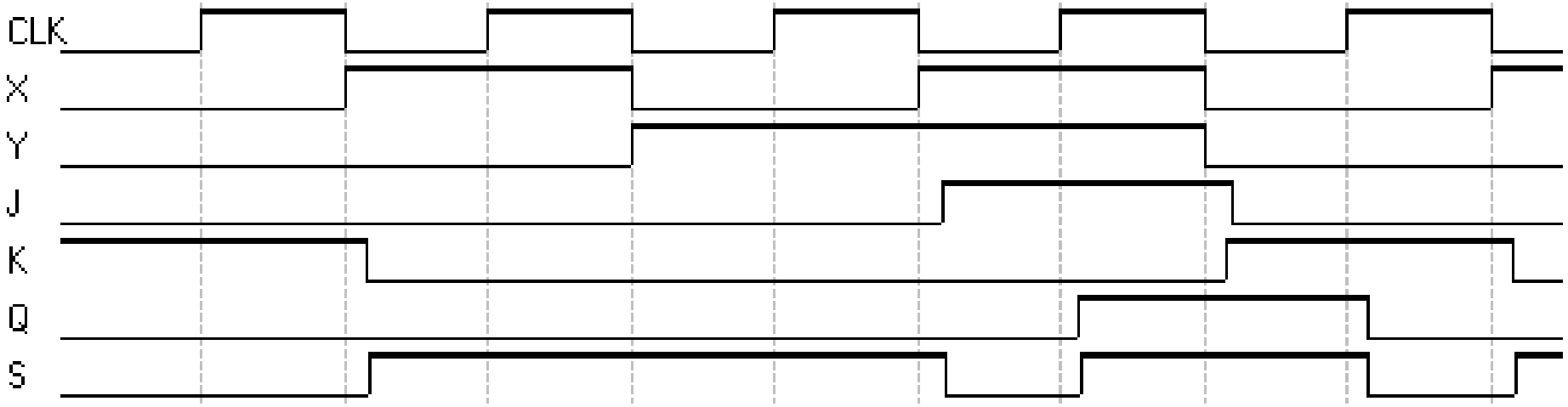
- Diseñar un sumador en serie usando el procedimiento de lógica secuencial (tabla de estados). Utilizar flip-flops JK.

Estado Presente	Entradas		Proximo Estado	Salida	Flip-Flops	
Q	x	y	Q+	S	JQ	KQ
0	0	0	0	0	0	X
0	0	1	0	1	0	X
0	1	0	0	1	0	X
0	1	1	1	0	1	X
1	0	0	0	1	X	1
1	0	1	1	0	X	0
1	1	0	1	0	X	0
1	1	1	1	1	X	0

El estado presente Q es el valor presente del carry.



Ejercicio 4: Simulación

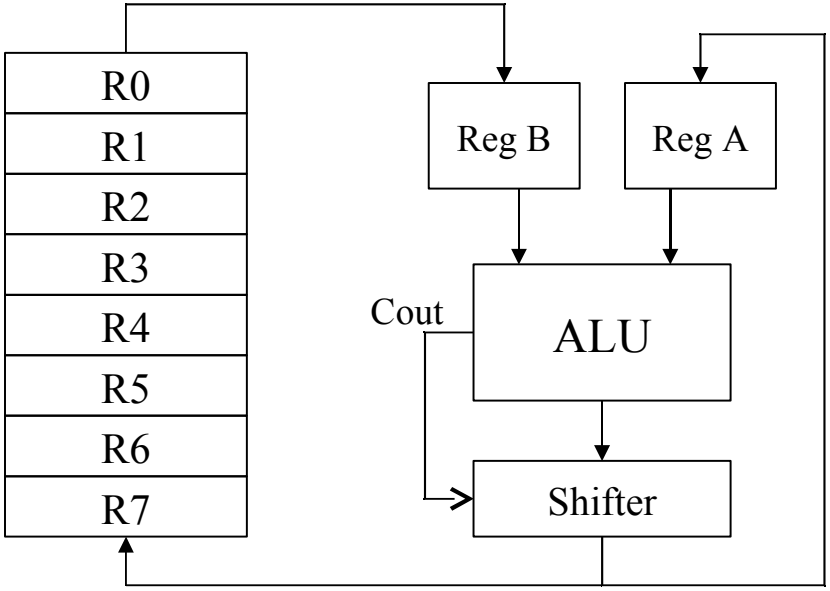


Ejercicio 5

Universidad del Valle

Es necesario calcular el valor promedio de cuatro números binarios sin signo almacenados en los registros R0, R1, R2 y R3 del procesador definido en la figura. El valor promedio se debe almacenar en el registro R4. Se debe tener cuidado de no causar sobrecapacidad. Utilizar el menor numero de estados posibles.

- Dar la lista de la secuencia de operaciones en RTL.
- Listar las señales de control binarias correspondientes



Ejercicio 5

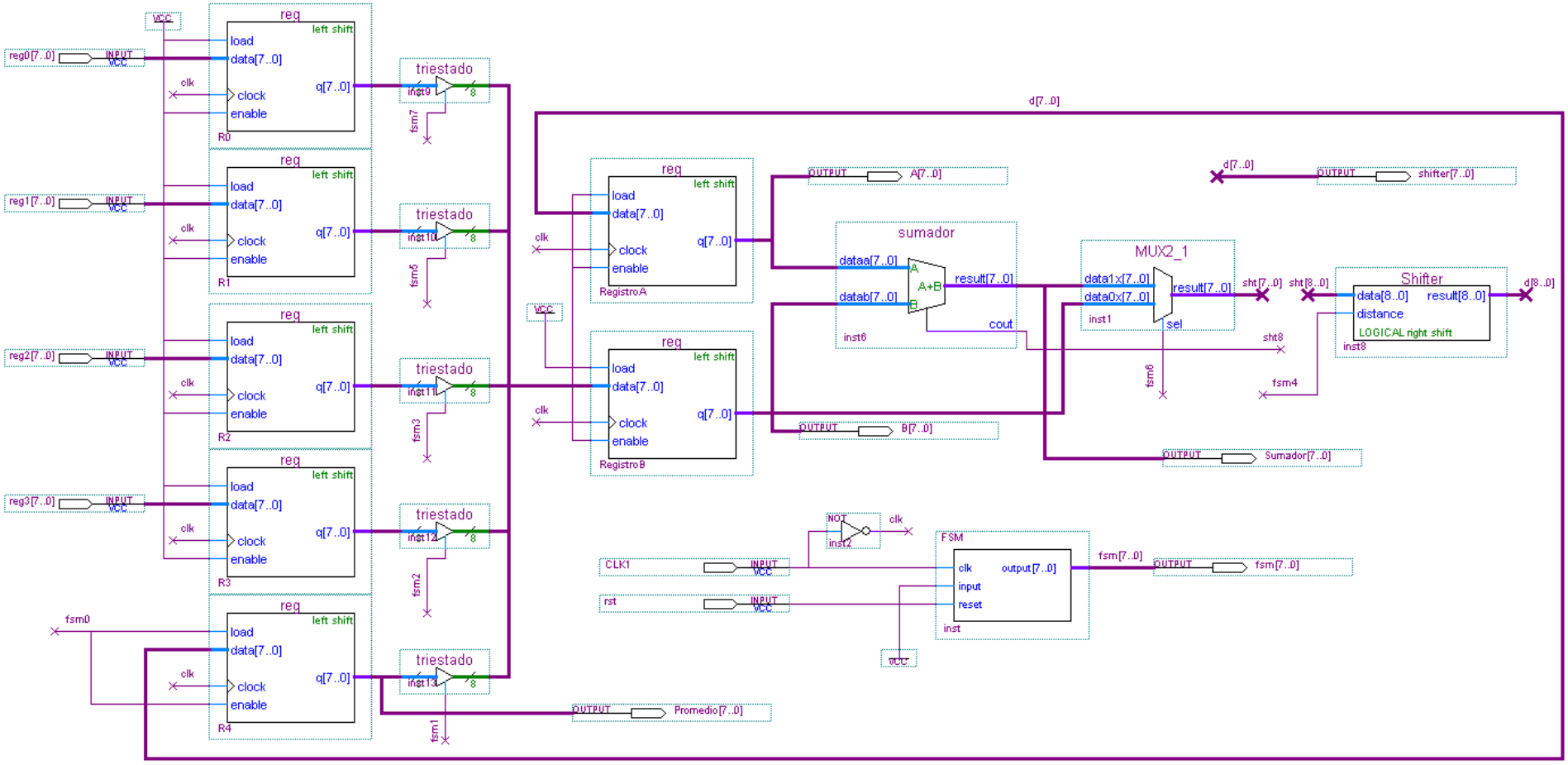
Universidad del Valle

Para evitar la sobrecapacidad en la ALU, primero se debe realizar el promedio entre R0 y R1, y almacenar el resultado temporal en el registro R4. Lo mismo se hace con los registros R2 y R3 almacenando el resultado en RA. Finalmente se calcula el promedio entre R4 y RA obteniendo el promedio total.

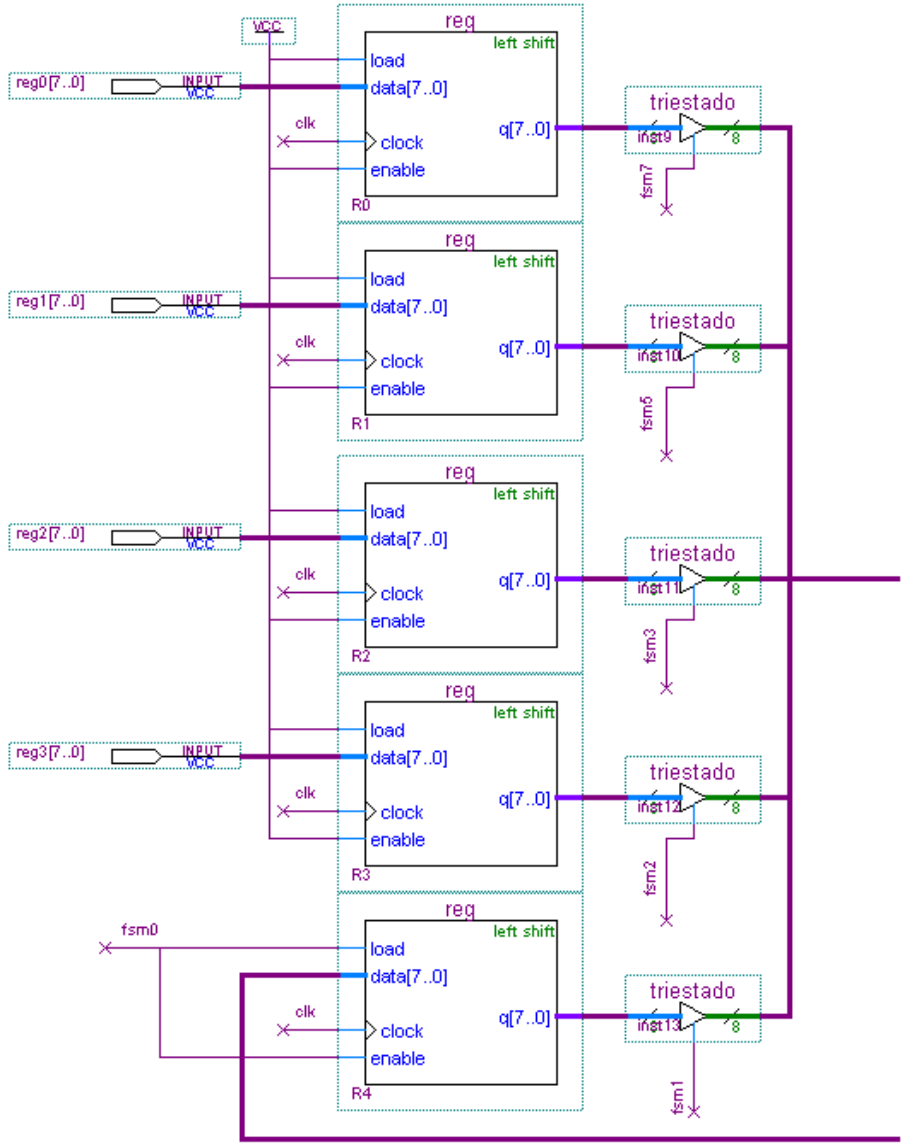
S0:	RB ← R0	OC_0
S1:	RA ← RB, RB ← R1	ALU = 0, OC_1
S2:	R4 ← shr(RA + RB), RB ← R2	EN_4, ALU = 1, SHR, OC_2
S3:	RA ← RB, RB ← R3	ALU = 0, OC_3
S4:	RA ← shr(RA + RB), RB ← R4	ALU = 1, SHR, OC_4
S5:	R4 ← shr(RA + RB)	ALU = 1, SHR, EN_4

ALU: 0: Transferir B (no suma)
1: Sumar A + B

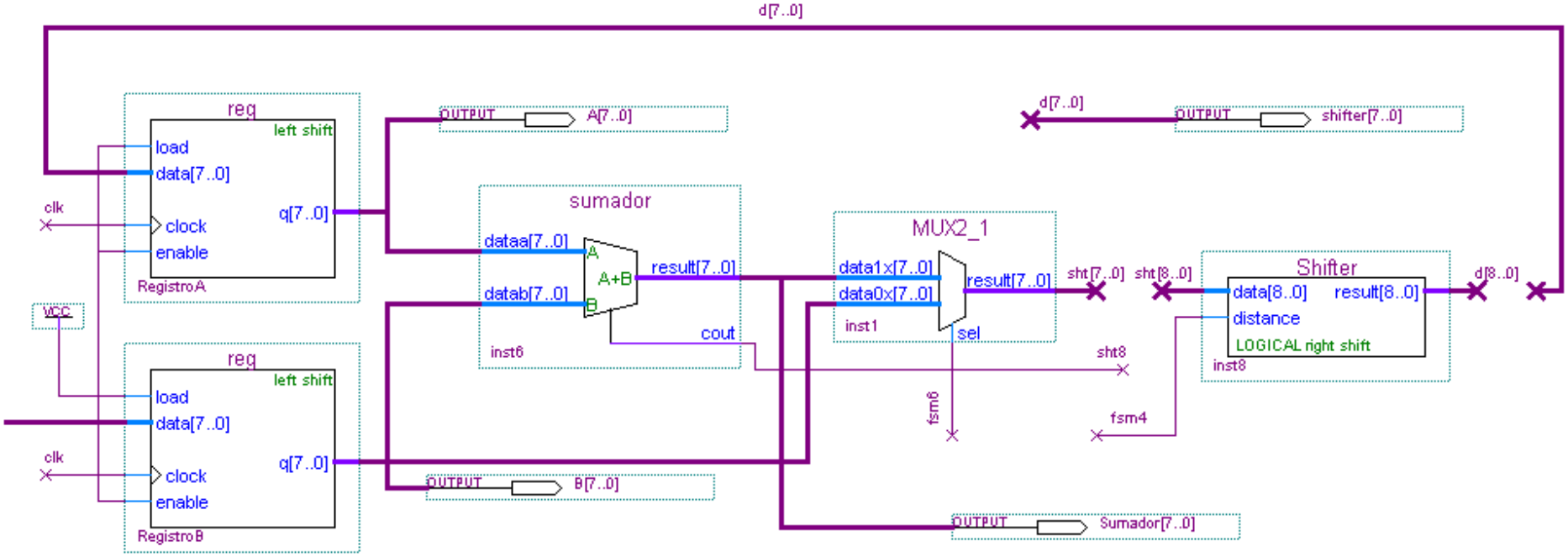
Ejercicio 5: Quartus II (Datapath)



Ejercicio 5: Quartus II (Banco de Registros)

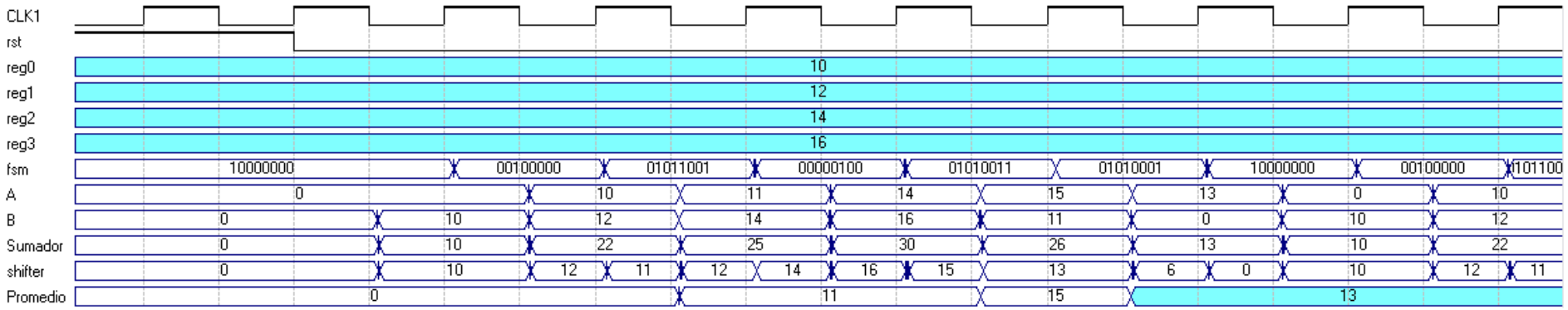


Ejercicio 5: Quartus II (Unidad de Ejecución)



Ejercicio 5: Simulación

Universidad del Valle

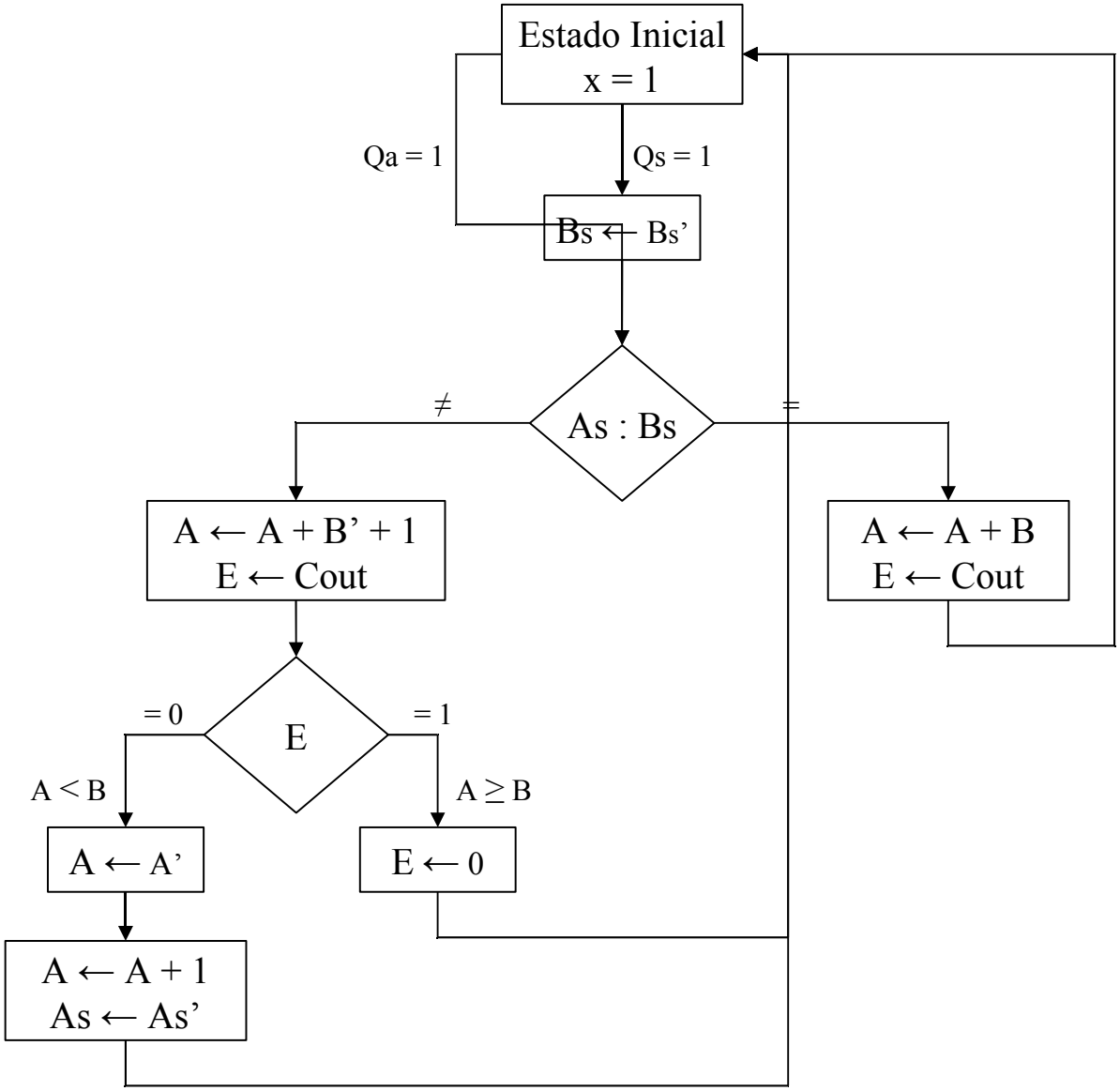


Ejercicio 6

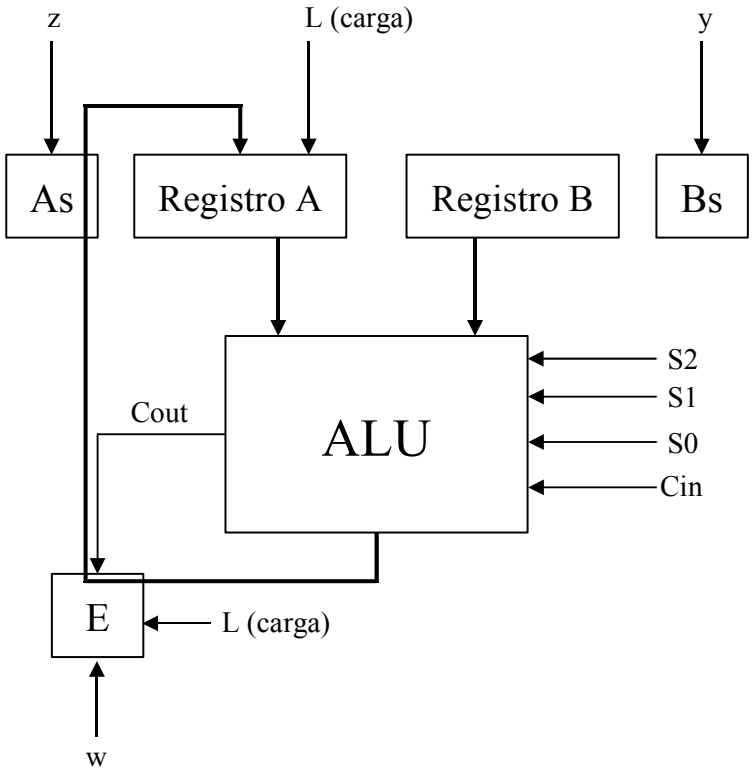
Universidad del Valle

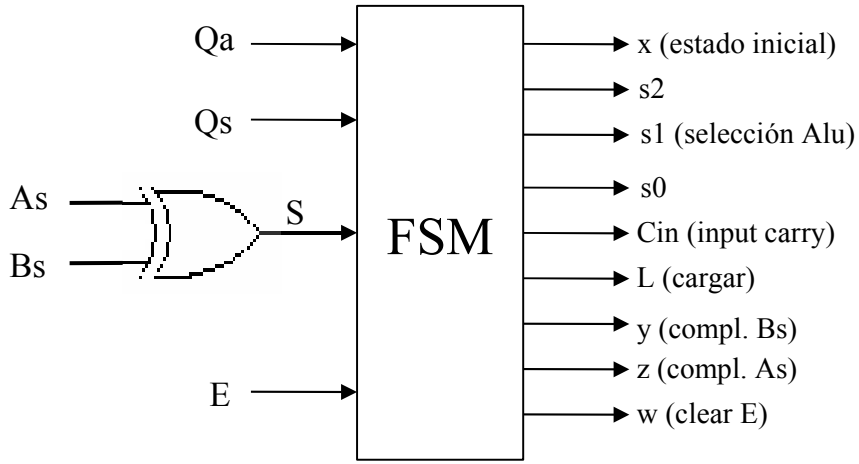
- Diseñar un circuito que realice la adición y sustracción de dos números binarios de punto fijo representado en forma de signo-magnitud. Se puede usar aritmética complementada siempre y cuando el resultado final esté en la forma de signo-magnitud. El circuito debe tener un flip-flop para almacenar el bit de desbordamiento por sobrecapacidad. (Controlador cableado)

Ejercicio 6

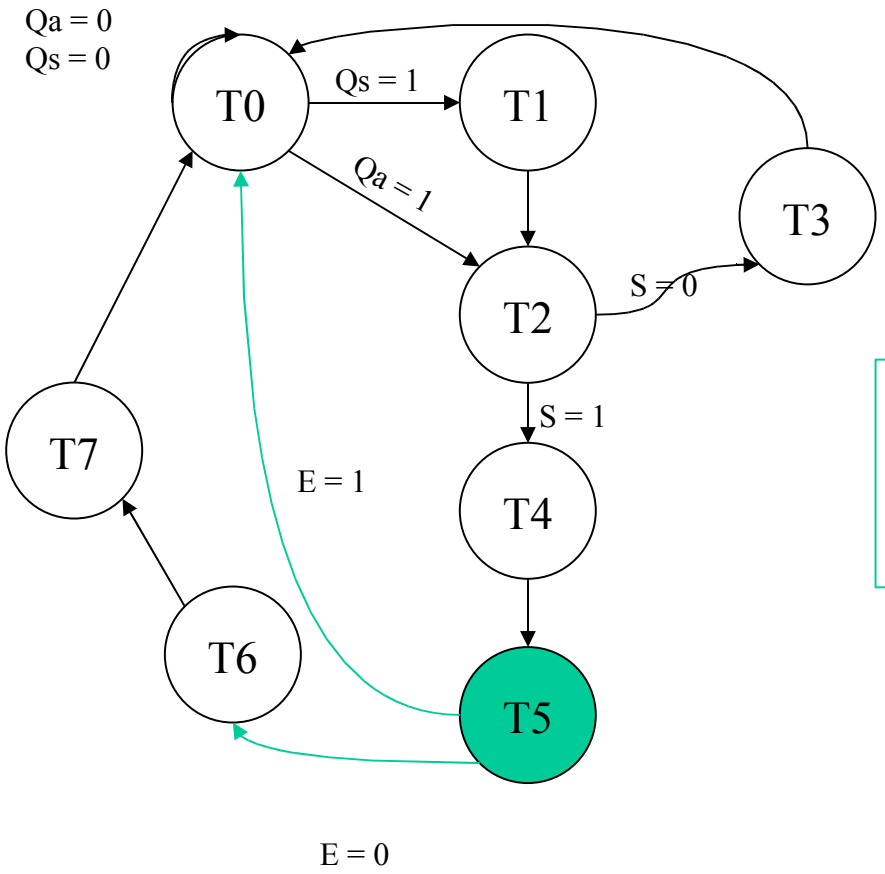

Estado Inicial
x = 1
Qa = 1
Qs = 1
Bs ← Bs'
As : Bs
≠
=
A ← A + B' + 1
E ← Cout
A ← A + B
E ← Cout
E
= 0
= 1
A < B
A ≥ B
A ← A'
E ← 0
A ← A + 1
As ← As'

Ejercicio 6


z
L (carga)
y
As
Registro A
Registro B
Bs
ALU
Cout
S2
S1
S0
Cin
E
L (carga)
w


Qa
Qs
As
Bs
S
FSM
E
x (estado inicial)
s2
s1 (selección Alu)
s0
Cin (input carry)
L (cargar)
y (compl. Bs)
z (compl. As)
w (clear E)

Ejercicio 6



Qa = 1	Sumar	(in3)
Qs = 1	Restar	(in2)
S = 0	Signos iguales	(in1)
S = 1	Signos diferentes	(in1)
E	Output Carry	(in0)

Ejercicio 6

Universidad del Valle

	x (fsm8)	s2	s1	s0	Cin	L	Y	z	W (fsm0)
T0: Estado Inicial x = 1	1	0	0	0	0	0	0	0	0
T1: Bs ← Bs'	0	0	0	0	0	0	1	0	0
T2: E ← 0	0	0	0	0	0	0	0	1	0
T3: A ← A + B, E ← Cout	0	0	0	1	0	1	0	0	0
T4: A ← A + B' + 1, E ← Cout	0	0	1	0	1	1	0	0	0
T5: nada	0	0	0	0	0	0	0	0	0
T6: A ← A'	0	1	1	1	0	1	0	0	0
T7: A ← A + 1, As ← As'	0	0	0	0	1	1	0	1	0

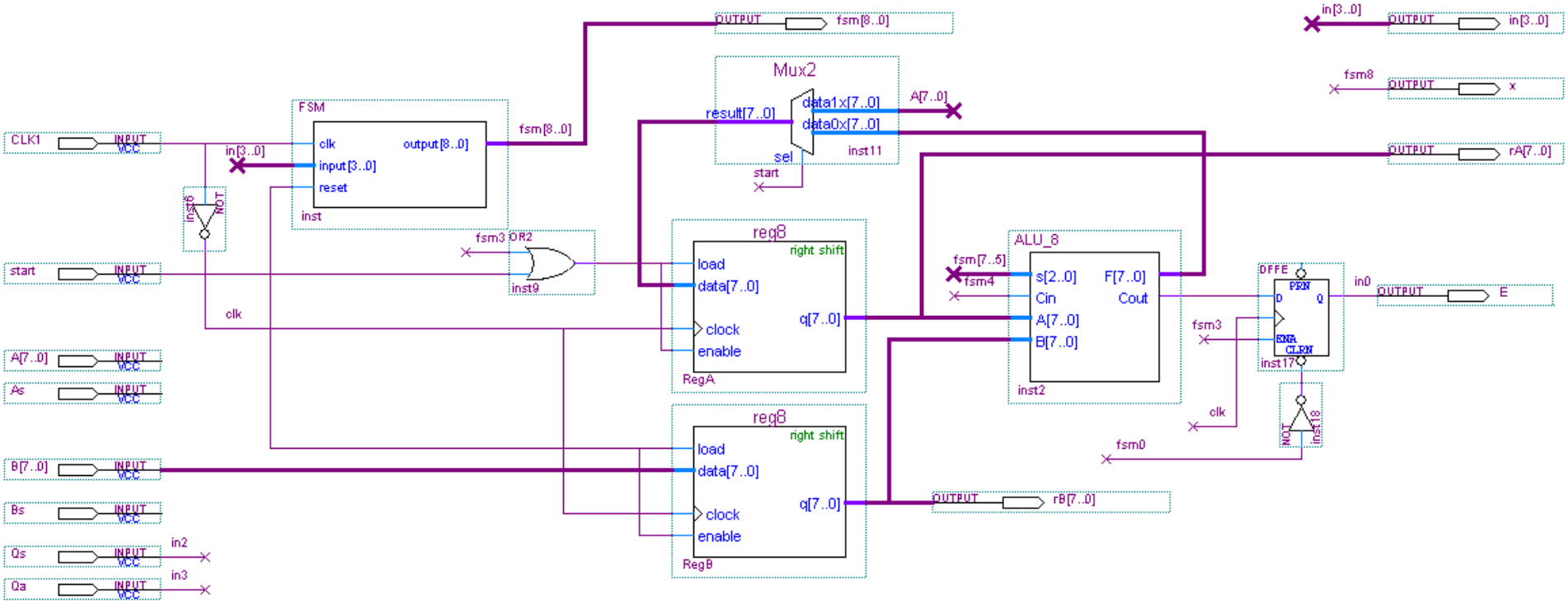
Salidas de la FSM por estado

Ejercicio 6: Funciones de la ALU

Universidad del Valle

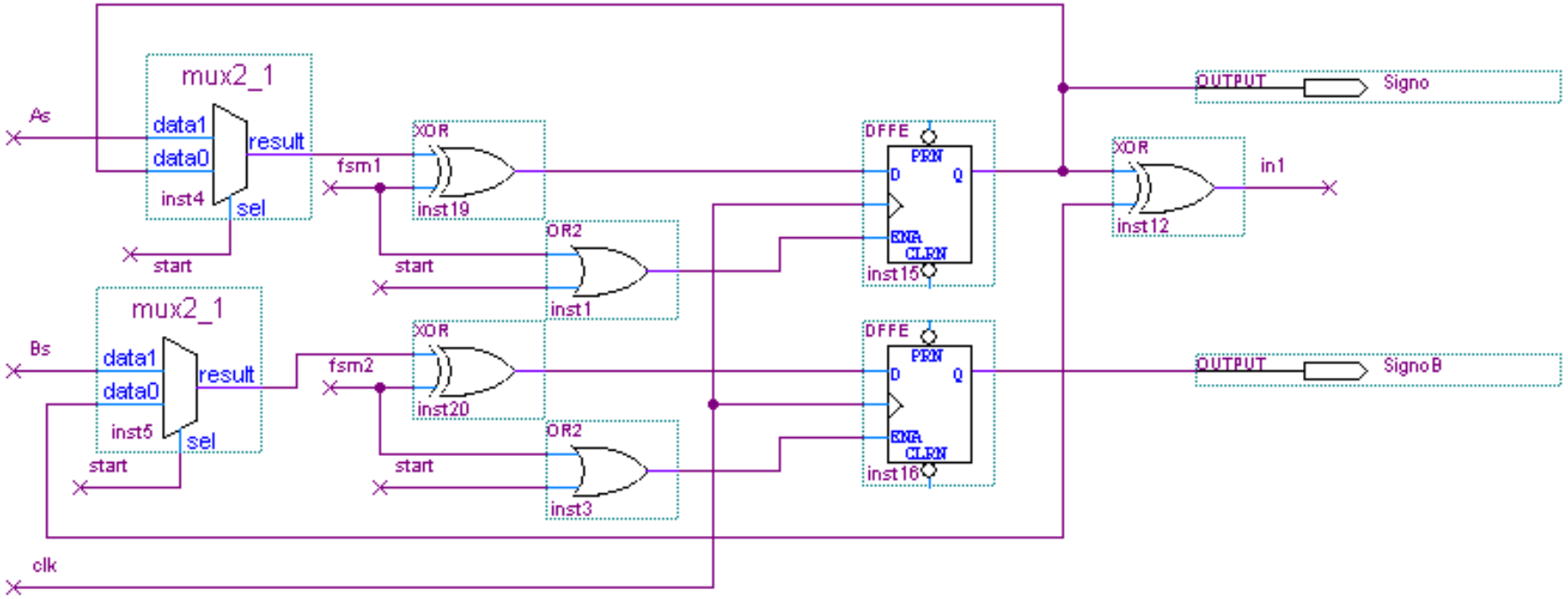
Selección					
S2	**S1**	**S0**	**Cin**	**Salida**	**Función**
0	0	0	0	F=A	Transferir A
0	0	0	1	F=A+1	Incrementar A
0	0	1	0	F=A+B	Suma
0	0	1	1	F=A+B+1	Suma con carry
0	1	0	0	F=A-B-1	Resta con
0	1	0	1	F=A-B	Sustracción
0	1	1	0	F=A-1	Decrementar A
0	1	1	1	F=A	Transferir A
1	0	0	X	F=A OR B	OR
1	0	1	X	F=A XOR B	XOR
1	1	0	X	F=A AND B	AND
1	1	1	X	F=A’	Complementar A

Ejercicio 6: Quartus II (Datapath)

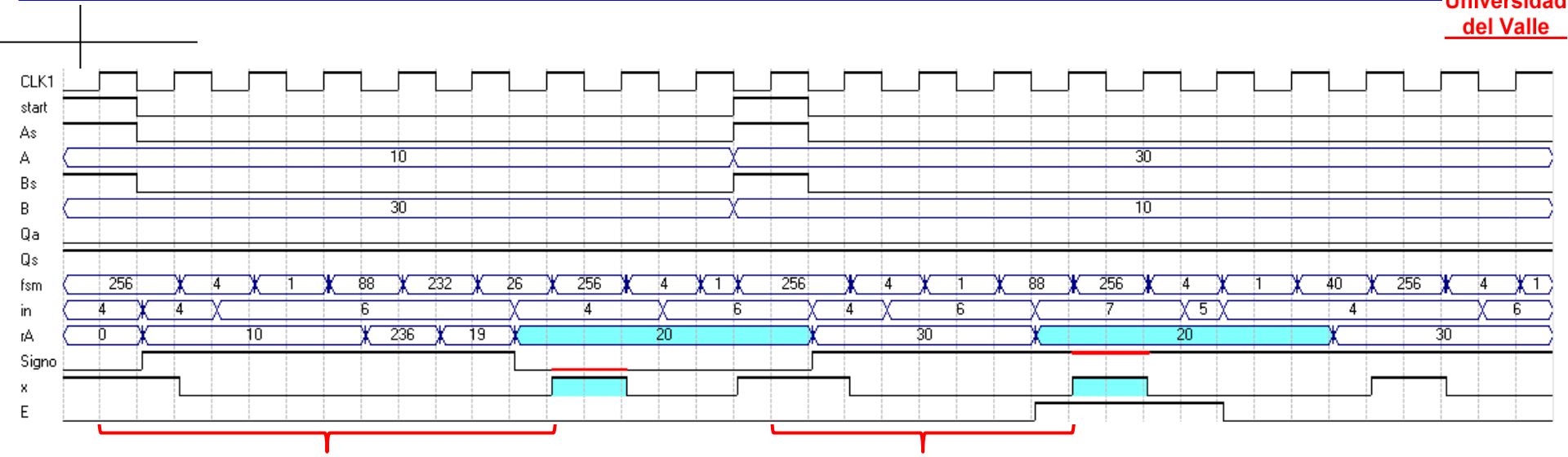


Ejercicio 6: Quartus II (Flip-flops de Signo)

Universidad del Valle



Ejercicio 6: Simulación



A= -10 - (-30)

A = -30 - (-10)

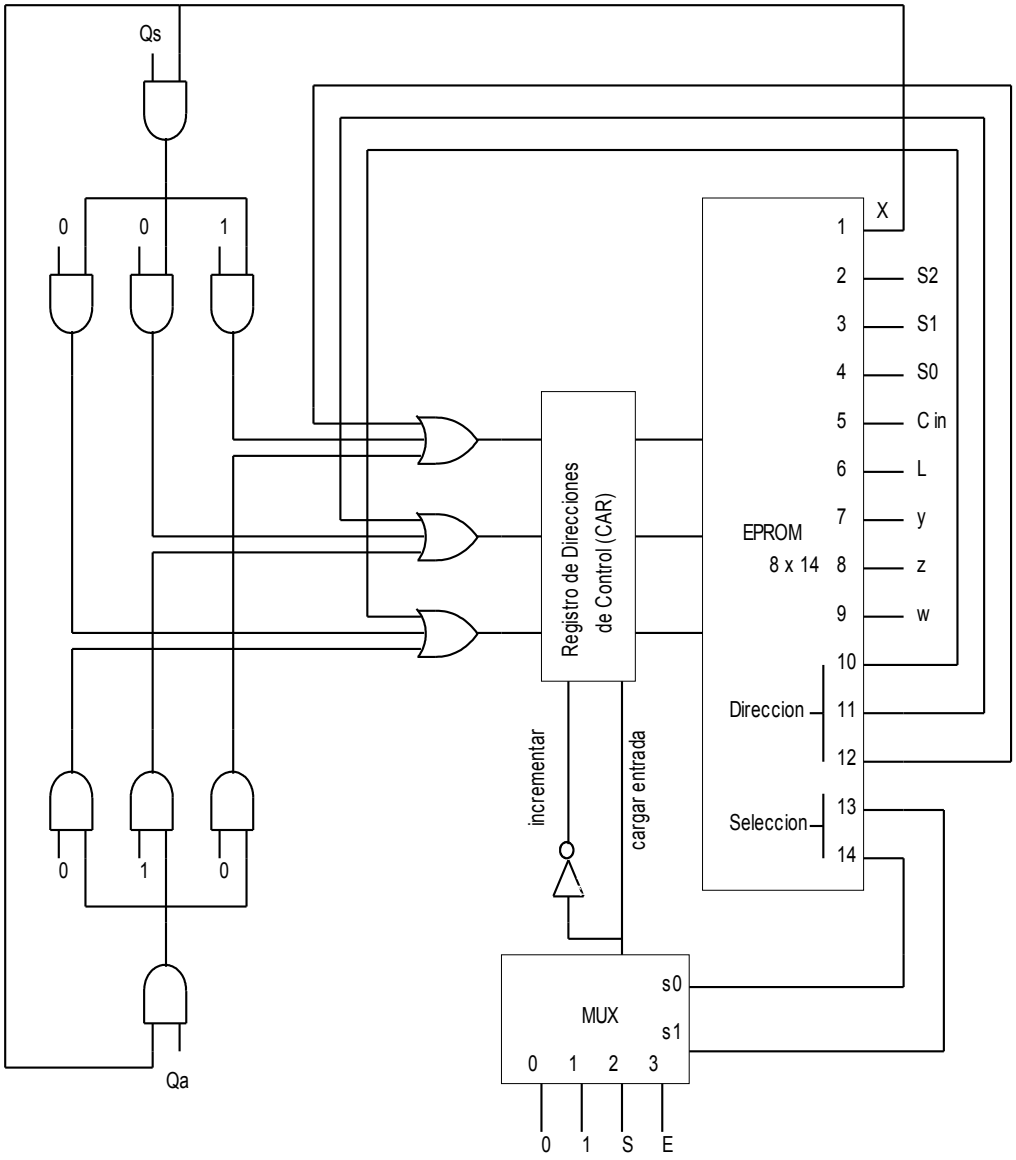
Start : comenzar operación
As : signo de A
Bs : signo de B
Qa : sumar
Qs : restar
fsm : salidas de la FSM
In : entradas de la FSM
rA : contenido final del reg. A
Signo: signo final de A
X : señal de T0 (indica el comienzo y el final de una operación)

Ejercicio 7

Universidad del Valle

- Implementar el diseño del ejercicio anterior utilizando un registro de desplazamiento para la secuencia del algoritmo y una memoria ROM para la lógica combinacional de las señales de control necesarias. El registro tiene capacidad de carga en paralelo con el fin de poder hacer “saltos” en la secuencia del programa. Se puede utilizar otros componentes si es necesario (Controlador micro-programado).

Ejercicio 7


Qs
0
0
1
0
1
0
Qa
Registro de Direcciones de Control (CAR)
incrementar
cargar entrada
MUX
s0
s1
0
1
2
3
0
1
S
E
EPROM
8 x 14
X
1
2
3
4
5
6
7
8
9
10
11
12
13
14
S2
S1
S0
C in
L
y
z
w
Direccion
Seleccion

Ejercicio 7

Bits de ROM 13	Bits de ROM 14	Función de selección del MUX
0	0	Incrementar el CAR
0	1	Cargar la entrada al CAR
1	0	Cargar las entradas al CAR si S=1, incrementar el CAR si S=0
1	1	Cargar las entradas al CAR si E=1, incrementar el CAR si E=0

Dirección De ROM	Microinstrucción	Comentarios
0	x=1; si (Qs=1) entonces (va a 1); si (Qa=1) entonces (va a 2); si (Qs OR Qa =0) entonces (va a 0)	Cargar 0 o dirección externa
1	Bs ← Bs'	Qs = 1, comenzar sustracción
2	si (S = 1) entonces (va a 4)	Qa = 1, comenzar suma
3	A ← A + B; E ← Cout; va a 0	Sumar magnitudes y regreso
4	A ← A + B' + 1; E ← Cout	Sustraer Magnitudes
5	Si (E = 1) entonces (va a 0); E ← 0	Operación Finalizada si E = 1
6	A ← A'	E = 0, complementar A
7	A ← A + 1; As ← As'; va a 0	Terminado, regresar a la dirección 0

Microprograma simbólico para la memoria de control

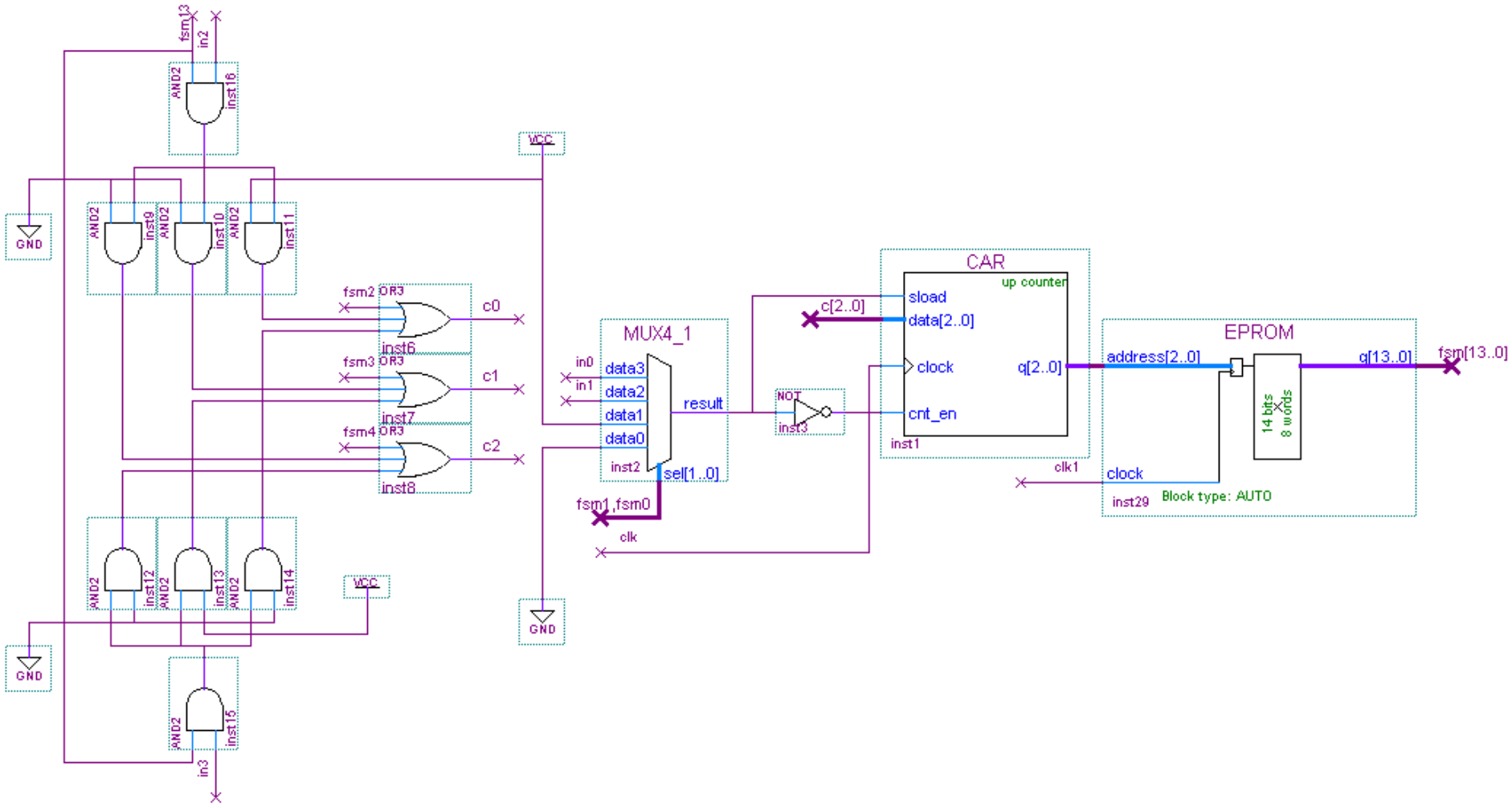
Ejercicio 7

Dirección EPROM			Salidas de EPROM												
			›	s2	s1	s0	Cin	L	y	z	w	Dirección			Selección	
			(	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0	'	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	(	0	0	0	0	0	1	0	0	0	1	0	0	1
0	1	0	(	0	0	0	0	0	0	0	1	1	0	0	1	0
0	1	1	(	0	0	1	0	1	0	0	0	0	0	0	0	1
1	0	0	(	0	1	0	1	1	0	0	0	1	0	1	0	1
1	0	1	(	0	0	0	0	0	0	0	0	0	0	0	1	1
1	1	0	(	1	1	1	0	0	0	0	0	1	1	1	0	1
1	1	1	(	0	0	0	1	1	0	1	0	0	0	0	0	1

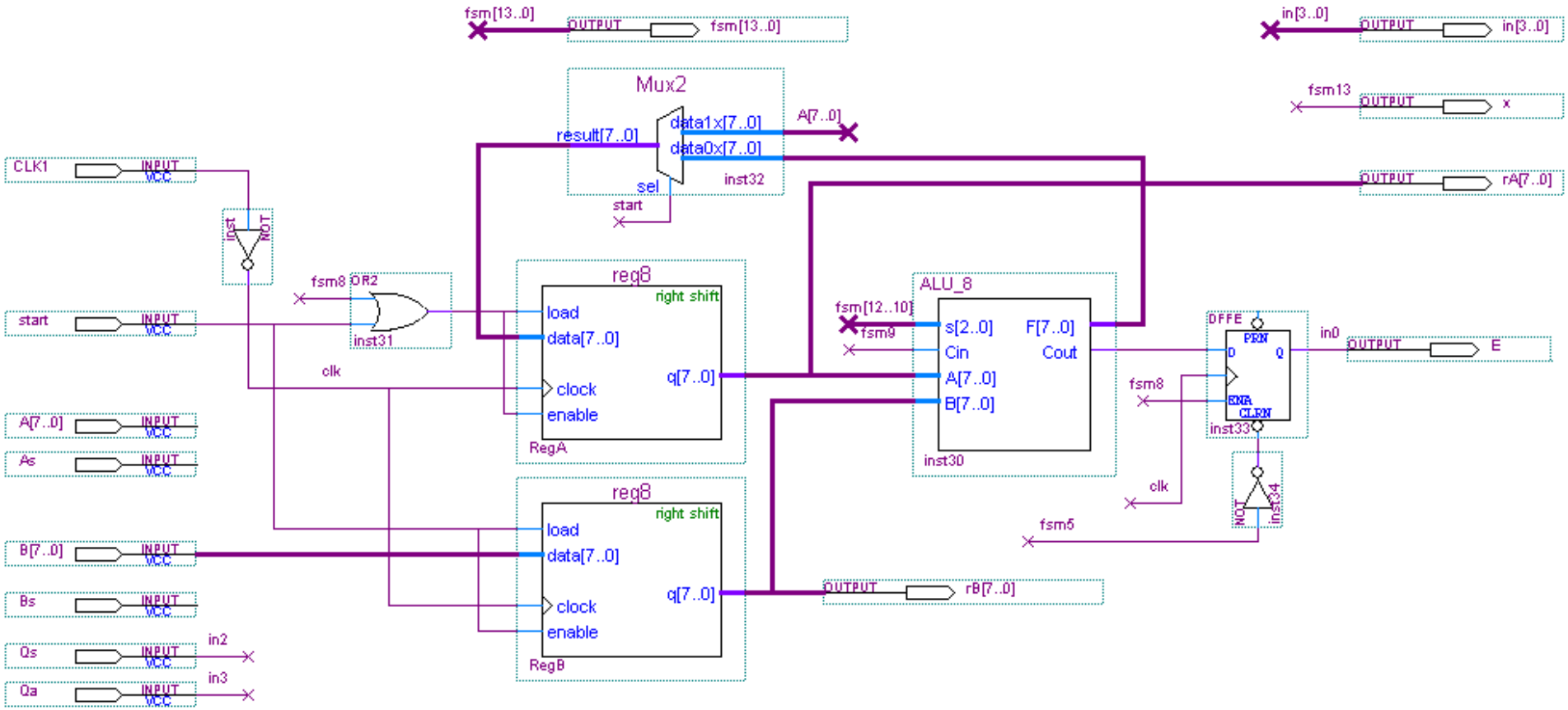
Microprograma binario para la memoria de control

Ejercicio 7: Quartus II (Control Path)

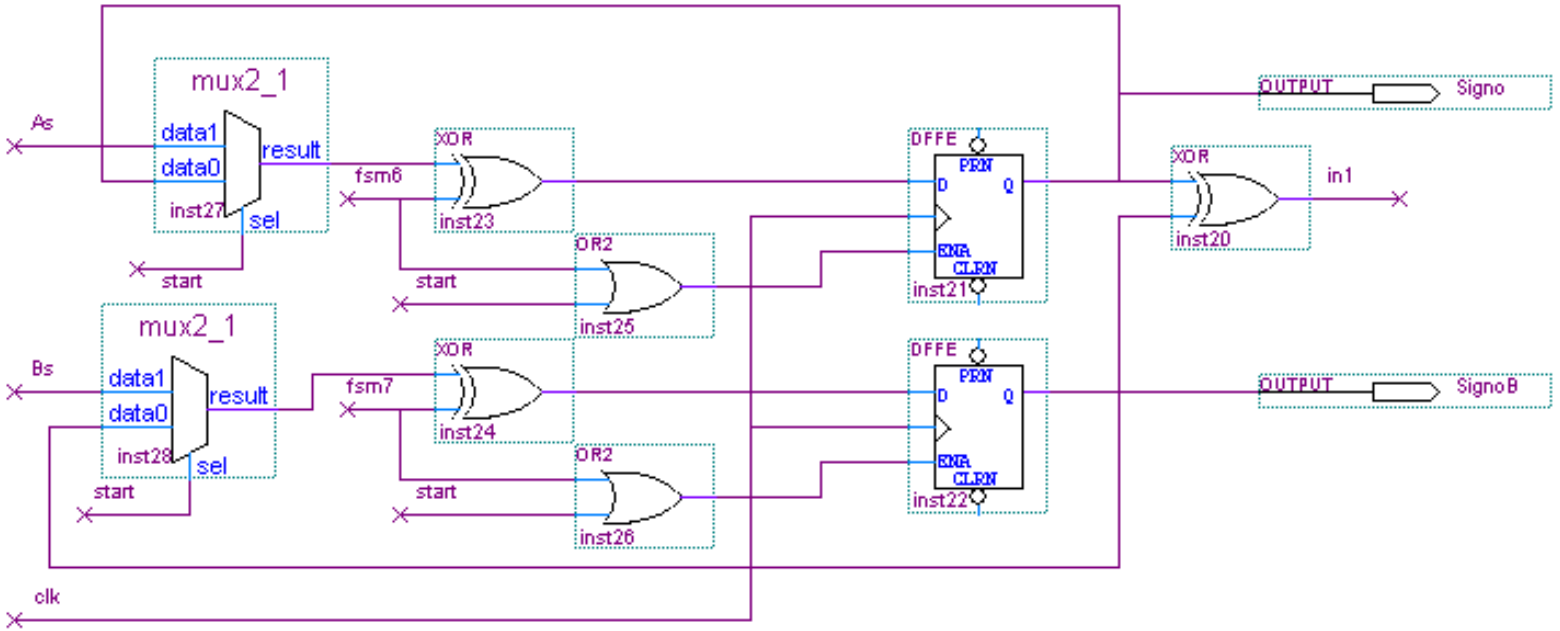
Universidad del Valle



Ejercicio 7: Quartus II (Data Path)

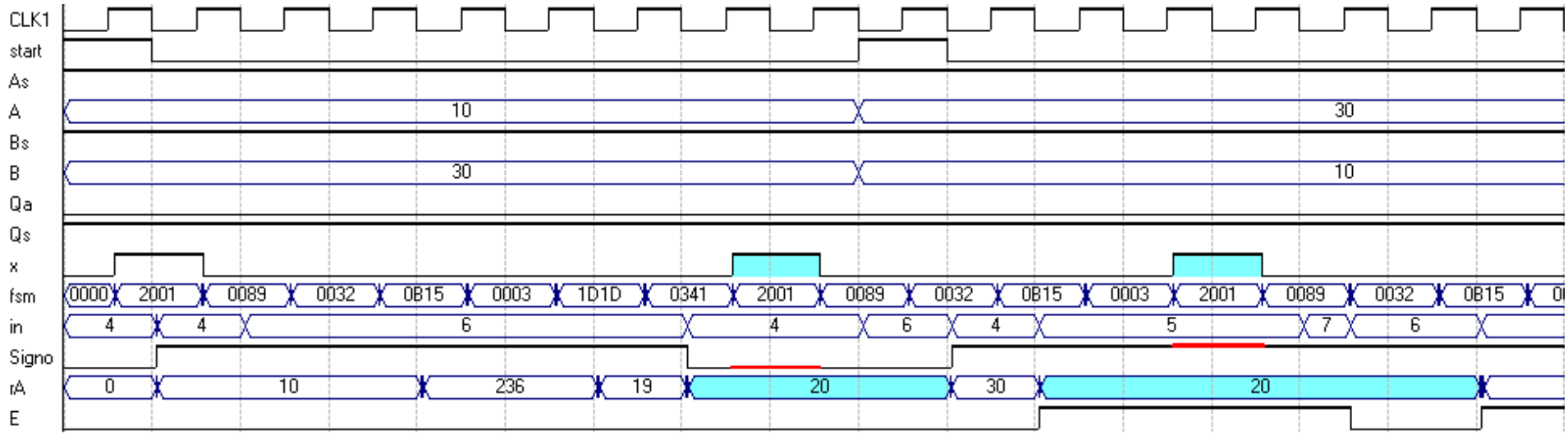


Ejercicio 7: Quartus II (Flip-flops de signo)



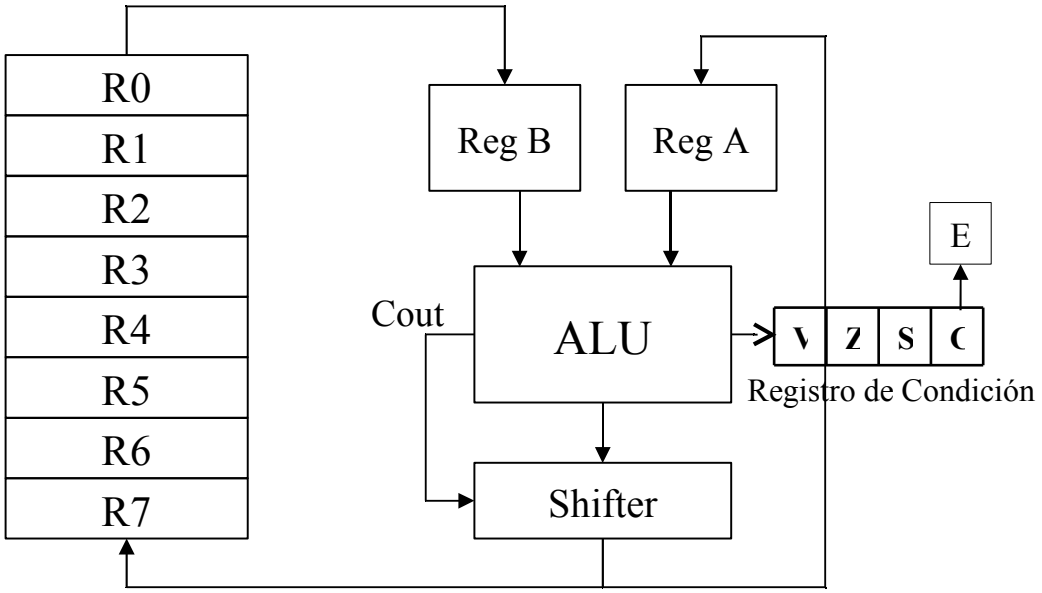
Ejercicio 7: Simulación

Universidad del Valle



Ejercicio 8

- Diseñe el control (Diagrama ASM) para un circuito que compara dos números binarios sin signo almacenados en R0 y R1. El registro que contiene el numero menor se borra. Si los dos números son iguales se borran ambos registros. Indique las señales de control asociadas a cada estado.

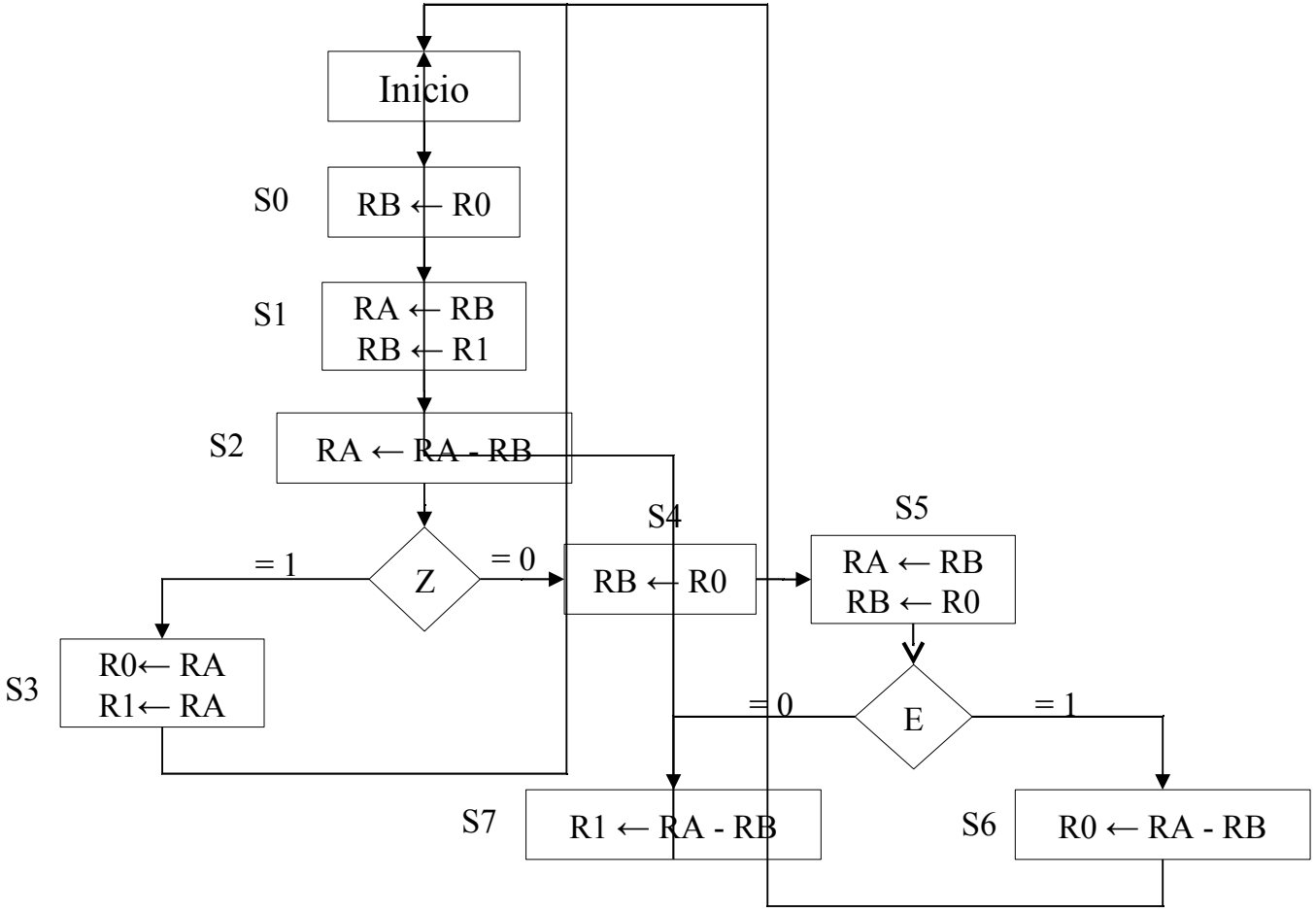


C: output carry
S: signo
Z: cero
V: sobrecapacidad

ALU: 00: Transferir A
01: Sumar A +B
10: Restar A – B
11: Transferir B

Ejercicio 8

- Se debe tener en cuenta que en una operación de sustracción de dos números binarios sin signo, su magnitud relativa está determinada por el carry de salida y no por el flag de signo. De esta manera, si el carry es 1 es porque, o los números son iguales, o B es menor que A. Por el contrario si el carry es 0 es porque A es menor que B.



Ejercicio 8

Universidad del Valle

S0:	RB ← R0	OC_0
S1:	RA ← RB, RB ← R1	ALU=11, OC_1
S2:	RA ← RA – RB	ALU=10, load E

(El contenido del carry se transfiere al flipflop E.)

S3:	R0← RA, R1← RA	EN_0, EN_1, ALU=00

(Como el resultado es cero, se transfiere a R0 y R1 para borrarlos)

S4:	RB ← R0	OC_0
S5:	RA ← RB, RB ← R0	ALU=11, OC_0

(Generación de un cero para borrar a R0 o a R1)

S6:	R0 ← RA – RB	ALU=10, EN_0
S7:	R1 ← RA – RB	ALU=10, EN_1

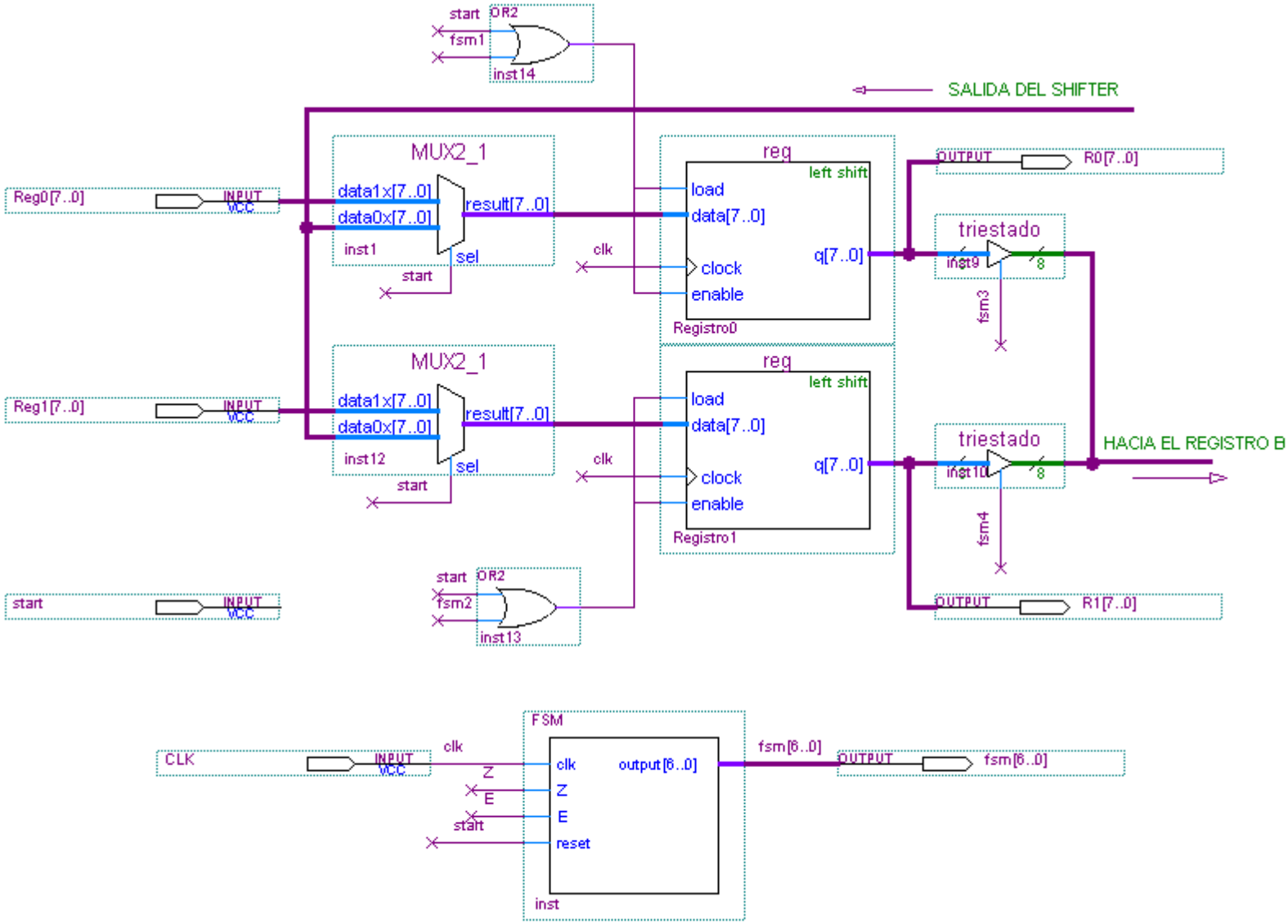
Ejercicio 8

Universidad del Valle

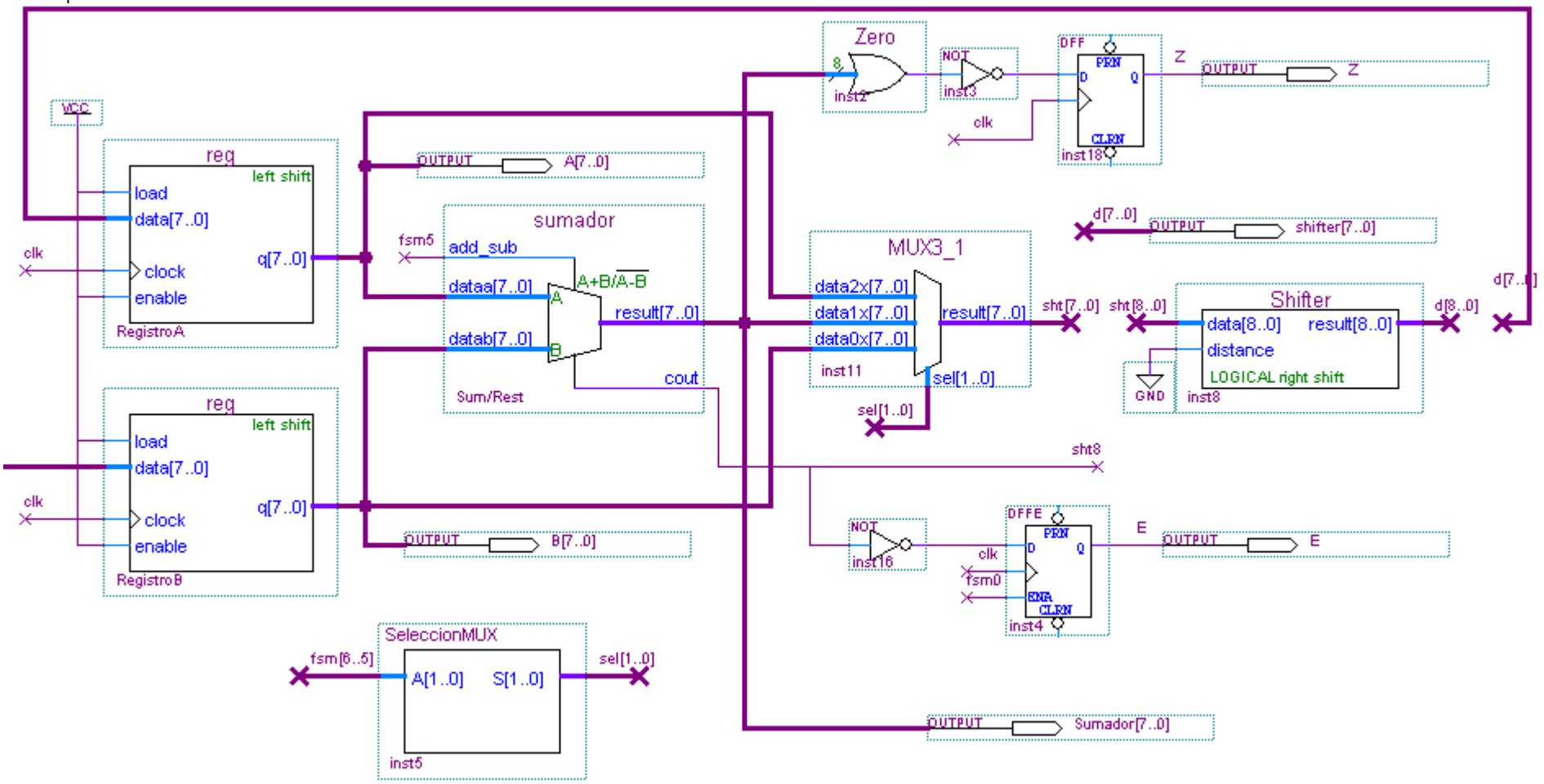
Estado	ALU_1	ALU_0	OE_1	OE_0	EN_1	EN_0	LE
S0	0	0	0	1	0	0	0
S1	1	1	1	0	0	0	0
S2	1	0	0	0	0	0	1
S3	0	0	0	0	1	1	0
S4	0	0	0	1	0	0	0
S5	1	1	0	1	0	0	0
S6	1	0	0	0	0	1	0
S7	1	0	0	0	1	0	0

Asignación de señales de control por estado

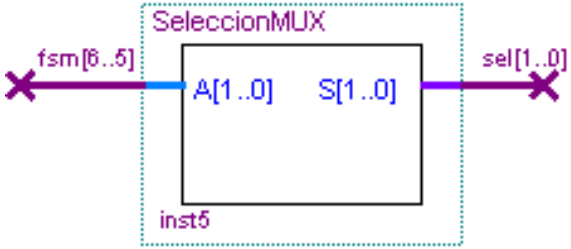
Ejercicio 8: Quartus II (Data Path 1)



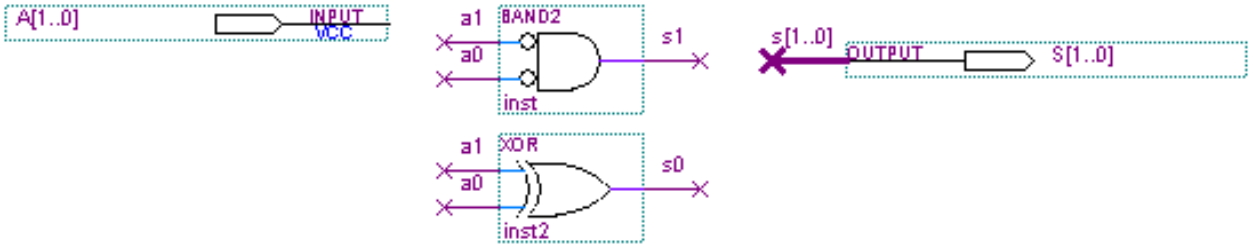
Ejercicio 8: Quartus II (Data Path 2)



Ejercicio 8: Conversor entre ALU – MUX

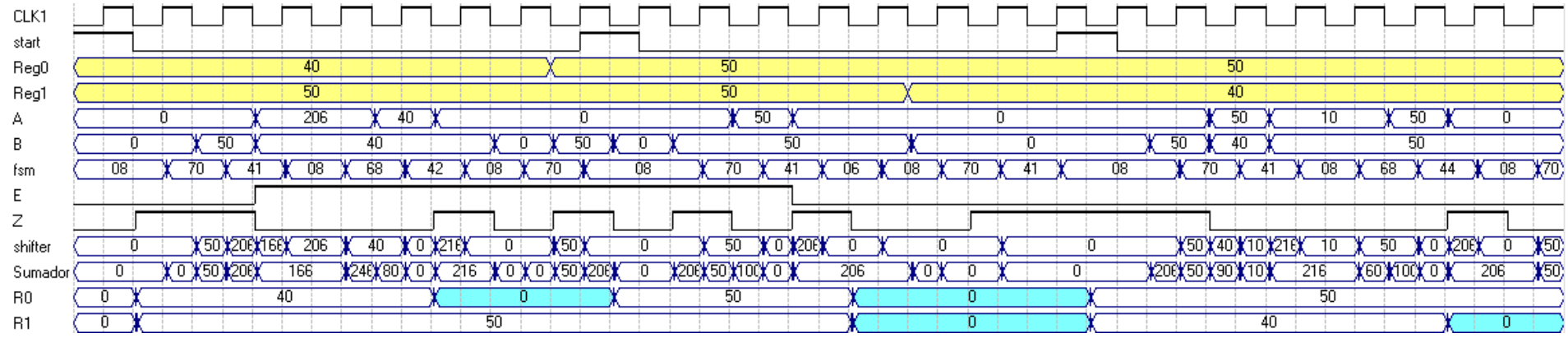


ALU_1	ALU_0	S_1	S_0
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	0



Ejercicio 8: Simulación

Universidad del Valle



Ejercicio 9

- Diseñe un circuito que ejecute la multiplicación de dos números binarios de 8 bits por el método de sumas sucesivas.

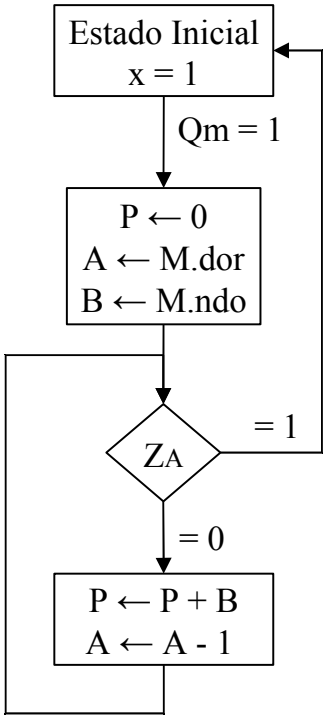


Diagrama ASM

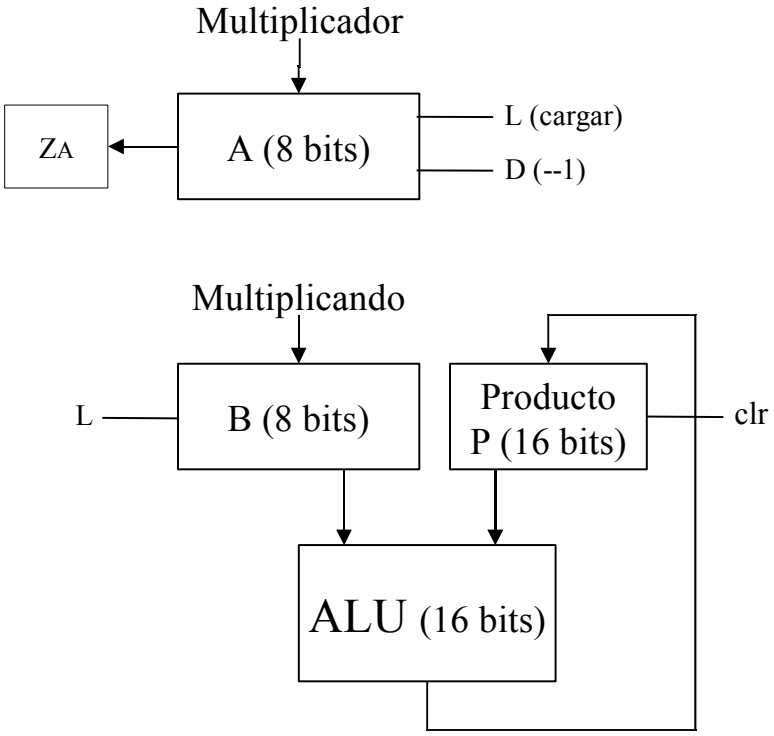
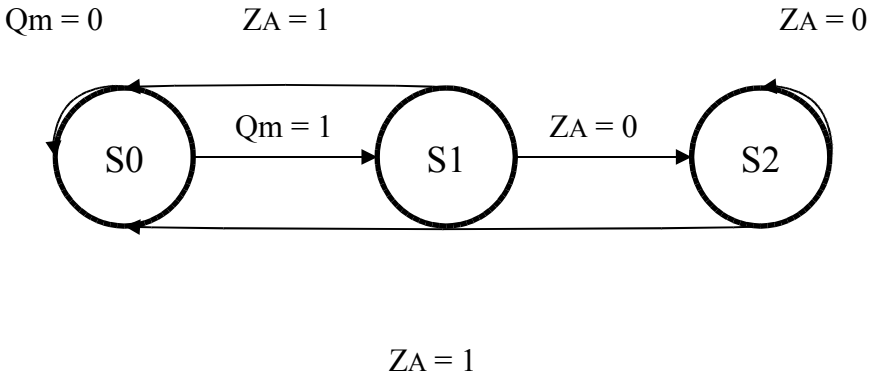


Diagrama de Bloques

Ejercicio 9



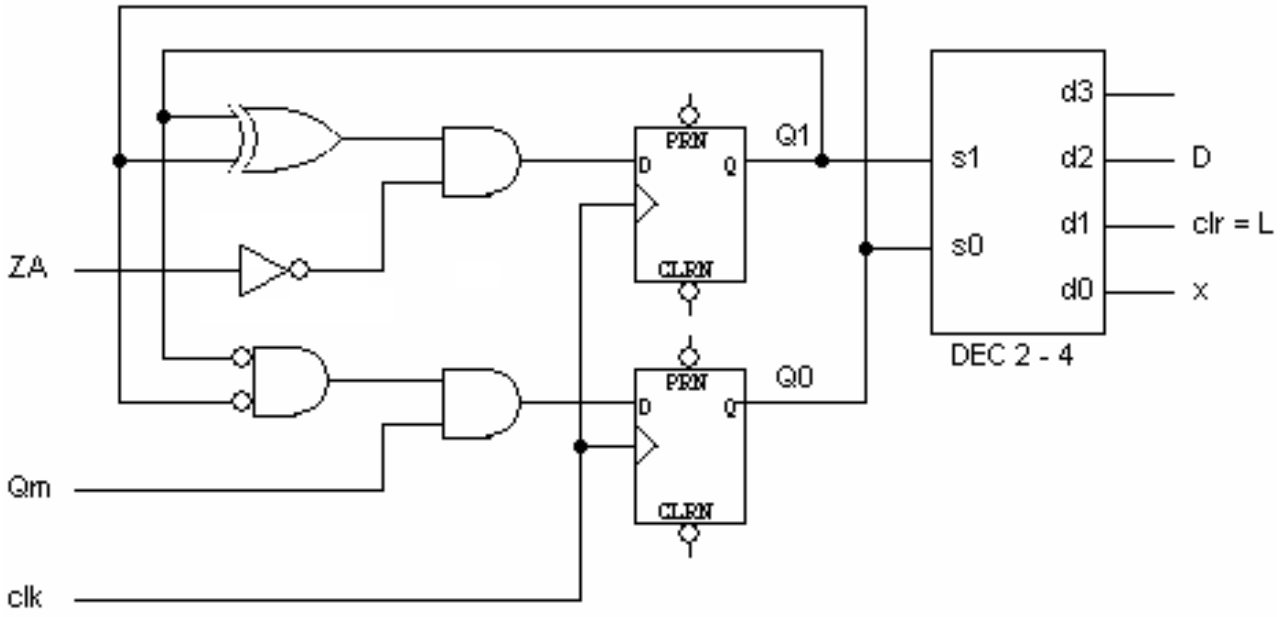
Estado			Salidas			
	Q1	Q0	x	clr	L	D
S0	0	0	1	0	0	0
S1	0	1	0	1	1	0
S2	1	0	0	0	0	1

Ejercicio 9

Estado Presente		Entradas		Próximo Estado		Salidas			
Q1	Q0	Qm	ZA	Q1+	Q0+	x	clr	L	D
0	0	0	X	0	0	1	0	0	0
0	0	1	X	0	1	1	0	0	0
0	1	X	0	1	0	0	1	1	0
0	1	X	1	0	0	0	1	1	0
1	0	X	0	1	0	0	0	0	1
1	0	X	1	0	0	0	0	0	1

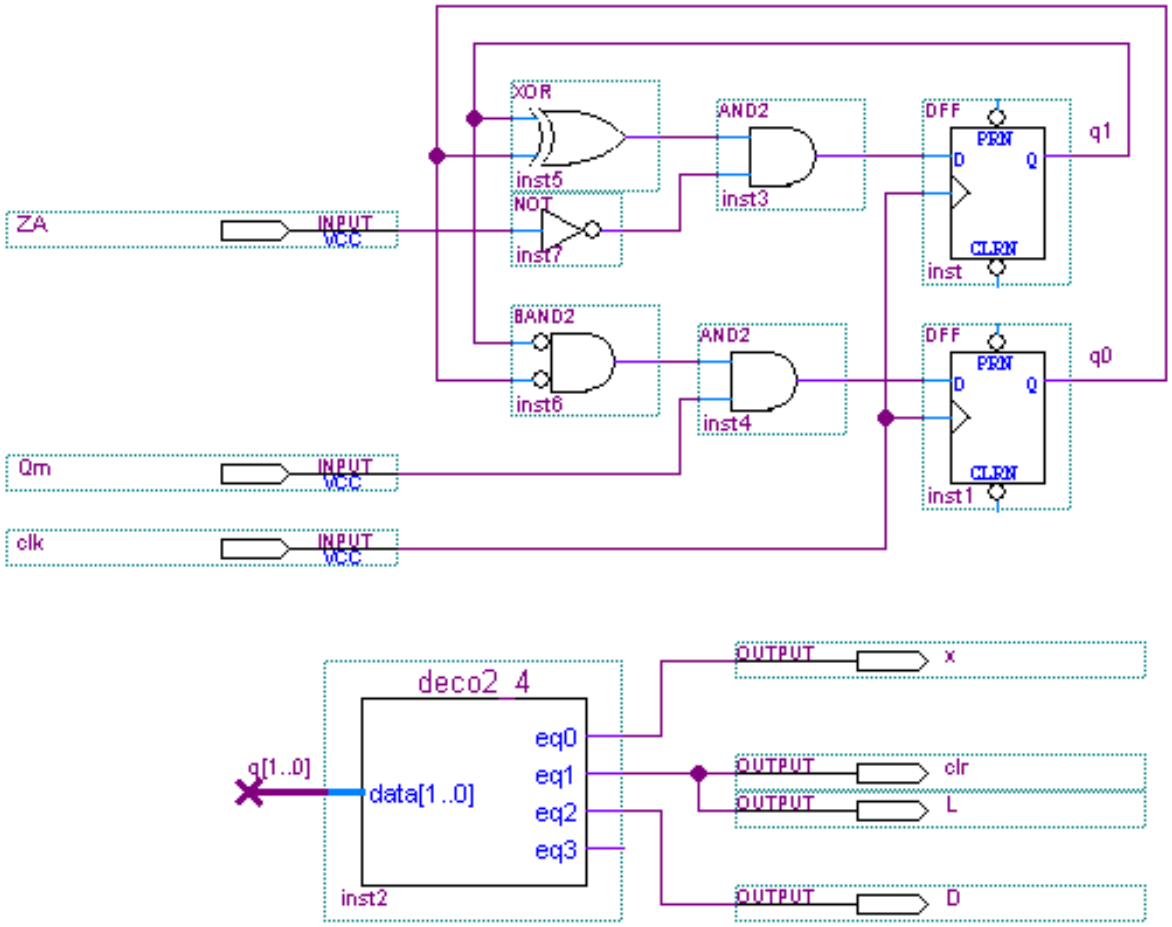
Tabla de Estados

Ejercicio 9

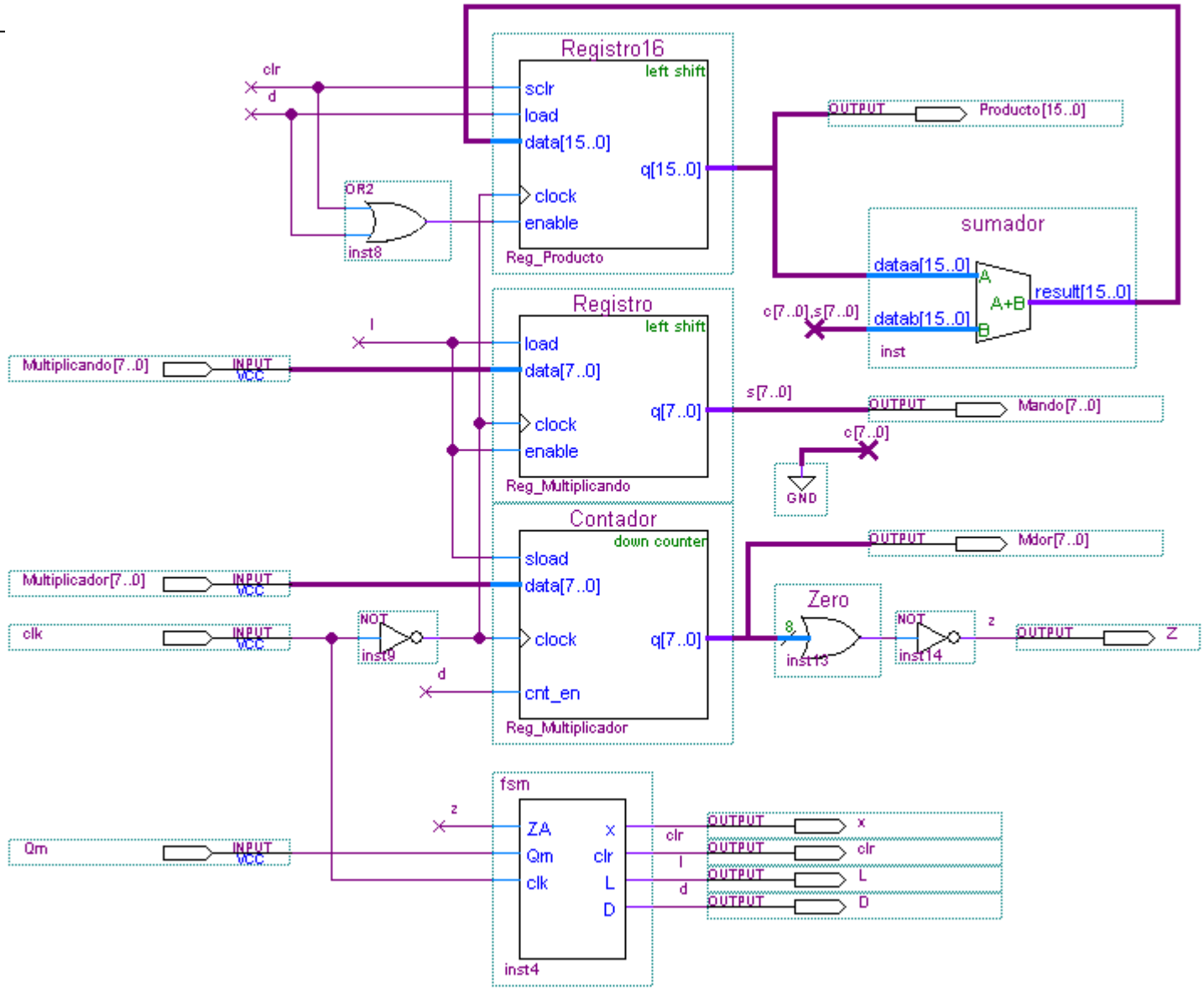


Circuito controlador

Ejercicio 9: Quartus II (FSM)

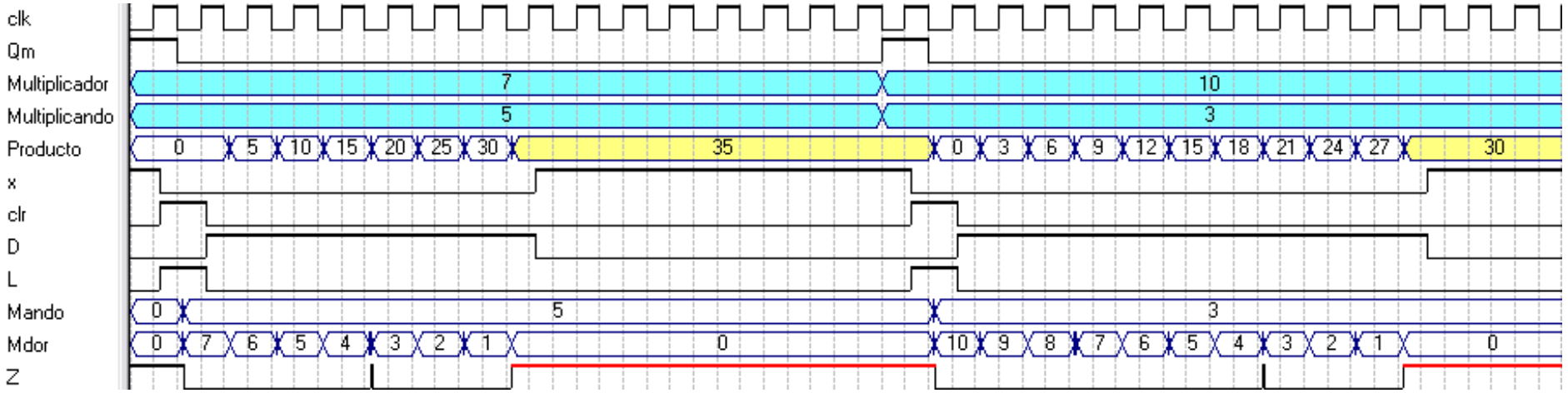


Ejercicio 9: Quartus II (DataPath)



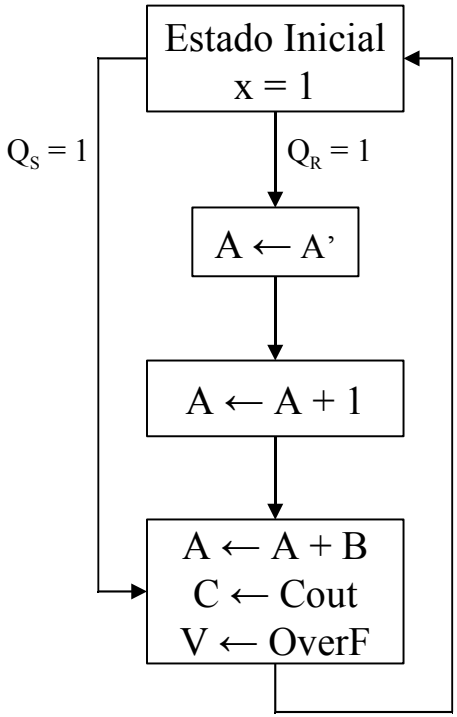
Ejercicio 9: Simulación

Universidad del Valle

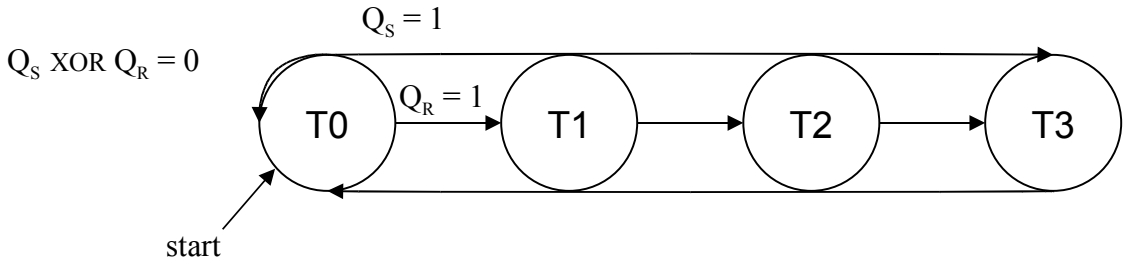


Ejercicio 10

- Diseñe un sistema digital que sume y reste dos números binarios de punto fijo representados en la forma de signo-complemento de 2 (B - A). Incluya una indicación de sobrecapacidad.



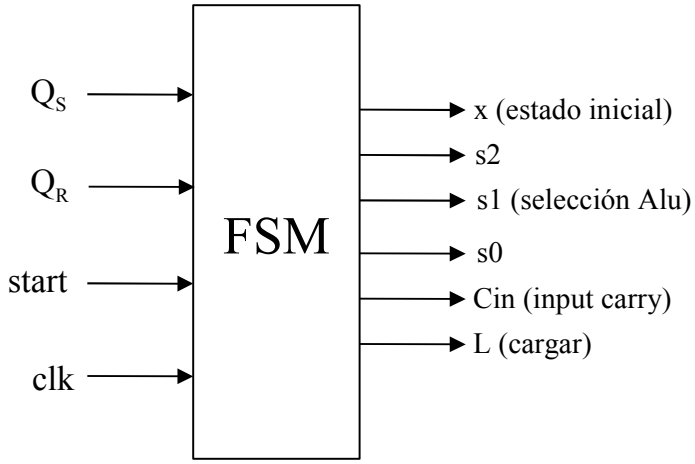
Ejercicio 10: FSM

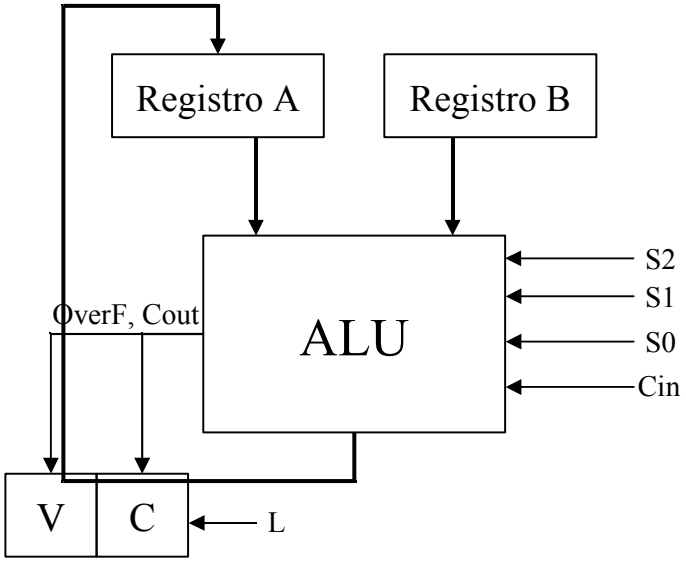


Estado	x	S2	S1	S0	Cin	L
T0	1	0	0	0	0	0
T1	0	1	1	1	0	0
T2	0	0	0	0	1	0
T3	0	0	0	1	0	1

Salidas por Estado

Ejercicio 10: DataPath



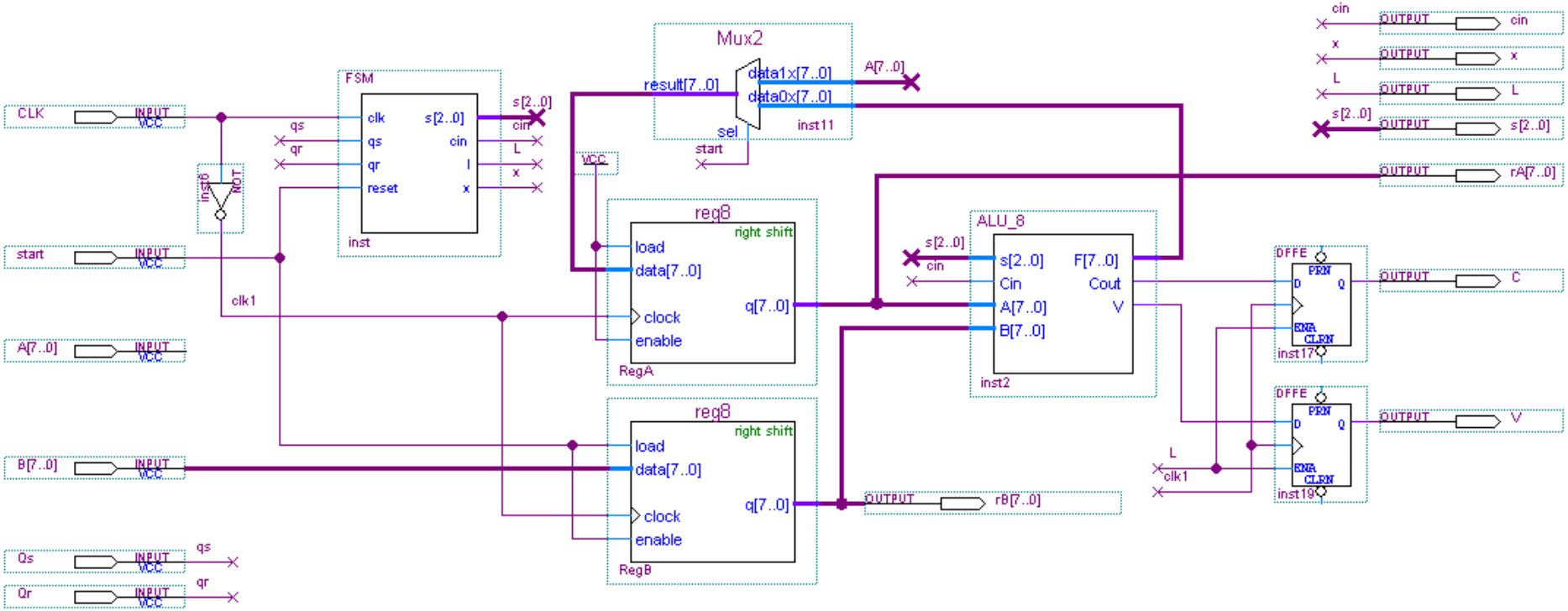


Ejercicio 10: Funciones de la ALU

Universidad del Valle

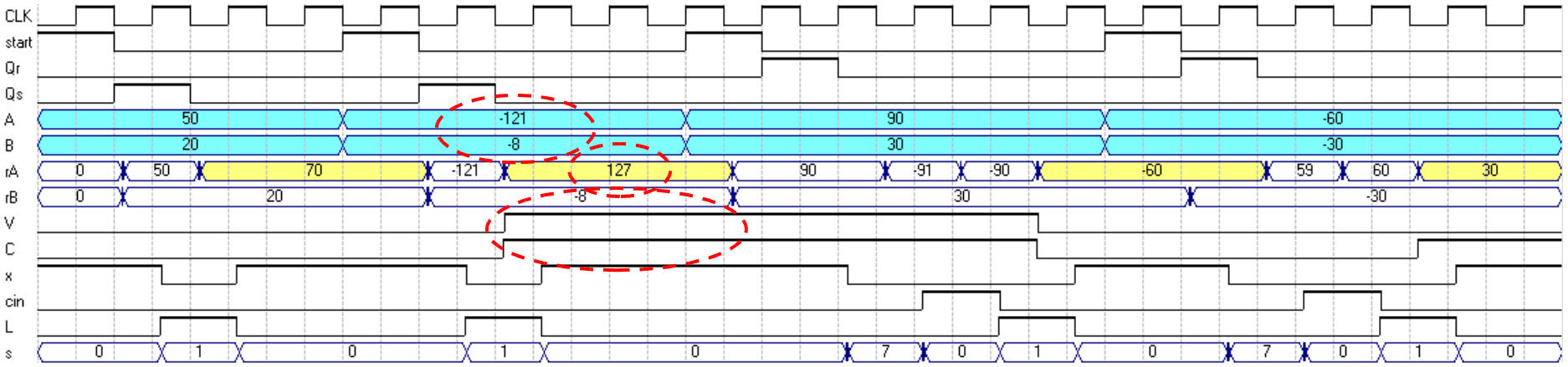
Selección					
S2	**S1**	**S0**	**Cin**	**Salida**	**Función**
0	0	0	0	F=A	Transferir A
0	0	0	1	F=A+1	Incrementar A
0	0	1	0	F=A+B	Suma
0	0	1	1	F=A+B+1	Suma con carry
0	1	0	0	F=A-B-1	Resta con Préstamo
0	1	0	1	F=A-B	Sustracción
0	1	1	0	F=A-1	Decrementar A
0	1	1	1	F=A	Transferir A
1	0	0	X	F=A OR B	OR
1	0	1	X	F=A XOR B	XOR
1	1	0	X	F=A AND B	AND
1	1	1	X	F=A’	Complementar A

Ejercicio 10: Quartus II: DataPath



Ejercicio 10: Simulación

Universidad del Valle



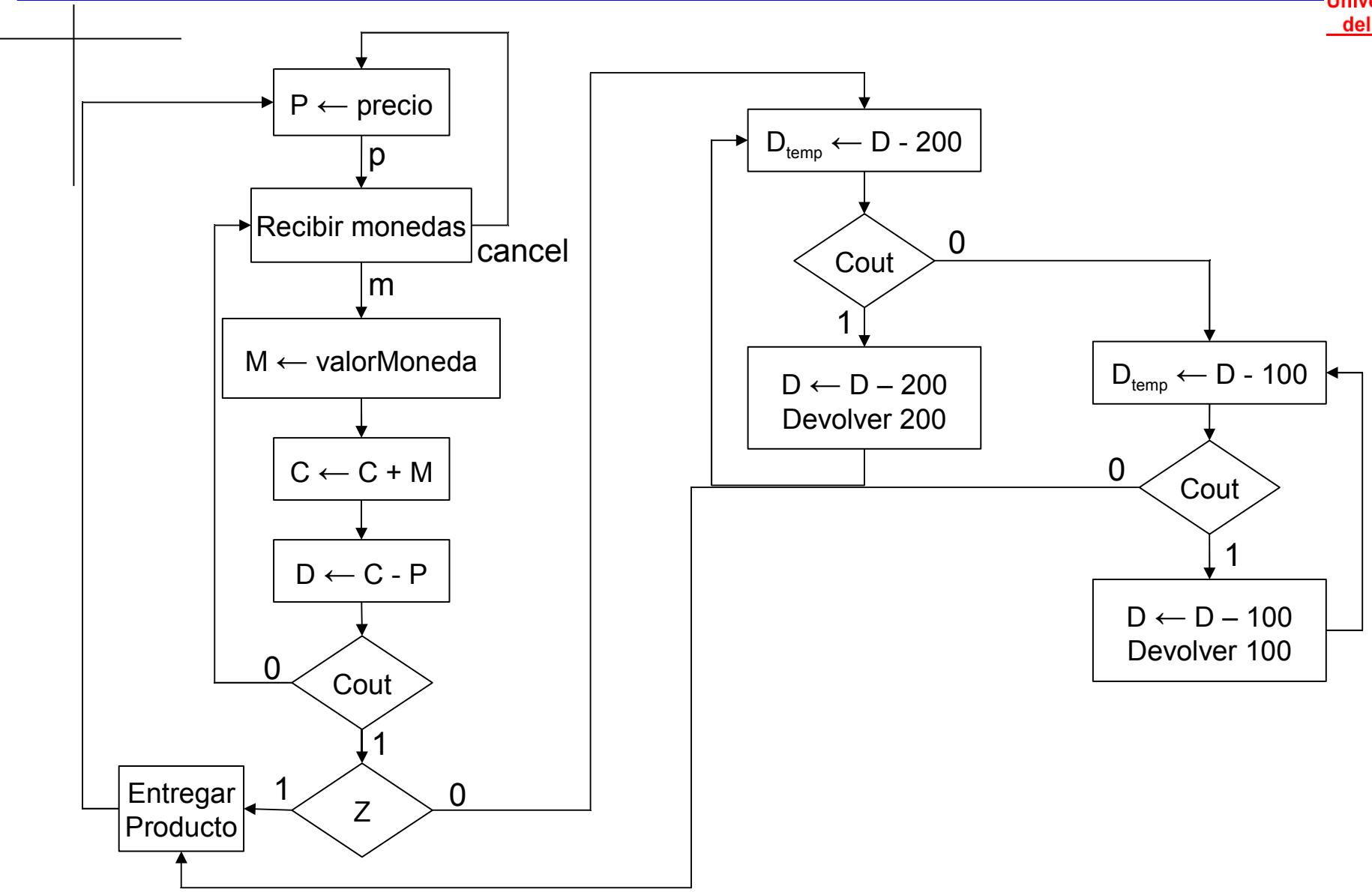
Ejercicio 11: Diseño de una maquina de gaseosas

Universidad del Valle

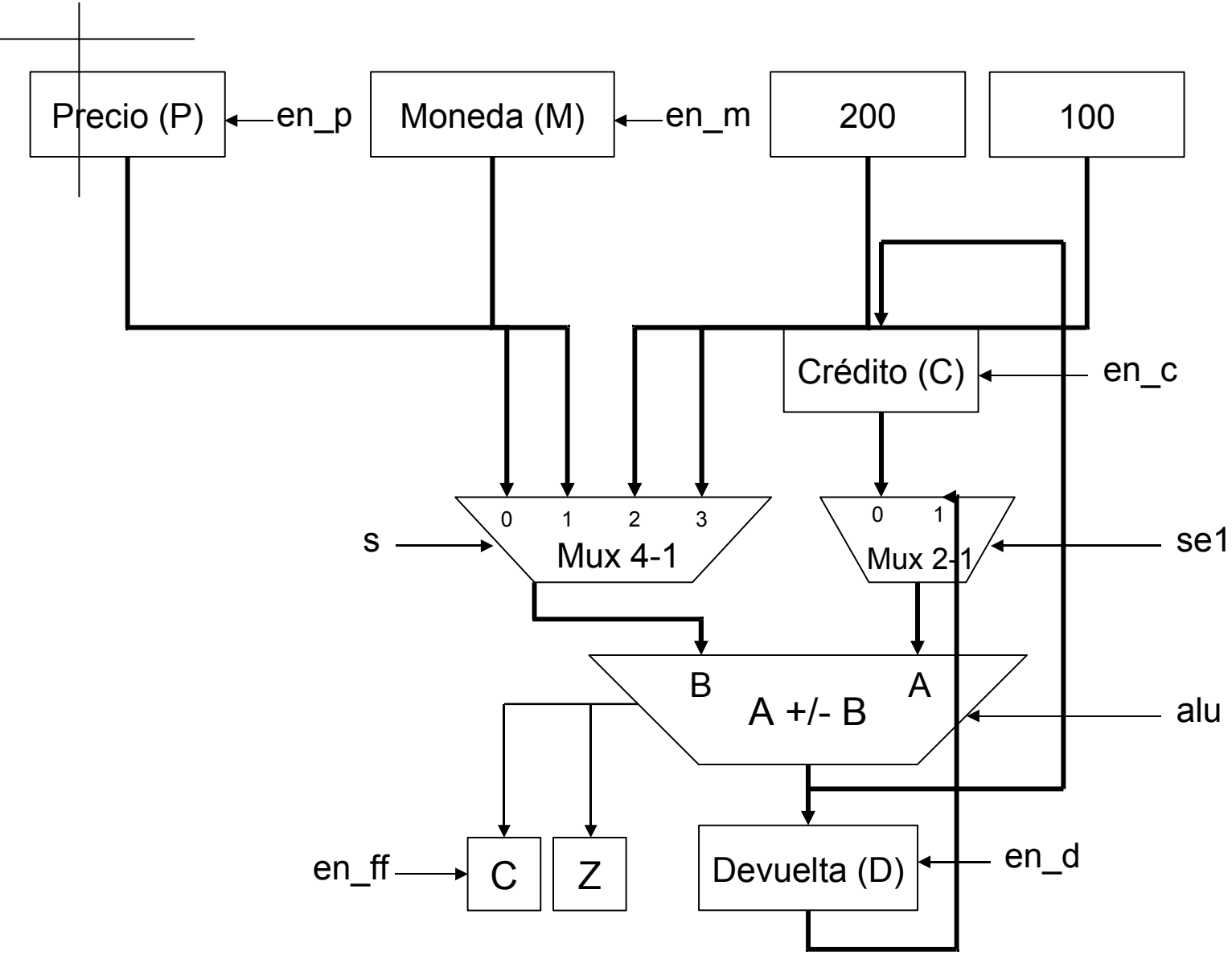
- Diseñe una maquina vendedora de gaseosas, la cual pueda manejar al menos 4 productos diferentes y pueda recibir monedas de dos denominaciones diferentes. La maquina debe dar devueltas en caso de que sea necesario y debe tener una opción para cancelar una transacción y devolver las monedas que un usuario le haya insertado.

- Establezca las señales de control necesarias para hacer el control interno de la maquina, como compuertas o actuadores mecánicos.

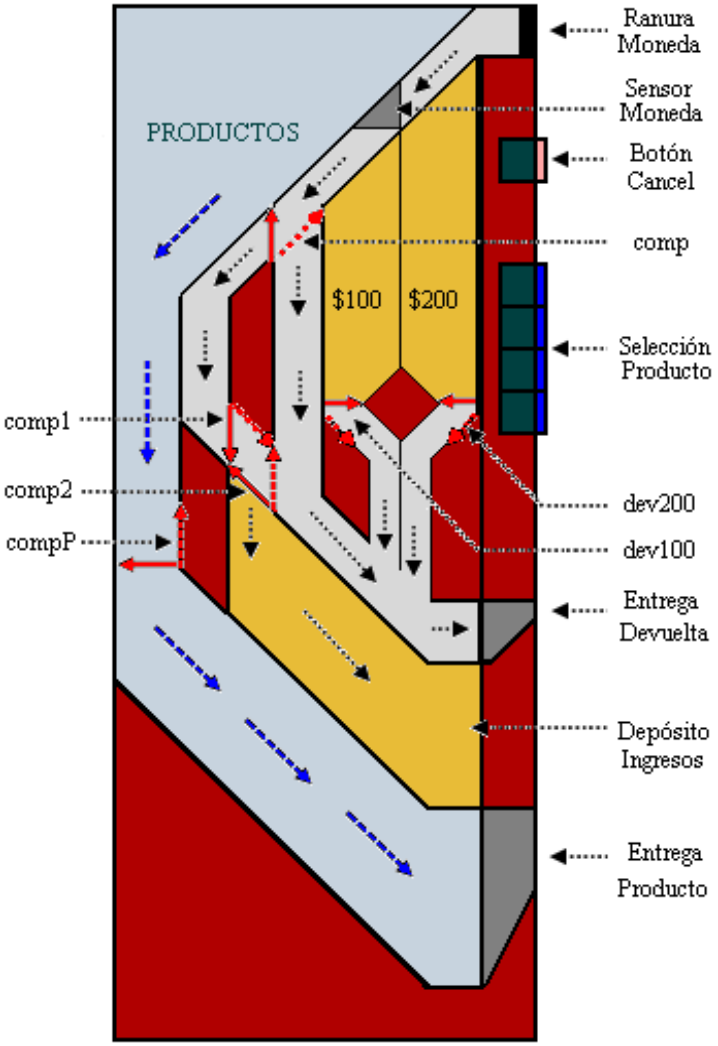
Ejercicio 11: Diagrama ASM



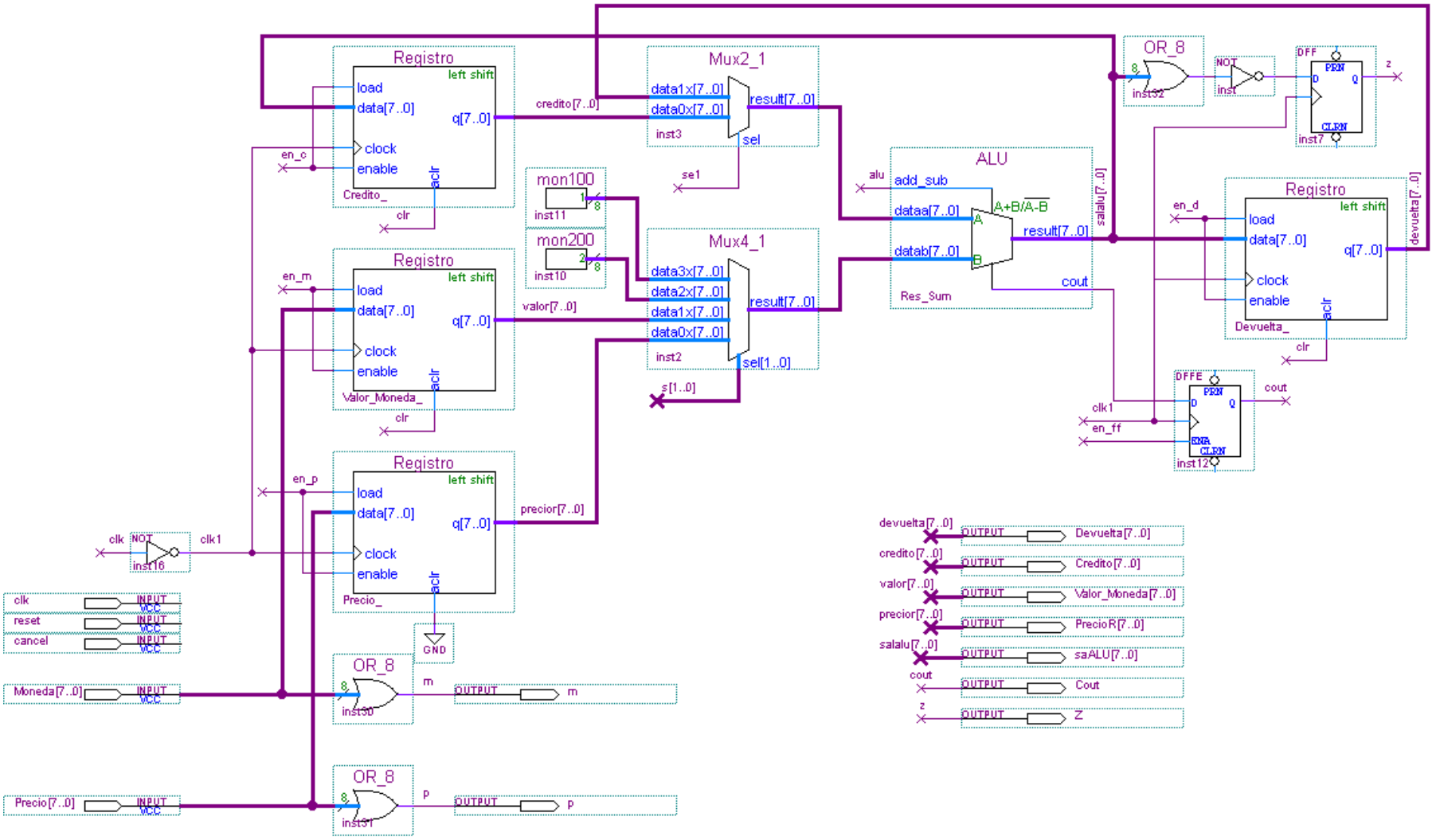
Ejercicio 11: DataPath



Ejercicio 11: Diagrama Mecánico



Ejercicio 11: Quartus II: DataPath



Ejercicio 11: Implementación de la FSM en VHDL (1)

Universidad del Valle

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;

ENTITY FSM IS

    PORT(
        clk     : IN STD_LOGIC;
        reset   : IN STD_LOGIC;
        cancel  : IN STD_LOGIC;
        p, m, cout, zero  : IN STD_LOGIC;

        clr, en_p, en_m, en_c, en_d, en_ff, compP,comp,comp1,comp2, alu, se1, dev100, dev200 : OUT STD_LOGIC;

        s : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
        estado : OUT STD_LOGIC_VECTOR(3 downto 0)

    );

END ENTITY;
```

Ejercicio 11: Implementación de la FSM en VHDL (2)

Universidad del Valle

```
ARCHITECTURE rtl OF FSM IS

TYPE state_type IS (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9);
signal state : state_type;

BEGIN

    process (clk, reset)
    begin
        if reset = '1' then
            state <= s0;
        elsif (rising_edge(clk)) then
            case state is

                when s0=>
                    if p = '1' then
                        state <= s1;
                    else
                        state <= s0;
                    end if;

                when s1=>
                    if cancel = '1' then
                        state <= s0;
                        elsif m = '1' then
                            state <= s2;
                            else
                            state <= s1;
                    end if;

                when s2=>
                    state <= s3;

                when s3=>
                    state <= s4;

                when s4=>
                    if cout = '0' then
                        state <= s1;
                    elsif zero = '1' then
                        state <= s5;
                    else
                        state <= s6;
                    end if;

                when s5=>
                    state <= s0;

                when s6=>
                    if cout = '1' then
                        state <= s7;
                    else
                        state <= s8;
                    end if;

                when s7=>
                    if cout = '1' then
                        state <= s6;
                    else
                        state <= s8;
                    end if;

                when s8=>
                    if cout = '1' then
                        state <= s9;
                    else
                        state <= s5;
                    end if;

                when s9=>
                    if cout = '1' then
                        state <= s8;
                    else
                        state <= s5;
                    end if;

            end case;
        end if;
    end process;
```

Ejercicio 11: Implementación de la FSM en VHDL (3)

Universidad del Valle

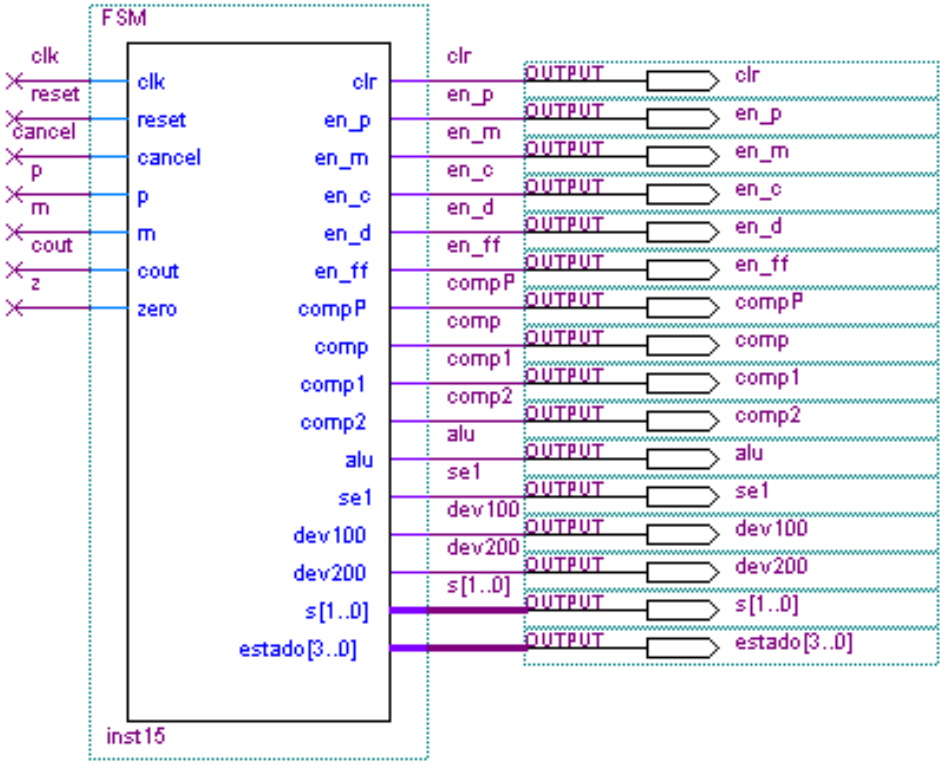
```
PROCESS(state)
BEGIN
    case state is

        when s0 =>
            dev100 <= '0';
            dev200 <= '0';
            en_p <= '1';
            en_c <= '0';
            en_ff <= '0';
            en_d <= '0';
            en_m <= '0';
            alu <= '1';
            se1 <= '0';
            clr <= '1';
            comp <= '0';
            comp1 <= '1';
            comp2 <= '0';
            compP <= '0';
            se1 <= '0';
            s <= "00";
            estado <= "0000";

        when s1 =>
            dev100 <= '0';
            dev200 <= '0';
            clr <= '0';
            en_p <= '0';
            en_m <= '0';
            en_c <= '0';
            en_d <= '0';
            en_ff <= '0';
            alu <= '1';
            se1 <= '0';
            comp <= '1';
            comp1 <= '0';
            comp2 <= '0';
            compP <= '0';
            s <= "00";
            estado <= "0001";

        when s2 =>
            clr <= '0';
            en_p <= '0';
            en_m <= '1';
            en_c <= '0';
            en_d <= '0';
            en_ff <= '0';
            alu <= '1';
            se1 <= '0';
            dev100 <= '0';
            dev200 <= '0';
            comp <= '1';
            comp1 <= '0';
            comp2 <= '0';
            compP <= '0';
            s <= "00";
            estado <= "0010";

        when s3 =>
            clr <= '0';
            en_p <= '0';
            en_m <= '0';
            en_c <= '1';
            en_d <= '0';
            en_ff <= '0';
            se1 <= '0';
            s <= "01";
            alu <= '1';
            dev100 <= '0';
            dev200 <= '0';
            comp <= '1';
            comp1 <= '0';
            comp2 <= '0';
            compP <= '0';
            estado <= "0011";

        when s4 =>
            clr <= '0';
            en_p <= '0';
            en_m <= '0';
            en_c <= '0';
            en_d <= '1';
            se1 <= '0';
            dev100 <= '0';
            dev200 <= '0';
            s <= "00";
            alu <= '0';
            en_ff <= '1';
            comp <= '1';
            comp1 <= '0';
            comp2 <= '0';
            compP <= '0';
            estado <= "0100";

        when s5 =>
            clr <= '0';
            en_p <= '0';
            en_m <= '0';
            en_c <= '0';
            en_d <= '0';
            en_ff <= '0';
            alu <= '1';
            se1 <= '0';
            dev100 <= '0';
            dev200 <= '0';
            comp <= '1';
            comp1 <= '1';
            comp2 <= '1';
            compP <= '1';
            s <= "00";
            estado <= "0101";

        when s6 =>
            clr <= '0';
            en_p <= '0';
            en_m <= '0';
            en_c <= '0';
            en_d <= '0';
            s <= "10";
            en_d <= '0';
            se1 <= '1';
            alu <= '0';
            dev100 <= '0';
            dev200 <= '0';
            en_ff <= '1';
            comp <= '0';
            comp1 <= '1';
            comp2 <= '1';
            compP <= '0';
            estado <= "0110";

        when s7 =>
            clr <= '0';
            en_p <= '0';
            en_m <= '0';
            en_c <= '0';
            en_d <= '1';
            dev200 <= '1';
            dev100 <= '0';
            comp <= '0';
            comp1 <= '0';
            comp2 <= '0';
            compP <= '0';
            s <= "10";
            se1 <= '1';
            alu <= '0';
            en_ff <= '1';
            estado <= "0111";
```

Ejercicio 11: Implementación de la FSM en VHDL (4)

Universidad del Valle

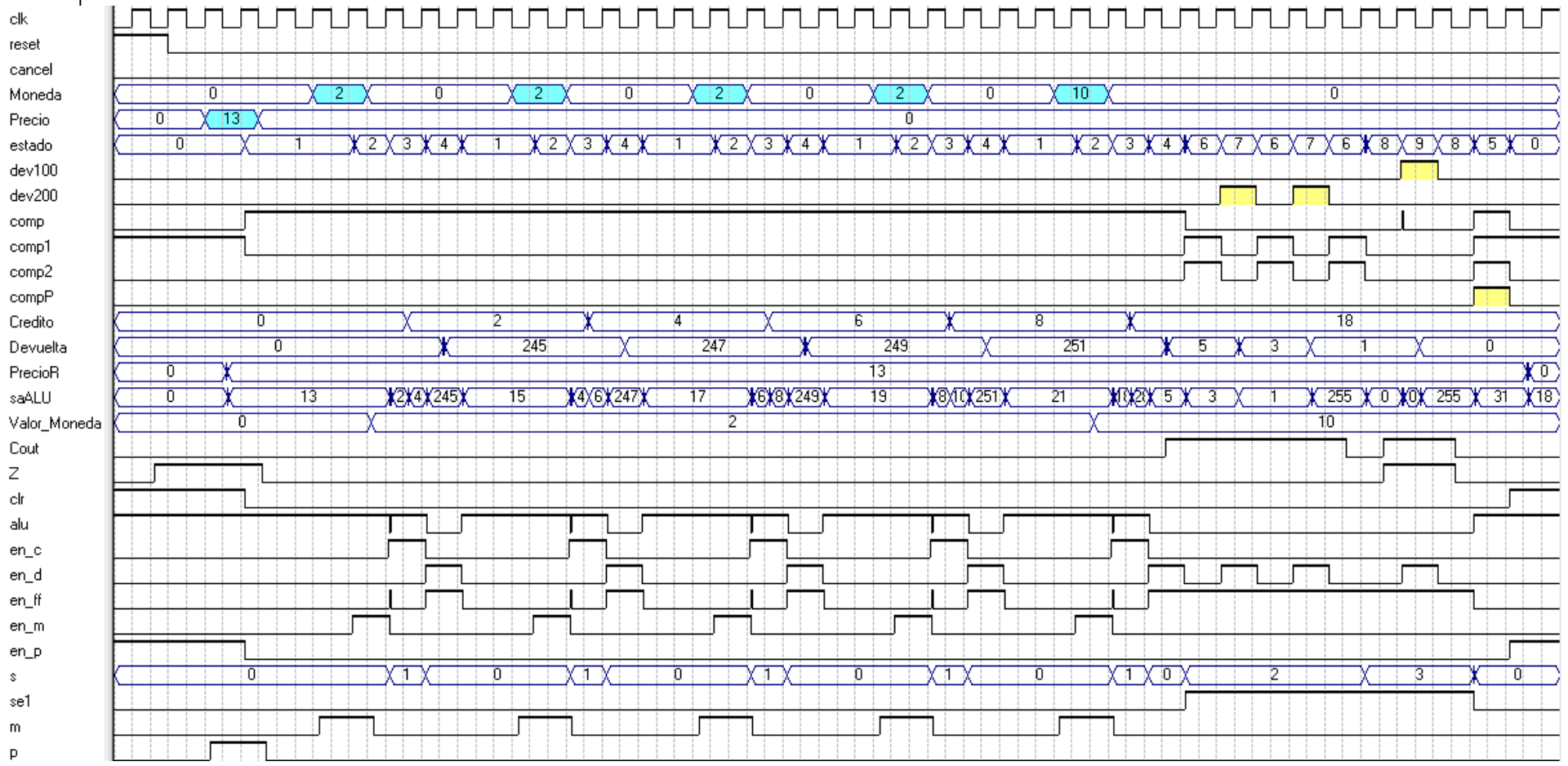
```
            when s8 =>
                clr <= '0';
                en_p <= '0';
                en_m <= '0';
                en_c <= '0';
                en_d <= '0';
                dev100 <= '0';
                dev200 <= '0';
                comp <= '0';
                comp1 <= '0';
                comp2 <= '0';
                compP <= '0';
                s <= "11";
                sel <= '1';
                alu <= '0';
                en_ff <= '1';
                estado <= "1000";

            when s9 =>
                clr <= '0';
                en_p <= '0';
                en_m <= '0';
                en_c <= '0';
                en_d <= '1';
                comp <= '0';
                comp1 <= '0';
                comp2 <= '0';
                compP <= '0';
                dev100 <= '1';
                dev200 <= '0';
                s <= "11";
                sel <= '1';
                alu <= '0';
                en_ff <= '1';
                estado <= "1001";

            end case;
        end process;
end rtl;
```

Ejercicio 11: Quartus II: FSM



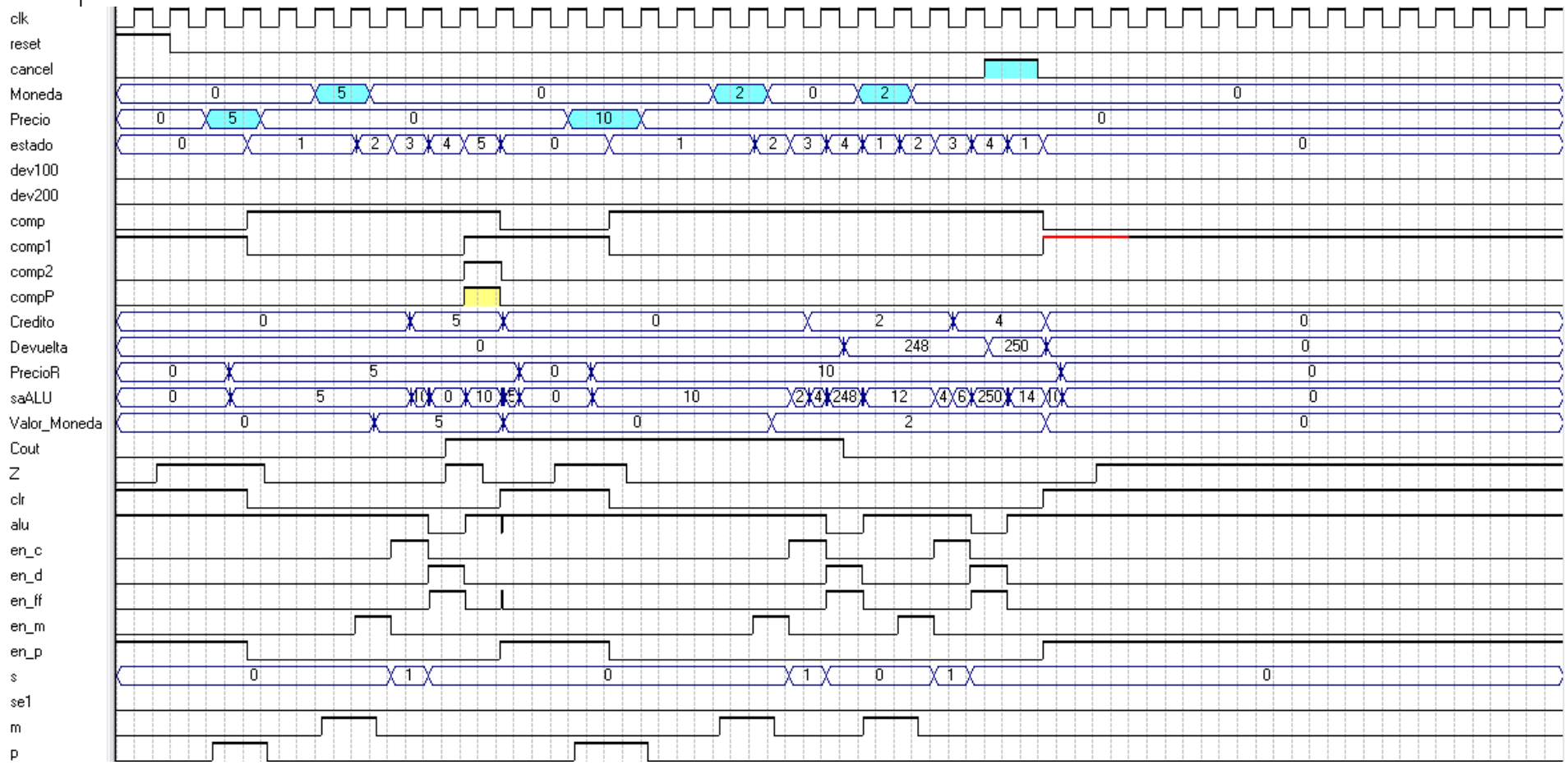
Ejercicio 11: Simulacion (1)

Universidad del Valle



Ejercicio 11: Simulacion (2)

Universidad del Valle

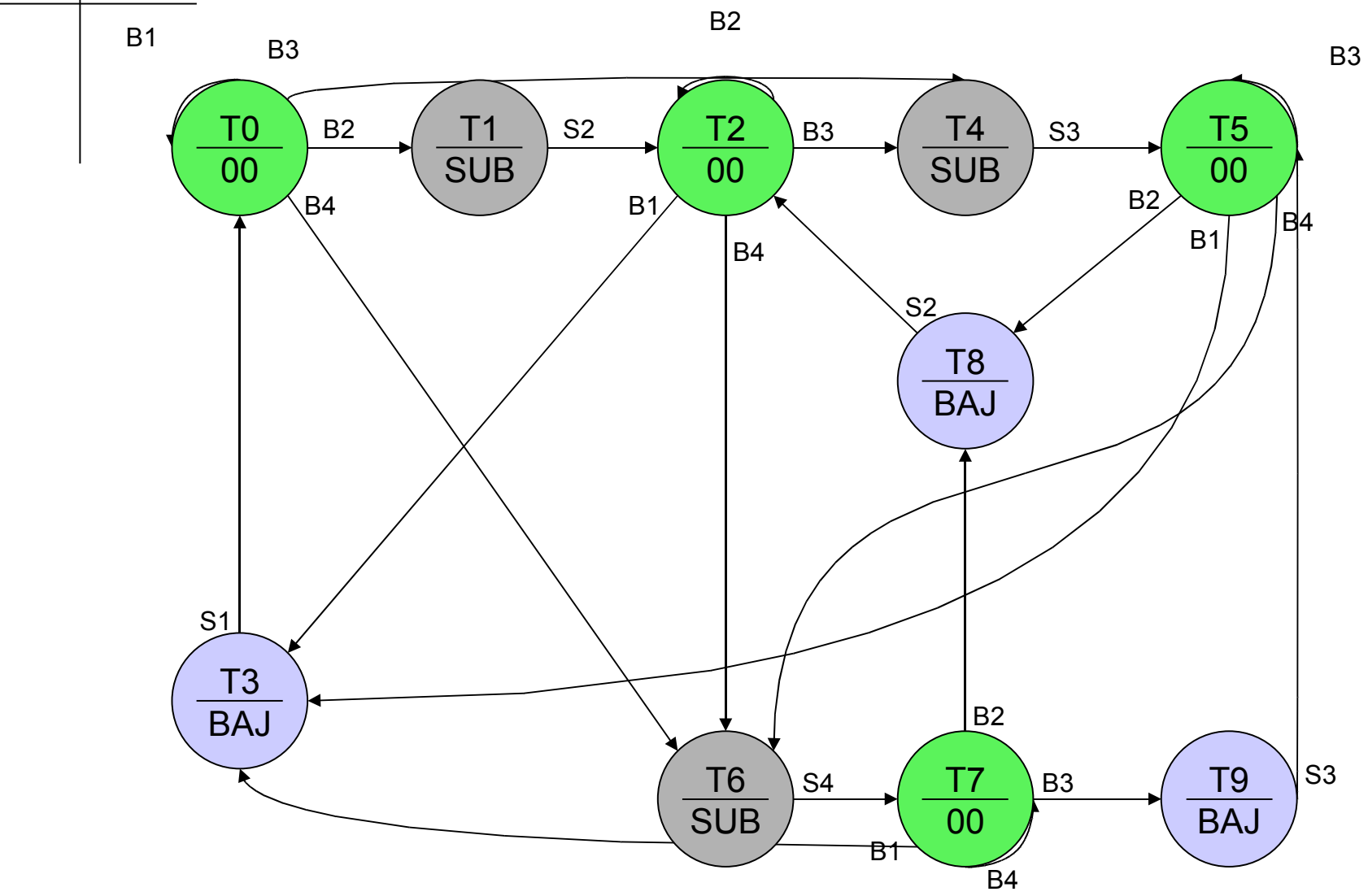


Ejercicio 12: Diseño de un ascensor

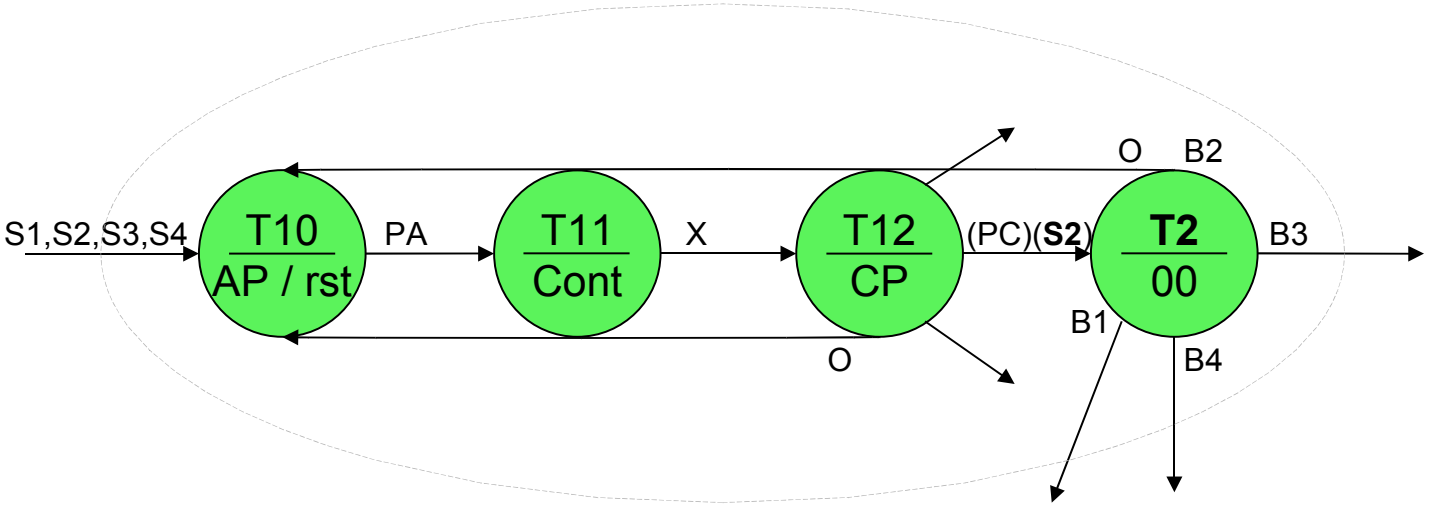
Universidad del Valle

- Diseñar una maquina de estados tipo Moore que controle un ascensor que funciona en un edificio de 4 pisos. El ascensor tiene botones externos para ser llamado desde otro piso, e internos para llevarlo a cualquiera ellos. Además cuenta con un sistema de sensores, el cual detecta en que piso se encuentra actualmente.
- Se debe definir una lógica de prioridad para que funcione en caso de que el ascensor sea llamado desde dos o mas pisos al mismo tiempo.
- Se pueden utilizar flip-flops o latches adicionales a los de la FSM para el almacenamiento temporal de una petición para el ascensor (útil cuando el ascensor es llamado mientras esta subiendo o bajando).

Ejercicio 12: Diagrama de Estados General



Ejercicio 12: Diagrama de Estados Internos (T2)



Entradas:

por usuario:

- O : botón Abrir
- C : botón Cerrar

por control:

- X : Cerrar Puerta
- PA : Puerta Abierta
- PC : Puerta Cerrada
- T : Tiempo de espera

Salidas:

- AP : Abrir Puerta
- CP : Cerrar Puerta
- Cont : Habilitar Conteo
- rst : Reiniciar Conteo
- E2 : Borrar flip-flop B2

E1 = S1 AND T10
E2 = S2 AND T10
E3 = S3 AND T10
E4 = S4 AND T10

C (close)	O (open)	T (tiempo)	**X**
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**0**
1	1	1	0

Lógica del control para cerrar la puerta

Ejercicio 12: Lógica de Prioridad

Universidad del Valle

Botones Presionados				Prioridad					
					Piso 2: T2		Piso 3: T5		
B1	B2	B3	B4	Piso 1: T0	Subiend	Bajando	Subiend	Bajando	Piso 4: T7
0	0	0	0	-	-	-	-	-	-
0	0	0	1	B4	B4	B4	B4	B4	-
0	0	1	0	B3	B3	B3	-	-	B3
0	0	1	1	B3	B3	B3	B4	B4	B3
0	1	0	0	B2	-	-	B2	B2	B2
0	1	0	1	B2	B4	B4	B4	B2	B2
0	1	1	0	B2	B3	B3	B2	B2	B3
0	1	1	1	B2	B3	B3	B4	B2	B3
1	0	0	0	-	B1	B1	B1	B1	B1
1	0	0	1	B4	B4	B1	B4	B1	B1
1	0	1	0	B3	B3	B1	B1	B1	B3
1	0	1	1	B3	B3	B1	B4	B1	B3
1	1	0	0	B2	B1	B1	B2	B2	B2
1	1	0	1	B2	B4	B1	B4	B2	B2
1	1	1	0	B2	B3	B1	B2	B2	B3
1	1	1	1	B2	B3	B1	B4	B2	B3

Ejemplo 12: Implementación de la FMS en VHDL (1)

Universidad del Valle

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;

ENTITY FSM IS

    PORT(
        clk     : IN STD_LOGIC;
        reset   : IN STD_LOGIC;
        B1,B2,B3,B4,S1,S2,S3,S4,PA,PC,X,O  : IN STD_LOGIC;

        SUB,BAJ,AP,CP,Cont,rst: OUT STD_LOGIC;
        estado : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );

END ENTITY;

ARCHITECTURE rtl OF FSM IS

TYPE state_type IS (T0, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12);
SIGNAL state : state_type;

SIGNAL salidas : STD LOGIC VECTOR(9 DOWNTO 0);
```

Ejemplo 12: Implementación de la FMS en VHDL (2)

Universidad del Valle

```
BEGIN

    process (clk, reset)
    begin
        if reset = '1' then
            state <= T1;
        elsif (rising_edge(clk)) then
            case state is

                when T0=>                          --ESTADOS DE PISO
                    if O = '1' OR B1 = '1' then
                        state <= T10;
                    elsif B2 = '1' then
                        state <= T1;
                    elsif B3 = '1' then
                        state <= T4;
                    elsif B4 = '1' then
                        state <= T6;
                    else
                        state <= T0;
                    end if;

                when T2=>
                    if O = '1' OR B2 = '1' then
                        state <= T10;
                    elsif B1 = '1' then
                        state <= T3;
                    elsif B3 = '1' then
                        state <= T4;
                    elsif B4 = '1' then
                        state <= T6;
                    else
                        state <= T2;
                    end if;

                when T5=>
                    if O = '1' OR B3 = '1' then
                        state <= T10;
                    elsif B1 = '1' then
                        state <= T3;
                    elsif B2 = '1' then
                        state <= T8;
                    elsif B4 = '1' then
                        state <= T6;
                    else
                        state <= T5;
                    end if;

                when T7=>
                    if O = '1' OR B4 = '1' then
                        state <= T10;
                    elsif B1 = '1' then
                        state <= T3;
                    elsif B2 = '1' then
                        state <= T8;
                    elsif B3 = '1' then
                        state <= T9;
                    else
                        state <= T7;
                    end if;
```

Ejemplo 12: Implementación de la FMS en VHDL (3)

Universidad del Valle

```
--ESTADOS DE SUBIDA

when T1=>
    if S2 = '1' then
        state <= T10;
    else
        state <= T1;
    end if;

when T4=>
    if S3 = '1' then
        state <= T10;
    else
        state <= T4;
    end if;

when T6=>
    if S4 = '1' then
        state <= T10;
    else
        state <= T6;
    end if;
```

```
--ESTADOS DE BAJADA

when T3=>
    if S1 = '1' then
        state <= T10;
    else
        state <= T3;
    end if;

when T8=>
    if S2 = '1' then
        state <= T10;
    else
        state <= T8;
    end if;

when T9=>
    if S3 = '1' then
        state <= T10;
    else
        state <= T9;
    end if;
```

Ejemplo 12: Implementación de la FMS en VHDL (4)

```
            when T10=>                  --ESTADOS INTERNOS
                if PA = '1' then
                    state <= T11;
                else
                    state <= T10;
                end if;

            when T11=>
                if X = '1' then
                    state <= T12;
                else
                    state <= T11;
                end if;

            when T12=>
                if O = '1' then
                    state <= T10;
                elsIF PC = '1' THEN

                    IF S1 = '1' THEN
                        state <= T0;
                    END IF;
                    IF S2 = '1' THEN
                        state <= T2;
                    END IF;
                    IF S3 = '1' THEN
                        state <= T5;
                    END IF;
                    IF S4 = '1' THEN
                        state <= T7;
                    END IF;
                ELSE
                    state <= T12;
                end if;
        end case;
    end if;
end process;
```

Ejemplo 12: Implementación de la FMS en VHDL (5)

Universidad del Valle

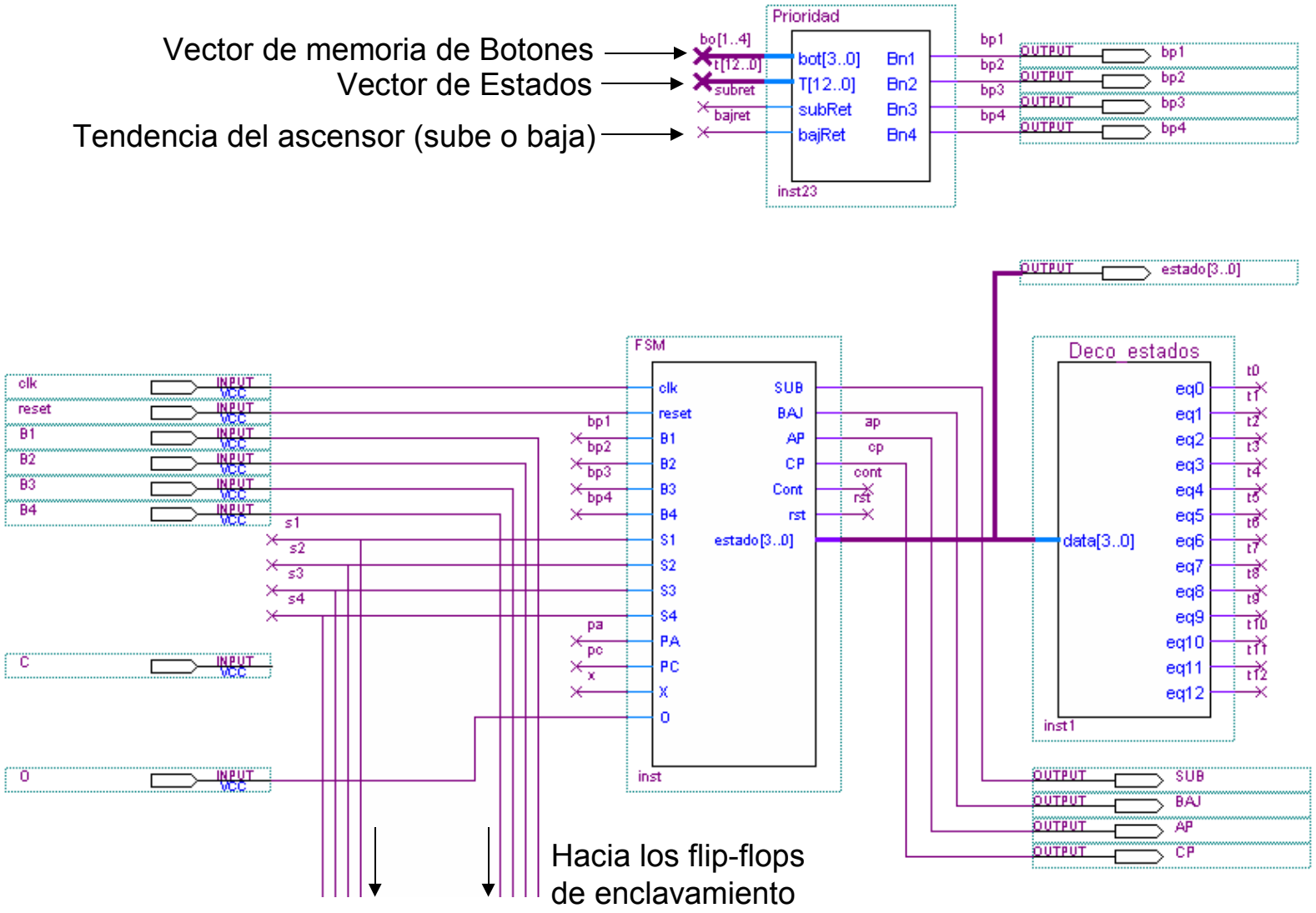
```
PROCESS(state)
    BEGIN
        case state is   --  SUB,BAJ,AP,CP,Cont,rst,estado(3..0)

            when T10 =>
                salidas <= "0010011010";    --INTERNOS
            when T11 =>
                salidas <= "0000101011";
            when T12 =>
                salidas <= "0001001100";

            when T1 =>
                salidas <= "1000000001";    --SUBIR
            when T4 =>
                salidas <= "1000000100";
            when T6 =>
                salidas <= "1000000110";

            when T3 =>
                salidas <= "0100000011";    --BAJAR
            when T8 =>
                salidas <= "0100001000";
            when T9 =>
                salidas <= "0100001001";

            when T0 =>
                salidas <= "0000000000";    --PISOS
            when T2 =>
                salidas <= "0000000010";
            when T5 =>
                salidas <= "0000000101";
            when T7 =>
                salidas <= "0000000111";

            end case;
        end process;
```

Ejemplo 12: Implementación de la FMS en VHDL (6)

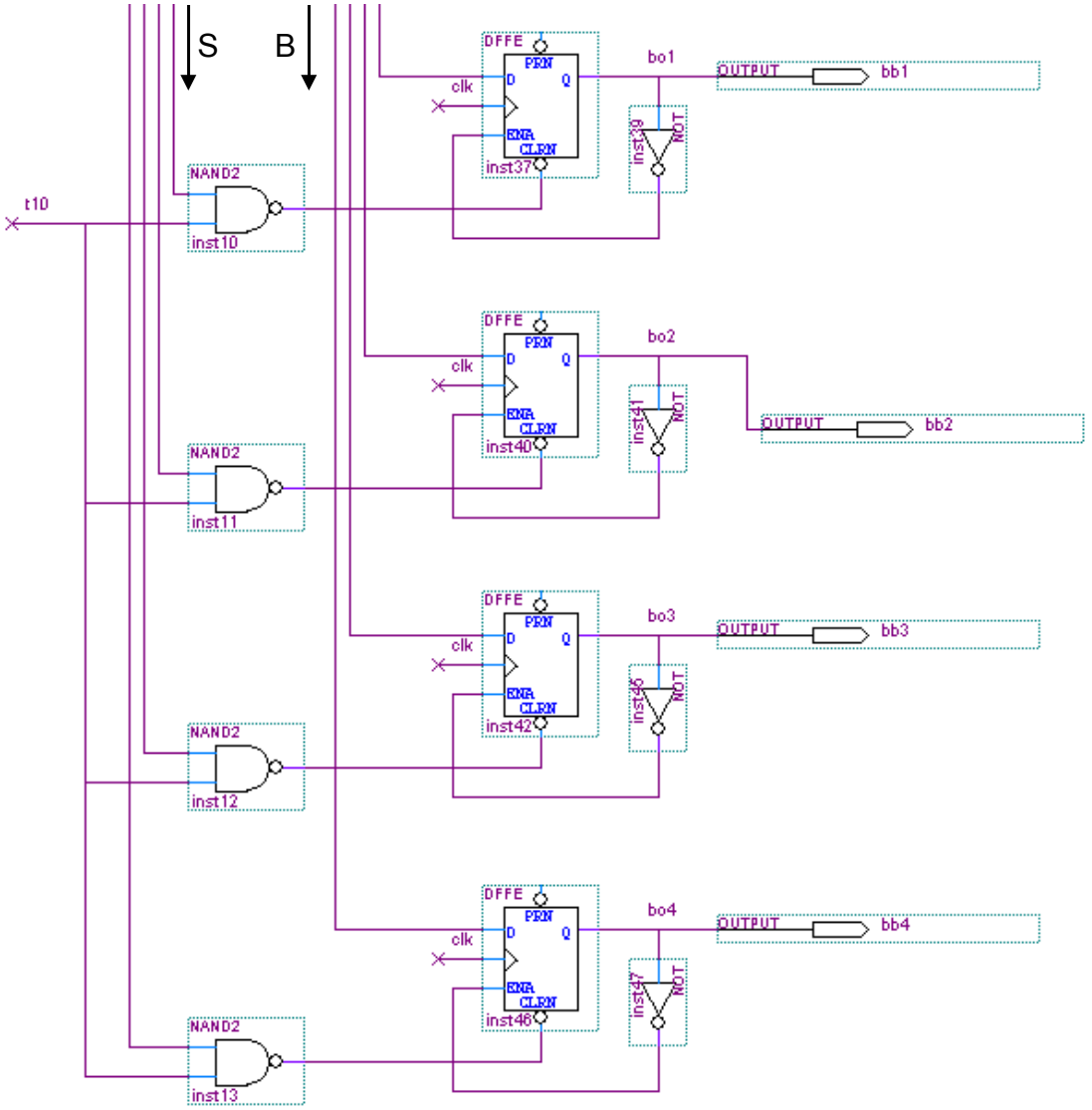
Universidad del Valle

```
--  salidas = SUB,BAJ,AP,CP,Cont,rst,estado(3..0)

SUB <= salidas(9);
BAJ <= salidas(8);
AP <= salidas(7);
CP <= salidas(6);
Cont <= salidas(5);
rst <= salidas(4);
estado <= salidas(3 DOWNTO 0);

end rtl;
```

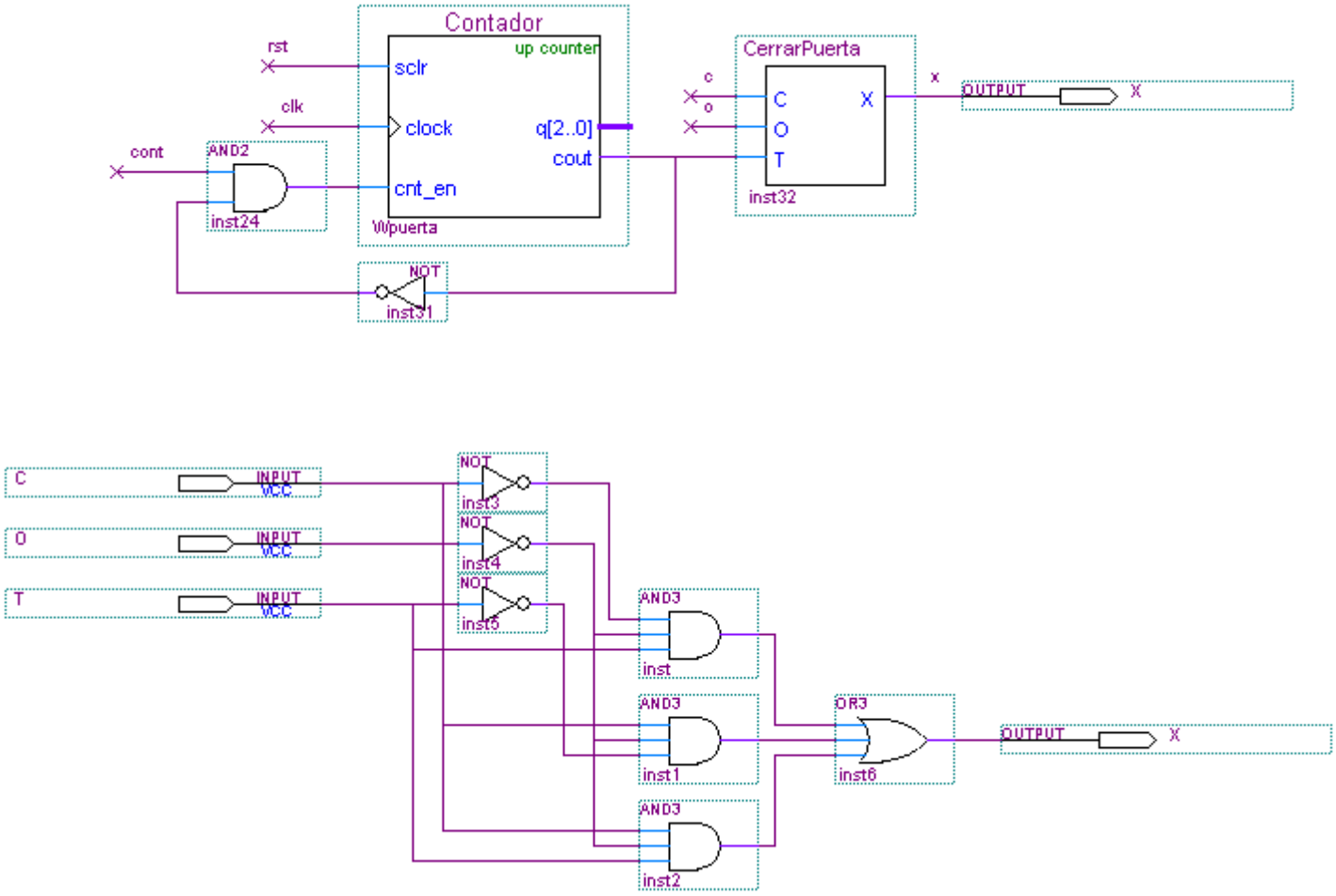
Ejercicio 12: Quartus II



Ejercicio 12: Quartus II: Enclavamiento de los botones

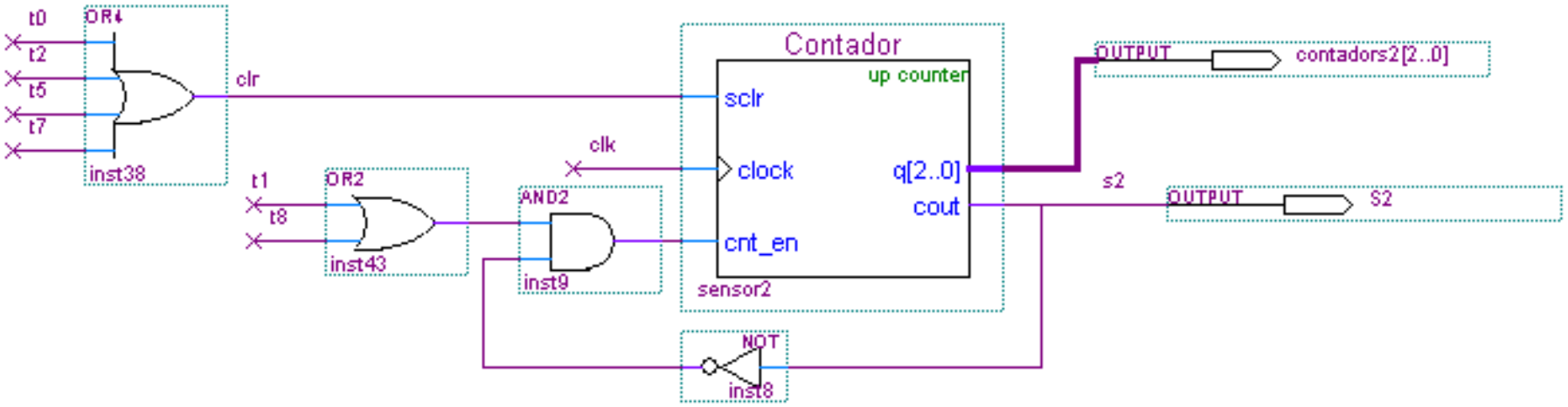


Ejercicio 12: Generador de Retardo y Circuito para cerrar la puerta del ascensor

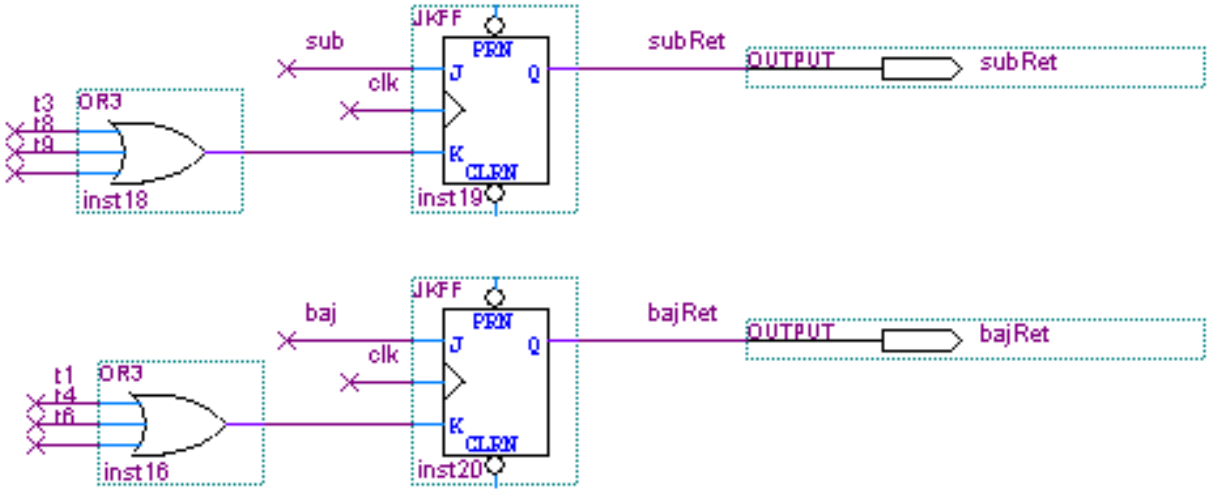


Ejercicio 12: Generador de Retardo para simular el tiempo de subida hacia el Piso 2

Universidad del Valle

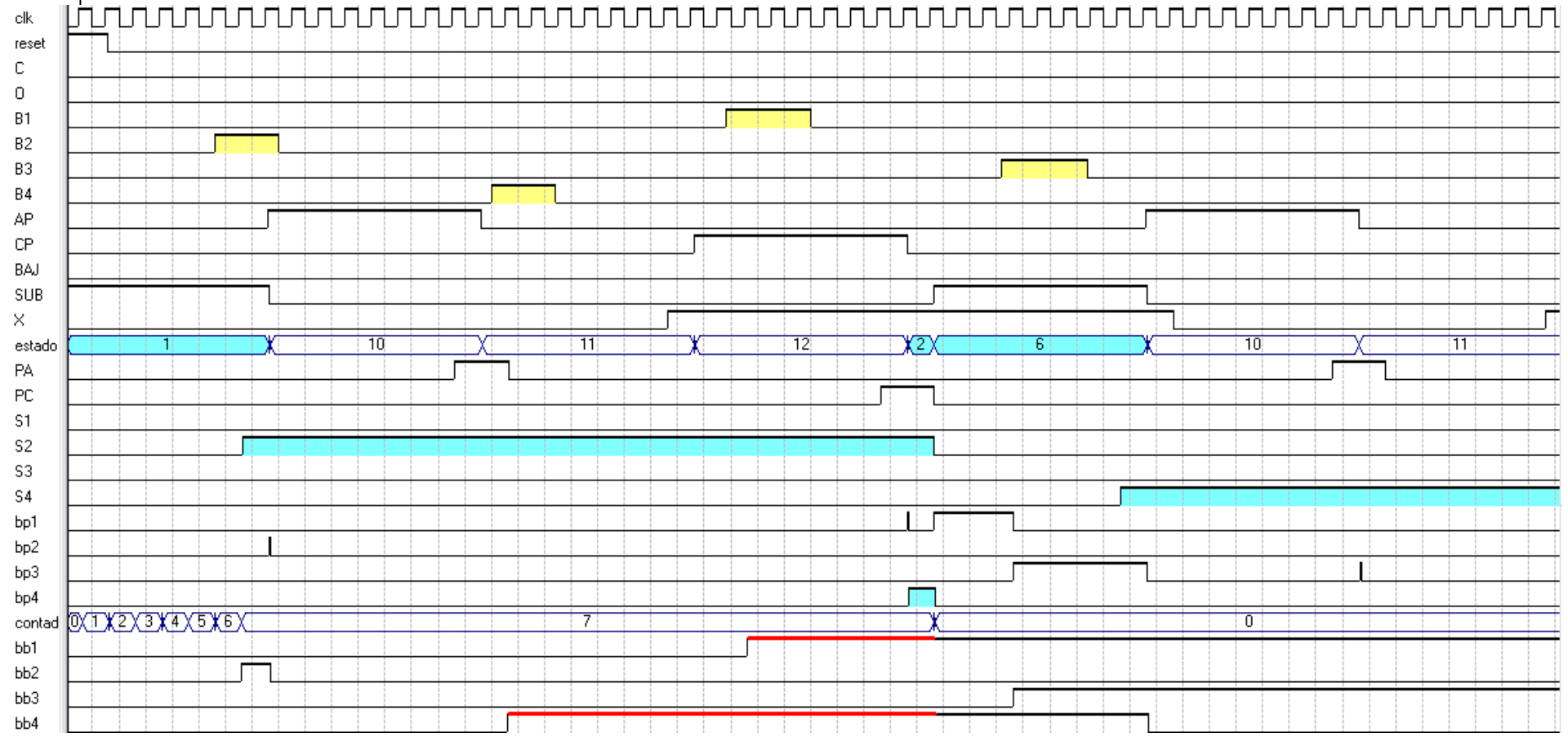


Ejercicio 12: Generadores de Tendencia



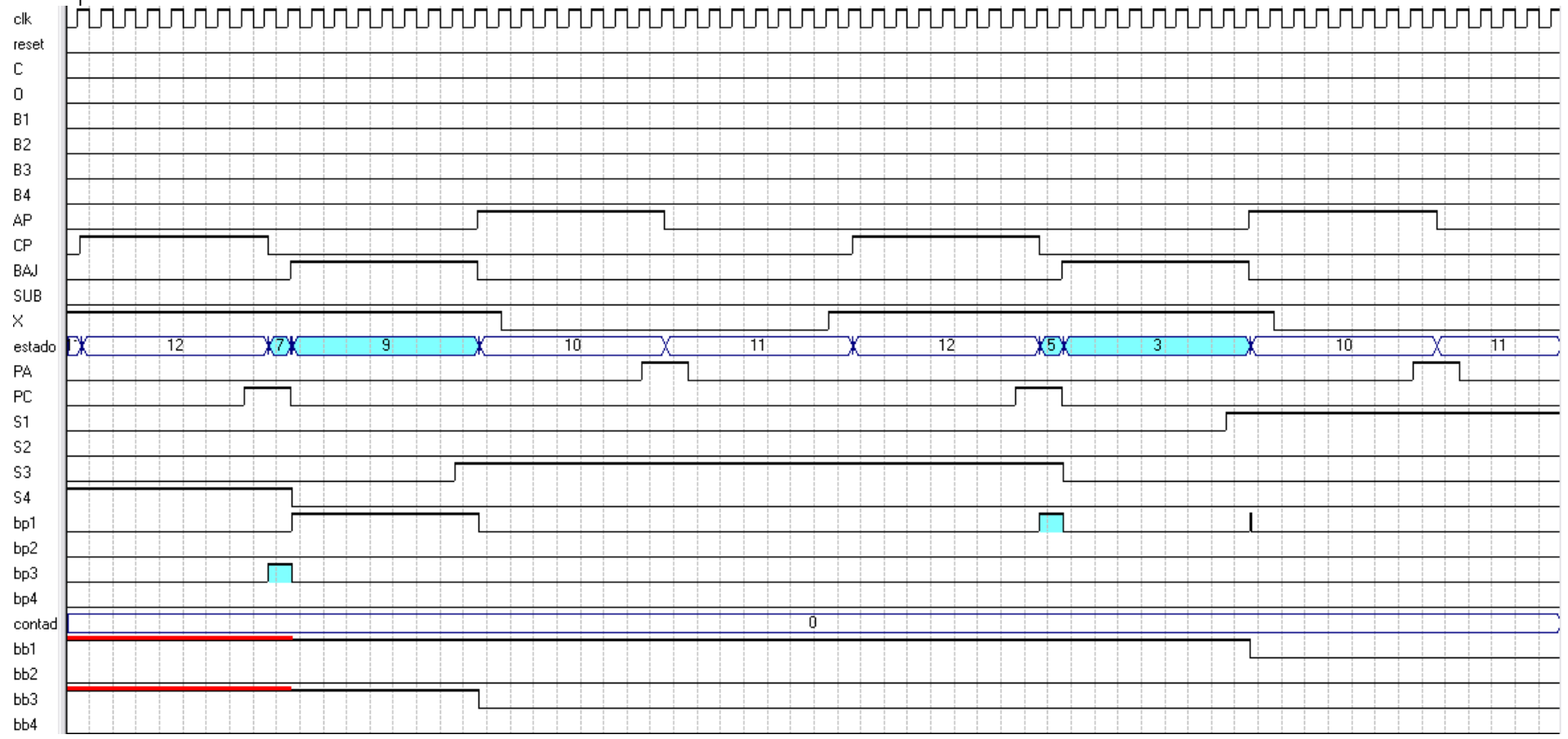
Ejercicio 12: Simulación 1-a

Universidad del Valle



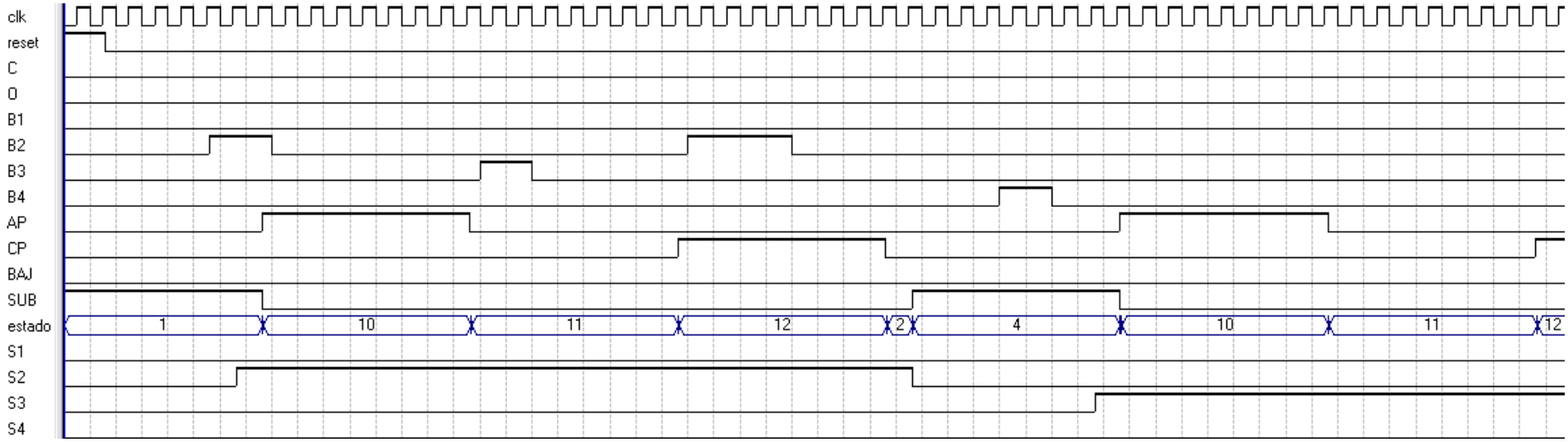
Ejercicio 12: Simulación 1-b

Universidad del Valle



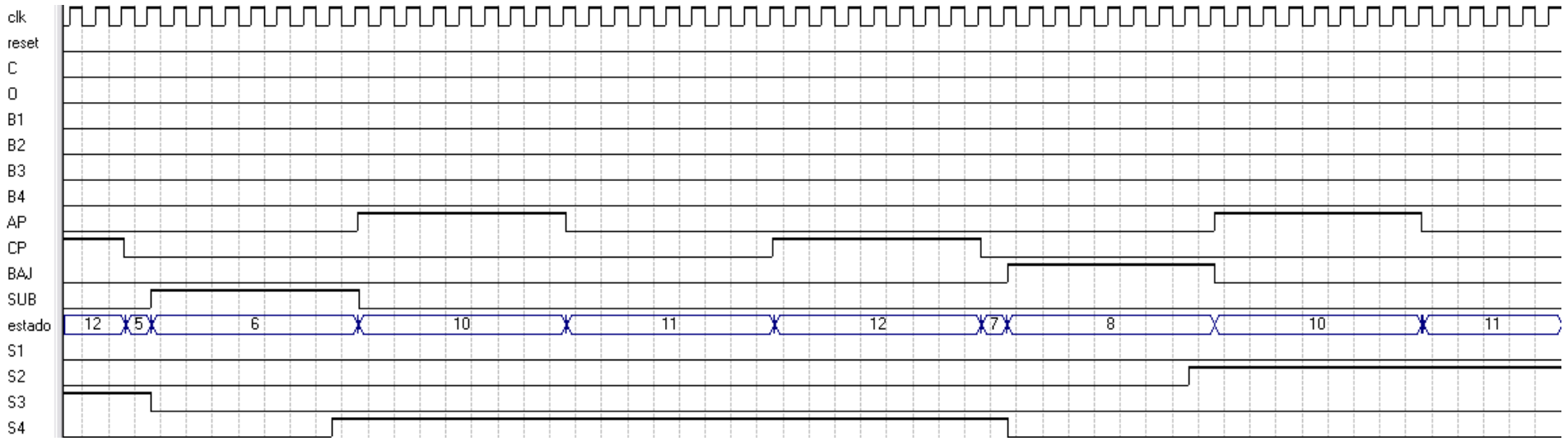
Ejercicio 12: Simulación 2-a

Universidad del Valle



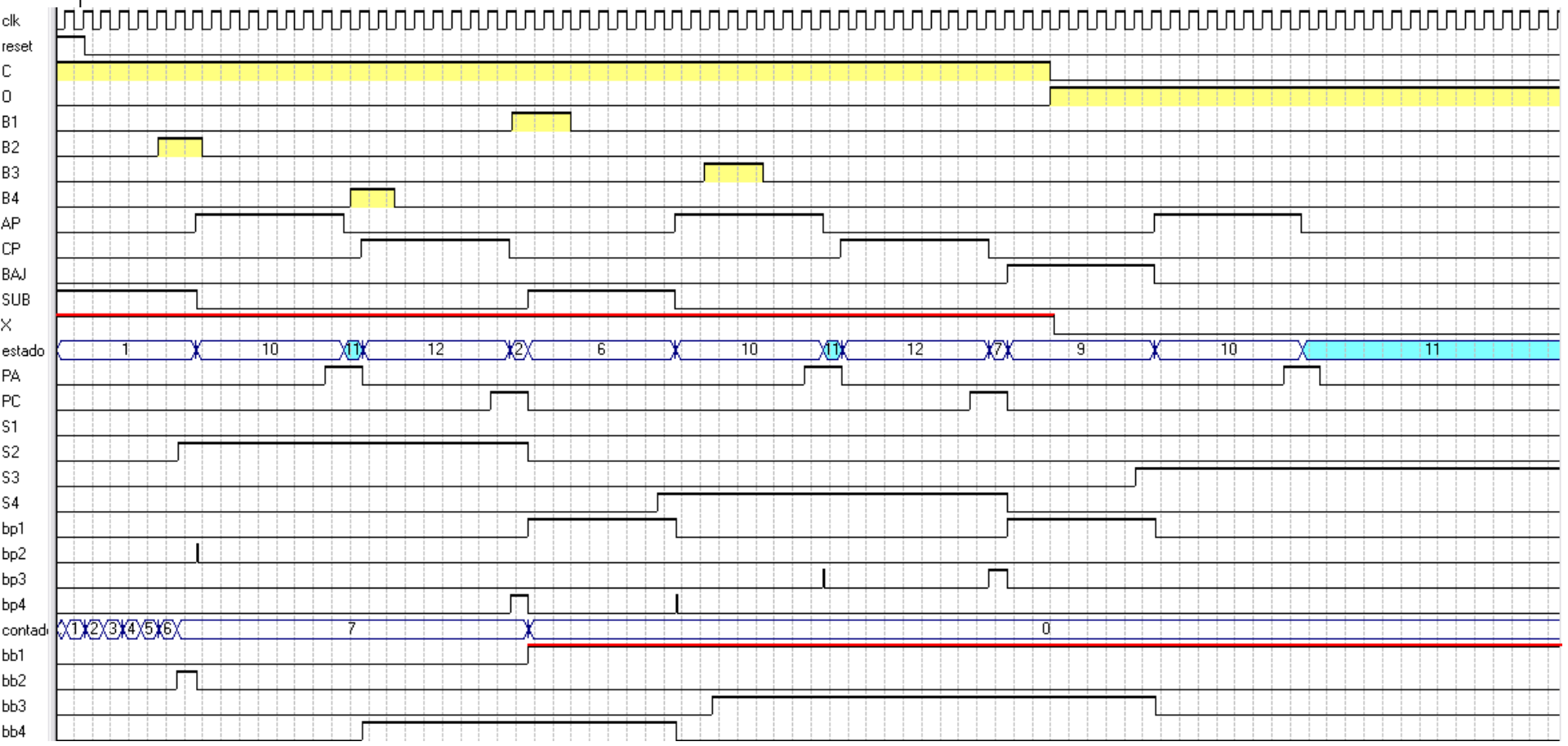
Ejercicio 12: Simulación 2-b

Universidad del Valle

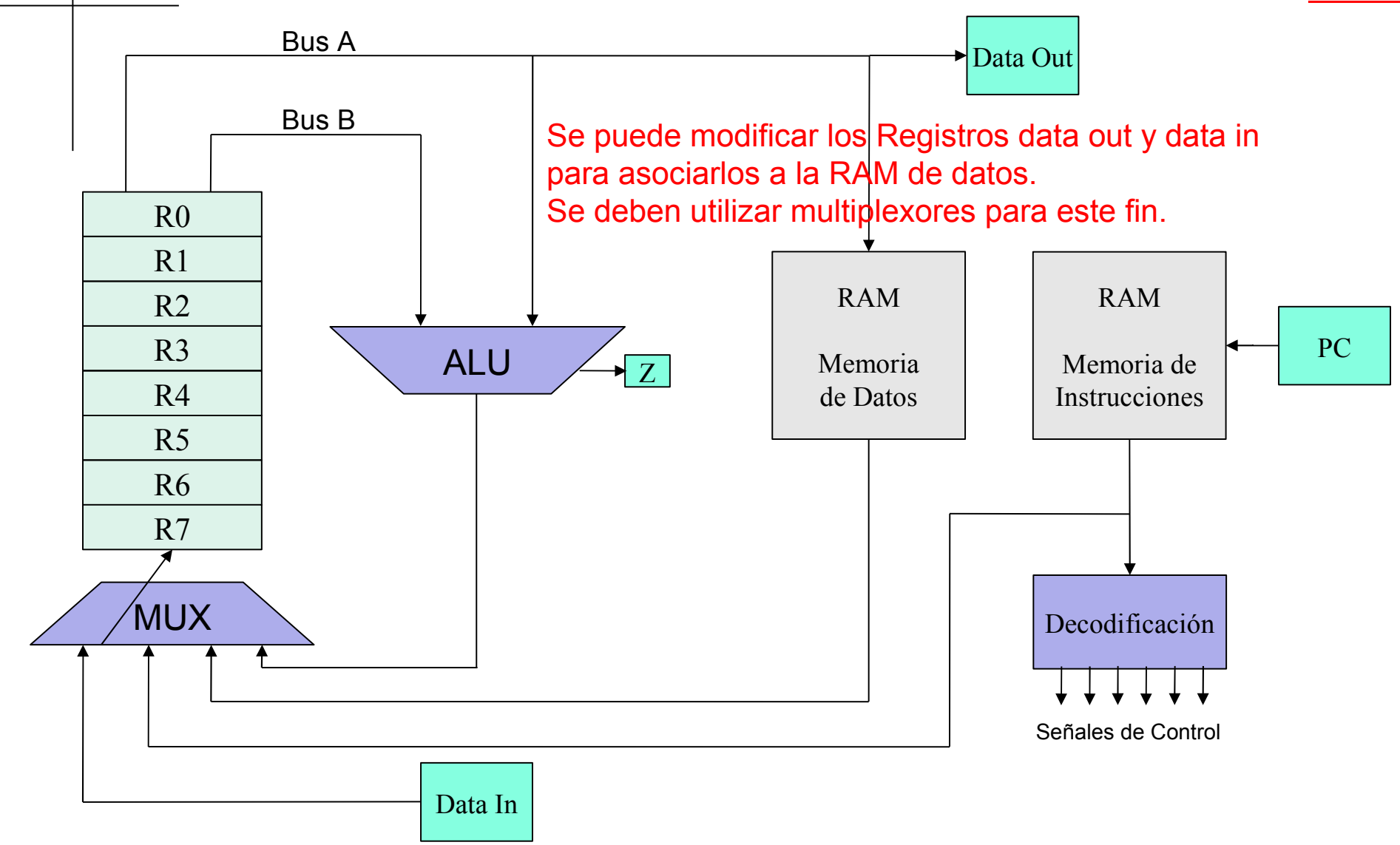


Ejercicio 12: Simulación 3 (manejo de puertas)

Universidad del Valle



Ejercicio 13: Diseño de un Procesador Monociclo



Ejercicio 13: Diseño de un Procesador Monociclo

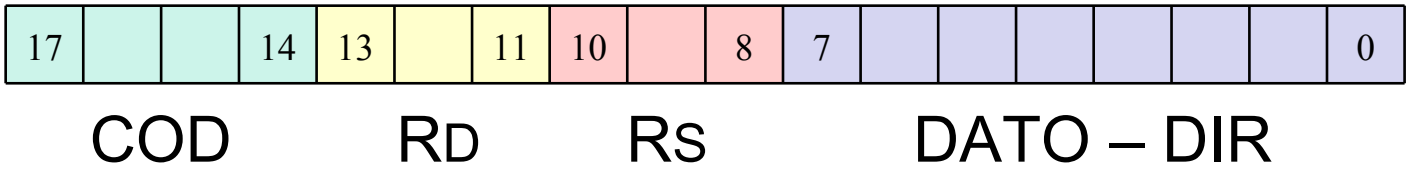
Universidad del Valle

Conjunto de Instrucciones

- 0000 STORE
- 0001 LOAD
- 0010 OUT
- 0011 IN
- 0100 JZ
- 0101 AND
- 0110 OR
- 0111 XOR
- 1000 NOT
- 1001 ADD
- 1010 SUB
- 1011 MULT B
- 1100 MULT A
- 1101 MOV (REG)
- 1110 MOV (DATO)
- 1111 NOP

Ejercicio 13: Diseño de un Procesador Monociclo

Formato de Instrucción



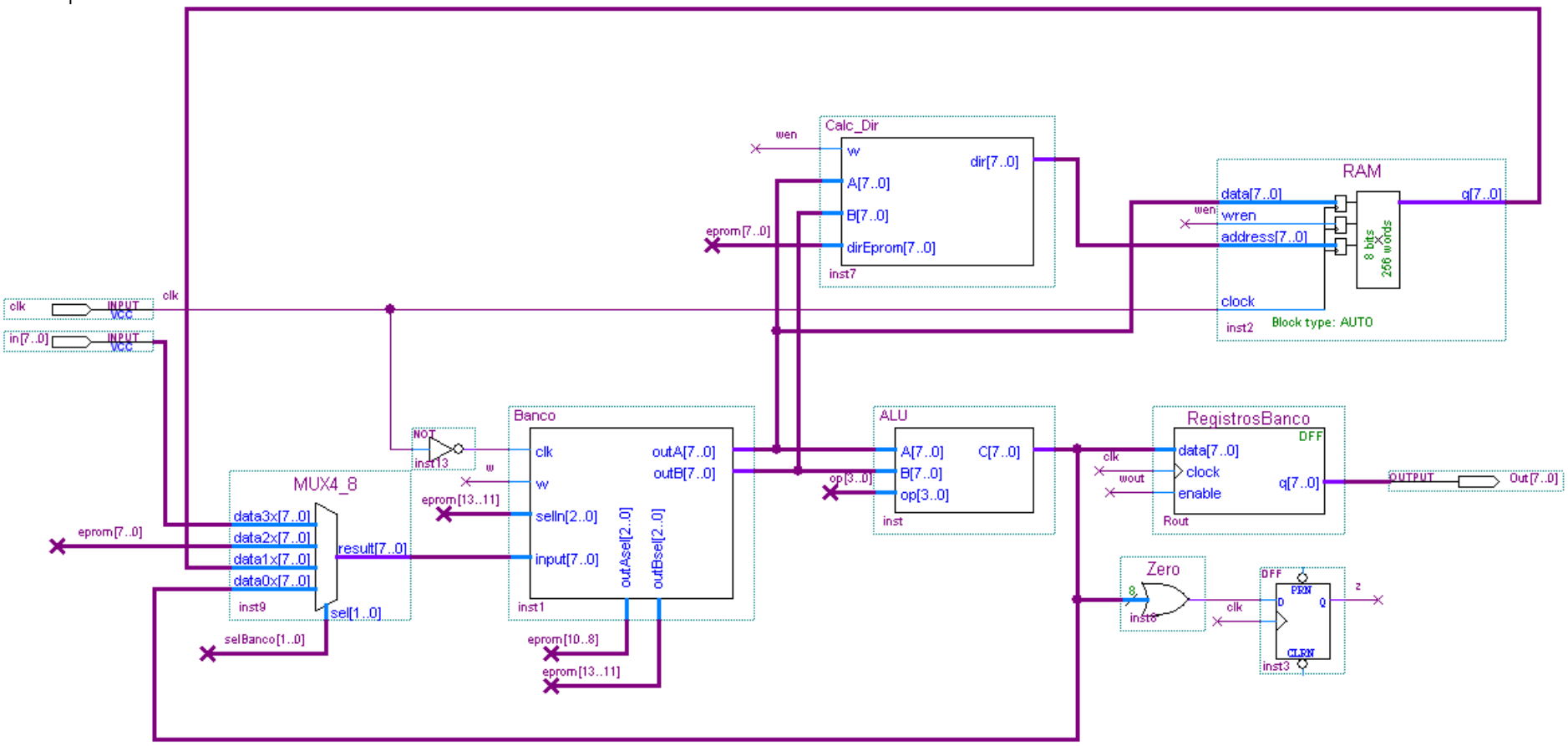
Numero de Instrucciones	$= 2^4$	= 16
Numero de Registros	$= 2^3$	= 8
Posiciones de Memoria	$= 2^8$	= 256

R3 ← R1	MOV R3, R1	=	1101 011 001 xxxxxxxx
R3 ← "10"	MOV R3, "10"	=	1110 011 xxx 00001010
RAM[11 + R3] ← R2	STORE [11+R3], R2	=	0000 011 010 00000011
R1 ← **R1** + R7	ADD R1, R7	=	1001 001 111 xxxxxxxx

Ejercicio 13: Diseño de un Procesador Monociclo

Universidad del Valle

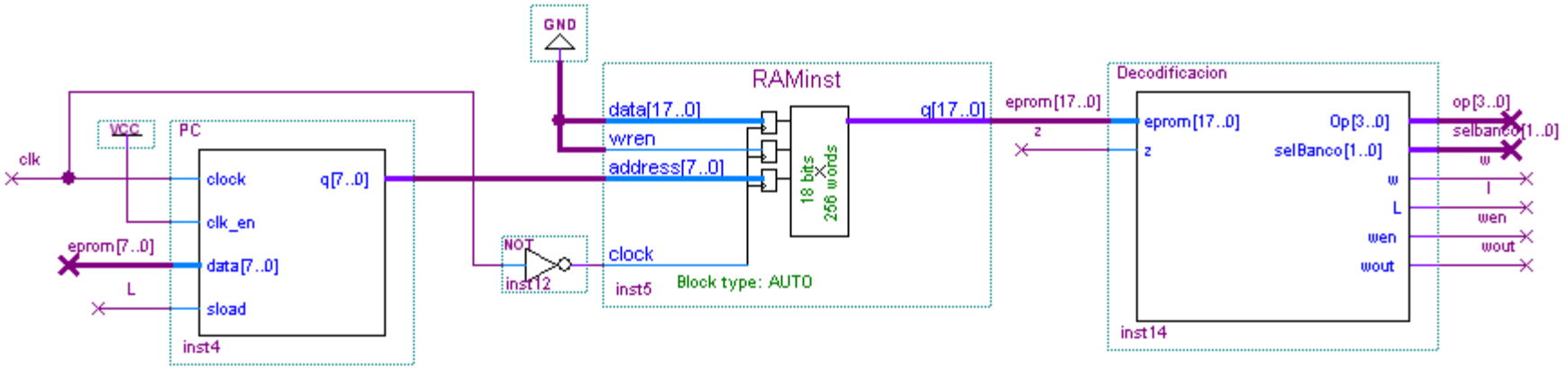
Implementación en Quartus II: Unidad Operativa



Ejercicio 13: Diseño de un Procesador Monociclo

Universidad del Valle

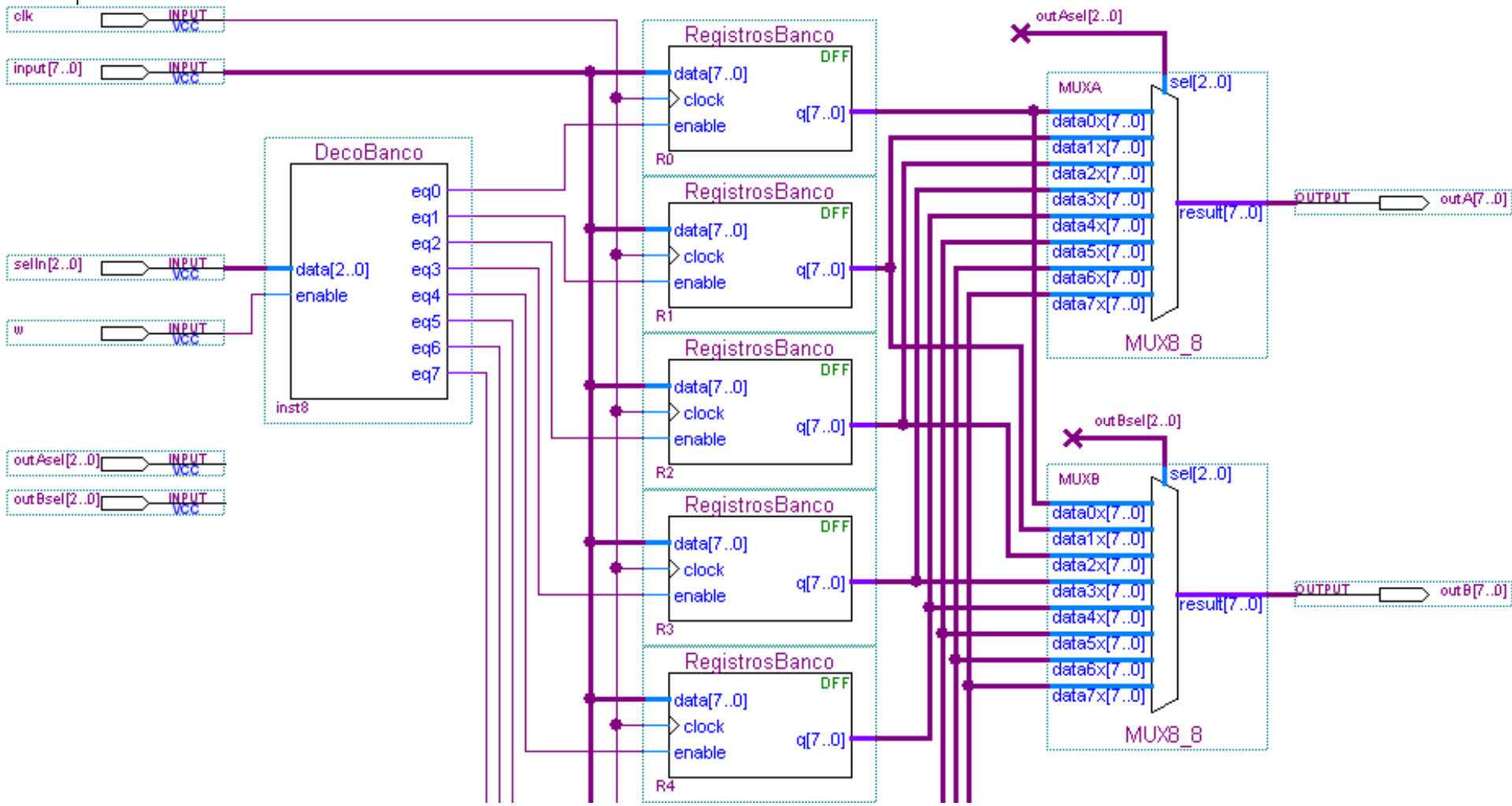
Implementación en Quartus II: Unidad de Control



Ejercicio 13: Diseño de un Procesador Monociclo

Universidad del Valle

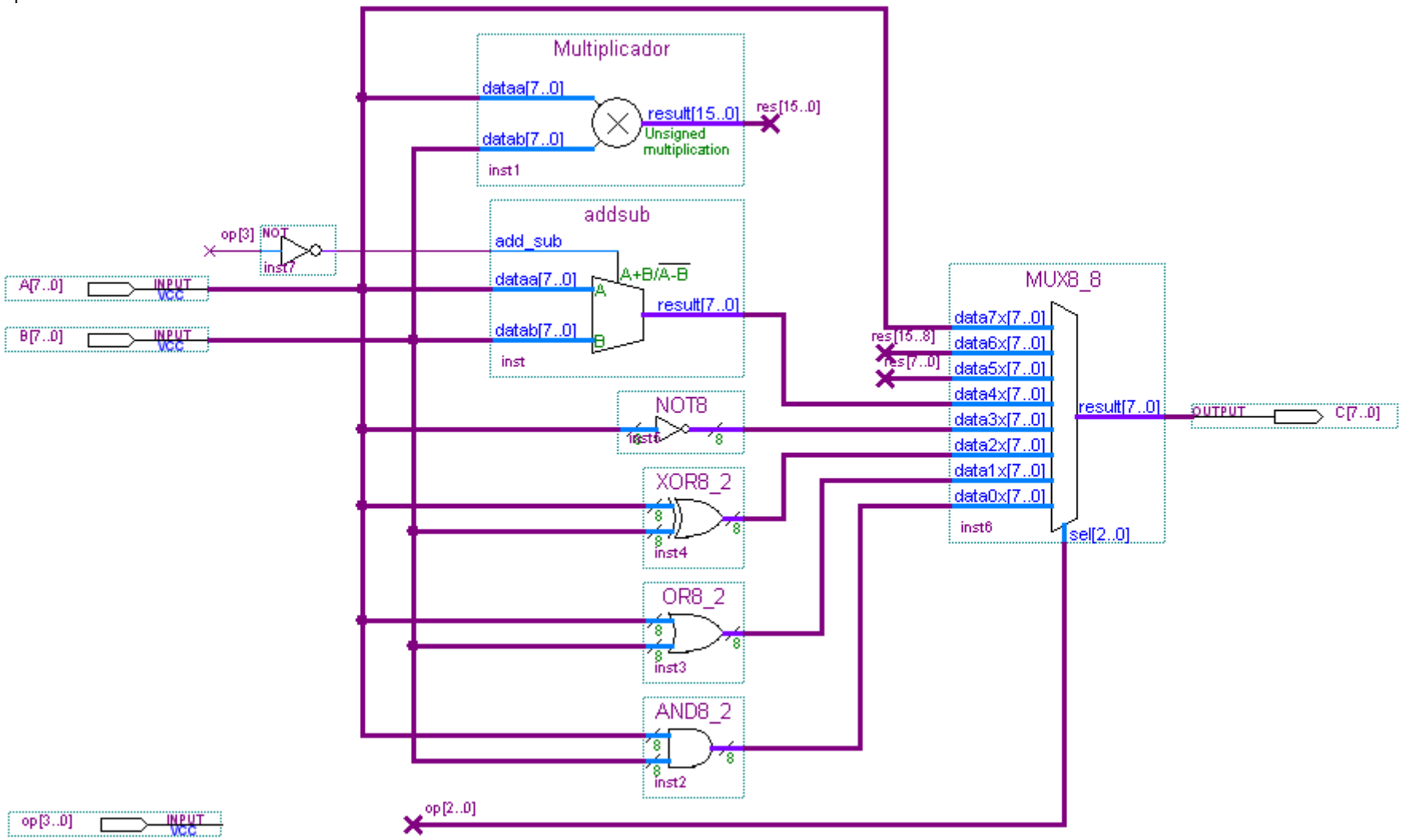
Banco de Registros



Ejercicio 13: Diseño de un Procesador Monociclo

Universidad del Valle

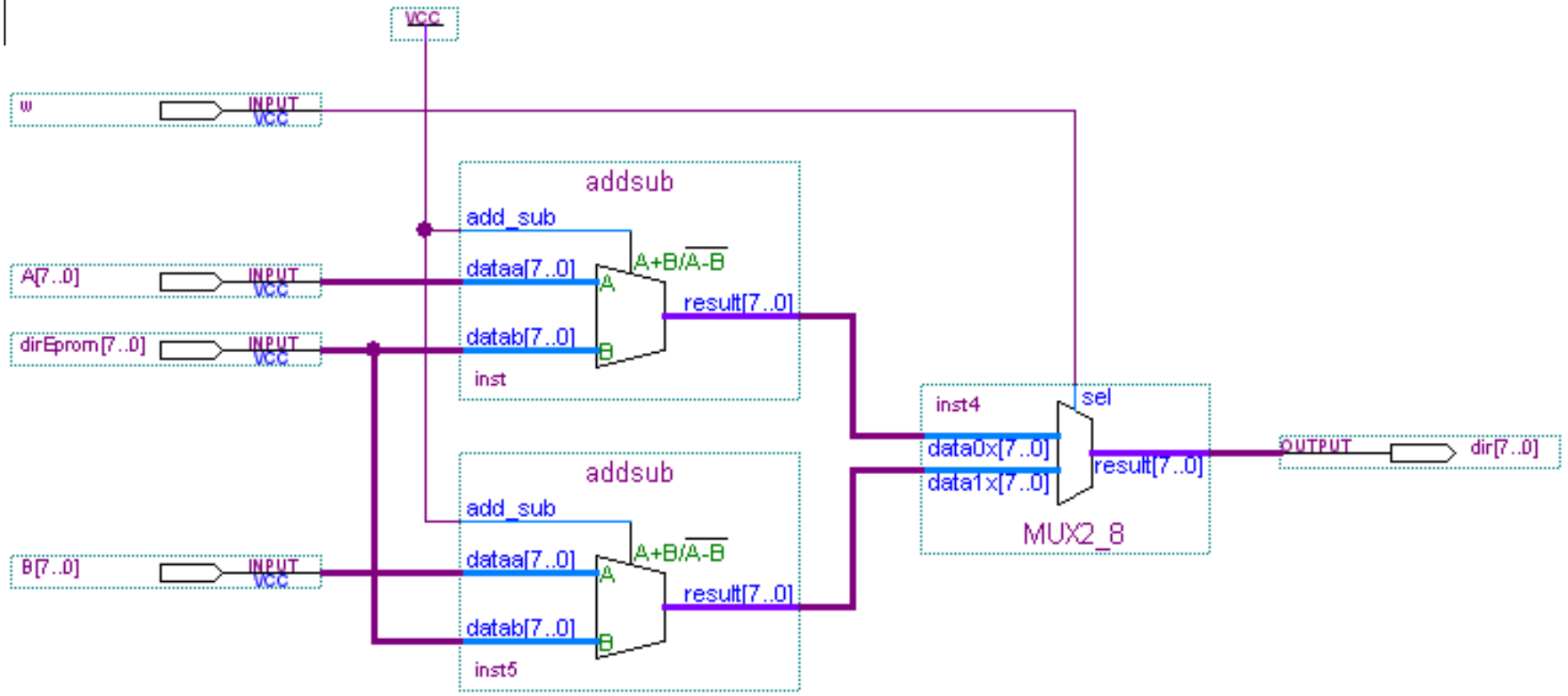
Unidad Lógica Aritmética ALU



Ejercicio 13: Diseño de un Procesador Monociclo

Universidad del Valle

Bloque Generador de Direcciones



Ejercicio 13: Diseño de un Procesador Monociclo

Universidad del Valle

Decodificación

- Genera el Código de Operación para la ALU.
- Establece la Fuente de información para el Banco de Registros.
- Genera las señales de control de escritura para el Banco de Registros, la RAM y el Registro de Salida.
- Activa la señal de carga “L” para realizar saltos en el PC.

Ejercicio 13: Diseño de un Procesador Monociclo

Universidad del Valle

Decodificación: Interpretación del Código.

COD	Instrucción	OP	Operación	selBanco	L	Wb	Wen	Wout
0000	STORE	1111	C ← A	11	0	0	1	0
0001	LOAD	1111	C ← A	01	0	1	0	0
0010	OUT	1111	C ← A	11	0	0	0	1
0011	IN	1111	C ← A	11	0	1	0	0
0100	JZ	1111	C ← A	11	"Z"	0	0	0
0101	AND	0000	C ← A AND B	00	0	1	0	0
0110	OR	0001	C ← A OR B	00	0	1	0	0
0111	XOR	0010	C ← A XOR B	00	0	1	0	0
1000	NOT	0011	C ← NOT(A)	00	0	1	0	0
1001	ADD	0100	C ← A + B	00	0	1	0	0
1010	SUB	1100	C ← A - B	00	0	1	0	0
1011	MULT B	0101	C ← L(A * B)	00	0	1	0	0
1100	MULT A	0110	C ← H(A * B)	00	0	1	0	0
1101	MOV (REG)	1111	C ← A	00	0	1	0	0
1110	MOV (DATO)	1111	C ← A	10	0	1	0	0
1111	NOP	1111	C ← A	11	0	0	0	0

Ejercicio 13: Diseño de un Procesador Monociclo

Programa de Ejemplo: Generación de la serie de Fibonacci

0.	R0 <= 0	111000000000000000
1.	R1 <= 1	111000100000000001
2.	R4 <= 0	111010000000000000
3.	R7 <= RAM[0 + R0]	000111100000000000
4.	R5 <= 0	111010100000000000
5.	R5 <= R7 - R5	101010111100000000
6.	JZ[10]	010000000000001010
7.	R2 <= R1	110101000100000000
8.	R2 <= R4 - R2	101001010000000000
9.	JZ[42]	010000000000101010
10.	R2 <= 1	111001000000000001
11.	R3 <= 1	111001100000000001
12.	R0 <= R0 + R1	100100000100000000
13.	RAM[0 + R0] <= R2	000000001000000000
14.	R5 <= R1	110110100100000000
15.	R5 <= R7 - R5	101010111100000000
16.	JZ[20]	010000000000010100
17.	R5 <= R1	110110100100000000
18.	R5 <= R4 - R5	101010110000000000
19.	JZ[32]	010000000000100000
20.	R6 <= R7	110111011100000000
21.	R5 <= 255	111010100011111111
22.	R6 <= R5 - R6	101011010100000000
23.	R6 <= R6 + R1	100111000100000000
24.	R0 <= R0 + R1	100100000100000000
25.	RAM[0 + R0] <= R3	000000001100000000
26.	R3 <= R3 + R2	100101101000000000
27.	R2 <= R3 - R2	101001001100000000
28.	R6 <= R6 + R1	100111000100000000
29.	R4 <= R5	110110010100000000
30.	R4 <= R6 - R4	101010011000000000
31.	JZ[24]	010000000000011000
32.	R6 <= R7	110111011100000000
33.	R6 <= R5 - R6	101011010100000000
34.	R4 <= 1	111010000000000001
35.	R3 <= RAM[0 + R4]	000101110000000000
36.	Rout <= R3	001000001100000000
37.	R6 <= R6 + R1	100111000100000000
38.	R4 <= R4 + R1	100110000100000000
39.	R2 <= R5	110101010100000000
40.	R2 <= R6 - R2	101001011000000000
41.	JZ[35]	010000000000100011

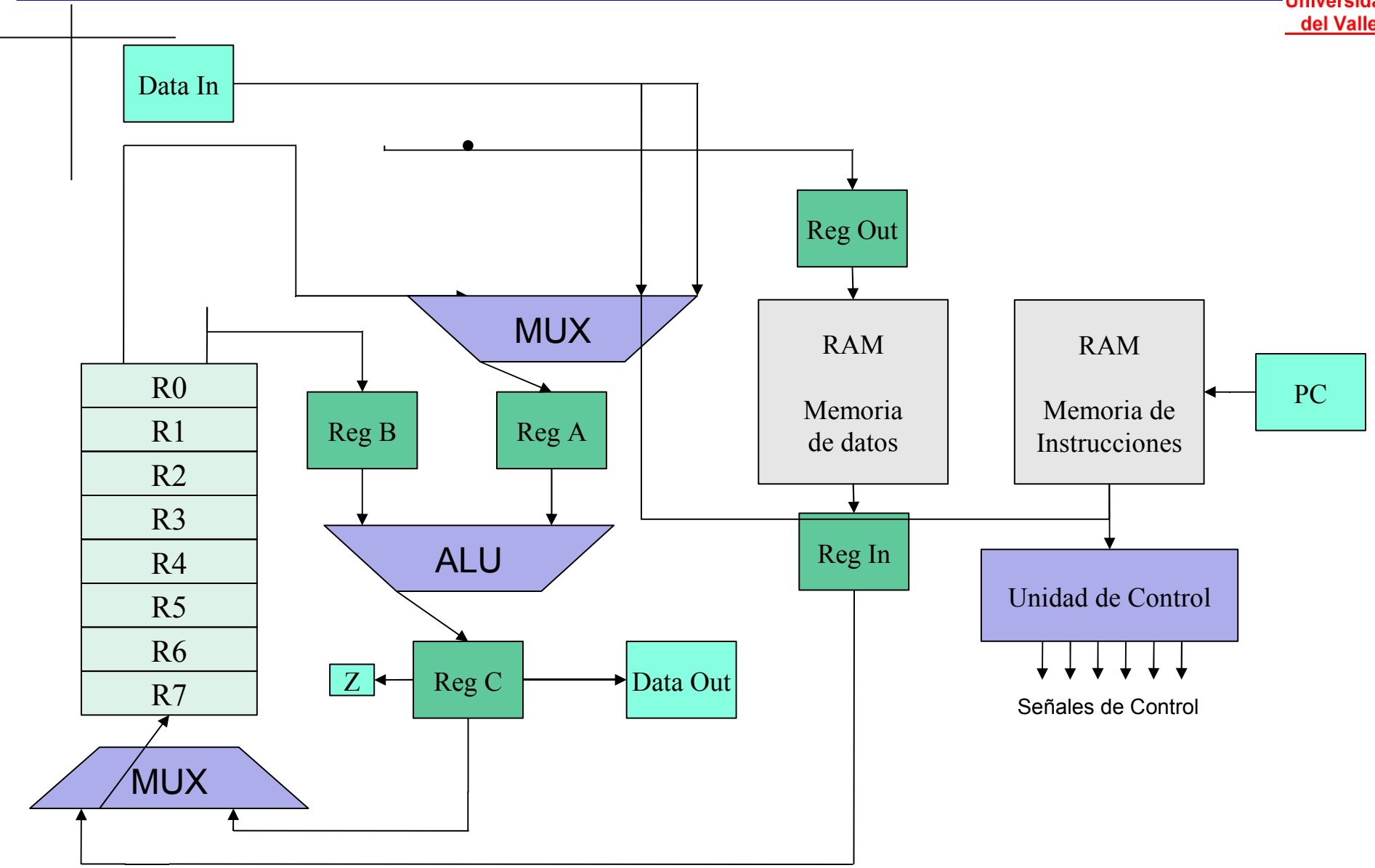
Ejercicio 13: Diseño de un Procesador Monociclo

Universidad del Valle

Resumen de Desempeño:

- Dispositivo de Prueba: Stratix II – EP2S15F484C3
- ALUT's Combinacionales: 166 / 12480
- Registros Logicos: 81 / 12480
- Máxima Frecuencia de Operación: 50.45MHz
- Tiempo de Ejecución de los primeros 6 términos de las serie de Fibonacci: 2.6 uS

Ejercicio 14: Diseño de un Procesador RISC "Pipeline"



Características de un Procesador RISC

- Acceso a Memoria solo con LOAD y STORE.
- Manipulación de Datos "Registro – Registro".
- Numero limitado de modos de direccionamiento.
- Formatos de Instrucciones de longitud fija
- Las instrucciones ejecutan operaciones básicas.

  - Alto Rendimiento
  - Alta velocidad de ejecución
  - Evita accesos a memoria
  - Banco de Registros grande
  - Unidad de control cableada*
  - Pipeline

Ejercicio 14: Diseño de un Procesador "Pipeline"

Universidad del Valle

Ciclo de Instrucción:

	Fetch	Decode	Load	Execute	Store
Ciclo 1	MOV				
Ciclo 2	ADD	MOV			
Ciclo 3	MULT	ADD	MOV		
Ciclo 4	XOR	MULT	ADD	MOV	
Ciclo 5	SUB	XOR	MULT	ADD	MOV
Ciclo 6	NOT	SUB	XOR	MULT	ADD

Ejercicio 14: Diseño de un Procesador "Pipeline"

Universidad del Valle

Problemas

- Conflicto de Datos (dependencia de datos)
  - Solución Software:
    - Introducción de NOP's:
      - Reduce el "Throughput".
  - Soluciones Hardware:
    - Inserción de Burbujas (NOP's):
      - Reduce el "Throughput".
    - **Anticipación de Datos:**
      - No reduce el "Throughput".
      - Añade Retardo Combinacional.

Ejercicio 14: Diseño de un Procesador "Pipeline"

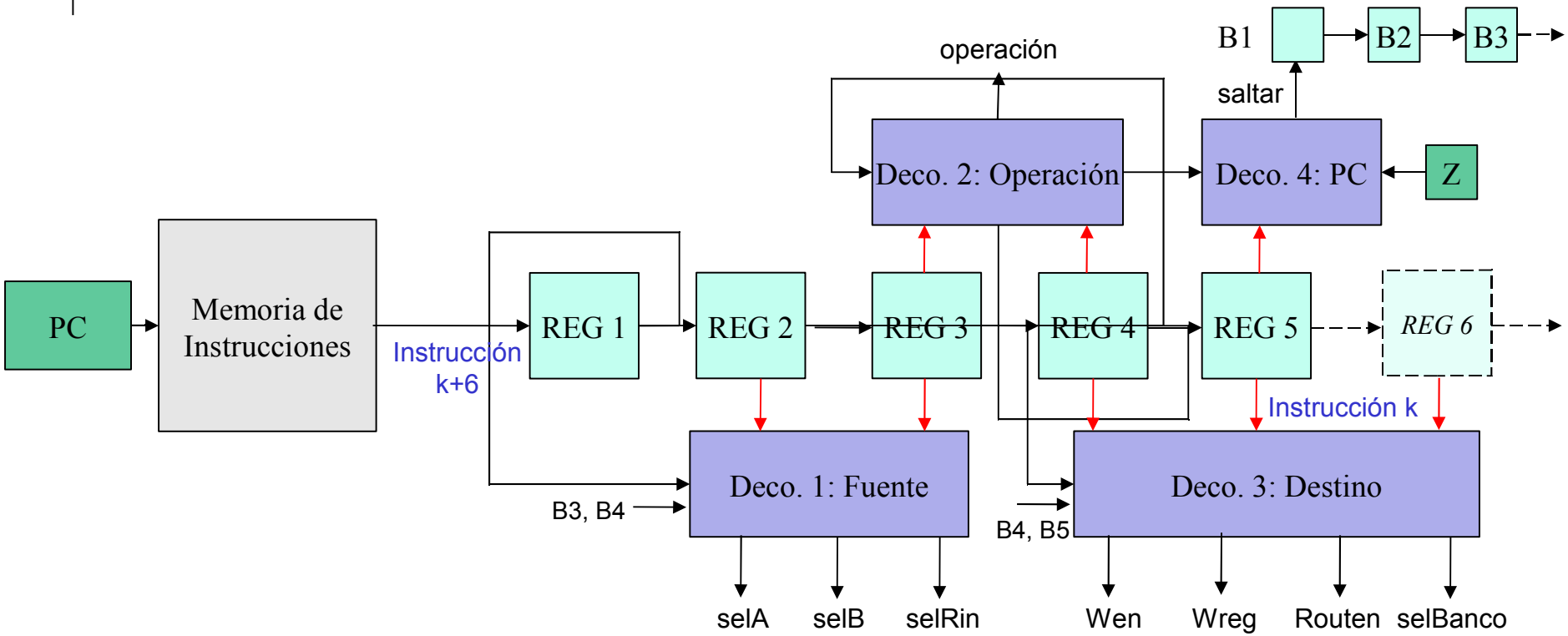
Universidad del Valle

Problemas

- Conflictos de Control (saltos o bifurcaciones)
  - Solución Software:
    - Bifurcación Retardada (NOP's):
      - Reduce el "Throughput".
  - Soluciones Hardware:
    - Parada de Conflicto de bifurcación (detener PC):
      - Reduce el "Throughput".
    - **Predicción de Salto** (asumir que los saltos no se ejecutan):
      - No reduce "Throughput" cuando no se ejecuta un salto

Ejercicio 14: Diseño de un Procesador "Pipeline"

Unidad de Control (Registros de Segmentación)



Ejercicio 14: Diseño de un Procesador "Pipeline"

Universidad del Valle

Conjunto de Instrucciones

0000	STORE	1000	NOT
0001	LOAD	1001	ADD
0010	OUT	1010	SUB
0011	IN	1011	MULT B
0100	BNEQ	1100	MULT A
0101	AND	1101	MOV (REG)
0110	OR	1110	MOV (DATO)
0111	XOR	1111	NOP

Ejercicio 14: Diseño de un Procesador "Pipeline"

Universidad del Valle

Aplicación: Generación de la Serie de Fibonacci

Algoritmo en C++:

```
public class Fibonacci
{
        public static void main (String [] args)
        {
                int f1=1, f2=1;
                System.out.println (f1);
                for(int k = 1, k<N, k++)
                {
                        System.out.println(f2);
                        f2+=f1;
                        f1 = f2 - f1;
                }
        }
}
```

Ejercicio 14: Diseño de un Procesador "Pipeline"

Universidad del Valle

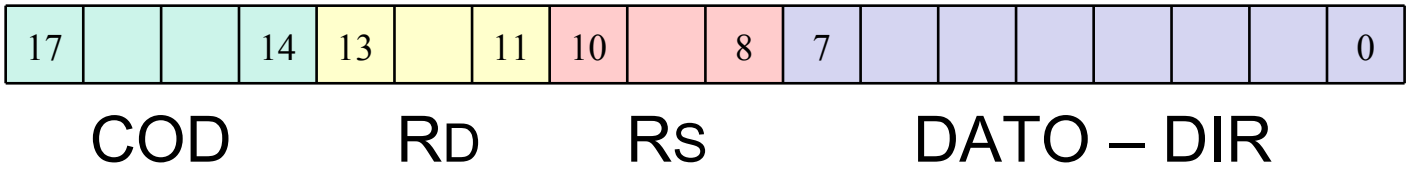
Aplicación: Generación de la Serie de Fibonacci

Código en "Ensamblador"

```
0.  R0 <= 0
3.  R1 <= 1
4.  R4 <= 0
5.  R7 <= RAM[0 + R0]          (R7 <= N: numero de términos que se desea generar)
6.  Bneq R4, R7 [6]            (saltar a línea 6 si N ≠ 0, no salta si N = 0)
7.  Bneq R4, R1 [30]           (salir, este salto siempre se ejecuta porque R4 ≠ R1)
8.  R2 <= 1                    (primer termino de la serie)
9.  R3 <= 1                    (segundo termino de la serie)
10. R0 <= R0 + R1              (incrementar R0)
11. RAM[0 + R0] <= R2          (Almacenar primer termino de la serie en RAM)
12. Bneq R1, R7 [12]           (saltar a línea 12 si N ≠ 1, no salta si N = 1)
13. Bneq R4, R1 [22]           (ir a la secuencia de visualización)
14. R6 <= R7                   (R6 <= N)
15. R5 <= 255                  (condición de parada)
16. R6 <= R5 – R6              (R6 <= 255 – N)
17. R6 <= R6 + R1              (incrementar R6: R6 <= 255 – N + 1)
for(int k = 1, k<N, k++) Cálculo de los términos de la serie
19. R0 <= R0 + R1              (incrementar R0)
20. RAM[0 + R0] <= R3          (Almacenar termino k de la serie en RAM)
21. R3 <= R3 + R2              (f2+=f1;            )
22. R2 <= R3 – R2              (f1 = f2 - f1;)
23. R6 <= R6 + R1              (incrementar R6)
24. Bneq R6, R5 [16]           (saltar a 16 mientras R6 ≠ 255)
Subrutina de Visualización de los términos
26. R6 <= R7                   (R6 <= N)
27. R6 <= R5 – R6              (R6 <= 255 – N)
28. R4 <= 1                    (posición "1" de la RAM)
29. R3 <= RAM[0 + R4]          (cargar dato k de la RAM a R4)
30. Rext <= R3                 (visualizar termino k de la serie)
31. R6 <= R6 + R1              (incrementar R6)
32. R4 <= R4 + R1              (incrementar R4: incrementar posición de memoria)
33. Bneq R5, R6 [25]           (saltar a 25 mientras R6 ≠ 255)
```

Ejercicio 14: Diseño de un Procesador "Pipeline"

Formato de Instrucción

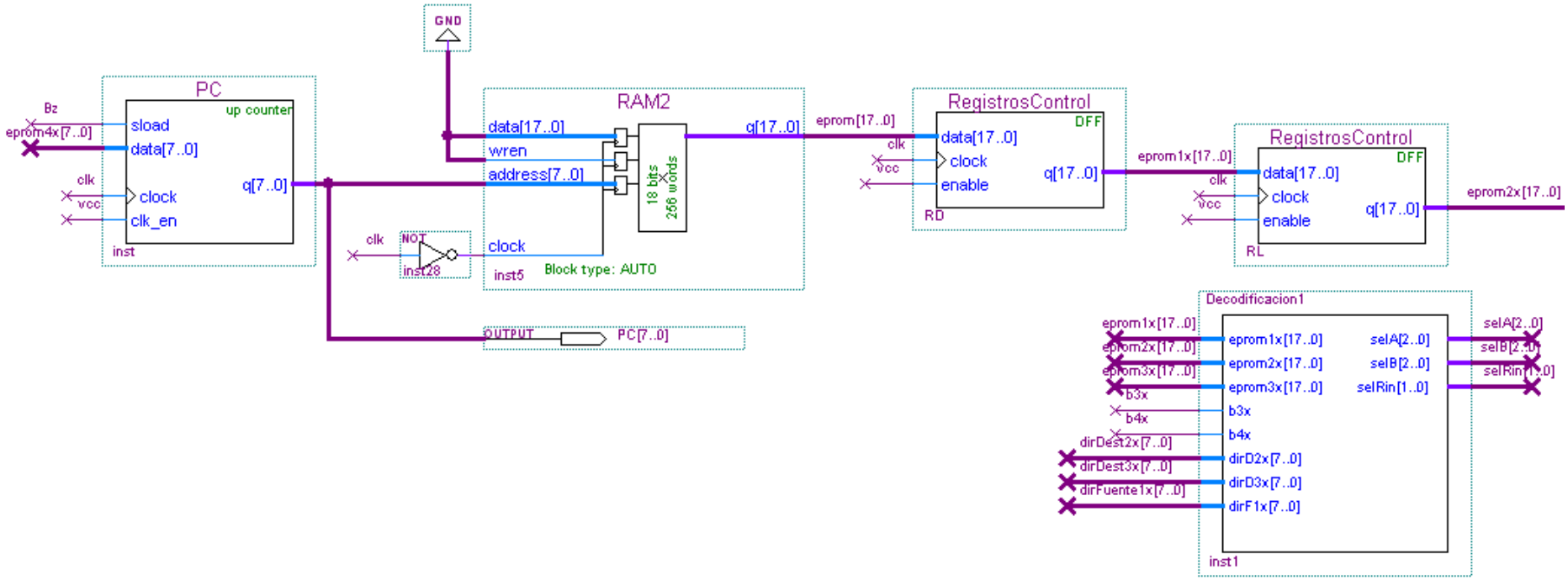


Numero de Instrucciones	$= 2^4$	$= 16$
Numero de Registros	$= 2^3$	$= 8$
Posiciones de Memoria	$= 2^8$	$= 256$

R3 ← R1	MOV R3, R1	=	1101 011 001 xxxxxxxx
R3 ← "10"	MOV R3, "10"	=	1110 011 xxx 00001010
RAM[11 + R3] ← R2	STORE [11+R3], R2	=	0000 011 010 00000011
R1 ← **R1** + R7	ADD R1, R7	=	1001 001 111 xxxxxxxx

Ejercicio 14: Diseño de un Procesador "Pipeline"

Unidad de Control: Ciclos de Búsqueda (Fetch) y Decodificación



Ejercicio 14: Diseño de un Procesador "Pipeline"

Universidad del Valle

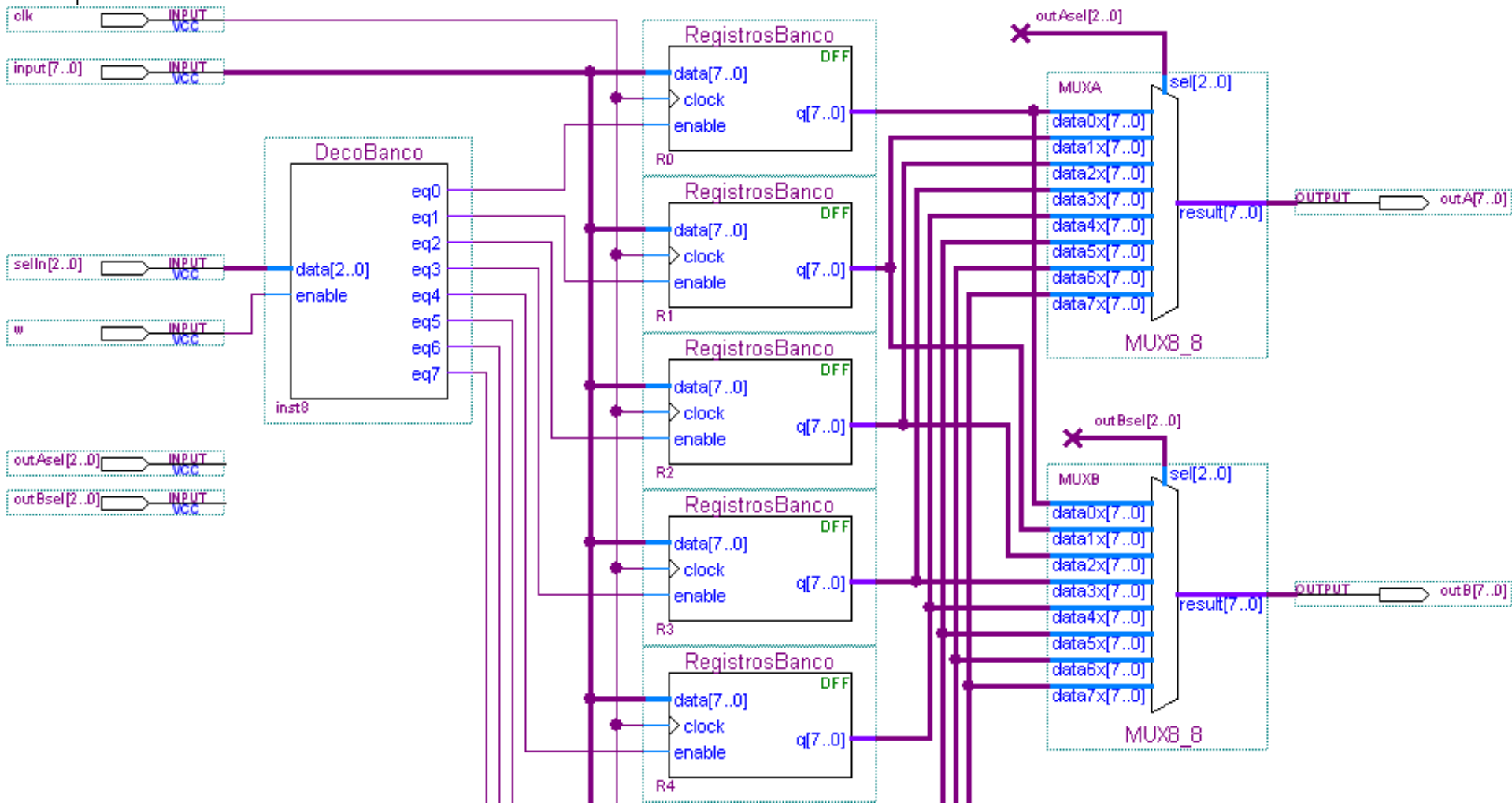
Decodificacion1: Búsqueda del Operando.

- Selecciona qué datos del banco de registros se van a "leer" (selA y selB) y qué datos se van a escribir en los registros RA y RB (Banco, Data In, dato inmediato).
- Selecciona qué dato debe leerse en Rin en una operación "Load".
- En caso de presentarse Dependencia de Datos, se modifican los selectores para que la información se obtenga de una etapa diferente (Rout, Rout2, RAM, RC, ALU, Rin).
- Se detecta de qué manera se llegó a la instrucción actual (salto o secuencia) para establecer de donde se deben obtener los datos.

Ejercicio 14: Diseño de un Procesador "Pipeline"

Universidad del Valle

Banco de Registros



Ejercicio 14: Diseño de un Procesador "Pipeline"

Universidad del Valle

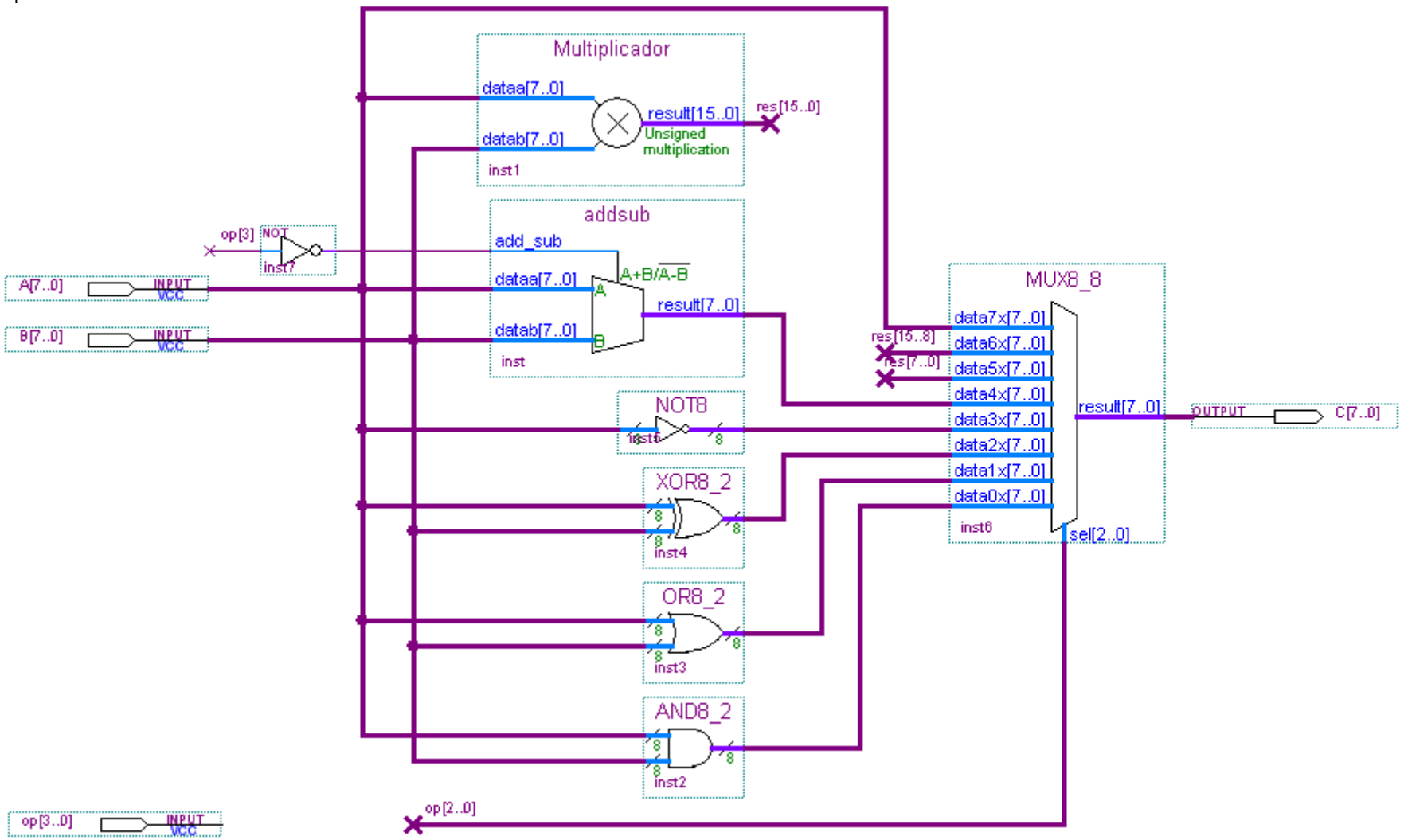
Decodificacion2: Interpretación del Código.

COD	Instrucción	OP	Operación
0000	STORE	1111	C ← A
0001	LOAD	1111	C ← A
0010	OUT	1111	C ← A
0011	IN	1111	C ← A
0100	BEQ	1100	C ← A - B
0101	AND	0000	C ← A AND B
0110	OR	0001	C ← A OR B
0111	XOR	0010	C ← A XOR B
1000	NOT	0011	C ← NOT(A)
1001	ADD	0100	C ← A + B
1010	SUB	1100	C ← A - B
1011	MULT B	0101	C ← L(A * B)
1100	MULT A	0110	C ← H(A * B)
1101	MOV (REG)	1111	C ← A
1110	MOV (DATO)	1111	C ← A
1111	NOP	1111	C ← A

Ejercicio 14: Diseño de un Procesador "Pipeline"

Universidad del Valle

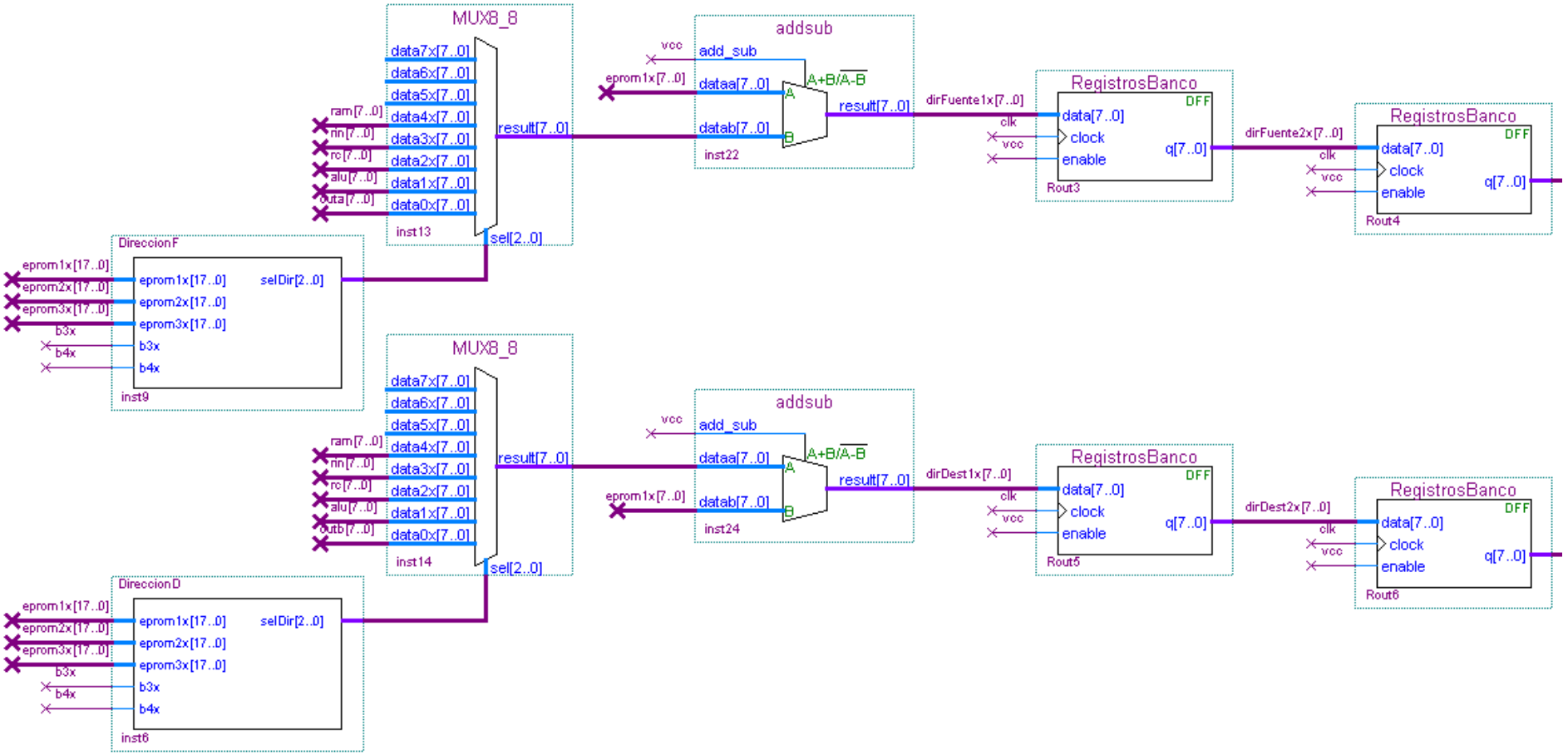
Unidad Lógica Aritmética ALU



Ejercicio 14: Diseño de un Procesador "Pipeline"

Universidad del Valle

Unidad de Direcciones



Ejercicio 14: Diseño de un Procesador "Pipeline"

Universidad del Valle

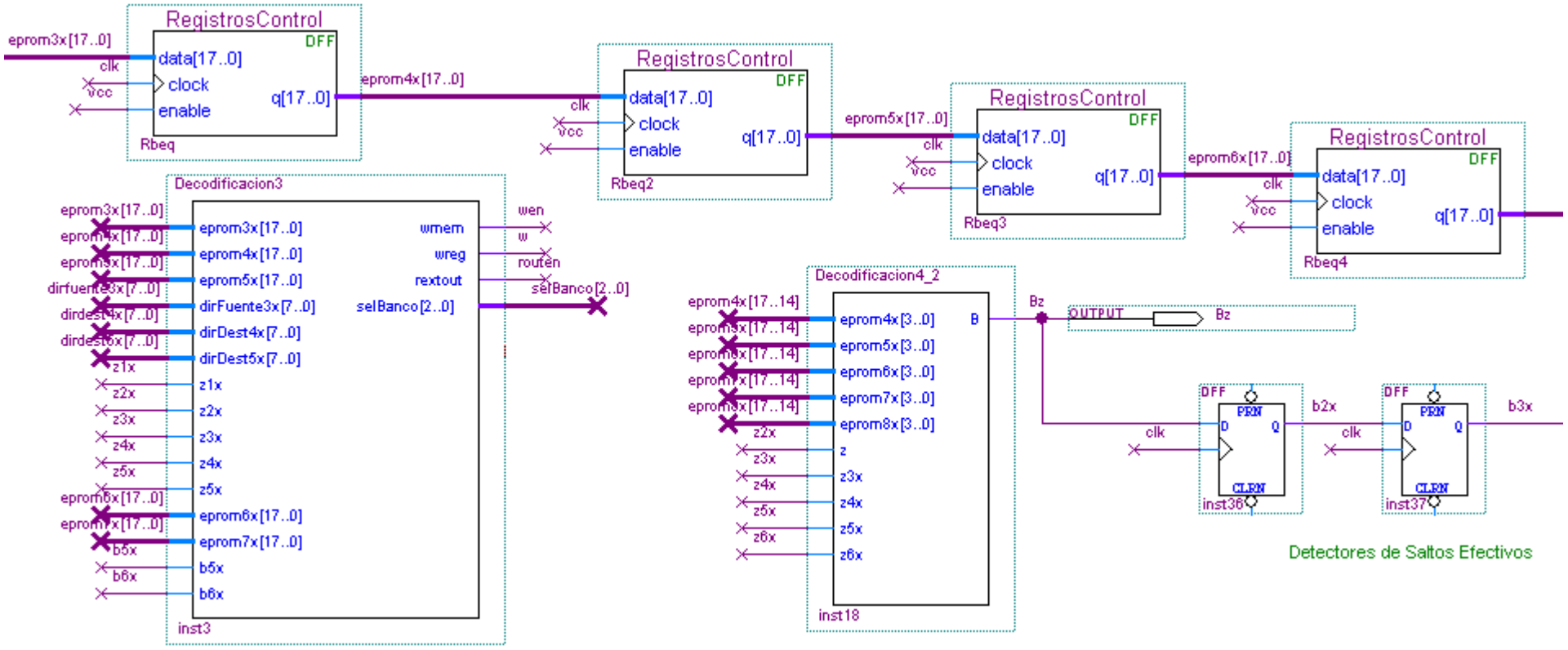
Decodificacion3: Fase de Escritura.

- Habilita las señales de escritura de la RAM, del Banco de Registros y del Registro de Salida.
- Selecciona el dato que va a ser almacenado en el Banco de Registros.
- Tiene en cuenta la dependencia de datos, la ocurrencia de saltos y la precedencia de la instrucción actual.

Ejercicio 14: Diseño de un Procesador "Pipeline"

Universidad del Valle

Decodificacion3 y Decodificacion4



Decodificacion4: Manejo del PC.

- Establece si se cumplen o no las condiciones para ejecutar un salto. Genera la señal de carga del PC.
- En caso de la presencia de dos saltos consecutivos o intercalados por otra instrucción, desactiva la señal de carga si el primer salto evaluado se ejecuta.
- Cuando la condición de salto se cumple, se habilita la carga del PC para forzarlo a una dirección especifica.

Ejercicio 14: Diseño de un Procesador "Pipeline"

Universidad del Valle

Resumen de Desempeño:

- Dispositivo de Prueba: Stratix II – EP2S15F484C3
- ALUT's Combinacionales: 283 / 12480
- Registros Lógicos: 211 / 12480
- Máxima Frecuencia de Operación: 117.58 MHz
- Tiempo de Ejecución de los primeros 6 términos de las serie de Fibonacci: 1.26 uS