

Simulador de circuitos digitales



David Kutugata A01138865 Humberto Acosta A01195234

6 de Mayo, 2015

ITESM

Table of Contents

[Descripción del Proyecto 3](#_Toc418646061)

[Objetivos y Alcance 3](#_Toc418646062)

[Análisis de Requerimientos 3](#_Toc418646063)

[Descripción de Casos de Prueba 3](#_Toc418646064)

[Bitácora 4](#_Toc418646065)

[Descripción del Lenguaje 5](#_Toc418646066)

[Descripción Genérica 5](#_Toc418646067)

[Descripción de Errores 5](#_Toc418646068)

[Descripción del Compilador 6](#_Toc418646069)

[Equipo de Cómputo, Lenguaje y Utilerías necesarios 6](#_Toc418646070)

[Análisis de Léxico 6](#_Toc418646071)

[Análisis de Sintaxis 7](#_Toc418646072)

[Generación de Cuádruplos y Análisis Semántico 9](#_Toc418646073)

[Diagramas de Sintaxis 9](#_Toc418646074)

[Consideraciones Semánticas: 16](#_Toc418646075)

[Administración de Memoria 16](#_Toc418646076)

[Descripción de la Máquina Virtual 17](#_Toc418646077)

[Equipo de Cómputo, Lenguaje y Utilerías 17](#_Toc418646078)

[Administración de Memoria 17](#_Toc418646079)

[Pruebas de Funcionamiento 19](#_Toc418646080)

[Pruebas Básicas 19](#_Toc418646081)

[Ejemplo 1: 19](#_Toc418646082)

[Ejemplo 2: 21](#_Toc418646083)

[Ejemplo 3: 23](#_Toc418646084)

[Ejemplo 4: 25](#_Toc418646085)

[Ejemplo 5: 27](#_Toc418646086)

[Pruebas Reales 30](#_Toc418646087)

[Ejemplo 6-Unidad de Control 30](#_Toc418646088)

[Ejemplo 7-Unidad Aritmética-Lógica 35](#_Toc418646089)

[Listados 38](#_Toc418646090)

[Anexos 42](#_Toc418646091)

[Código Flex 42](#_Toc418646092)

[Código Bison 42](#_Toc418646093)

[Código Máquina Virtual 46](#_Toc418646094)

# Descripción del Proyecto

## Objetivos y Alcance

El objetivo del proyecto es proporcionar una herramienta para simulación de circuitos digitales con la ventaja de ser fácil de usar y ser más veloz a los actuales. La herramienta más utilizada para éste fin es el ISE Design Suite de Xilinx, el cuál toma mucha memoria de ejecución y tiene un proceso de compilación lento, dado que baja el programa a su tarjeta especial. Ésta nueva herramienta permitirá realizar numerosas pruebas solo proporcionando en código la arquitectura del circuito a simular, y después dando los valores de entrada a probar, los desplegará en pantalla y dará opción de realizar cuantas pruebas más sean necesarias para el usuario.

## Análisis de Requerimientos

Los requerimientos necesarios para la utilización de este lenguaje son los siguientes:

* Conocimientos básicos de programación
* Conocimientos básicos de algebra booleana
* Conocimiento de conversión decimal a booleana e inversa
* Maquina con 185Kb de espacio de Disco Duro

Una vez que el usuario haya comprendido el correcto funcionamiento de la sintaxis del lenguaje Kuac por medio del manual de usuario, se debería de poder ejecutar sin problema alguno de una manera más eficiente que el anteriormente mencionado en el punto de Objetivos y Alcances.

## Descripción de Casos de Prueba

Los casos de prueba son de dos tipos, básicos y reales, los básicos solo son para probar los diferentes estatutos del lenguaje y los reales prueban diferentes elementos de un procesador como la unidad aritmética y lógica o la unidad de control.

Los casos de prueba básicos son:

* Estatuto if
* Estatuto if-else
* Estatuto case
* Asignaciones y operaciones aritméticas/lógicas
* Todas las anteriores combinadas

Los casos de prueba reales son:

* Unidad aritmética lógica
* Unidad de control

## Bitácora

A lo largo de los últimos meses se estuvo trabajando en la realización de este proyecto de una manera continua con entregas semanales, teniendo como avances parciales los siguientes puntos en las fechas aquí mencionadas.

Marzo 12 – Definición de la primera propuesta de proyecto, junto con su Léxico y Sintaxis.

Marzo 18 – Definición de la segunda propuesta de proyecto, adaptando Léxico y Sintaxis de la primera propuesta a la nueva y definición de Tablas de Variables.

Marzo 25 – Programación de Tablas de Variables, Cubo Semántico y Generación de Cuádruplos de Expresiones Aritméticas.

Abril 8 – Generación de Cuádruplos de Asignación, Lectura y Escritura y se comenzó a trabajar en Generación de Cuádruplos Condicionales.

Abril 15 – Definición del mapa de memoria y esqueleto de máquina virtual.

Abril 22 – Programación de expresiones aritméticas en máquina virtual.

Abril 29 – Finalización de generación de Cuadruplos secuenciales e inicializo la programación de estatutos secuenciales en máquina virtual y comienzo de documentación.

Mayo 1 – Finalización de la primera versión de la máquina virtual funcional básica.

Mayo 4 – Inicio Debugeo de Máquina Virtual en búsqueda de errores.

Mayo 5 – Máquina Virtual completamente funcional, con pruebas grandes realizadas y finalización de la documentación.

# Descripción del Lenguaje

El lenguaje es nombrado Kuac por la combinación de los apellidos (Kutugata-Acosta) de los autores y desarrolladores del mismo.

## Descripción Genérica

El Kuac se compone de dos grandes partes, la declaración de variables y la descripción del circuito.

La primera se usa para definir las entradas y salidas del circuito, marcadas como variables “in” o “out”, también se permite la declaración de variables tipo “signal” las cuales permiten la simplificación del código.

La segunda se usa para describir el comportamiento del circuito, es decir, el estado de las variables de salida de acuerdo a los diferentes casos de las variables de entrada.

## Descripción de Errores

Dejando fuera los errores de léxico y de sintaxis, el error principal de semántica es el tratar de utilizar una variable de salida como entrada para un if o un case. También es un error de semántica el tratar de asignarle un valor a una variable de entrada, ya que esos valores son definidos y modificados en ejecución.

# Descripción del Compilador

## Equipo de Cómputo, Lenguaje y Utilerías necesarios

Requerimientos de Hardware:

* 256MB de Memoria
* 185KB de Disco Duro

Requerimientos de Software:

* Sistema Operativo: Cualquier versión de Windows o Linux
* Editor de texto

## Análisis de Léxico

Lista de palabras reservadas:

|  |  |
| --- | --- |
| **Palabra** | **Significado** |
| entity | Indicador para iniciar el programa y la declaración de variables |
| is | Indicador de léxico |
| in | Indicador de variable de entrada |
| out | Indicador de variable de salida |
| signal | Indicador de variable tipo señal |
| bit | Indicador de que la variable es un bit |
| vector | Indicador de que la variable es un arreglo de bits |
| to | Indicador de léxico |
| end | Indicador de léxico para indicar fin de bloque |
| architecture | Indicador para iniciar el bloque de descripción del circuito |
| of | Indicador de léxico |
| begin | Indicador de inicio de arquitectura |
| if | Indicador del estatuto if |
| then | Indicador del estatuto if |
| else | Indicador del estatuto if |
| case | Indicador del estatuto case |
| when | Indicador del estatuto case |
| break | Indicador del estatuto case |
| other | Indicador del estatuto case |
| SHL | Operador “shift left” |
| SHR | Operador “shift right” |

Lista de tokens:

|  |  |
| --- | --- |
| **Token** | **Significado** |
| "(" | Operador para abrir paréntesis |
| ")" | Operador para cerrar paréntesis |
| ":" | Indicador usado en declaración de variables y en el estatuto case |
| ";" | Indicador de fin de línea |
| "==" | Comparador de igualdad |
| "!=" | Comparador de diferencia |
| "=" | Operador de asignación |
| "<" | Comparador de menor que |
| ">" | Comparador de mayor que |
| "&" | Operador “Y” |
| "|" | Operador “O” |
| "!" | Operador “NO” |
| "^" | Operador “O exclusivo” |

Lista de expresiones regulares

|  |  |
| --- | --- |
| **Expresión** | **Significado** |
| [a-zA-Z][\_a-zA-Z0-9]\* | ID |
| [0-9]+ | Entero |

## Análisis de Sintaxis

Lo siguiente describe la gramática formal del compilador:

PROGRAM :VARS BLOQUE

;

VARS :entity ID is '(' VAR1 ')' end ID ';'

;

VAR1 :VAR VAR2

;

VAR2 :

|VAR1

;

VAR :ID ':' INOUT TIPO ';'

;

INOUT :in

|out |signal ;

TIPO :vector '(' CTE\_INT ')'

|bit

;

BLOQUE :architecture of ID is begin ESTATUTO1 end ID ';'

;

ESTATUTO1 :ESTATUTO ESTATUTO2

;

ESTATUTO2 :

|ESTATUTO1

;

ESTATUTO :ASIGNACION

|CONDIF

|CONDCASE

;

ASIGNACION :ID '=' EXP ';'

;

CONDIF :IF '(' EXPRESION ')' THEN ESTATUTO1 CONDIF1

;

CONDIF1 :end IF ';'

|ELSE ESTATUTO1 end IF ';'

;

CONDCASE :CASE ID is CONDCASE1 end CASE ';'

;

CONDCASE1 :WHEN OPCASE ':' ESTATUTO1 BREAK ';' CONDCASE1

|

;

OPCASE :CTE\_INT

|OTHER

;

EXPRESION :EXP EXPRESION1

;

EXPRESION1 :

|'==' EXP

|'!=' EXP

|'<' EXP

|'>' EXP ;

EXP :LOGIC LOGIC1

|'!' EXP

;

LOGIC1 :

|'&' EXP

|'|' EXP |'^' EXP ;

LOGIC :TERMINO TERMINO1

;

TERMINO1 :

|'+' LOGIC |'-' LOGIC ;

TERMINO :FACTOR FACTOR1

;

FACTOR1 :

|SHL TERMINO |SHR TERMINO ;

FACTOR :'(' EXP ')'

|VAR\_CTE

;

VAR\_CTE :ID

|CTE\_INT ;

## Generación de Cuádruplos y Análisis Semántico

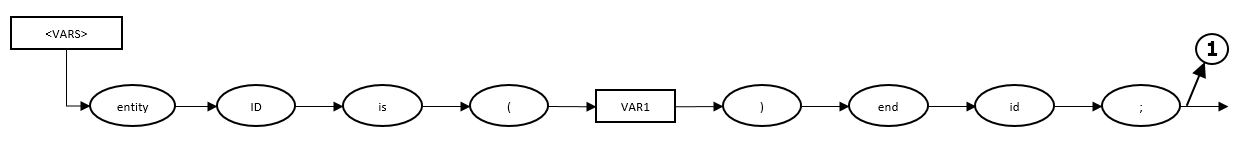
### Diagramas de Sintaxis

Los siguientes diagramas representan el proceso semántico y de generación de cuádruplos:

<PROGRAM>

VARS

BLOQUE



<VAR>

INOUT

TIPO

<VAR1>

VAR

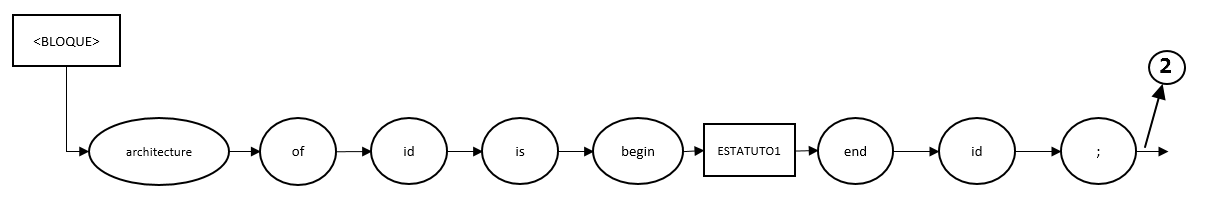
VAR2

<VAR2>

VAR1

<INOUT>

<TIPO>



<ESTATUTO1>

ESTATUTO

ESTATUTO2

<ESTATUTO2>

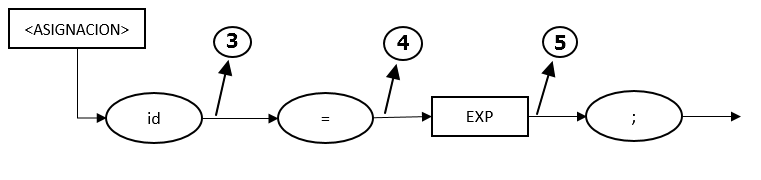
ESTATUTO1

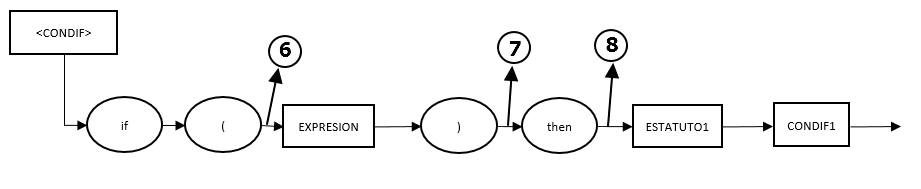
<ESTATUTO>

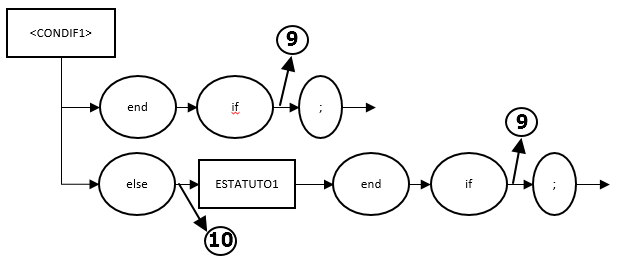
ASGINACION

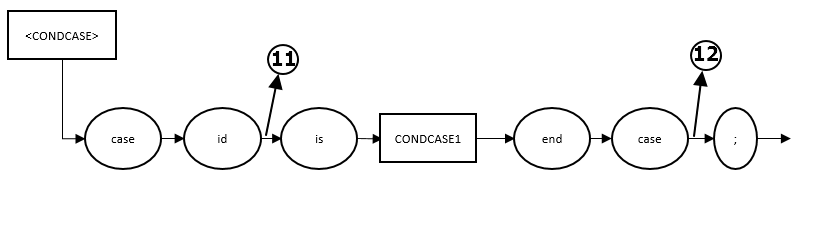
CONDIF

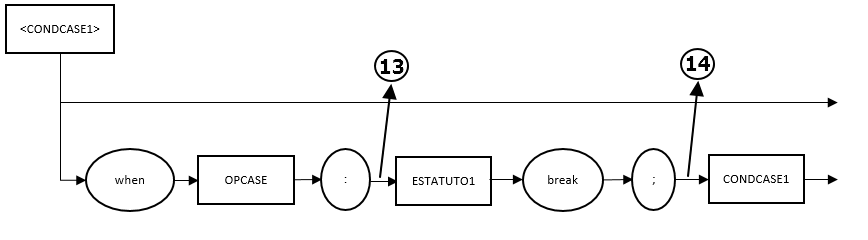
CONDCASE

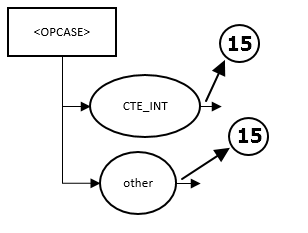








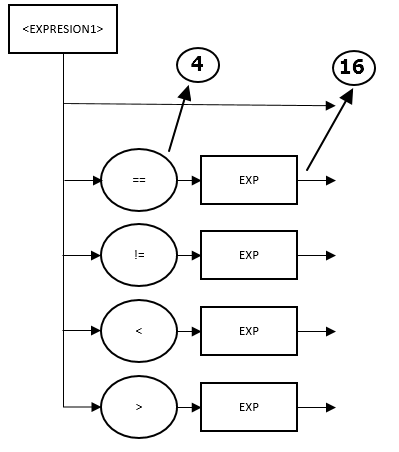




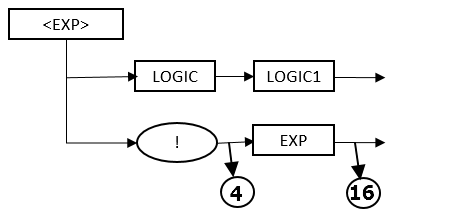
<EXPRESION>

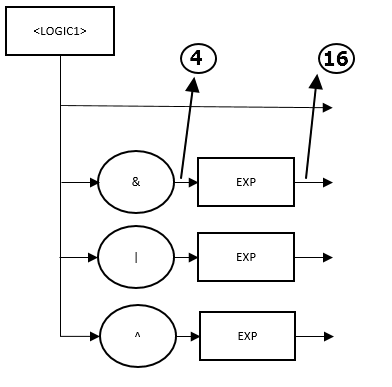
EXP

EXPRESION1



El diagrama anterior hace las mismas acciones en todos sus casos.



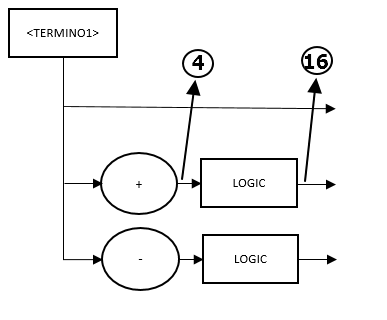


El diagrama anterior hace las mismas acciones en todos sus casos.

<LOGIC>

TERMINO

TERMINO1

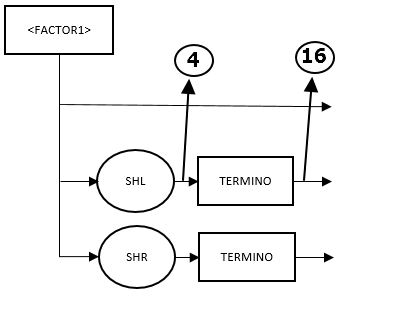


El diagrama anterior hace las mismas acciones en todos sus casos.

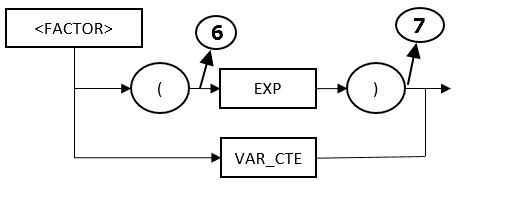
<TERMINO>

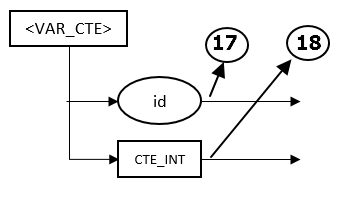
FACTOR

FACTOR1



El diagrama anterior hace las mismas acciones en todos sus casos.





A continuación se describen las acciones semánticas y de generación de código intermedio:

1: Actualizar la tabla de variables y revisar que ambos ID's sean iguales

2: Crear el archivo objeto con código intermedio y revisar que ambos ID's sean iguales

3: Agregar a Pila de Operandos. Si se agrega una variable de entrada, error semántico.

4: Agregar a Pila de Operadores

5: Crear Cuádruplo

6: Marcar fondo falso

7: Sacar Marca de fondo falso

8: Crear Cuádruplo y guardar CONT en Pila de Saltos

9: Rellenar tope de Pila de Saltos con CONT

10: Crear Cuádruplo, guardar CONT en Pila de Saltos y rellenar tope de Pila de Saltos con CONT-1

11: Meter una marca a Pila Saltos y Meter a Pila de Operandos. Si se mete una variable de salida, error semántico.

12: Sacar de Pila de Operandos y llenar todos los GOTO pendientes con CONT

13: Mientras no se encuentre la marca en Pila de Saltos, rellenar todos los GOTOV con CONT+1, crear Cuádruplo y guardar CONT-1 en Pila de Saltos

14: Llenar el tope de Pila de Saltos con CONT+1, crear Cuádruplo y guardar CONT-1 en Pila de Saltos

15: Crear Cuádruplo de comparación y Cuádruplo de GOTOV, volver a guardar el ID en Pila de Operandos y guardar CONT-1 en Pila de Saltos

16: Crear Cuádruplo y agregar temporal a Pila de Operandos

17: Meter a Pila de Operandos. Si se mete una variable de salida, error semántico.

18: Meter a Pila de Operandos

### Consideraciones Semánticas:

La siguiente tabla es el cubo semántico, el cual describe las posibles combinaciones de variables de entrada, salida y señales.

**Operadores**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Operandos** | | **=** | **==** | **<<** | **>>** | **!=** | **<** | **>** | **&** | **|** | **!** | **^** | **+** | **-** |
| **in** | **in** | x | in | in | in | in | in | in | in | in | in | in | in | in |
| **in** | **out** | x | x | x | x | x | x | x | x | x | x | x | x | x |
| **in** | **signal** | x | signal | signal | signal | signal | signal | signal | signal | signal | signal | signal | signal | signal |
| **out** | **in** | out | x | x | x | x | x | x | x | x | x | x | x | x |
| **out** | **out** | out | x | x | x | x | x | x | x | x | x | x | x | x |
| **out** | **signal** | out | x | x | x | x | x | x | x | x | x | x | x | x |
| **signal** | **in** | signal | signal | signal | signal | signal | signal | signal | signal | signal | signal | signal | signal | signal |
| **signal** | **out** | x | x | x | x | x | x | x | x | x | x | x | x | x |
| **signal** | **signal** | signal | signal | signal | signal | signal | signal | signal | signal | signal | signal | signal | signal | signal |

## Administración de Memoria

La memoria de ésta aplicación se divide en cuatro partes: memoria de entrada, de salida, de señales y de temporales. La siguiente tabla muestra cómo está repartida la memoria:

|  |  |
| --- | --- |
| **Tipo de Variable** | **Rango de Memoria** |
| In | 1000-1999 |
| Out | 2000-2999 |
| Signal | 3000-3999 |
| Temporal | 6000-6999 |

# Descripción de la Máquina Virtual

## Equipo de Cómputo, Lenguaje y Utilerías

La máquina virtual implementada para el lenguaje Kuac está programada en su totalidad en lenguaje C apoyado de las 2 librerías base stdio.h y stdlib.h para la lectura y procesamiento básico de las instrucciones, adicionalmente se hace uso de la librería math.h para la realización de algunos cálculos de operaciones especiales necesarios para el correcto funcionamiento de la máquina virtual al momento de definir los límites máximos y mínimos de memoria.

## Administración de Memoria

La memoria se encuentra dividida en cuatro pilas principales, las cuales son Variables de Entrada, Variables de Salida, Variables de Señales y Variables Temporales cada una contando con un espacio de memoria de 1000 localidades las cuales son suficientes como para la construcción de un procesador pequeño, el cual es el mayor reto de programación de este lenguaje desarrollado.

|  |  |
| --- | --- |
| 0 | Constantes |
| 999 |
| 1000 | IN |
| 1999 |
| 2000 | OUTS |
| 2999 |
| 3000 | Signals |
| 3999 |
| 6000 | Temporales |
| 6999 |

Mapa de Memoria

La primera sección de la máquina virtual se encuentra destinada a las operaciones de memoria que involucra el uso de constantes, la segunda división al guardado de variables de entrada proporcionadas por el usuario al momento de estar cargando los valores con los cuales la maquina trabajara sobre el programa previamente compilado.

La tercera sección son las salidas las cuales son el resultado final del programa y estas son modificadas únicamente por el programa una vez compilado en su totalidad.

La cuarta sección son las variables de tipo Signal, estas son las encargadas de realizar todas las operaciones intermedias entre las variables de entrada y las de salida para haciendo de puente entre estas, estas variables solo pueden ser modificadas de dos maneras, declarándolas e inicializando como tal desde el momento de escribir el código a compilar o durante el proceso de ejecución que por alguna operación su valor cambie.

Finalmente en la última sección del mapa de memoria se localizan las variables temporales, las cuales son las encargadas de almacenar por breves periodos de tiempo los resultados de las operaciones aritméticas y booleanas, antes de que estas sean guardadas y asignadas ya sea a una variable de tipo Signal o una variable de tipo Out.

Durante el periodo de compilación los cuádruplos que se generan para ser interpretados por la máquina virtual definen un número específico de memoria que va en un rango del 0 al 7000, una vez que estos cuádruplos son leídos por máquina virtual se realiza una serie de condicionales anidados dependiendo el caso para poder diferenciar la pila correspondiente de cada uno, ya siendo la pila de variables de entrada, salida, señales o temporales y una vez que ha sido identificada se resta el offset necesario ya sea 1000 en el caso de las entradas, 2000 en salidas, 3000 en señales o 6000 en temporales y el número que quede una vez hecha la resta es el subíndice donde se buscara o almacenara la información de la pila correspondiente.

Para la realización de los saltos se implementó un contador general que iba recorriendo los cuádruplos uno por uno para verificar la instrucción que seguía fuera realmente la indicada y en el momento que se indicara un salto de línea ya sea hacia adelante o hacia tras esta se pudiese realizar sin problema con la facilidad de cambiar el índice correspondiente por el que pida el cuádruplo y restando uno al indicado por el mismo, por la misma estructura de la máquina que al finalizar la instrucción correspondiente esta siempre suma una unidad al índice de movimiento permitiendo siempre ir a la par con el correspondiente.

# Pruebas de Funcionamiento

## Pruebas Básicas

### Ejemplo 1:

entity P is(

A: in bit;

B: in bit;

C: out bit;

D: out vector (5);

E: signal vector (2);

) end P;

architecture of P is

begin

case A is

when 0:

C=A;

break;

when 1:

D=B;

break;

when 2:

C=B;

break;

when other:

E=1;

break;

end case;

end P;

**Cuádruplos:**

16 1000 -1 1

16 1001 -1 1

16 2000 -1 1

16 2001 -1 5

16 3000 -1 2

1 1000 0 6000

14 6000 -1 8

13 -1 -1 10

0 1000 -1 2000

13 -1 -1 24

1 1000 1 6001

14 6001 -1 13

13 -1 -1 15

0 1001 -1 2001

13 -1 - 1 24

1 1000 2 6002

14 6002 -1 18

13 -1 -1 20

0 1001 -1 2000

13 -1 -1 24

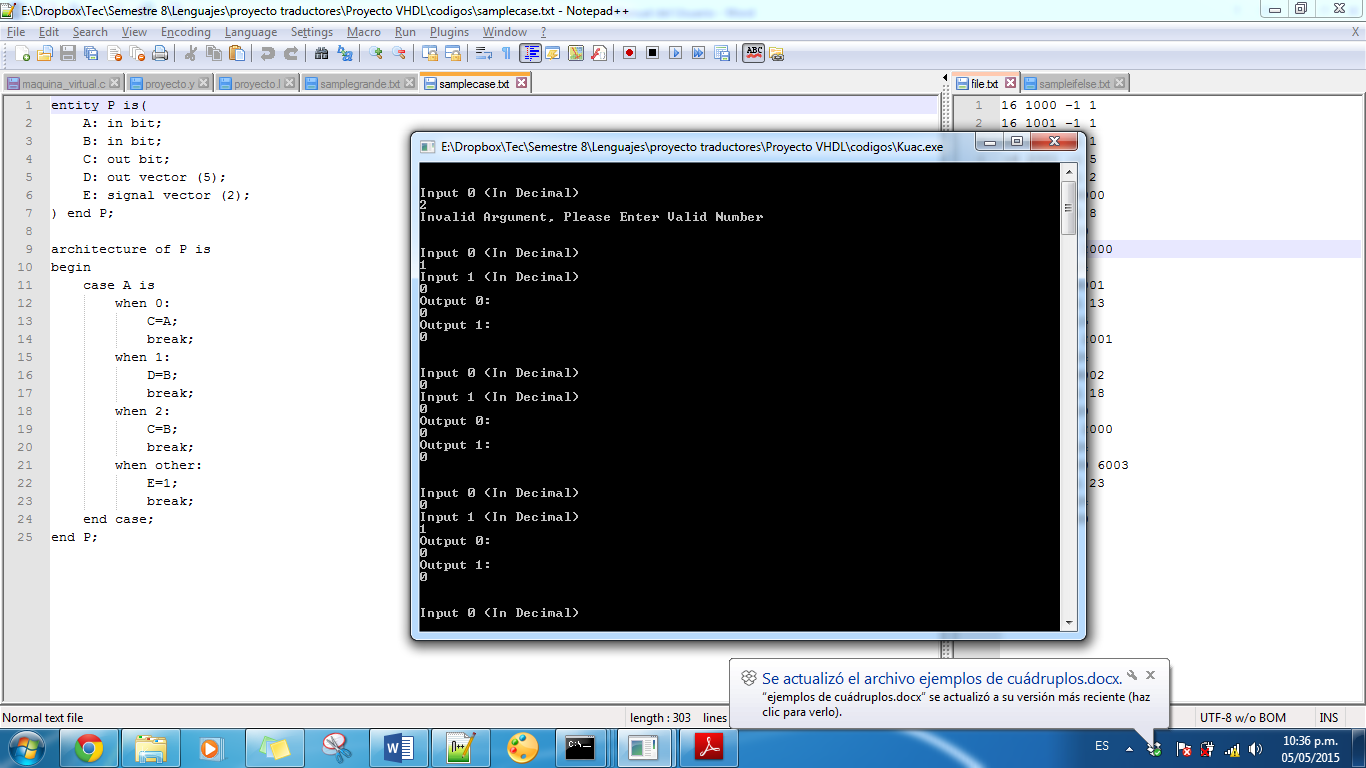
1 1000 1000 6003

14 6003 -1 23

13 -1 -1 24

0 1 -1 3000

**Ejecución:**

****

### Ejemplo 2:

entity P is(

A: in vector (3);

B: in bit;

C: out bit;

D: out vector (5);

E: signal bit;

) end P;

architecture of P is

begin

if(B>A) then

D=100;

C=1;

end if;

end P;

**Cuádruplos:**

16 1000 -1 3

16 1001 -1 1

16 2000 -1 1

16 2001 -1 5

16 3000 -1 1

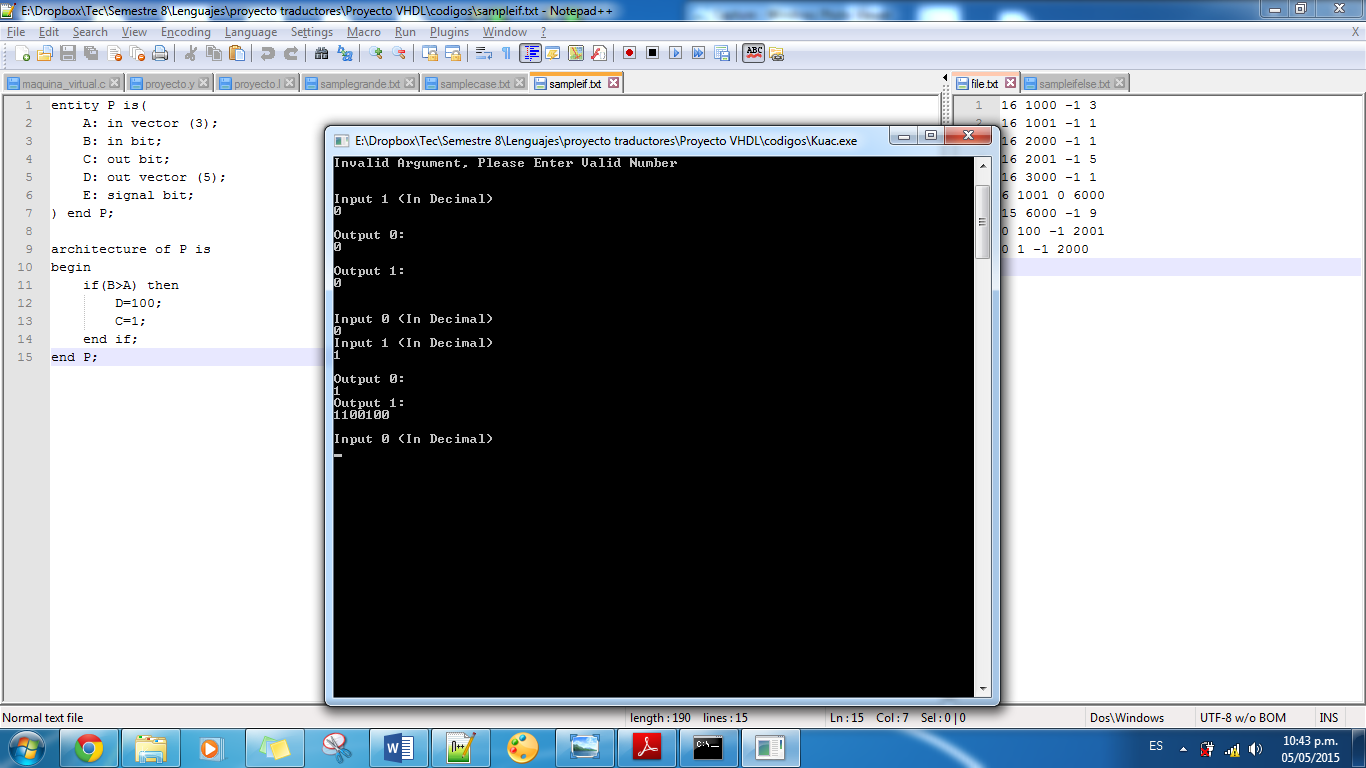
6 1001 0 6000

15 6000 -1 9

0 100 -1 2001

0 1 -1 2000

**Ejecución:**



### Ejemplo 3:

entity P is(

A: in vector (3);

B: in bit;

C: out bit;

D: out vector (5);

E: signal bit;

) end P;

architecture of P is

begin

if(A<B) then

D=A+B;

else

D=A SHL B;

end if;

end P;

**Cuádruplos:**

16 1000 -1 3

16 1001 -1 1

16 2000 -1 1

16 2001 -1 5

16 3000 -1 1

5 1001 1000 6000

15 6000 -1 10

11 1001 1000 6001

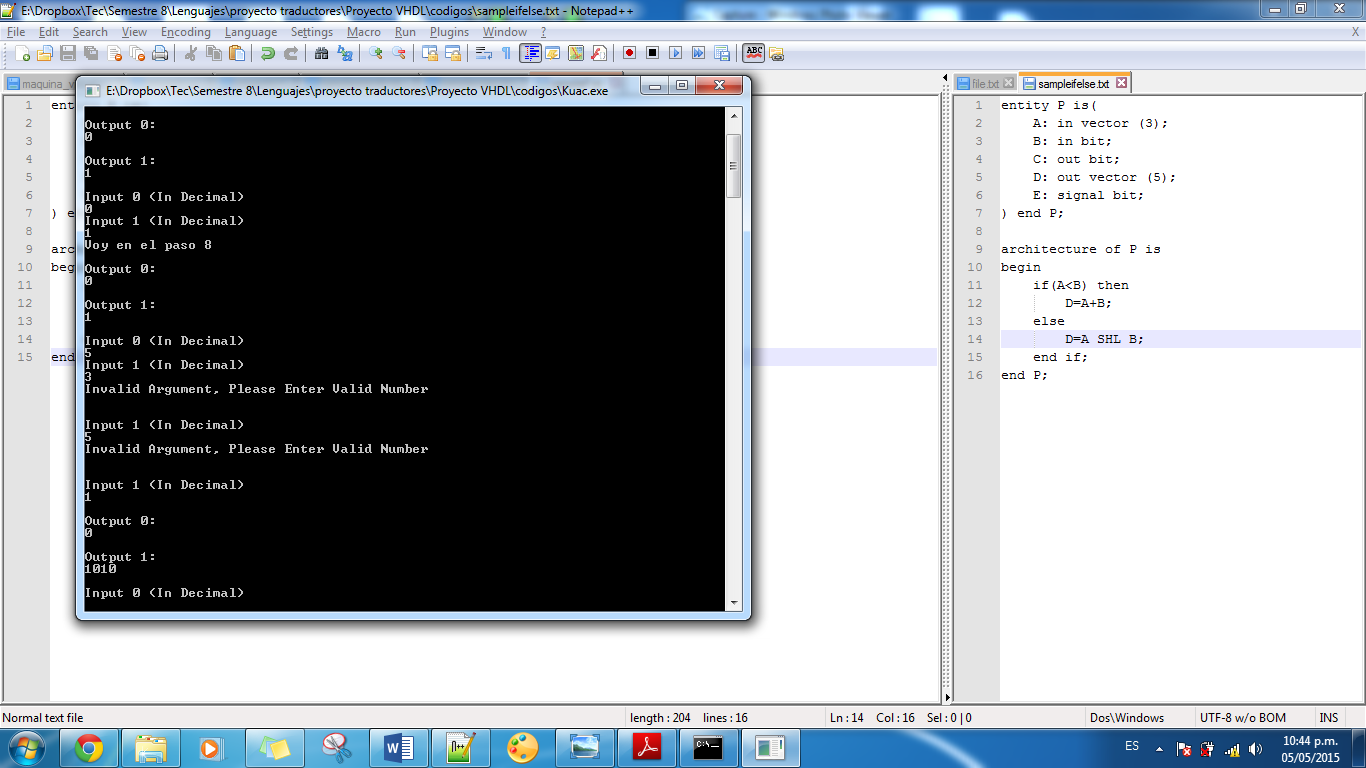
0 6001 -1 2001

13 -1 -1 12

2 1000 1001 6002

0 6002 -1 2001

**Ejecución:**



### Ejemplo 4:

entity P is(

A: in vector (3);

B: in bit;

C: out bit;

D: out vector (5);

E: signal bit;

F: signal vector (5);

) end P;

architecture of P is

begin

E=B;

F=A+B;

D=A|E^B&!F;

end P;

**Cuádruplos:**

16 1000 -1 3

16 1001 -1 1

16 2000 -1 1

16 2001 -1 5

16 3000 -1 1

16 3001 -1 5

0 1001 -1 3000

11 1001 1000 6000

0 6000 -1 3001

9 3001 -1 6001

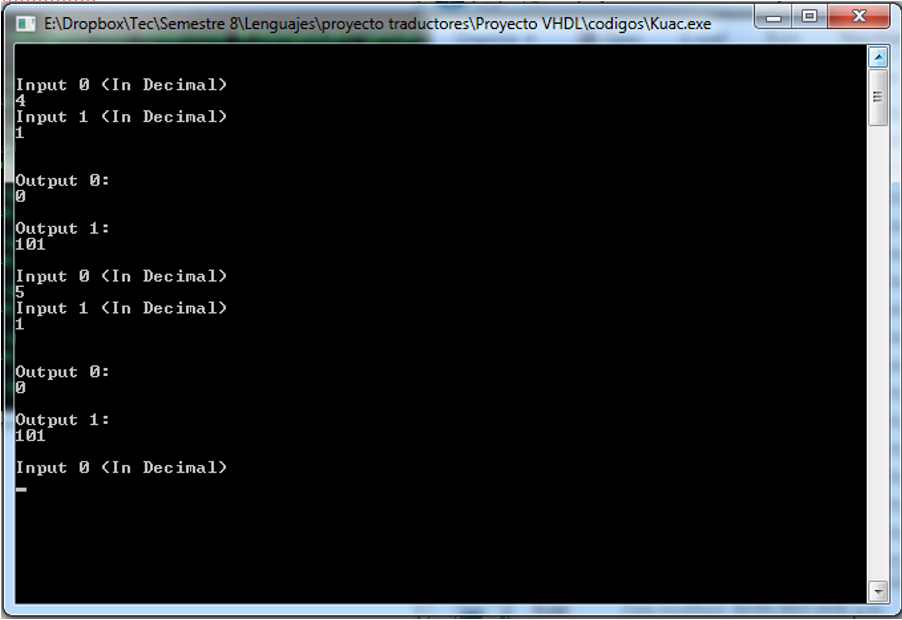
7 6001 1001 6002

10 6002 3000 6003

8 6003 1000 6004

0 6004 -1 2001

**Ejecución:**



### Ejemplo 5:

entity P is(

A: in vector (2);

B: in bit;

C: in bit;

D: out vector (3);

E: signal vector (2);

) end P;

architecture of P is

begin

E=A|B;

case E is

when 0:

if(C==0) then

D=000;

else

D=101;

end if;

break;

when 1:

D=A+B&C;

break;

when 2:

if(C==1) then

D=010;

end if;

break;

when 3:

D=(A-B) SHL C;

break;

when other:

D=111;

break;

end case;

end P;

**Cuádruplos:**

16 1000 -1 2

16 1001 -1 1

16 1002 -1 1

16 2000 -1 3

16 3000 -1 2

8 1001 1000 6000

0 6000 -1 3000

1 3000 0 6001

14 6001 -1 10

13 -1 -1 16

1 0 1002 6002

15 6002 -1 14

0 0 -1 2000

13 -1 -1 15

0 101 -1 2000

13 -1 -1 41

1 3000 1 6003

14 6003 -1 19

13 -1 -1 23

11 1001 1000 6004

7 1002 6004 6005

0 6005 -1 2000

13 -1 -1 41

1 3000 2 6006

14 6006 -1 26

13 -1 -1 30

1 1 1002 6007

15 6007 -1 29

0 10 -1 2000

13 -1 -1 41

1 3000 3 6008

14 6008 -1 33

13 -1 -1 37

12 1001 1000 6009

2 6009 1002 6010

0 6010 -1 2000

13 -1 -1 41

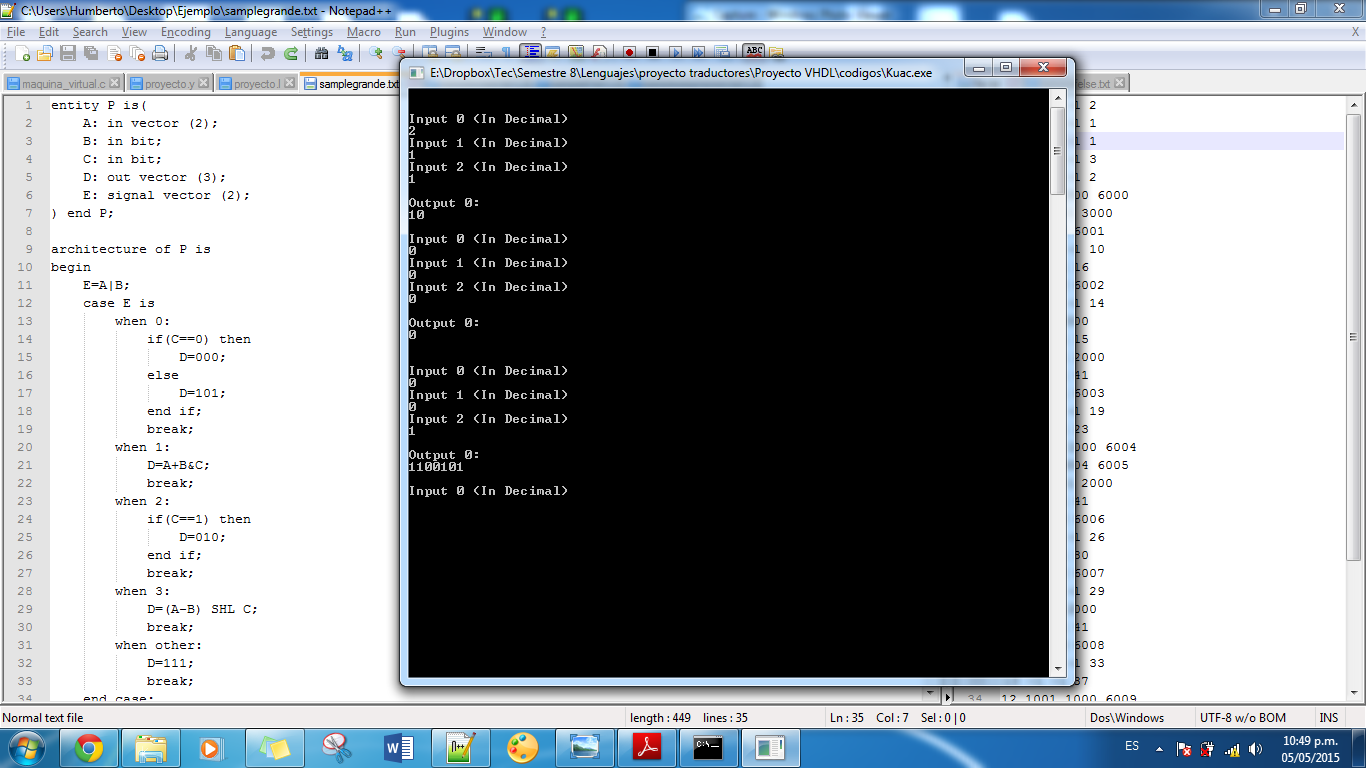
1 3000 3000 6011

14 6011 -1 40

13 -1 -1 41

0 111 -1 2000

**Ejecución:**



## Pruebas Reales

### Ejemplo 6-Unidad de Control

entity U is(

I : in vector (6);

D : out bit;

B : out bit;

J : out bit;

R : out bit;

M : out bit;

O : out vector (2);

W : out bit;

S : out bit;

E : out bit;

) end U;

architecture of U is

begin

case I is

when 0:

D = 1;

B = 0;

J = 0;

R = 0;

M = 0;

O = 2;

W = 0;

S = 0;

E = 1;

break;

when 35:

D = 0;

B = 0;

J = 0;

R = 1;

M = 1;

O = 0;

W = 0;

S = 1;

E = 1;

break;

when 43:

D = 0;

B = 0;

J = 0;

R = 0;

M = 0;

O = 0;

W = 1;

S = 1;

E = 0;

break;

when 4:

D = 0;

B = 1;

J = 0;

R = 0;

M = 0;

O = 1;

W = 0;

S = 0;

E = 0;

break;

when 2:

D = 0;

B = 0;

J = 1;

R = 0;

M = 0;

O = 00;

W = 0;

S = 0;

E = 0;

break;

when other:

D = 0;

B = 0;

J = 0;

R = 0;

M = 0;

O = 0;

W = 0;

S = 1;

E = 1;

break;

end case;

end U;

**Cuádruplos:**

16 1000 -1 6

16 2000 -1 1

16 2001 -1 1

16 2002 -1 1

16 2003 -1 1

16 2004 -1 1

16 2005 -1 2

16 2006 -1 1

16 2007 -1 1

16 2008 -1 1

1 1000 0 6000

14 6000 -1 13

13 -1 -1 23

0 1 -1 2000

0 0 -1 2001

0 0 -1 2002

0 0 -1 2003

0 0 -1 2004

0 2 -1 2005

0 0 -1 2006

0 0 -1 2007

0 1 -1 2008

13 -1 -1 87

1 1000 35 6001

14 6001 -1 26

13 -1 -1 36

0 0 -1 2000

0 0 -1 2001

0 0 -1 2002

0 1 -1 2003

0 1 -1 2004

0 0 -1 2005

0 0 -1 2006

0 1 -1 2007

0 1 -1 2008

13 -1 -1 87

1 1000 43 6002

14 6002 -1 39

13 -1 -1 49

0 0 -1 2000

0 0 -1 2001

0 0 -1 2002

0 0 -1 2003

0 0 -1 2004

0 0 -1 2005

0 1 -1 2006

0 1 -1 2007

0 0 -1 2008

13 -1 -1 87

1 1000 4 6003

14 6003 -1 52

13 -1 -1 62

0 0 -1 2000

0 1 -1 2001

0 0 -1 2002

0 0 -1 2003

0 0 -1 2004

0 1 -1 2005

0 0 -1 2006

0 0 -1 2007

0 0 -1 2008

13 -1 -1 87

1 1000 2 6004

14 6004 -1 65

13 -1 -1 75

0 0 -1 2000

0 0 -1 2001

0 1 -1 2002

0 0 -1 2003

0 0 -1 2004

0 0 -1 2005

0 0 -1 2006

0 0 -1 2007

0 0 -1 2008

13 -1 -1 87

1 1000 1000 6005

14 6005 -1 78

13 -1 -1 87

0 0 -1 2000

0 0 -1 2001

0 0 -1 2002

0 0 -1 2003

0 0 -1 2004

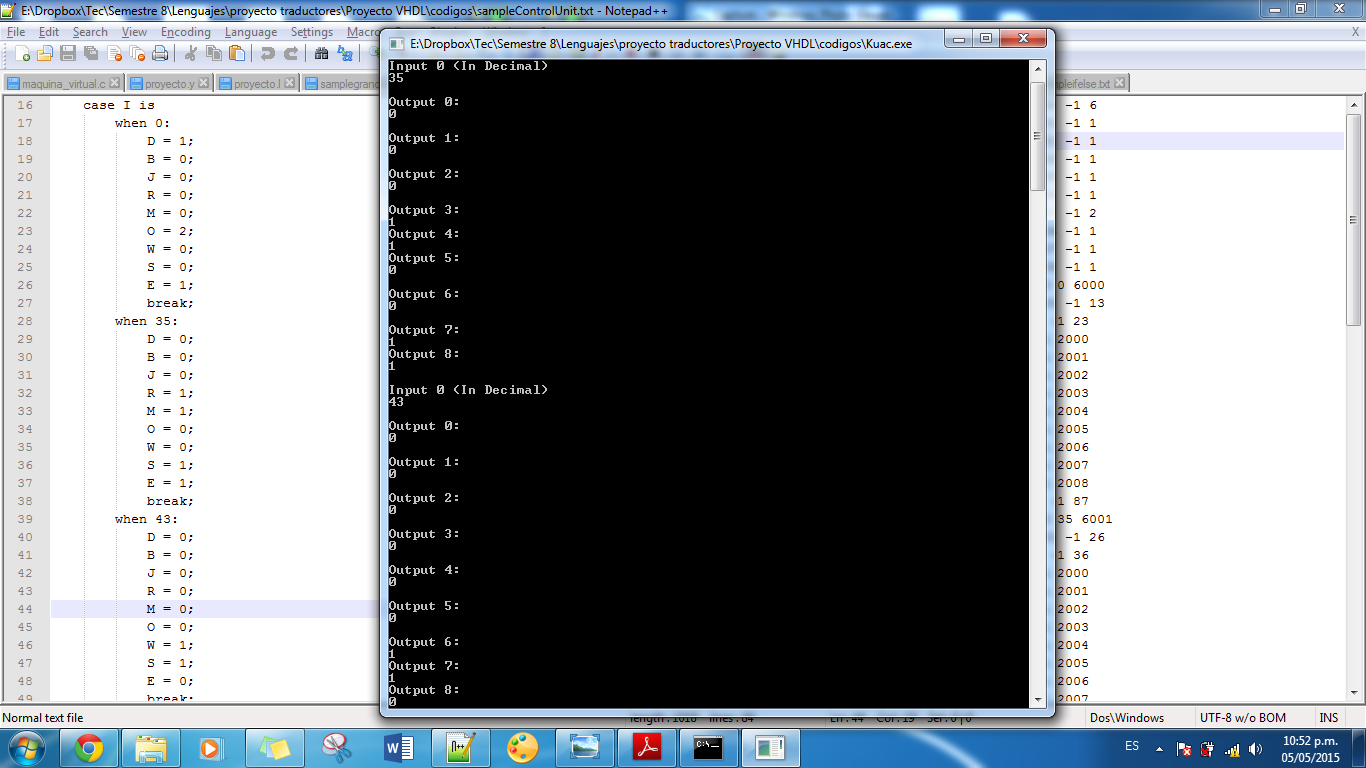
0 0 -1 2005

0 0 -1 2006

0 1 -1 2007

0 1 -1 2008

**Ejecución:**

****

### Ejemplo 7-Unidad Aritmética-Lógica

entity L is(

A : in vector (8);

B : in vector (8);

C : in vector (3);

Z : out bit;

O : out bit;

F : out vector (8);

I : signal vector (8);

S : signal vector (8);

)end L;

architecture of L is

begin

case C is

when 0:

I=A&B;

break;

when 1:

I=A|B;

break;

when 2:

S=A+B;

if(S>510) then

O=1;

end if;

break;

when 3:

I=A;

break;

when 6:

I=A-B;

break;

when 7:

if (A<B) then

I=1;

else

I=0;

end if;

break;

when other:

I=0;

break;

end case;

if (I==0) then

Z=1;

else

Z=0;

end if;

F=I;

end L;

**Cuádruplos:**

16 1000 -1 8

16 1001 -1 8

16 1002 -1 3

16 2000 -1 1

16 2001 -1 1

16 2002 -1 8

16 3000 -1 8

16 3001 -1 8

1 1002 0 6000

14 6000 -1 11

13 -1 -1 14

7 1001 1000 6001

0 6001 -1 3000

13 -1 -1 53

1 1002 1 6002

14 6002 -1 17

13 -1 -1 20

8 1001 1000 6003

0 6003 -1 3000

13 -1 -1 53

1 1002 2 6004

14 6004 -1 23

13 -1 -1 29

11 1001 1000 6005

0 6005 -1 3001

6 3001 0 6006

15 6006 -1 28

0 1 -1 2001

13 -1 -1 53

1 1002 3 6007

14 6007 -1 32

13 -1 -1 34

0 1000 -1 3000

13 -1 -1 53

1 1002 6 6008

14 6008 -1 37

13 -1 -1 40

12 1001 1000 6009

0 6009 -1 3000

13 -1 -1 53

1 1002 7 6010

14 6010 -1 43

13 -1 -1 49

5 1001 1000 6011

15 6011 -1 47

0 1 -1 3000

13 -1 -1 48

0 0 -1 3000

13 -1 -1 53

1 1002 1002 6012

14 6012 -1 52

13 -1 -1 53

0 0 -1 3000

1 0 3000 6013

15 6013 -1 57

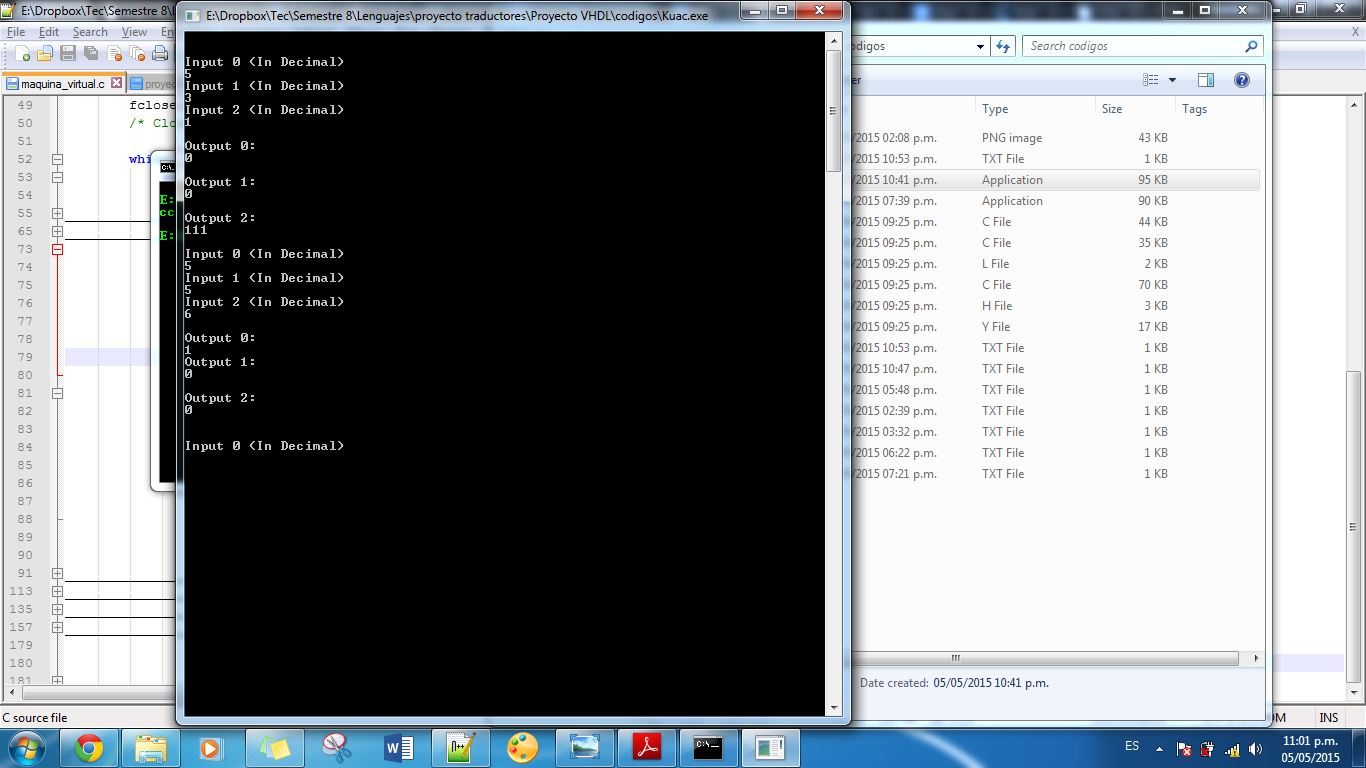
0 1 -1 2000

13 -1 -1 58

0 0 -1 2000

0 3000 -1 2002

**Ejecución:**



# Listados

Los cuádruplos generables con el lenguaje de programación Kuac se describen a continuación:

**Cuádruplo 0 – Asignación**

Token: =

Estructura:

0 Operando1 -1 Resultado

El operando puede ser una variable de entrada o una señal, y el resultado es una variable de salida. El cuádruplo da el valor del operando 1 al resultado.

**Cuádruplo 1 – Comparación**

Token: ==

Estructura:

1 Operando1 Operando2 Resultado

Los operandos pueden ser variables de entrada o señales, y el resultado es una variable temporal. El cuádruplo regresa uno si los operandos son iguales, de lo contrario, regresa cero.

**Cuádruplo 2 – Shift Left**

Token: SHL

Estructura:

2 Operando1 Operando2 Resultado

Los operandos pueden ser variables de entrada o señales, y el resultado es una variable temporal. El cuádruplo regresa el primer operando recorrido hacia la izquierda por el segundo operando.

**Cuádruplo 3 – Shift Rigth**

Token: SHR

Estructura:

3 Operando1 Operando2 Resultado

Los operandos pueden ser variables de entrada o señales, y el resultado es una variable temporal. El cuádruplo regresa el primer operando recorrido hacia la derecha por el segundo operando.

**Cuádruplo 4 – Diferencia**

Token: !=

Estructura:

4 Operando1 Operando2 Resultado

Los operandos pueden ser variables de entrada o señales, y el resultado es una variable temporal. El cuádruplo regresa uno si los operandos son diferentes, de lo contrario, regresa cero.

**Cuádruplo 5 – Menor que**

Token: <

Estructura:

5 Operando1 Operando2 Resultado

Los operandos pueden ser variables de entrada o señales, y el resultado es una variable temporal. El cuádruplo regresa uno si el primer operando es menor que el segundo, de lo contrario, regresa cero.

**Cuádruplo 6 – Mayor que**

Token: >

Estructura:

6 Operando1 Operando2 Resultado

Los operandos pueden ser variables de entrada o señales, y el resultado es una variable temporal. El cuádruplo regresa uno si el primer operando es mayor que el segundo, de lo contrario, regresa cero.

**Cuádruplo 7 – And**

Token: &

Estructura:

7 Operando1 Operando2 Resultado

Los operandos pueden ser variables de entrada o señales, y el resultado es una variable temporal. El cuádruplo regresa la operación lógica AND.

**Cuádruplo 8 – Or**

Token: |

Estructura:

8 Operando1 Operando2 Resultado

Los operandos pueden ser variables de entrada o señales, y el resultado es una variable temporal. El cuádruplo regresa la operación lógica OR.

**Cuádruplo 9 – Not**

Token: !

Estructura:

9 Operando1 -1 Resultado

El operando puede ser una variable de entrada o una señal, y el resultado es una variable temporal. El cuádruplo regresa la operación lógica NOT.

**Cuádruplo 10 – Xor**

Token: ^

Estructura:

10 Operando1 Operando2 Resultado

Los operandos pueden ser variables de entrada o señales, y el resultado es una variable temporal. El cuádruplo regresa la operación lógica OR Exclusivo.

**Cuádruplo 11– Suma**

Token: +

Estructura:

11 Operando1 Operando2 Resultado

Los operandos pueden ser variables de entrada o señales, y el resultado es una variable temporal. El cuádruplo regresa la operación aritmética suma.

**Cuádruplo 12 – Resta**

Token: -

Estructura:

12 Operando1 Operando2 Resultado

Los operandos pueden ser variables de entrada o señales, y el resultado es una variable temporal. El cuádruplo regresa la operación aritmética resta.

**Cuádruplo 13 – Salto**

Estructura:

13 -1 -1 Dirección

El operando dirección es la instrucción a la cual se va a brincar.

**Cuádruplo 14 – Salto en verdadero**

Estructura:

14 Operando1 -1 Dirección

El operando dirección es la instrucción a la cual se va a brincar. Se brinca solo si el operando es igual a uno.

**Cuádruplo 15 – Salto en falso**

Estructura:

15 Operando1 -1 Dirección

El operando dirección es la instrucción a la cual se va a brincar. Se brinca solo si el operando es igual a cero.

**Cuádruplo 16 – Declaración de variable**

Estructura:

16 Dirección -1 Dimensión

El operando dirección es la dirección de memoria de la variable a declarar y la variable dimensión indica el tamaño del dato.

# Anexos

## Código Flex

%{

#include "proyecto.tab.h"

%}

%%

"entity" return entity;

"is" return is;

"in" return in;

"out" return out;

"signal" return signal;

"bit" return bit;

"vector" return vector;

"to" return to;

"end" return end;

"architecture" return architecture;

"of" return of;

"begin" return begin;

"if" return IF;

"then" return THEN;

"else" return ELSE;

"case" return CASE;

"when" return WHEN;

"break" return BREAK;

"other" return OTHER;

"(" return yytext[0];

")" return yytext[0];

":" return yytext[0];

";" return yytext[0];

"==" return yytext[0];

"SHL" return SHL;

"SHR" return SHR;

"!=" return yytext[0];

"=" return yytext[0];

"<" return yytext[0];

">" return yytext[0];

"&" return yytext[0];

"|" return yytext[0];

"!" return yytext[0];

"^" return yytext[0];

"+" return yytext[0];

"-" return yytext[0];

[a-zA-Z][\_a-zA-Z0-9]\* {yylval.id = yytext[0]; return ID;}

[0-9]+ {yylval.integer = atoi(yytext); return CTE\_INT;}

[ \t\n] ;

. {ECHO; yyerror("ERROR\n");}

%%

int yywrap(void)

{

//[0-1]+ {yylval.integer = atoi(yytext); return BYTE;}

return 1;

}

## Código Bison

%{

void yyerror (char \*s);

#include <stdio.h>

#include <stdlib.h>

int pilaOperandos [100];

int pilaOperadores [100];

int pilaFondoFalso [100];

int pilaSaltos [100];

int operadores=0,operandos=0,fondo\_falso=0, saltos=0;

int temp=6000;

int mem\_in=1000;

int mem\_out=2000;

int mem\_signal=3000;

int mem\_const=4000;

int cuad1[100], cuad2[100], cuad3[100], cuad4[100];

int cuad\_cont=0;

int cont;

int cont\_tranza;

int flag=1;

//-----------cubos semanticos----------------------------------------------------------//

int semantica2 [5][15]= {{-1,-1,0,1,2,3,4,5,6,7,8,9,10,11,12},

{100,100,100,100,100,100,100,100,100,100,100,-1,100,101,100},

{100,101,-1,100,101,101,100,100,100,101,101,-1,101,101,101},

{101,100,101,100,101,101,100,100,100,101,101,-1,101,101,101},

{101,101,101,100,101,101,100,100,100,101,101,-1,101,101,101}};

int semantica1 [10][15]= {{-1,-1,0,1,2,33,4,5,6,7,8,9,10,11,12},

{200,200,-1,200,200,200,200,200,200,200,200,200,200,200,200},

{200,201,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},

{200,202,-1,202,202,202,202,202,202,202,202,202,202,202,202},

{201,200,201,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},

{201,201,201,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},

{201,202,201,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},

{202,200,202,202,202,202,202,202,202,202,202,202,202,202,202},

{202,201,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},

{202,202,202,202,202,202,202,202,202,202,202,202,202,202,202}};

//----------Fin de cubos semanticos----------------------------------------------------//

//-------------------Tabla de variables------------------------------------------------//

char Table\_var[100][3];

int Table\_var\_mem[100];

int contab=0;

//----------------Fin de tabla de variables--------------------------------------------//

%}

%union {int integer; char id; char tipos;}

%start PROGRAM

%token entity

%token is

%token to

%token end

%token architecture

%token of

%token begin

%token IF

%token THEN

%token ELSE

%token CASE

%token WHEN

%token BREAK

%token OTHER

%token SHL

%token SHR

%token <integer> CTE\_INT

%token <id> ID

%token <tipos> in out signal bit vector

%type <integer> VARS VAR1 VAR2 VAR EXPRESION EXPRESION1 EXP LOGIC LOGIC1 TERMINO TERMINO1 FACTOR FACTOR1 VAR\_CTE

%type <id> BLOQUE INOUT TIPO ESTATUTO1 ESTATUTO2 ESTATUTO ASIGNACION CONDIF CONDCASE CONDCASE1

%%

PROGRAM :VARS BLOQUE {;}

;

VARS :entity ID is '(' VAR1 ')' end ID ';' {if($2!=$8)

printf("error semantico\n");

for(cont=0; cont<contab; cont++){

cuad1[cuad\_cont]=16;

cuad2[cuad\_cont]=Table\_var\_mem[cont];

cuad3[cuad\_cont]=-1;

cuad4[cuad\_cont]=Table\_var[cont][2];

cuad\_cont++;

}

}

;

VAR1 :VAR VAR2 {;}

;

VAR2: {;}

|VAR1 {;}

;

VAR :ID ':' {Table\_var[contab][0]=$1;}

INOUT TIPO ';' {;}

;

INOUT :in {Table\_var[contab][1]='i';Table\_var\_mem[contab]=mem\_in++;}

|out {Table\_var[contab][1]='o';Table\_var\_mem[contab]=mem\_out++;}

|signal {Table\_var[contab][1]='s';Table\_var\_mem[contab]=mem\_signal++;}

;

TIPO :vector '(' CTE\_INT ')' {Table\_var[contab][2]=$3; contab=contab+1;}

|bit {Table\_var[contab][2]=1; contab=contab+1;}

;

BLOQUE :architecture of ID is begin ESTATUTO1 end ID ';' {if($3!=$8)

printf("error semantico\n");

FILE \*f = fopen("file.txt", "w");

int n=cuad\_cont;

int i;

for(i=0; i<n; i++)

fprintf(f, "%d %d %d %d\n", cuad1[i], cuad2[i], cuad3[i], cuad4[i]);

fclose(f);}

;

ESTATUTO1 :ESTATUTO ESTATUTO2 {;}

;

ESTATUTO2 : {;}

|ESTATUTO1 {;}

;

ESTATUTO :ASIGNACION {;}

|CONDIF {;}

|CONDCASE {;}

;

ASIGNACION :ID {for(cont=0; cont<=contab; cont++) if(Table\_var[cont][0]==$1) if(Table\_var\_mem[cont]<2000){

printf("error semantico\n asignacion a valor de entrada\n");

} else{

operandos++;

pilaOperandos[operandos]=Table\_var\_mem[cont]; }

}

'=' {operadores++;pilaOperadores[operadores]=0;}

EXP {cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad3[cuad\_cont]=-1;

cuad4[cuad\_cont]=pilaOperandos[operandos];operandos--;

//operandos++; pilaOperandos[operandos]=cuad4[cuad\_cont];

cuad\_cont++;

}

';'

;

CONDIF :IF '(' {fondo\_falso++; pilaFondoFalso[fondo\_falso]=pilaOperadores[operadores];} EXPRESION ')' {fondo\_falso--;} THEN {operadores++;pilaOperadores[operadores]=15;

cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad3[cuad\_cont]=-1;

cuad4[cuad\_cont]=-1;

cuad\_cont++;

saltos++;pilaSaltos[saltos]=cuad\_cont-1;}

ESTATUTO1 CONDIF1 {;}

;

CONDIF1 :end IF {cuad4[pilaSaltos[saltos]]=cuad\_cont; saltos--;}

';' {;}

|ELSE {operadores++;pilaOperadores[operadores]=13;

cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad2[cuad\_cont]=-1;

cuad3[cuad\_cont]=-1;

cuad4[cuad\_cont]=-1;

cuad\_cont++;

cuad4[pilaSaltos[saltos]]=cuad\_cont; saltos--;

saltos++;pilaSaltos[saltos]=cuad\_cont-1;}

ESTATUTO1 end IF {cuad4[pilaSaltos[saltos]]=cuad\_cont; saltos--;}

';' {;}

;

CONDCASE :CASE ID {saltos++;pilaSaltos[saltos]=-5;

for(cont=0; cont<=contab; cont++)

if(Table\_var[cont][0]==$2)

if(Table\_var\_mem[cont]>1999&&Table\_var\_mem[cont]<3000){

printf("error semantico\n mal uso de variable de salida\n"); }

else{

operandos++;

pilaOperandos[operandos]=Table\_var\_mem[cont];

}

}

is CONDCASE1 end CASE {operandos--;

cont\_tranza=cuad\_cont;

while(cont\_tranza>1){

if(cuad1[cont\_tranza]==13&&cuad4[cont\_tranza]==-1){

cuad4[cont\_tranza]=cuad\_cont;

}

cont\_tranza--;

}

//cuad4[pilaSaltos[saltos]]=cuad\_cont;

//saltos--;

}

';' {;}

;

CONDCASE1 :WHEN OPCASE ':' {while(pilaSaltos[saltos]!=-5){

if(cuad1[pilaSaltos[saltos]]==14){

cuad4[pilaSaltos[saltos]]=cuad\_cont+1;

}

saltos--;

}

cuad1[cuad\_cont]=13;

cuad2[cuad\_cont]=-1;

cuad3[cuad\_cont]=-1;

cuad4[cuad\_cont]=-1;

cuad\_cont++;

saltos++;pilaSaltos[saltos]=cuad\_cont-1;

}

ESTATUTO1 BREAK ';' {if(flag==1){

cuad4[pilaSaltos[saltos]]=cuad\_cont+1;

saltos--;

cuad1[cuad\_cont]=13;

cuad2[cuad\_cont]=-1;

cuad3[cuad\_cont]=-1;

cuad4[cuad\_cont]=-1;

cuad\_cont++;

saltos++;pilaSaltos[saltos]=cuad\_cont-1;

}

flag=1;

}

CONDCASE1 {;}

| {;}

;

OPCASE :CTE\_INT {cuad1[cuad\_cont]=1;

cuad2[cuad\_cont]=pilaOperandos[operandos];

cuad3[cuad\_cont]=$1;

cuad4[cuad\_cont]=temp++;

cuad\_cont++;

cuad1[cuad\_cont]=14;

cuad2[cuad\_cont]=cuad4[cuad\_cont-1];

cuad3[cuad\_cont]=-1;

cuad4[cuad\_cont]=-1;

cuad\_cont++;

//operandos--;

operandos++;pilaOperandos[operandos]=cuad2[cuad\_cont-2];

saltos++;pilaSaltos[saltos]=cuad\_cont-1;

}

|OTHER {cuad1[cuad\_cont]=1;

cuad2[cuad\_cont]=pilaOperandos[operandos];

cuad3[cuad\_cont]=pilaOperandos[operandos];

cuad4[cuad\_cont]=temp++;

cuad\_cont++;

cuad1[cuad\_cont]=14;

cuad2[cuad\_cont]=cuad4[cuad\_cont-1];

cuad3[cuad\_cont]=-1;

cuad4[cuad\_cont]=-1;

cuad\_cont++;

//operandos--;

operandos++;pilaOperandos[operandos]=cuad2[cuad\_cont-2];

saltos++;pilaSaltos[saltos]=cuad\_cont-1;

flag=0;}

;

EXPRESION :EXP EXPRESION1 {;}

;

EXPRESION1 : {;}

|'==' {operadores++;pilaOperadores[operadores]=1;}

EXP {cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad3[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad4[cuad\_cont]=temp++;

operandos++; pilaOperandos[operandos]=cuad4[cuad\_cont];

cuad\_cont++;

}

|'!=' {operadores++; pilaOperadores[operadores]=4;}

EXP {cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad3[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad4[cuad\_cont]=temp++;

operandos++; pilaOperandos[operandos]=cuad4[cuad\_cont];

cuad\_cont++;

}

|'<' {operadores++; pilaOperadores[operadores]=5;}

EXP {cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad3[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad4[cuad\_cont]=temp++;

operandos++; pilaOperandos[operandos]=cuad4[cuad\_cont];

cuad\_cont++;

}

|'>' {operadores++; pilaOperadores[operadores]=6;}

EXP {cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad4[cuad\_cont]=temp++;

operandos++; pilaOperandos[operandos]=cuad4[cuad\_cont];

cuad\_cont++;

}

;

EXP :LOGIC LOGIC1 {;}

|'!' {operadores++; pilaOperadores[operadores]=9;}

EXP {if(pilaOperadores[operadores]==7|pilaOperadores[operadores]==8|pilaOperadores[operadores]==9|pilaOperadores[operadores]==10){ cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad3[cuad\_cont]=-1;

cuad4[cuad\_cont]=temp++;

operandos++; pilaOperandos[operandos]=cuad4[cuad\_cont];

cuad\_cont++;

}}

;

LOGIC1 {;}

|'&' {operadores++; pilaOperadores[operadores]=7;}

EXP {if(pilaOperadores[operadores]==7|pilaOperadores[operadores]==8|pilaOperadores[operadores]==9|pilaOperadores[operadores]==10){

cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad3[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad4[cuad\_cont]=temp++;

operandos++; pilaOperandos[operandos]=cuad4[cuad\_cont];

cuad\_cont++;

}}

|'|' {operadores++; pilaOperadores[operadores]=8;}

EXP {if(pilaOperadores[operadores]==7|pilaOperadores[operadores]==8|pilaOperadores[operadores]==9|pilaOperadores[operadores]==10){ cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad3[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad4[cuad\_cont]=temp++;

operandos++; pilaOperandos[operandos]=cuad4[cuad\_cont];

cuad\_cont++;

}}

|'^' {operadores++; pilaOperadores[operadores]=10;}

EXP {if(pilaOperadores[operadores]==7|pilaOperadores[operadores]==8|pilaOperadores[operadores]==9|pilaOperadores[operadores]==10){ cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad3[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad4[cuad\_cont]=temp++;

operandos++; pilaOperandos[operandos]=cuad4[cuad\_cont];

cuad\_cont++;

}}

;

LOGIC :TERMINO TERMINO1 {;}

;

TERMINO1 : {;}

|'+' {operadores++; pilaOperadores[operadores]=11;}

LOGIC {if(pilaOperadores[operadores]==11|pilaOperadores[operadores]==12){

cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad3[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad4[cuad\_cont]=temp++;

operandos++; pilaOperandos[operandos]=cuad4[cuad\_cont];

cuad\_cont++;

}}

|'-' {operadores++; pilaOperadores[operadores]=12;}

LOGIC {if(pilaOperadores[operadores]==11|pilaOperadores[operadores]==12){

cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad3[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad4[cuad\_cont]=temp++;

operandos++; pilaOperandos[operandos]=cuad4[cuad\_cont];

cuad\_cont++;

}}

;

TERMINO: FACTOR FACTOR1 {;}

;

FACTOR1: {;}

|SHL {operadores++; pilaOperadores[operadores]=2;}

TERMINO {if(pilaOperadores[operadores]==3|pilaOperadores[operadores]==2){

cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad3[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad4[cuad\_cont]=temp++;

operandos++; pilaOperandos[operandos]=cuad4[cuad\_cont];

cuad\_cont++;

}}

|SHR {operadores++; pilaOperadores[operadores]=3;}

TERMINO {if(pilaOperadores[operadores]==3|pilaOperadores[operadores]==2){

cuad1[cuad\_cont]=pilaOperadores[operadores];operadores--;

cuad3[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad2[cuad\_cont]=pilaOperandos[operandos];operandos--;

cuad4[cuad\_cont]=temp++;

operandos++; pilaOperandos[operandos]=cuad4[cuad\_cont];

cuad\_cont++;

}}

;

FACTOR: '(' {fondo\_falso++; pilaFondoFalso[fondo\_falso]=pilaOperadores[operadores];}

EXP ')' {fondo\_falso--;}

|VAR\_CTE {;}

;

VAR\_CTE :ID {for(cont=0; cont<=contab; cont++)

if(Table\_var[cont][0]==$1)

if(Table\_var\_mem[cont]>1999&&Table\_var\_mem[cont]<3000){

printf("error semantico\n mal uso de variable de salida\n");

}

else{

operandos++;

pilaOperandos[operandos]=Table\_var\_mem[cont];

}

}

|CTE\_INT {operandos++; pilaOperandos[operandos]=$1;}

;

%%

int main(void){

return yyparse();

}

void yyerror (char \*s) {printf("%s\n", s);}

## Código Máquina Virtual

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

void binario(int num){

int aux;

if(num==0)

return;

aux=num%2;

num=num/2;

binario(num);

printf("%d",aux);

}

void Seguidor(int a){

printf("Voy en el paso %i \n", a+1);

}

int i=0,j=0,m=0,o=0,s=0,a=1;

int dim=0;

int flag=1;

unsigned int cuad1[100], cuad2[100], cuad3[100], cuad4[100];

char alfabeto[27] = {'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};

int main(void)

{

printf("\n Simulador de Circuitos Digitales\n");

printf(" Kuac v2.0.1\n\nRecuerda que para el uso de este software se debio\n compilar previamente el código del circuito\n");

printf(" haciendo uso de Kuac\_Compiler.exe\n\n");

/\*for(i=0;i<27;i++)

printf("%c",alfabeto[i]);\*/

while(flag){

unsigned int mem\_in [100] = {0};

unsigned int mem\_in\_dim[100] = {0};

unsigned int mem\_out [100] = {0};

unsigned int mem\_out\_dim[100] = {0};

unsigned int mem\_sig[100] = {0};

unsigned int mem\_sig\_dim[100] = {0};

unsigned int mem\_temp [100] = {0};

i=0,j=0,m=0,o=0,s=0,a=1;

/\* Open File to Read Quadruples \*/

FILE \*f;

f =fopen("file.txt","r");

while(!feof(f))

{

fscanf(f, "%i %i %i %i \n", &cuad1[i], &cuad2[i], &cuad3[i], &cuad4[i]);

i++;

}

fclose(f);

/\* Close File to Read Quadruples \*/

while(j<i){

switch(cuad1[j]){

case 0://=

if(cuad2[j]<1000){ // Asignación de Constantes

if(cuad4[j]>=1000 && cuad4[j]<2000)

mem\_in[cuad4[j]-1000] = cuad2[j];// Se asigna Const en In

else if(cuad4[j]>=2000 && cuad4[j]<3000)

mem\_out[cuad4[j]-2000] = cuad2[j];//Se asigna Const en Out

else if(cuad4[j]>=3000 && cuad4[j]<4000)

mem\_sig[cuad4[j]-3000] = cuad2[j];//Se asigna Const en Signal

else if(cuad4[j]>=6000 && cuad4[j]<7000)

mem\_in[cuad4[j]-6000] = cuad2[j];//Se asigna Const en Temp

}

else if(cuad2[j]>=1000 && cuad2[j]<2000){//Asignación de Ins

if(cuad4[j]>=2000 && cuad4[j]<3000)

mem\_out[cuad4[j]-2000] = mem\_in[cuad2[j]-1000];//Se asigna In en Out

else if(cuad4[j]>=3000 && cuad4[j]<4000)

mem\_sig[cuad4[j]-3000] = mem\_in[cuad2[j]-1000];//Se asigna In en Signal

else if(cuad4[j]>=6000 && cuad4[j]<7000)

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000];//Se asigna In en Temp

}

else if(cuad2[j]>=3000 && cuad2[j]<4000){//Asignación de Signals

if(cuad4[j]>=2000 && cuad4[j]<3000)

mem\_out[cuad4[j]-2000] = mem\_sig[cuad2[j]-3000];//Se asigna Signal en Out

else if(cuad4[j]>=3000 && cuad4[j]<4000)

mem\_temp[cuad4[j]-3000] = mem\_sig[cuad2[j]-3000];//Se asigna Signal en Signal

else if(cuad4[j]>=6000 && cuad4[j]<7000)

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000];//Se asigna Signal en Temp

}

else if(cuad2[j]>=6000 && cuad2[j]<7000){//Asignación de Temporales

if(cuad4[j]>=2000 && cuad4[j]<3000)

mem\_out[cuad4[j]-2000] = mem\_temp[cuad2[j]-6000];//Se asigna Temp en Out

else if(cuad4[j]>=3000 && cuad4[j]<4000)

mem\_sig[cuad4[j]-3000] = mem\_temp[cuad2[j]-6000];//Se asigna Temp en Signal

else if(cuad4[j]>=6000 && cuad4[j]<7000)

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000];//Se asigna Temp en Temp

}

break;

case 1://==

if(cuad2[j]<1000){ // Constantes

if(cuad3[j]<1000) // Con Constantes

if(cuad2[j] == cuad3[j])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(cuad2[j] == mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(cuad2[j] == mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(cuad2[j] == mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

}

else if(cuad2[j]>=1000 && cuad2[j]<2000){ // Ins

if(cuad3[j]<1000) // Con Constantes

if(mem\_in[cuad2[j]-1000] == cuad3[j])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(mem\_in[cuad2[j]-1000] == mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(mem\_in[cuad2[j]-1000] == mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(mem\_in[cuad2[j]-1000] == mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

}

else if(cuad2[j]>=3000 && cuad2[j]<4000){ // Signals

if(cuad3[j]<1000) // Con Constantes

if(mem\_sig[cuad2[j]-3000] == cuad3[j])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(mem\_sig[cuad2[j]-3000] == mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(mem\_sig[cuad2[j]-3000] == mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(mem\_sig[cuad2[j]-3000] == mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

}

else if(cuad2[j]>=6000 && cuad2[j]<7000){ // Temporales

if(cuad3[j]<1000) // Con Constantes

if(mem\_temp[cuad2[j]-6000] == cuad3[j])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(mem\_temp[cuad2[j]-6000] == mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(mem\_temp[cuad2[j]-6000] == mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(mem\_temp[cuad2[j]-6000] == mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

}

break;

case 2://SHL

if(cuad2[j]<1000){ // Constantes

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = cuad2[j] << cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = cuad2[j] << mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = cuad2[j] << mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = cuad3[j] << mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=1000 && cuad2[j]<2000){ // Ins

if(cuad3[j]<1000) // Con Constante

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] << cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] << mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] << mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] << mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=3000 && cuad2[j]<4000){ // Signals

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] << cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] << mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] << mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] << mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=6000 && cuad2[j]<7000){ // Temporales

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] << cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] << mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] << mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] << mem\_temp[cuad3[j]-6000];

}

break;

case 3://SHR

if(cuad2[j]<1000){ // Constantes

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = cuad2[j] >> cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = cuad2[j] >> mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = cuad2[j] >> mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = cuad3[j] >> mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=1000 && cuad2[j]<2000){ // Ins

if(cuad3[j]<1000) // Con Constante

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] >> cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] >> mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] >> mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] >> mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=3000 && cuad2[j]<4000){ // Signals

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] >> cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] >> mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] >> mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] >> mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=6000 && cuad2[j]<7000){ // Temporales

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] >> cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] >> mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] >> mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] >> mem\_temp[cuad3[j]-6000];

}

break;

case 4:// !=

if(cuad2[j]<1000){ // Constantes

if(cuad3[j]<1000) // Con Constantes

if(cuad2[j] != cuad3[j])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(cuad2[j] != mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(cuad2[j] != mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(cuad2[j] != mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

}

else if(cuad2[j]>=1000 && cuad2[j]<2000){ // Ins

if(cuad3[j]<1000) // Con Constantes

if(mem\_in[cuad2[j]-1000] != cuad3[j])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(mem\_in[cuad2[j]-1000] != mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(mem\_in[cuad2[j]-1000] != mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(mem\_in[cuad2[j]-1000] != mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

}

else if(cuad2[j]>=3000 && cuad2[j]<4000){ // Signals

if(cuad3[j]<1000) // Con Constantes

if(mem\_sig[cuad2[j]-3000] != cuad3[j])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(mem\_sig[cuad2[j]-3000] != mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(mem\_sig[cuad2[j]-3000] != mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(mem\_sig[cuad2[j]-3000] != mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

}

else if(cuad2[j]>=6000 && cuad2[j]<7000){ // Temporales

if(cuad3[j]<1000) // Con Constantes

if(mem\_temp[cuad2[j]-6000] != cuad3[j])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(mem\_temp[cuad2[j]-6000] != mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(mem\_temp[cuad2[j]-6000] != mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(mem\_temp[cuad2[j]-6000] != mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

}

break;

case 5://<

if(cuad2[j]<1000){ // Constantes

if(cuad3[j]<1000) // Con Constantes

if(cuad2[j] < cuad3[j])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(cuad2[j] < mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(cuad2[j] < mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(cuad2[j] < mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

}

else if(cuad2[j]>=1000 && cuad2[j]<2000){ // Ins

if(cuad3[j]<1000) // Con Constantes

if(mem\_in[cuad2[j]-1000] < cuad3[j])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(mem\_in[cuad2[j]-1000] < mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(mem\_in[cuad2[j]-1000] < mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(mem\_in[cuad2[j]-1000] < mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

}

else if(cuad2[j]>=3000 && cuad2[j]<4000){ // Signals

if(cuad3[j]<1000) // Con Constantes

if(mem\_sig[cuad2[j]-3000]< cuad3[j])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(mem\_sig[cuad2[j]-3000] < mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(mem\_sig[cuad2[j]-3000] < mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(mem\_sig[cuad2[j]-3000] < mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

}

else if(cuad2[j]>=6000 && cuad2[j]<7000){ // Temporales

if(cuad3[j]<1000) // Con Constantes

if(mem\_temp[cuad2[j]-6000] < cuad3[j])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(mem\_temp[cuad2[j]-6000] < mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(mem\_temp[cuad2[j]-6000] < mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(mem\_temp[cuad2[j]-6000] < mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 0;

else

mem\_temp[cuad4[j]-6000] = 1;

}

break;

case 6://>

if(cuad2[j]<1000){ // Constantes

if(cuad3[j]<1000) // Con Constantes

if(cuad2[j] > cuad3[j])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(cuad2[j] > mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(cuad2[j] > mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(cuad2[j] > mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

}

else if(cuad2[j]>=1000 && cuad2[j]<2000){ // Ins

if(cuad3[j]<1000) // Con Constantes

if(mem\_in[cuad2[j]-1000] > cuad3[j])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(mem\_in[cuad2[j]-1000] > mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(mem\_in[cuad2[j]-1000] > mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(mem\_in[cuad2[j]-1000] > mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

}

else if(cuad2[j]>=3000 && cuad2[j]<4000){ // Signals

if(cuad3[j]<1000) // Con Constantes

if(mem\_sig[cuad2[j]-3000] > cuad3[j])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(mem\_sig[cuad2[j]-3000] > mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(mem\_sig[cuad2[j]-3000] > mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(mem\_sig[cuad2[j]-3000] > mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

}

else if(cuad2[j]>=6000 && cuad2[j]<7000){ // Temporales

if(cuad3[j]<1000) // Con Constantes

if(mem\_temp[cuad2[j]-6000] > cuad3[j])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

if(mem\_temp[cuad2[j]-6000] > mem\_in[cuad3[j]-1000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

if(mem\_temp[cuad2[j]-6000] > mem\_sig[cuad3[j]-3000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

if(mem\_temp[cuad2[j]-6000] > mem\_temp[cuad3[j]-6000])

mem\_temp[cuad4[j]-6000] = 1;

else

mem\_temp[cuad4[j]-6000] = 0;

}

break;

case 7://&

if(cuad2[j]<1000){ // Constantes

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = cuad2[j] & cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = cuad2[j] & mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = cuad2[j] & mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = cuad3[j] & mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=1000 && cuad2[j]<2000){ // Ins

if(cuad3[j]<1000) // Con Constante

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] & cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] & mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] & mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] & mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=3000 && cuad2[j]<4000){ // Signals

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] & cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] & mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] & mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] & mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=6000 && cuad2[j]<7000){ // Temporales

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] & cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] & mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] & mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] & mem\_temp[cuad3[j]-6000];

}

break;

case 8://|

if(cuad2[j]<1000){ // Constantes

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = cuad2[j] | cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = cuad2[j] | mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = cuad2[j] | mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = cuad3[j] | mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=1000 && cuad2[j]<2000){ // Ins

if(cuad3[j]<1000) // Con Constante

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] | cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] | mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] | mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] | mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=3000 && cuad2[j]<4000){ // Signals

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] | cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] | mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] | mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] | mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=6000 && cuad2[j]<7000){ // Temporales

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] | cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] | mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] | mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] | mem\_temp[cuad3[j]-6000];

}

break;

case 9:// !

if(cuad2[j]<1000) // Constantes

mem\_temp[cuad4[j]-6000] = !cuad2[j];

else if(cuad2[j]>=1000 && cuad2[j]<2000) // Ins

mem\_temp[cuad4[j]-6000] = !mem\_in[cuad2[j]-1000];

else if(cuad2[j]>=3000 && cuad2[j]<4000) // Signals

mem\_temp[cuad4[j]-6000] = !mem\_sig[cuad2[j]-3000];

else if(cuad2[j]>=6000 && cuad2[j]<7000) // Temporales

mem\_temp[cuad4[j]-6000] = !mem\_temp[cuad2[j]-6000];

break;

case 10://^

if(cuad2[j]<1000){ // Constantes

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = cuad2[j] ^ cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = cuad2[j] ^ mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = cuad2[j] ^ mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = cuad3[j] ^ mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=1000 && cuad2[j]<2000){ // Ins

if(cuad3[j]<1000) // Con Constante

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] ^ cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] ^ mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] ^ mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] ^ mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=3000 && cuad2[j]<4000){ // Signals

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] ^ cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] ^ mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] ^ mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] ^ mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=6000 && cuad2[j]<7000){ // Temporales

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] ^ cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] ^ mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] ^ mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] ^ mem\_temp[cuad3[j]-6000];

}

break;

case 11://+

if(cuad2[j]<1000){ // Constantes

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = cuad2[j] + cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = cuad2[j] + mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = cuad2[j] + mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = cuad3[j] + mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=1000 && cuad2[j]<2000){ // Ins

if(cuad3[j]<1000) // Con Constante

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] + cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] + mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] + mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_in[cuad2[j]-1000] + mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=3000 && cuad2[j]<4000){ // Signals

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] + cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] + mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] + mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad2[j]-3000] + mem\_temp[cuad3[j]-6000];

}

else if(cuad2[j]>=6000 && cuad2[j]<7000){ // Temporales

if(cuad3[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] + cuad3[j];

else if(cuad3[j]>=1000 && cuad3[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] + mem\_in[cuad3[j]-1000];

else if(cuad3[j]>=3000 && cuad3[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] + mem\_sig[cuad3[j]-3000];

else if(cuad3[j]>=6000 && cuad3[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad2[j]-6000] + mem\_temp[cuad3[j]-6000];

}

break;

case 12://-

if(cuad3[j]<1000){ // Constantes

if(cuad2[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = cuad3[j] - cuad2[j];

else if(cuad2[j]>=1000 && cuad2[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = cuad3[j] - mem\_in[cuad2[j]-1000];

else if(cuad2[j]>=3000 && cuad2[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = cuad3[j] - mem\_sig[cuad2[j]-3000];

else if(cuad2[j]>=6000 && cuad2[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = cuad3[j] - mem\_temp[cuad2[j]-6000];

}

else if(cuad3[j]>=1000 && cuad3[j]<2000){ // Ins

if(cuad2[j]<1000) // Con Constante

mem\_temp[cuad4[j]-6000] = mem\_in[cuad3[j]-1000] - cuad2[j];

else if(cuad2[j]>=1000 && cuad2[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_in[cuad3[j]-1000] - mem\_in[cuad2[j]-1000];

else if(cuad2[j]>=3000 && cuad2[j]<4000) // Con Signals

mem\_temp[cuad4[j]-6000] = mem\_in[cuad3[j]-1000] - mem\_sig[cuad2[j]-3000];

else if(cuad2[j]>=6000 && cuad2[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_in[cuad3[j]-1000] - mem\_temp[cuad2[j]-6000];

}

else if(cuad3[j]>=3000 && cuad3[j]<4000){ // Signals

if(cuad2[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad3[j]-3000] - cuad2[j];

else if(cuad2[j]>=1000 && cuad2[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad3[j]-3000] - mem\_in[cuad2[j]-1000];

else if(cuad2[j]>=3000 && cuad2[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad3[j]-3000] - mem\_sig[cuad2[j]-3000];

else if(cuad2[j]>=6000 && cuad2[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_sig[cuad3[j]-3000] - mem\_temp[cuad2[j]-6000];

}

else if(cuad3[j]>=6000 && cuad3[j]<7000){ // Temporales

if(cuad2[j]<1000) // Con Constantes

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad3[j]-6000] - cuad2[j];

else if(cuad2[j]>=1000 && cuad2[j]<2000) // Con Ins

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad3[j]-6000] - mem\_in[cuad2[j]-1000];

else if(cuad2[j]>=3000 && cuad2[j]<4000) // Con Signal

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad3[j]-6000] - mem\_sig[cuad2[j]-3000];

else if(cuad2[j]>=6000 && cuad2[j]<7000) // Con Temporales

mem\_temp[cuad4[j]-6000] = mem\_temp[cuad3[j]-6000] - mem\_temp[cuad2[j]-6000];

}

break;

case 13://go to

j = cuad4[j]-1;

break;

case 14://go to false

if(mem\_temp[cuad2[j]-6000]==1)

j = cuad4[j]-1;

break;

case 15://go to true

if(mem\_temp[cuad2[j]-6000]==0)

j = cuad4[j]-1;

break;

case 16://Asignación variables

if(cuad2[j]>=1000 && cuad2[j]<2000){//Inputs

mem\_in\_dim[m] = cuad4[m];

dim = pow(2,mem\_in\_dim[m])-1;

do{/\*Verify Inputs\*/

printf("Input %c (En Decimal)\n", alfabeto[j]);

scanf("%d", &mem\_in[cuad2[j]-1000]);

if(pow(2,mem\_in\_dim[m])-1 < mem\_in[cuad2[j]-1000]){

printf("Argumento Invalido, Favor de Introducir un Numero Valido\n");

printf("Dimension Máxima de la variable %d\n\n\n",dim);

}

}while(dim < mem\_in[cuad2[j]-1000]);

m++;

}

else if(cuad2[j]>=2000 && cuad2[j]<3000){//Outputs

mem\_out[o] = 0;

mem\_out\_dim[o] = cuad4[j];

o++;

}

else if(cuad2[j]>=3000 && cuad2[j]<4000){//Signals

mem\_sig[s] = 0;

mem\_sig\_dim[s] = cuad4[j];

s++;

}

break;

default:

printf("ERROR \n");

break;

}

j++;

a++;

}

j=0;

while(j<o){ /\*Print Outputs\*/

printf("\n");

if(mem\_out[j]==0)

printf("Output %i:\n0\n",j);

else{

printf("Output %i:\n",j);

binario(mem\_out[j]);

}

j++;

}

printf("\n\n");

}

return 0;

}