LP02 - EXAMEN FINAL

Propósito

El proyecto del curso consiste en analizar, diseñar e implementar un (pequeño) prototipo de lenguaje de programación imperativo, construyendo, de esta manera, los dos primeros módulos “Preprocessor” (preprocesador) y “Compiler” (compilador) de un sistema de procesamiento del lenguaje Al implementar estos dos módulos quedaría completo un sistema de procesamiento de lenguaje para una arquitectura computacional basada en Microprocesador Z80.

Integrantes

Integrante	Correo
Luis Alejandro Higuarán Serrano	lahiguarans@unal.edu.co
Yesid Alberto Ochoa Luque	yaochoal@unal.edu.co

Entregables

a. Diseño de módulos construidos:

Ventana principal:

La ventana principal es la encargada de gestionar el resto de clases "Z80,Editor Código,Editor Assembler, Editor de Memoria y así mismo es el puente para comunicar los datos y mostrar la arquitectura y funcionamiento del z80, esta hace uso de las clases mencionadas para obtener los datos del estado actual del z80 en su ejecución y para cargar el código re-localizable.

Editor de Código:

El editor de código esta implementado de tal forma que obtenga el texto de un fichero y este se pueda editar o guardar, adicional mente esta clase cuenta con la comunicación a las clases que ejecutan el compilador hecho en ANTLR el cual procesa el código para generar posteriormente el assembler del Z80 y así poderlo guardar en un nuevo fichero.

Editor de Assembler:

El editor de assembler esta implementado de tal forma que recibe el assembler y genera el código re-localizable y asigna las instrucciones que se van a ejecutar en formato hexadecimal y las guarda en una memoria temporal para posteriormente resolver las direcciones.

Z80:

La CPU Z80 fue implementada de tal manera que pueda ejecutar las instrucciones principales como cargar a los registros, ejecutar sumas, restas y asignar variables alterando las banderas dependiendo de cada instrucción, este recibe el hexadecimal generado por el assembler en las direcciones de memoria ya resueltas y finalmente ejecuta cada una de estas cuando se oprime ejecutar en la ventana principal dependiendo de los ciclos que se le pongan en la casilla, siendo 4 el número mínimo para ejecutar una instrucción o más dependiendo de la rapidez que se quiere que se ejecute el programa.

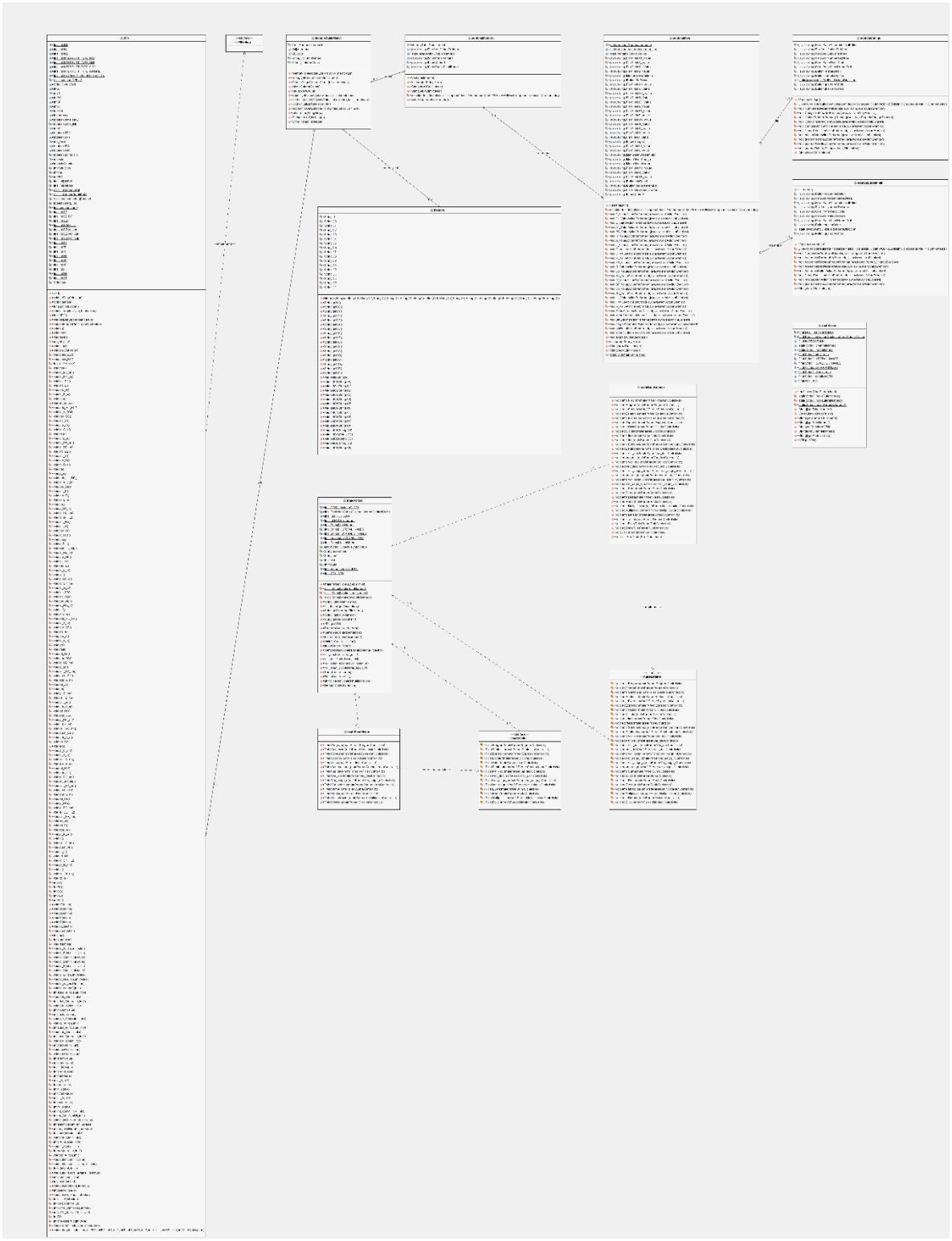


Diagrama de clases del Proyecto.

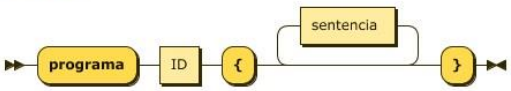
b. Categorías léxicas y sus expresiones regulares asociadas:

Categoría léxica	Expresión regular
PROGRAMA	'programa'
VAR	'var'
VEC	'vec'
SUMA	'+'
RESTA	'-'
MULT	'*'
DIVI	'/'
IF	'if'
ELSE	'else'
FOR	'for'
MAYORI	'>='
MENORQ	'<'
EQUIVA	'=='
ASIGNAR	'='
LLAVE_ABRE	'{'

Categoría léxica	Expresión regular
LLAVE_CIERRA	'}'
PAR_ABRE	'('
PAR_CIERRA	')'
SEMICOLON	';'
ID	[a-zA-Z_][a-zA-Z0-9_]*
NUMERO	[0-9]+
ESPACIOS	[\t\n\r]+

c y d. Gramáticas:

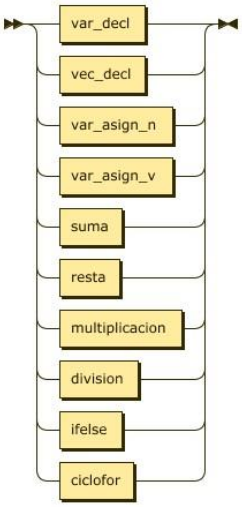
program:




```
program  ::= 'programa' ID '{' sentencia* '}'
```

no references

sentencia:

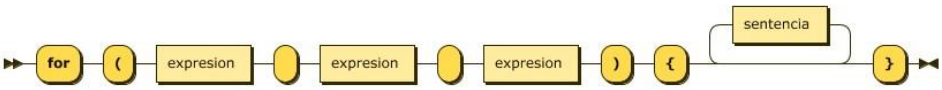



```
sentencia
         ::= var_decl
           | vec_decl
           | var_asign_n
           | var_asign_v
           | suma
           | resta
           | multiplicacion
           | division
           | ifelse
           | ciclofor
```

referenced by:

- ciclofor
- ifelse
- program

ciclofor:

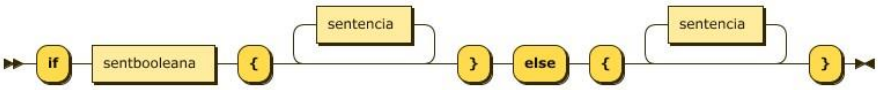



```
ciclofor ::= 'for' '(' expresion '' expresion '' expresion ')' '{' sentencia* '}'
```

referenced by:

- sentencia

ifelse:

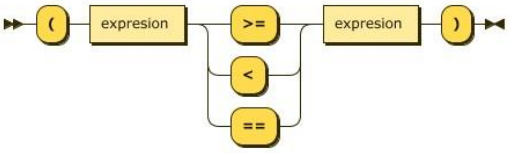



```
ifelse   ::= 'if' sentbooleana '{' sentencia* '}' 'else' '{' sentencia* '}'
```

referenced by:

- sentencia

sentbooleana:

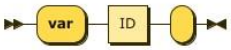



```
sentbooleana
        ::= '(' expresion ( '>=' | '<' | '==' ) expresion ')'
```

referenced by:

- ifelse

var_decl:

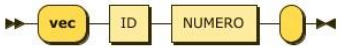



```
var_decl ::= 'var' ID ''
```

referenced by:

- sentencia

vec_decl:

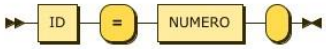



```
vec_decl ::= 'vec' ID NUMERO ''
```

referenced by:

- sentencia

var_asign_n:

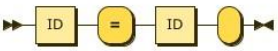



```
var_asign_n
        ::= ID '=' NUMERO ''
```

referenced by:

- sentencia

var_asign_v:

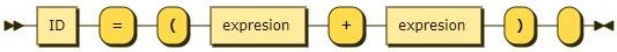



```
var_asign_v
        ::= ID '=' ID ''
```

referenced by:

- sentencia

suma:

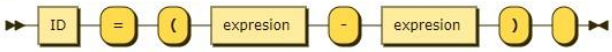



```
suma     ::= ID '=' '(' expresion '+' expresion ')' ''
```

referenced by:

- sentencia

resta:

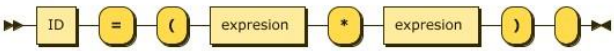



```
resta    ::= ID '=' '(' expresion '-' expresion ')' ''
```

referenced by:

- sentencia

multiplicacion:

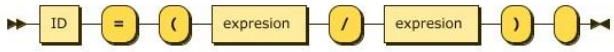



```
multiplicacion
        ::= ID '=' '(' expresion '*' expresion ')' ''
```

referenced by:

- sentencia

division:

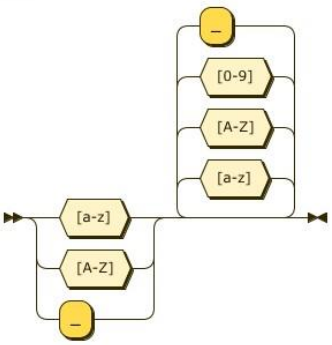



```
division ::= ID '=' '(' expresion '/' expresion ')' ''
```

referenced by:

- sentencia

ID:

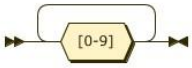



```
ID      ::= [a-zA-Z_] [a-zA-Z0-9_]*
```

referenced by:

- division
- multiplicacion
- program
- resta
- suma
- var_asign_n
- var_asign_v
- var_decl
- vec_decl

NUMERO:




```
NUMERO   ::= [0-9]+
```

referenced by:

- var_asign_n
- vec_decl

e. Fuentes de la aplicación construida:

Herramienta	Descripción	Fuentes
Z80 Lógica	Para la implementación de la lógica del Z80 se utilizó principalmente el manual de esta misma CPU como también varios videos de Youtube para entender su funcionamiento básico.	Manual Z80 Ciclo de Ejecución de la CPU
Z80 Instrucciones	Para las instrucciones se utilizó la tabla fuente como también el manual para implementar la logica	Tabla de instrucciones del Z80

Herramienta	Descripción	Fuentes
Z80 Assembler	Para aprender el assembler del z80 se utilizaron principalmente los 2 enlaces citados para aprender la sintaxis básica y ver ejemplos de programas que ejecuten operaciones matemáticas básicas.	Assembler for Dummies (Z80)Z80 Heaven Math
Analizador Léxico, sintáctico y Semántico	Para la implementación del Analizador léxico, sintáctico y semántico se utilizo la herramienta ANTLR y el manual llamado The Definitive ANTLR 4 Reference para utilizar esta misma adecuadamente	ANTLR The Definitive ANTLR 4 Reference

f. Manual de usuario y técnico:

1. Ventana principal:

La ventana principal de nuestro Z80 esta principalmente compuesta por la interfaz que muestra la arquitectura interna del Z80 mostrando los valores de sus registros los cuales se actualizan a cada momento que se ejecute un programa previamente cargado, adicional mente se agregan opciones para reiniciar la CPU y cambiar los ciclos de ejecución para visualizar paso a paso la ejecución del programa o que este se ejecute completamente de principio a fin. También contamos con un menú superior el cual nos permite acceder a un editor de código el cual explicaremos a continuación.

2. Editor de Código:

El editor de código cuenta con una pequeña interfaz la cual permite abrir un archivo con extensión ".ma" la cual es la que usaremos en el lenguaje diseñado para este proyecto el cual se bautizo como MAFE. Así mismo se permite visualizarlo y editarlo. Una vez que este terminado el código de nuestro programa se compila oprimiendo el botón de "Compilar" generando el Assembler correspondiente que recibirá el Z80,

este Assembler debe ser guardado con el botón "Guardar Assembler" en un archivo de extensión ".asm" la cual elegimos para identificar nuestros archivos de formato Assembler y posteriormente cargar en nuestro Editor de Assembler.

3. Editor de Assembler:

El editor de Assembler cuenta con una pequeña interfaz la cual permite abrir el archivos con extensión ".asm" que hayamos generado previamente en el editor de código, así mismo se permite visualizarlo y editarlo, una vez hayamos terminado de editar o verificar que nuestro Assembler es correcto procedemos a generar el código re-localizable a nuestro Z80 oprimiendo el botón Assemble y visualizando la dirección de memoria y la instrucción de maquina en la casilla de texto inferior.

4. Editor de Memoria y DEBUG:

Nota: Paso previo: Para visualizar el código maquina en el Editor de Memoria se debe haber oprimido previamente el botón "Cargar Assembler" en la ventana principal

El editor de Memoria cuenta con una pequeña tabla la cual permite visualizar y editar el código re-localizable que hayamos generado previamente en el editor de Assembler,una vez hayamos terminado de editar o verificar que nuestro memoria es correcta procedemos a oprimir el botón de **Ejecutar Programa** en la ventana principal visualizando el DEBUG en el editor de memoria de las instrucciones ejecutadas por el Z80.

5. Especificaciones técnicas del Z80 y el Lenguaje Implementado:

a. Z80 instrucciones soportadas:

Son soportadas todas las instrucciones principales desde la tabla "ver fuente Tabla de Instrucciones" del z80 00 hasta la FF.

b. Tipos de variable soportadas:

Son soportadas las variable de valor numérico y los vectores también de valor numérico que estén definidos en el dominio de 0 hasta 255.

```
ejemplo: ./variables.ma
programa main{
  var numero;
  numero = 3;

  vec vector 2;
  vector_1 = 1;
  vector_2 = 2;

}
```

c. Operaciones matemáticas soportadas por el lenguaje implementado:

Son soportadas la suma, resta, multiplicación y división de números enteros o variables asociadas a estos.

```
ejemplo: ./operaciones.ma
programa main{
  var a;
  var b;
  var suma;
  var resta;
  var multi;
  var divic;

  a = 10;
  b = 5;
  suma = (a+b);
  resta = (a-b);
  multi = (a*b);
  divic = (a/b);
}
```

d. Estructuras de selección soportadas por el lenguaje implementado:

Es soportado el if-else con operaciones boleanas de mayor igual ">=" menor que"<" y la equivalencia lógica"==".

```
ejemplo: ./ifelse.ma
programa main{
  var a;
  var b;
  var valor;
```

```
  a = 10;
  b = 5;

  if(a>=0){
      valor = (a+b);
  }else{
      valor = (a-b);
  }
}
```

e. Estructuras de control iterativas soportadas por el lenguaje implementado:

Es soportado el ciclo for el cual solo puede repetirse un x número de veces con un contador siempre de valor 1.

```
ejemplo: ./ciclofor.ma
programa main{
  var inicio;
  var fin;
  var incremento;
  var a;

  a = 1;
  inicio = 0;
  fin = 10;
  incremento = 1;

  for(inicio;fin;incremento){
      a = (a+1);
  }
}
```

g. Informe ejecutivo, ejemplos:

Requisitos

Java 1.8 o superior.

Ejemplo 1: Suma, resta, multiplicación y división:

1. Dirigirse al ejecutable de la aplicación z80.jar y abrir la aplicación.
2. Dar clic en el menú superior de "ver editor de código", crear un archivo ".ma" con la siguiente extensión y código y dar clic a el botón "abrir fichero" de la interfaz.

```
ejemplo: ./operaciones.ma
programa main{
  var a;
  var b;
  var suma;
  var resta;
  var multi;
  var divic;

  a = 10;
  b = 5;
  suma = (a+b);
  resta = (a-b);
  multi = (a*b);
  divic = (a/b);
}
```

3. Luego dar clic al botón "Compilar", una vez generado en la parte inferior el assembler correspondiente, dar clic en el botón "Guardar Assembler" y guardar el archivo con extensión ".asm" de la siguiente forma "operaciones.asm"

4. Una vez hecho esto dirigirse a la ventana principal y dar clic en el editor de Assembler, oprimir en "abrir fichero" y abrimos el fichero anteriormente creado "operaciones.asm". luego procedemos a dar clic a la opción "Assemble" y nos genera el código re-localizable.

5. Una vez terminado el paso anterior se procede a ir a la ventana principal y oprimir en el botón "Cargar Assembler" para resolver las direcciones de memoria y cargarlo a la memoria del z80, esto se puede ver dando clic al menu superior "Ver memoria y debug" en la ventana principal del Z80.

6. Una vez cargado en la memoria se procede a ejecutar el programa dando clic al botón "ejecutar programa" donde este mostrara en la ventana de "ver codigo y debug" las instrucciones del z80 que se ejecutan en ese momento.

7. Para visualizar los resultados de las operaciones, se guardan los resultados en la memoria a partir de la dirección 8000HEX de la tabla por lo cual para verificar que los resultado son correctos en la ventana "ver código y debug" nos dirigimos a esta parte de memoria y visualizamos los resultados.

Ejemplo 2: Ifelse:

1. Dirigirse al ejecutable de la aplicación z80.jar y abrir la aplicación.

2. Dar clic en el menú superior de "ver editor de código", crear un archivo ".ma" con la siguiente extensión y código y dar clic a el botón "abrir fichero" de la interfaz.

```
ejemplo: ./ifelse.ma
programa main{
  var a;
  var b;
  var valor;

  a = 10;
  b = 5;

  if(a>=0){
      valor = (a+b);
  }else{
      valor = (a-b);
  }
}
```

3. Luego dar clic al botón "Compilar", una vez generado en la parte inferior el assembler correspondiente, dar clic en el botón "Guardar Assembler" y guardar el archivo con extensión ".asm" de la siguiente forma "ifelse.asm"

4. Una vez hecho esto dirigirse a la ventana principal y dar clic en el editor de Assembler, oprimir en "abrir fichero" y abrimos el fichero anteriormente creado "ifelse.asm". luego procedemos a dar clic a la opción "Assemble" y nos genera el código re-localizable.

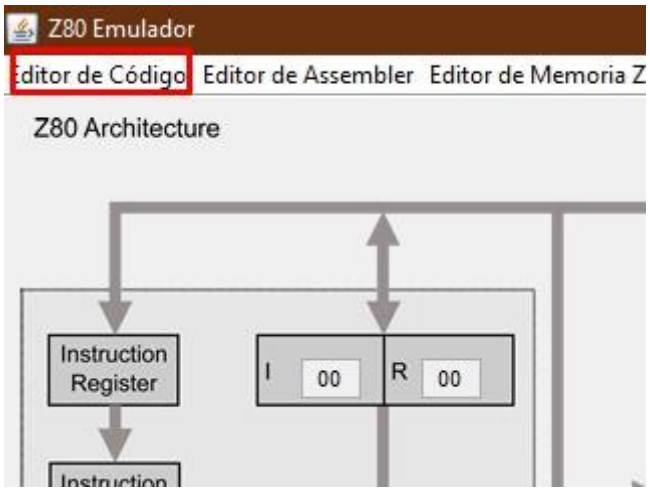
5. Una vez terminado el paso anterior se procede a ir a la ventana principal y oprimir en el botón "Cargar Assembler" para resolver las direcciones de memoria y cargarlo a la memoria del z80, esto se puede ver dando clic al menú superior "Ver memoria y debug" en la ventana principal del Z80.

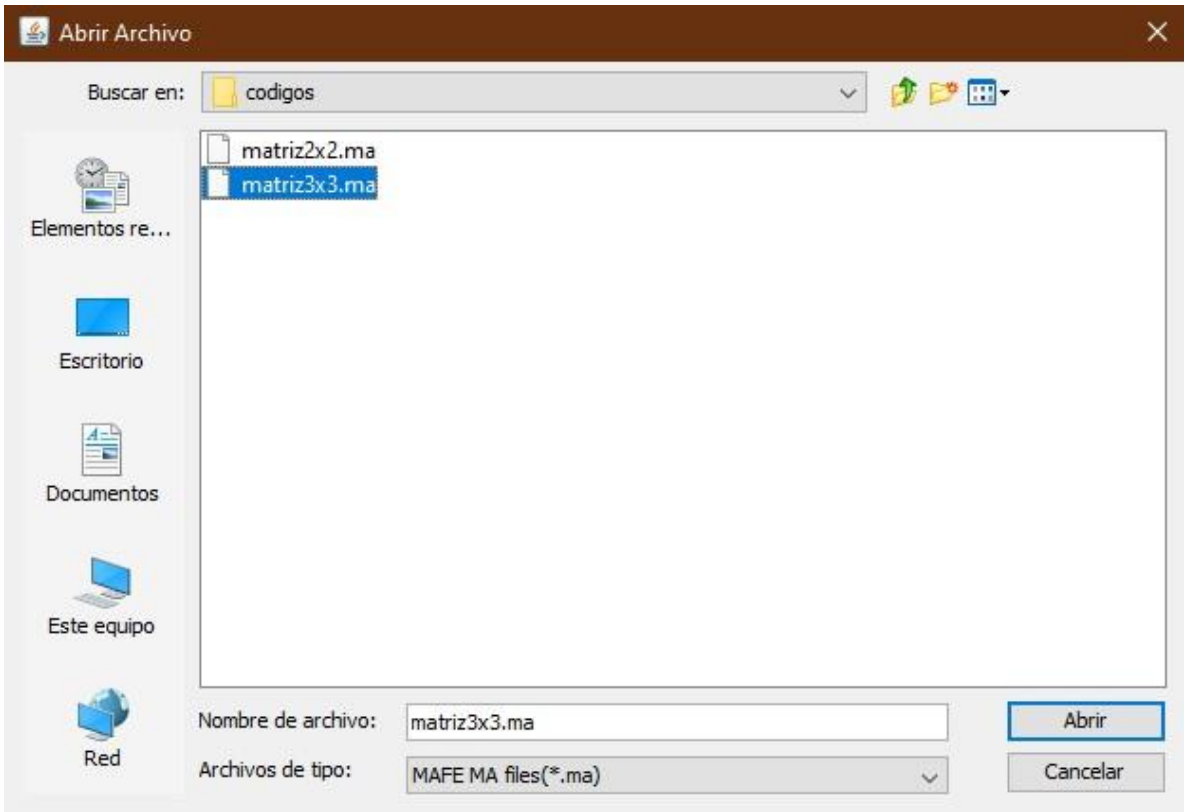
6. Una vez cargado en la memoria se procede a ejecutar el programa dando clic al botón "ejecutar programa" donde este mostrara en la ventana de "ver código y debug" las instrucciones del z80 que se ejecutan en ese momento.

7. Para visualizar los resultados de las operaciones, se guardan los resultados en la memoria a partir de la dirección 8000HEX de la tabla por lo cual para verificar que los resultado son correctos en la ventana "ver código y debug" nos dirigimos a esta parte de memoria y visualizamos los resultados.

Ejemplo 3: Multiplicación de Matrices 3x3:

1. Dirigirse al ejecutable de la aplicación z80.jar y abrir la aplicación.
2. Dar clic en el menú superior de "ver editor de código", crear un archivo ".ma" con la siguiente extensión y código y dar clic a el botón "abrir fichero" de la interfaz.





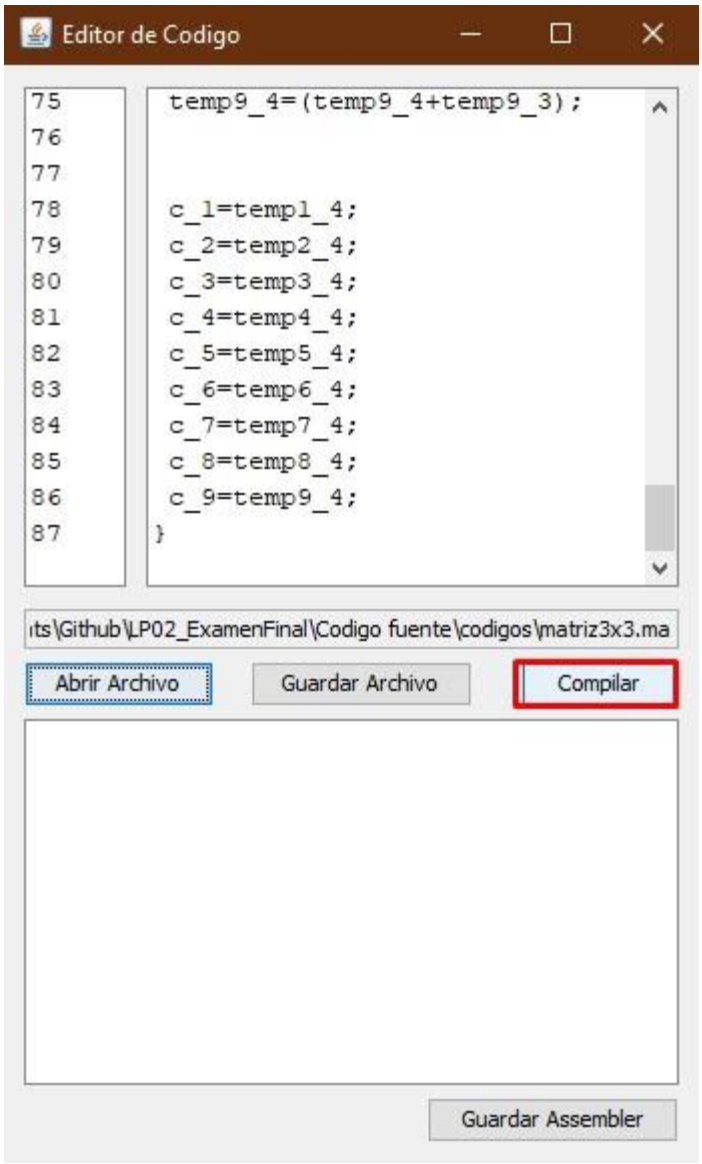
ejemplo: ./matrices3x3.ma

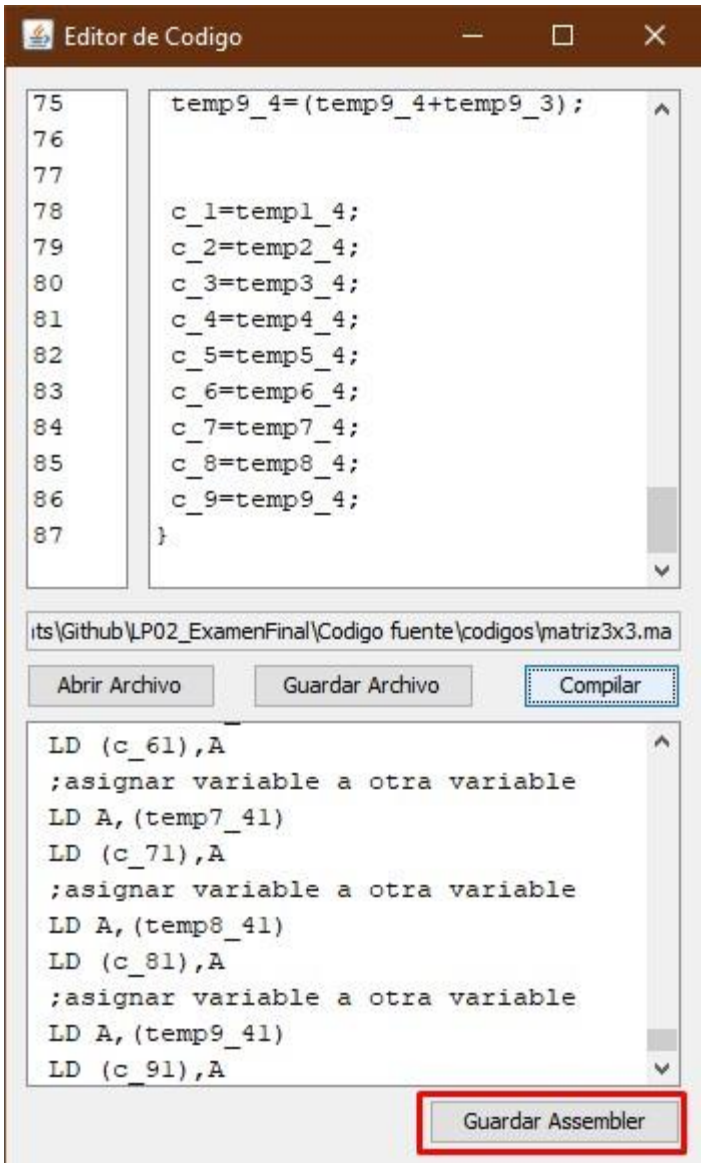
```
programa main{

 vec c 9;
 vec a 9;
 vec b 9;

 vec temp1 4;
 vec temp2 4;
 vec temp3 4;
 vec temp4 4;
 vec temp5 4;
 vec temp6 4;
 vec temp7 4;
 vec temp8 4;
 vec temp9 4;
```

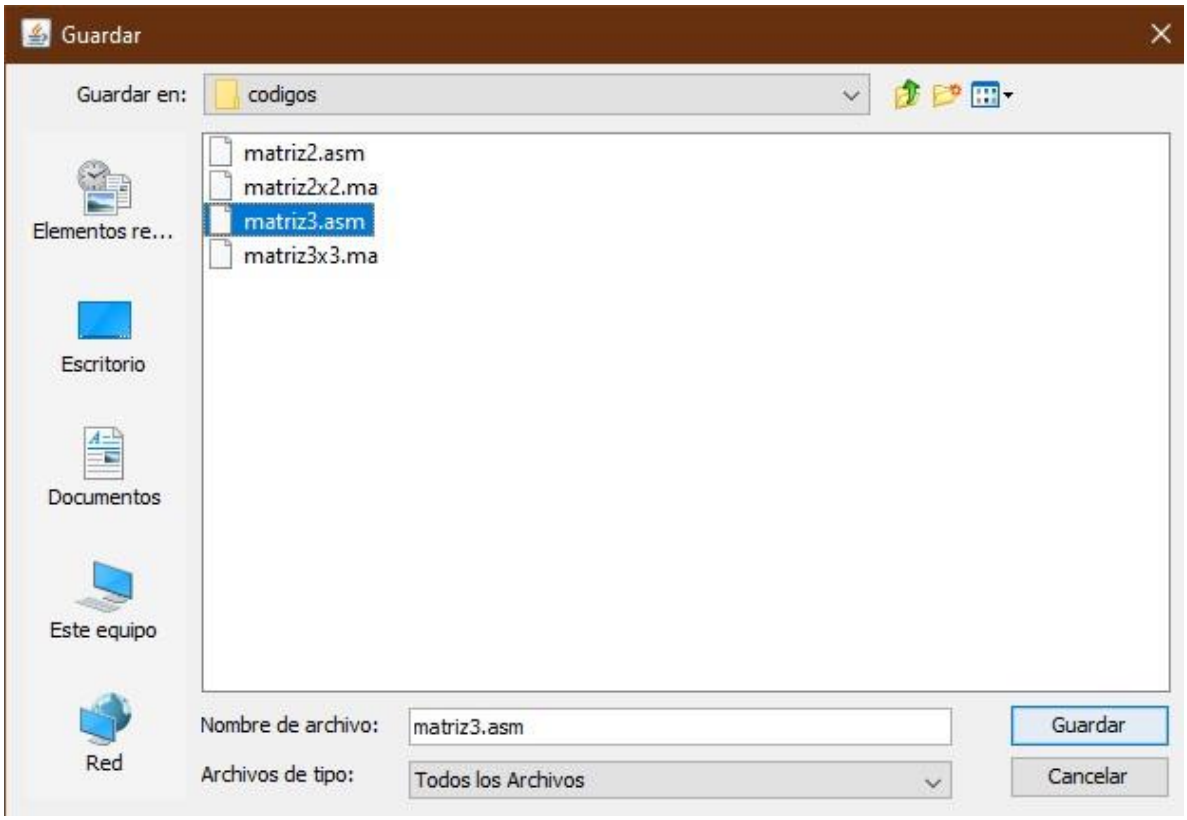
```
a_1=1;a_2=2;a_3=3;
a_4=4;a_5=5;a_6=6;
a_7=7;a_8=8;a_9=9;

b_1=1;b_2=2;b_3=3;
b_4=4;b_5=5;b_6=6;
b_7=7;b_8=8;b_9=9;

temp1_1=(a_1*b_1);
temp1_2=(a_2*b_4);
temp1_3=(a_3*b_7);
temp1_4=(temp1_1+temp1_2);
temp1_4=(temp1_4+temp1_3);

temp2_1=(a_1*b_2);
temp2_2=(a_2*b_5);
temp2_3=(a_3*b_8);
temp2_4=(temp2_1+temp2_2);
temp2_4=(temp2_4+temp2_3);

temp3_1=(a_1*b_3);
temp3_2=(a_2*b_6);
temp3_3=(a_3*b_9);
temp3_4=(temp3_1+temp3_2);
temp3_4=(temp3_4+temp3_3);

temp4_1=(a_4*b_1);
temp4_2=(a_5*b_4);
temp4_3=(a_6*b_7);
temp4_4=(temp4_1+temp4_2);
temp4_4=(temp4_4+temp4_3);

temp5_1=(a_4*b_2);
temp5_2=(a_5*b_5);
temp5_3=(a_6*b_8);
temp5_4=(temp5_1+temp5_2);
temp5_4=(temp5_4+temp5_3);

temp6_1=(a_4*b_3);
temp6_2=(a_5*b_6);
temp6_3=(a_6*b_9);
temp6_4=(temp6_1+temp6_2);
temp6_4=(temp6_4+temp6_3);

temp7_1=(a_7*b_1);
temp7_2=(a_8*b_4);
temp7_3=(a_9*b_7);
temp7_4=(temp7_1+temp7_2);
temp7_4=(temp7_4+temp7_3);

temp8_1=(a_7*b_2);
temp8_2=(a_8*b_5);
temp8_3=(a_9*b_8);
temp8_4=(temp8_1+temp8_2);
```

```
 temp8_4=(temp8_4+temp8_3);

 temp9_1=(a_7*b_3);
 temp9_2=(a_8*b_6);
 temp9_3=(a_9*b_9);
 temp9_4=(temp9_1+temp9_2);
 temp9_4=(temp9_4+temp9_3);


 c_1=temp1_4;
 c_2=temp2_4;
 c_3=temp3_4;
 c_4=temp4_4;
 c_5=temp5_4;
 c_6=temp6_4;
 c_7=temp7_4;
 c_8=temp8_4;
 c_9=temp9_4;

}
```

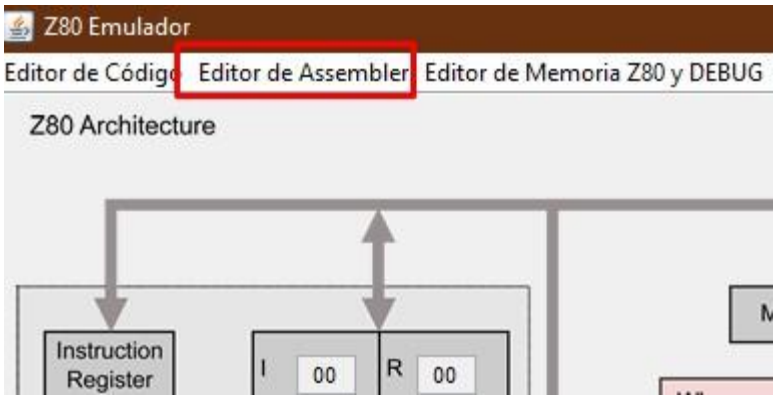
3. Luego dar clic al botón "Compilar", una vez generado en la parte inferior el assembler correspondiente, dar clic en el botón "Guardar Assembler" y guardar el archivo con extensión ".asm" de la siguiente forma "matrices3x3.asm"

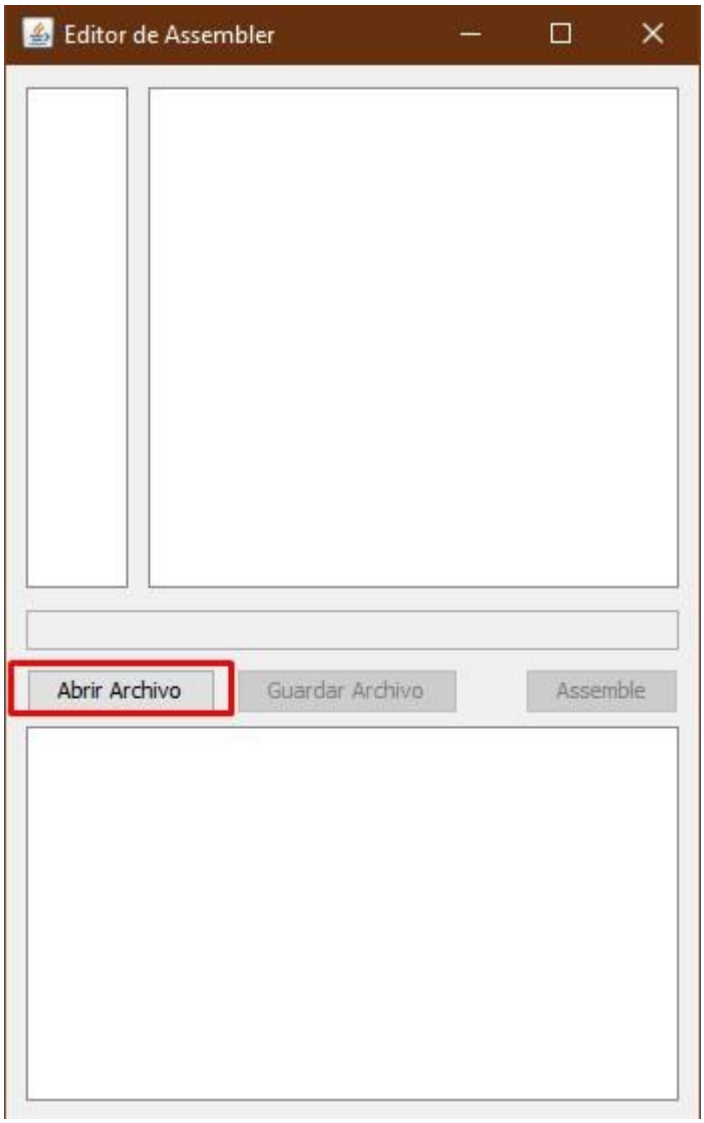

Editor de Codigo
75
76
77
78
79
80
81
82
83
84
85
86
87
temp9_4=(temp9_4+temp9_3);
c_1=temp1_4;
c_2=temp2_4;
c_3=temp3_4;
c_4=temp4_4;
c_5=temp5_4;
c_6=temp6_4;
c_7=temp7_4;
c_8=temp8_4;
c_9=temp9_4;
}
ıts\Github\LP02_ExamenFinal\Codigo fuente\codigos\matriz3x3.ma
Abrir Archivo
Guardar Archivo
Compilar
Guardar Assembler

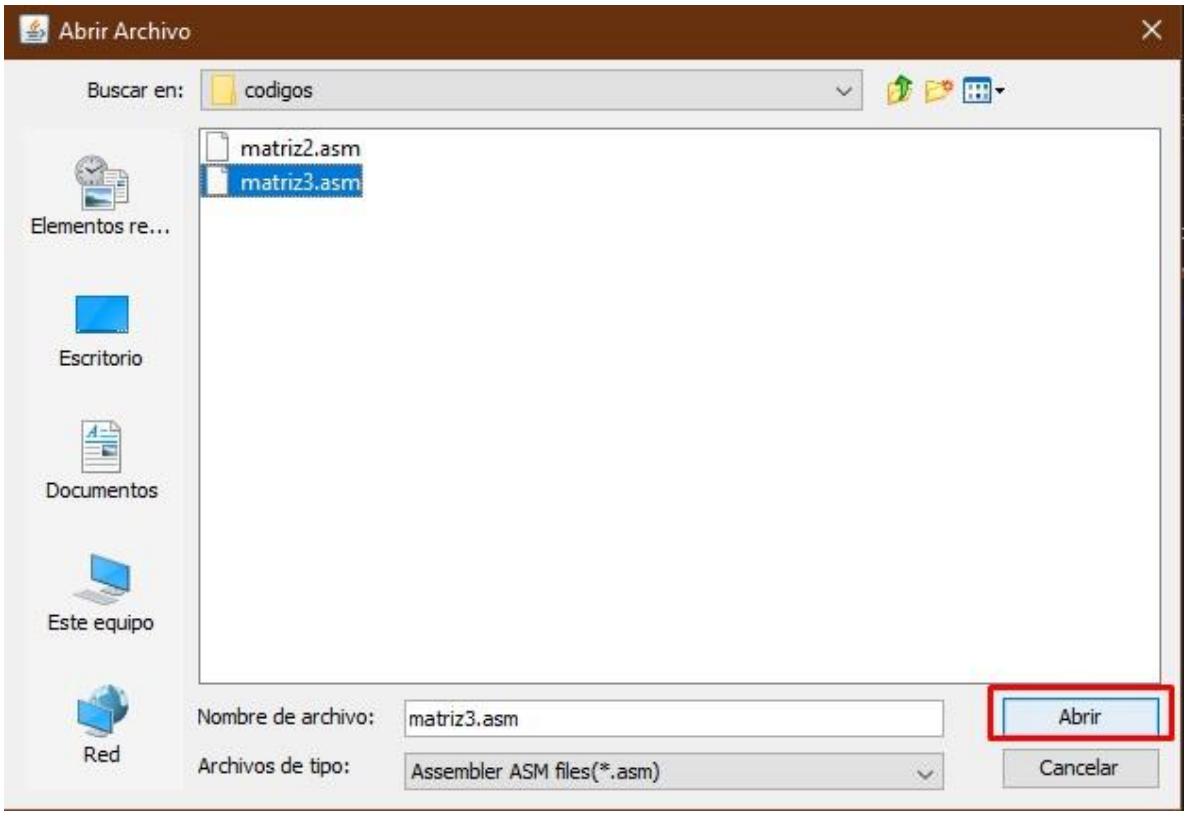

Editor de Codigo
75
76
77
78
79
80
81
82
83
84
85
86
87
temp9_4=(temp9_4+temp9_3);
c_1=temp1_4;
c_2=temp2_4;
c_3=temp3_4;
c_4=temp4_4;
c_5=temp5_4;
c_6=temp6_4;
c_7=temp7_4;
c_8=temp8_4;
c_9=temp9_4;
}
its\Github\LP02_ExamenFinal\Codigo fuente\codigos\matriz3x3.ma
Abrir Archivo
Guardar Archivo
Compilar
LD (c_61),A
;asignar variable a otra variable
LD A,(temp7_41)
LD (c_71),A
;asignar variable a otra variable
LD A,(temp8_41)
LD (c_81),A
;asignar variable a otra variable
LD A,(temp9_41)
LD (c_91),A
Guardar Assembler



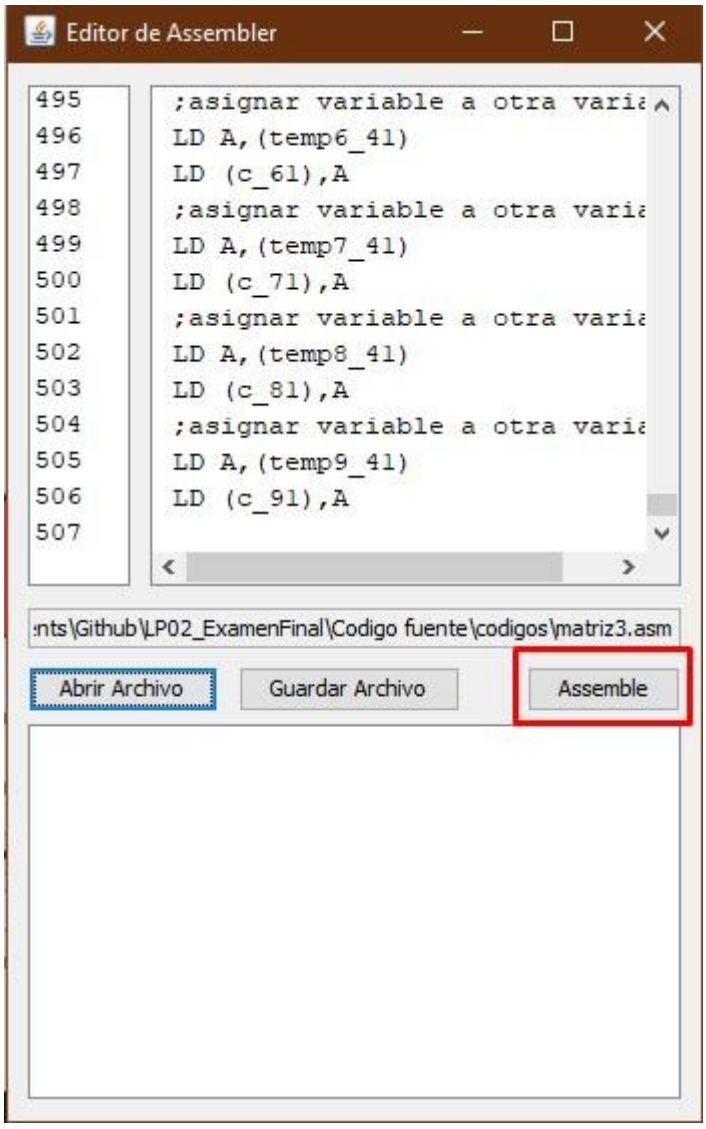
4. Una vez hecho esto dirigirse a la ventana principal y dar clic en el editor de Assembler, oprimir en "abrir fichero" y abrimos el fichero anteriormente creado "matrices3x3.asm". luego procedemos a dar clic a la opción "Assemble" y nos genera el código re-localizable.



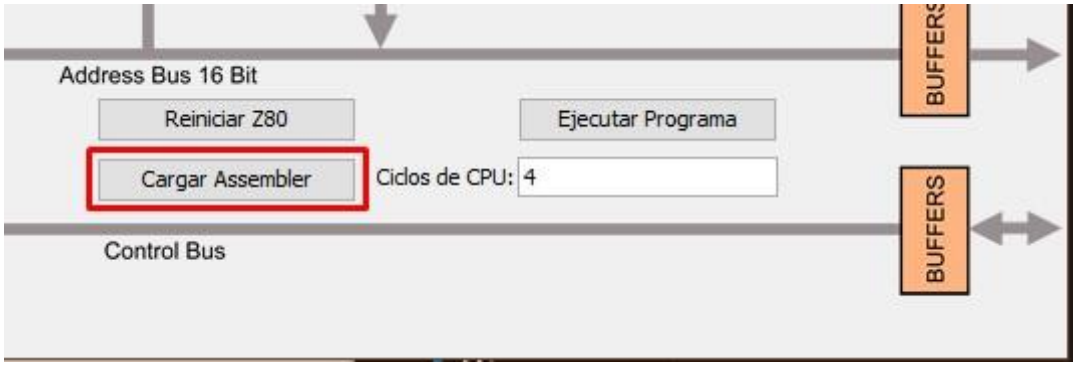

Editor de Assembler
Abrir Archivo
Guardar Archivo
Assemble

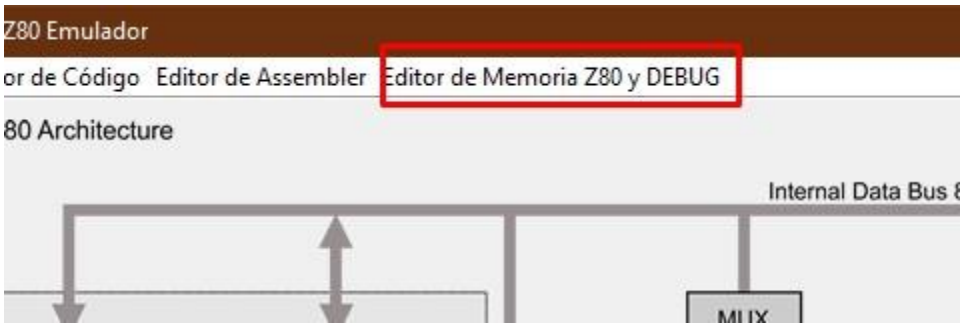


5. Una vez terminado el paso anterior se procede a ir a la ventana principal y oprimir en el botón "Cargar Assembler" para resolver las direcciones de memoria y cargarlo a la memoria del z80, esto se puede ver dando clic al menu superior "Ver memoria y debug" en la ventana principal del Z80.

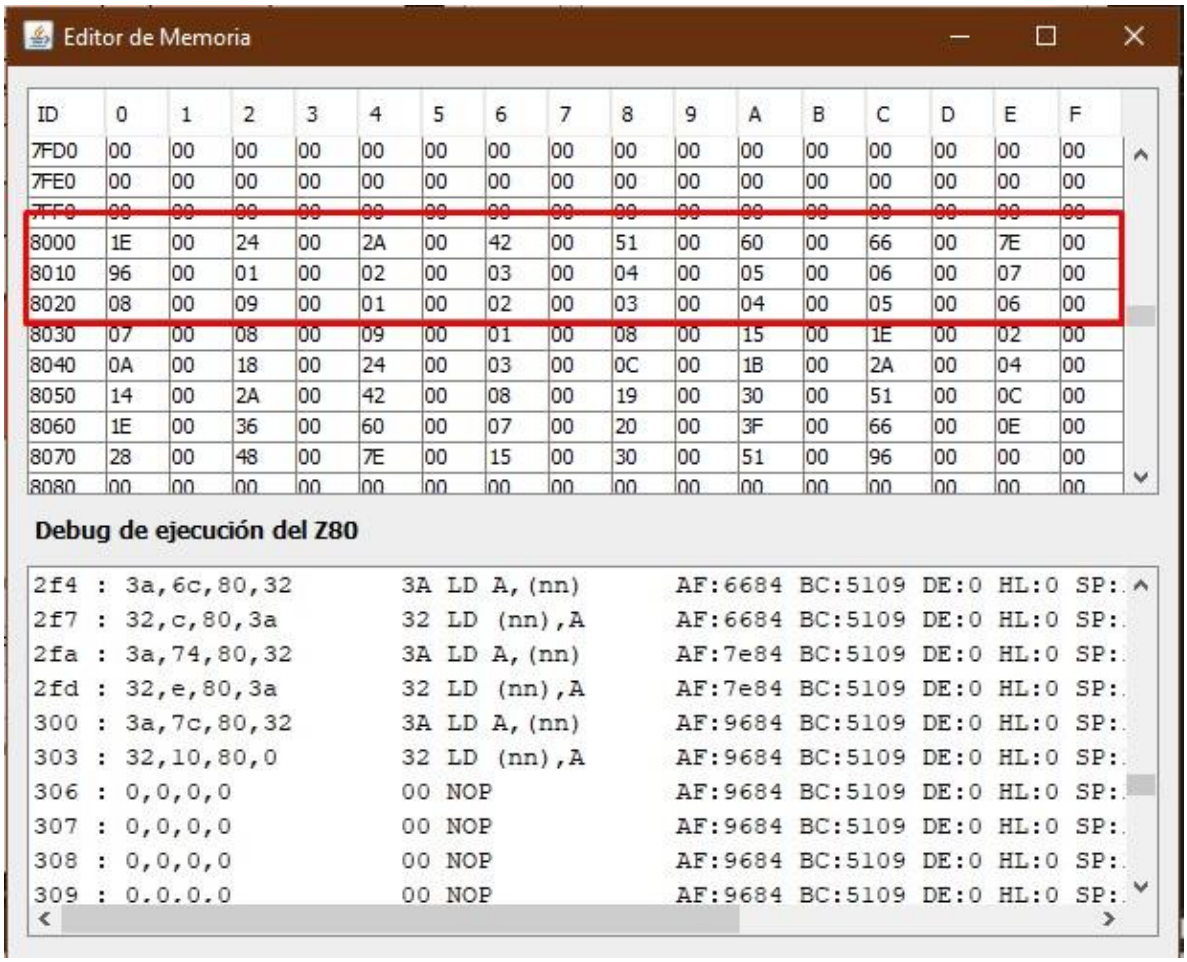


6. Una vez cargado en la memoria se procede a ejecutar el programa dando clic al botón "ejecutar programa" donde este mostrara en la ventana de "ver código y debug" las instrucciones del z80 que se ejecutan en ese momento.





7. Para visualizar los resultados de las operaciones, se guardan los resultados en la memoria a partir de la dirección 8000HEX de la tabla por lo cual para verificar que los resultado son correctos en la ventana "ver código y debug" nos dirigimos a esta parte de memoria y visualizamos los resultados.



Análisis de resultados.

Podemos observar que este proyecto nos hizo comprender de manera muy profunda el desarrollo no solo de un lenguaje de programación básico y todas las componentes que este tiene si no también como funciona un procesador hasta su más bajo nivel dándonos una perspectiva profunda y especializada de cómo se realizan estos desarrollos brindándonos herramientas de alta calidad para implementar y desarrollar herramientas a futuro.