UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ingeniería

Estructura y Programación de Computadoras

Proyecto 1: Compilador básico para el MC68HC11 de Motorola

Integrantes:

- Lázaro Martínez Annette Ariadna
- Martínez Jiménez María Fernanda
- Mendoza de la Vega Dulce Elizabeth
- Mendoza de los Santos Lirio Aketzalli

Grupo: 4

Profesor: M. I. Pedro Ignacio Rincón Gómez

Semestre 2021-1
Fecha de entrega: 01/diciembre/2020

Identificación con fotografía:

Lázaro Martínez Annette Ariadna

Martínez Jiménez María Fernanda

Mendoza de la Vega Dulce Elizabeth

Mendoza de los Santos Lirio Aketzalli

Reporte con evidencias:

Para la simulación del compilador MC68HC11 de Motorola, usamos el Entorno de Desarrollo Integrado NetBeans para el lenguaje de programación Java.

Con ayuda de esta IDE creamos una interfaz gráfica, cuya pantalla principal cuenta con 3 opciones diferentes.

Para la primera opción, “ejecutar”, se requiere ingresar el nombre de un documento con extensión .asc o .ASC que contenga código compilable por el MC68HC11 de Motorola, para analizar dicho código.

En la segunda opción, “visualizar”, se despliega una segunda ventana, la cual permite al usuario abrir un documento de texto desde el explorador de archivos, este documento es el que analiza el programa. Desde esta opción, se puede, además de visualizar el archivo original con el código, modificar dicho archivo y guardar los cambios generados.

La tercera opción, “sobre el programa”, abre el reporte actual del proyecto.

El programa realiza la generación de dos archivos en caso de que el código no contenga ningún error: un archivo con extensión .lst, el cual contiene tanto el código fuente como el código objeto generado, y un archivo .s19, el cual contiene el código objeto generado por el programa.

En caso de que el archivo ejecutado contenga errores, se genera únicamente un archivo .lst con el código fuente, el código objeto generado de las instrucciones sin errores y el señalamiento de los errores encontrados durante el programa, los cuales pueden ser los siguientes:

- 001. Constante inexistente
- 002. Variable inexistente
- 003. Etiqueta inexistente
- 004. Mnemónico inexistente
- 005. Instrucción carece de operando(s)
- 006. Instrucción no lleva operando(s)
- 007. Magnitud de operando errónea
- 008. Salto relativo muy lejos
- 009. Instrucción carece de al menos un espacio relativo al margen
- 010. No se encuentra END

Para identificar los mnemónicos correspondientes de los 6 diferentes modos de direccionamiento, se generaron HashTables (colección que nos permitió corroborar que no se almacenaran valores nulos), cuya información se guardó en diferentes archivos dentro de la carpeta del programa.

Se realizó una división del proyecto en diferentes clases para llevar un mejor control y organización del proyecto, a continuación se menciona brevemente lo que realiza cada una de ellas, mencionando entre paréntesis el nombre del integrante o integrantes que la elaboraron:

- ➔ Madre (Elizabeth): Genera un objeto de la clase Menu, dicho objeto se encarga de llamar y habilitar la interfaz gráfica (JFrame Menu).

- ➔ Menu (Elizabeth): Este JFrame contiene el diseño de la primera ventana con la que usuario podrá interactuar eligiendo entre tres distintos botones que realizan actividades diferentes, además se optó por un diseño bastante simple y agradable para el usuario. A continuación se describe la funcionalidad de cada botón:

  "Ejecutar", será necesario agregar la dirección del archivo a ejecutar, además del nombre del archivo ".lst" donde se guardará el archivo ya ejecutado, mostrando el formato y características antes planteadas por el profesor. En caso de ser válido "QUE EL ARCHIVO NO CUENTE CON ERRORES" además de crear y mostrar el archivo ".S19" correspondiente.

  "Visualizar" se desplegará una nueva ventana correspondiente a la JFrame Lectura.

  Por último, el botón "Sobre el programa" mostrará una este documento.

- ➔ Lectura (Elizabeth): Este JFrame contiene el diseño de la segunda ventana con la que el usuario va a interactuar, dicha interfaz contiene un área de texto donde se mostrará el contenido del archivo, además de tres distintos botones que cumpliran objetivos diferentes, se adecuo el diseño buscando que el contenido no se viera sobresaturado. A continuación se describe la funcionalidad de cada botón:

  "Seleccionar archivo" en este caso se muestra una ventana emergente que permitirá navegar entre los archivos del equipo de tipo ".S19", ".lst",".txt" y ".asc", al ya haber seleccionado dicho archivo y dar aceptar se desplegará el contenido del archivo.

  "Guardar Cambios" en este caso se muestra una ventana emergente que permitirá navegar entre los archivos del equipo de tipo ".lst",".txt" y ".asc", al ya haber seleccionado dicho archivo y dar aceptar se desplegará el contenido del archivo, por último se reescribirá el archivo o en caso de así desearlo se creará un nuevo archivo con dichos cambios.

  "Regresar" en este caso retornara a la ventana principal.

- ➔ GestionDoc (Elizabeth): Es esta clase se encontrarán tres métodos, los cuales son lo siguientes:

  BuscarA: Esta función debe recibir la ubicación completa del archivo a ejecutar, después de comprobar su existencia este envía la dirección del archivo a la clase métodos de lectura, el cual se encarga de compilarlo. En caso de no existir o encontrar dicho archivo muestra una ventana emergente de alerta que te solicita un archivo o dirección válida.

abrirArchivo: Esta función se encarga de recibir el nombre del archivo seleccionado en el JFrame Lectura, al seleccionar un archivo válido este leerá el flujo de contenido con su valor ASCII, después lo transformarà y concatenarà el contenido en una variable tipo String, la cual será retornada a la función JFrame correspondiente. En caso de no enviar o abrir el archivo, no mostrará nada.

guardarArchivo: Esta última función se encarga sobreescribir y guardar los cambios en el archivo seleccionado, en caso de tener una falla en el proceso, mostrará una pantalla emergente donde muestre dicho error, en caso contrario mostrará que el contenido se guardó de forma correcta.

➔ metodosDeLectura (Elizabeth, Fernanda, Lirio y Ariadna): Esta clase realiza la división principal del análisis del código fuente, pues es la que lee el archivo con extensión .asc y, dependiendo del contenido de cada línea, realiza diversas operaciones: Reconoce directivas de ensamblador, líneas en blanco, comentarios, etiquetas, variables o constantes, además de mandar la línea para su análisis a su clase correspondiente si encuentra algún mnemónico de alguno de los 6 modos de direccionamiento.
Esta clase también proporciona dos métodos para el almacenamiento de variables o constantes en otra colección para su uso durante el programa, el análisis de las diferentes directivas de ensamblador que el programa debe soportar (ORG, END, FCB y EQU), además de analizar las líneas en blanco o con comentarios.

En esta clase también se crean los archivos con extensión .LST y .S19 cuando es necesario.

➔ Mnemonicos (Elizabeth, Fernanda, Lirio y Ariadna): Esta clase genera las hashTables que almacenan el OPCODE correspondiente a cada instrucción, divididas según el modo de direccionamiento, además de los métodos necesarios para leer y escribir dichas colecciones en sus archivos correspondientes. Para validar los operandos de los modos de direccionamiento que así lo requieran, también se implementaron colecciones con el número de bytes requeridos, ya sea por la instrucción completa, o por el operando de cierta instrucción particular.

➔ IndexadoX (Elizabeth): en esta clase se realizará el análisis de las instrucciones del método de direccionamiento del tipo IndexadoX, para esto se debe analizar cada una de las características planteadas por el profesor, diferenciando si se trata de una un mnemónico existente o no, además de analizar si el operando es del tamaño correcto, si se trata de un valor decimal, hexadecimal, de tipo carácter, constante o variable. En caso de no cumplir con las características de cada uno de los casos antes mencionados, el algoritmo mostrará el error correspondiente.

➔ IndexadoY (Elizabeth): en esta clase se realizará el análisis de las instrucciones del método de direccionamiento del tipo IndexadoY, para esto se debe analizar cada una de las características planteadas por el profesor, diferenciando si se trata de una un

mnemónico existente o no, además de analizar si el operando es del tamaño correcto, si se trata de un valor decimal, hexadecimal, de tipo carácter, constante o variable. En caso de no cumplir con las características de cada uno de los casos antes mencionados, el algoritmo mostrará el error correspondiente.

- ➔ Inherente (Fernanda): En esta clase se realizará el análisis de las instrucciones del método de direccionamiento del tipo inherente, una vez entra a esta clase se valida si se trata de una un mnemónico de este tipo, además de analizar si el mnemónico tiene operandos ya que de ser así arroja un error indicando que este modo de direccionamiento no lleva operandos, también lee los comentarios en la línea sin tomarlos en cuenta. En caso de no cumplir con las características de cada uno de los casos antes mencionados, el algoritmo mostrará el error correspondiente.

- ➔ Inmediato (Ariadna): En esta clase caen las instrucciones cuyos mnemónicos pudieran ser del modo de direccionamiento Inmediato, Directo o Extendido, debido a que comparten mnemónicos.

  La diferenciación del modo de direccionamiento Inmediato fue la más sencilla, pues bastó con encontrar un '#' en el la segunda palabra de la línea, es decir, el operando y que su mnemónico correspondiente perteneciera a la lista de este modo de direccionamiento para identificar la instrucción.

  La diferenciación de los modos Directo y Extendido fue un poco más complicada porque, si bien, hay mnemónicos exclusivos para cada modo de direccionamiento, muchos de los mnemónicos correspondientes a estos dos modos de direccionamiento son idénticos, por lo que no solo fue necesario encontrar el mnemónico en la lista correspondiente, sino que también fue necesario analizar la longitud del operando para comprobar si pertenecía al modo directo, al modo extendido o era un operando erróneo.

  Esta clase contiene un algoritmo que identifica si los operandos de las instrucciones, ya sean valores decimales, hexadecimales, caracteres, constantes o variables, son correctos para el mnemónico correspondiente o no, además de mostrar en la pantalla de la IDE el código objeto y el código fuente de las instrucciones correspondientes a los 3 modos de direccionamiento mencionadas previamente, diferenciando entre el OPCODE de los mnemónicos con los operandos por medio de colores.

- ➔ Relativo (Lirio):
- ➔ Var_Cons_Etiq (Lirio):
- ➔ VarConst (Lirio):
- ➔ Variables (¿Lirio o Elizabeth?):
- ➔ Excepciones (Fernanda y Lirio): En esta clase se realizará el análisis de las cuatro instrucciones conocidas como excepciones, las cuales admiten más de un operando, para esto primero entra a la clase, se compara el mnemónico con los cuatro de excepción, una vez separado se valida el número de palabras correspondiente al

número de operandos del mnemónico, se valida que sean dos o más, ya que en caso contrario no es un mnemónico de excepción e indicará el error correspondiente. Se valida que el mnemónico sólo tenga dos operandos para las instrucciones que requieren de dos o que tenga 3 operandos para el caso de las instrucciones que requieren 3 operandos, se valida también que el primer operando sea directo o indexado respecto a X o a Y para así definir el opcode del mnemónico, una vez validado el modo de direccionamiento del primer operando se busca en el archivo correspondiente que contiene la hash-table de dicho modo, posteriormente se valida que el segundo operando sea del modo de direccionamiento inmediato y que el tercero en caso de tenerlo sea una etiqueta para salto correspondiente al modo relativo; además de analizar si el valor del operando es un valor decimal o hexadecimal para convertirlo dado el caso. Para este tercer operando de modo relativo….. (Lirio).....

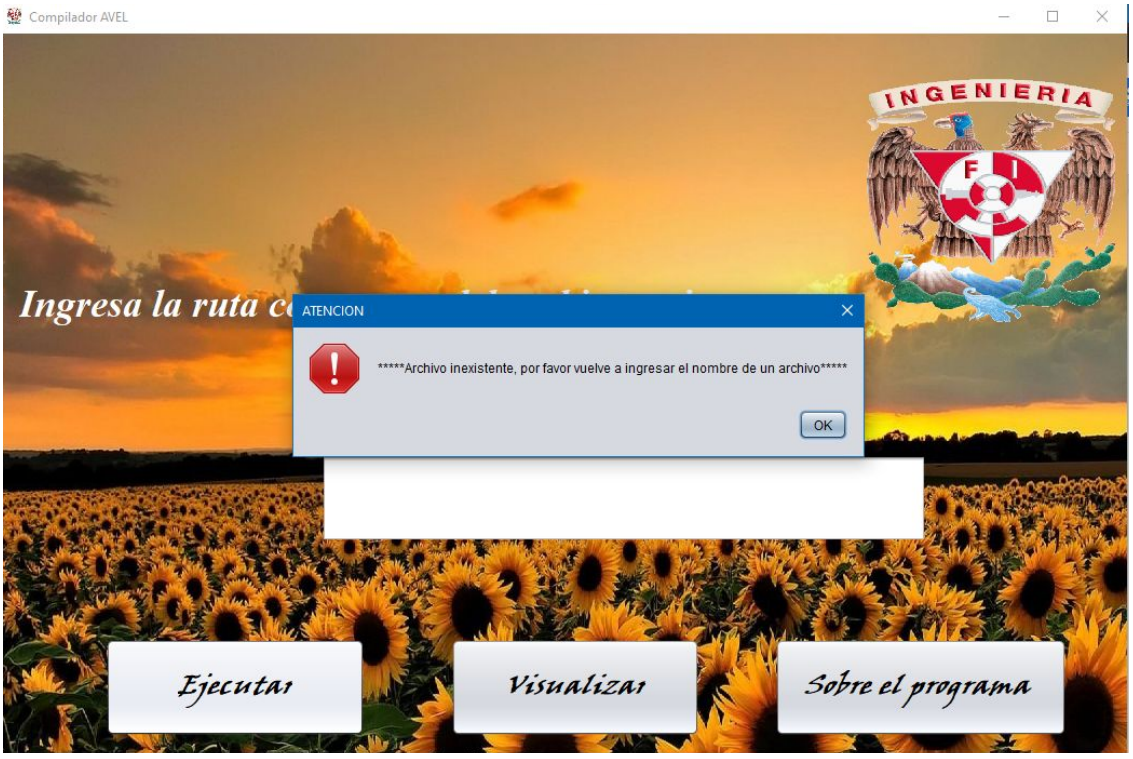
En caso de no cumplir con las características de cada uno de los casos antes mencionados, el algoritmo mostrará el error correspondiente.

Además, la realización de este reporte fue un trabajo en conjunto entre las 4 integrantes del equipo. Para dicha modalidad se usó un controlador de versiones llamado GitHub, lo cual nos facilitó compartir y actualizar los cambios realizados por cada uno de los integrantes.

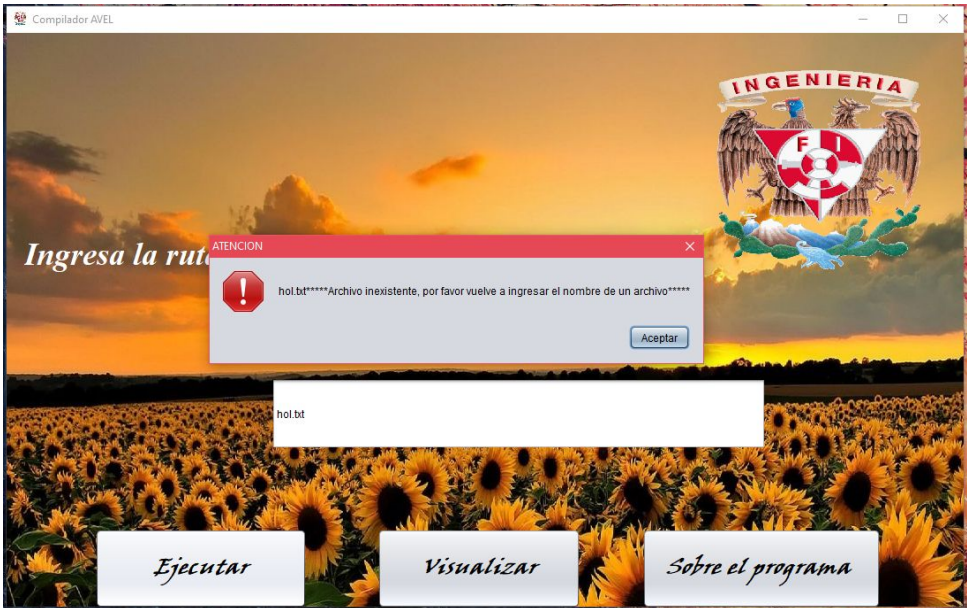
Evidencias de la funcionalidad del programa:

Al ejecutar el programa, se muestra la siguiente pantalla:

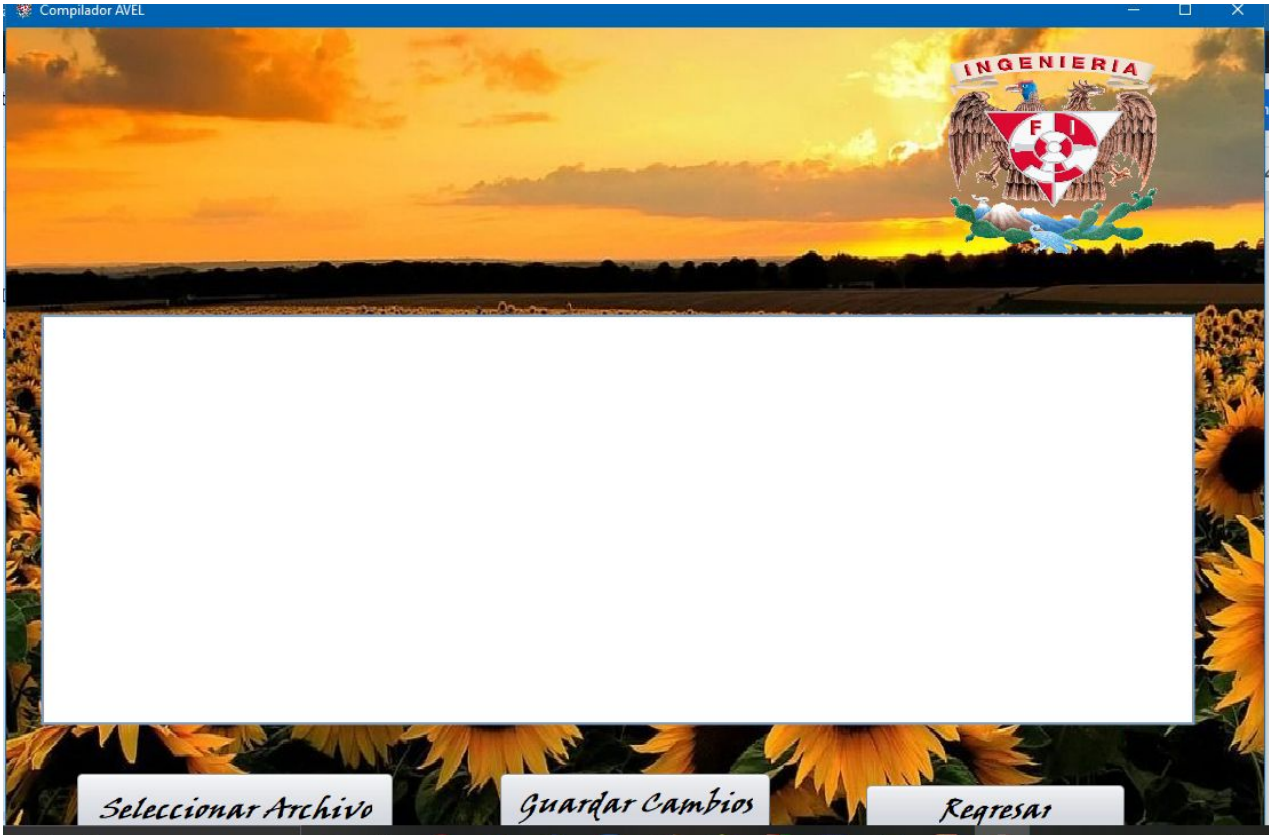
Si no se ingresa ningún valor en el cuadro de texto y se intenta ejecutar, aparece el siguiente mensaje:



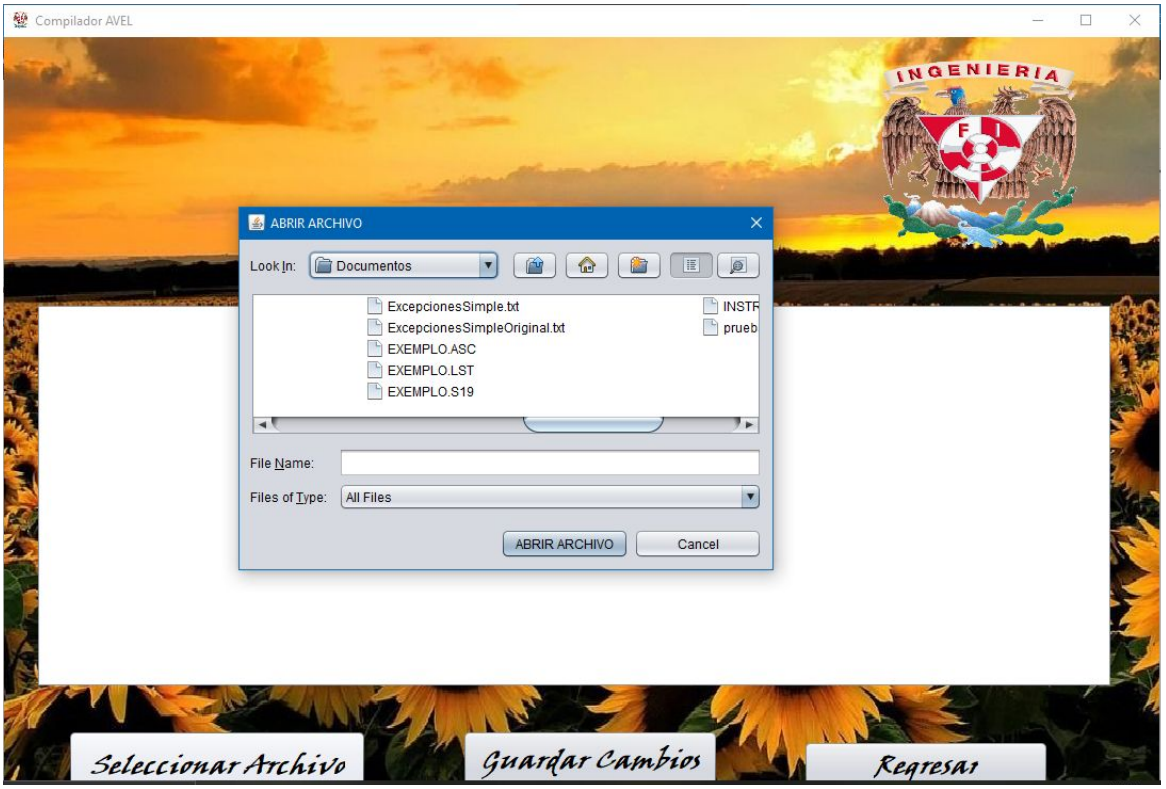
Si el archivo no se encuentra dentro de la carpeta del programa y no se especificó su ruta o si el archivo no existe, aparecerá el siguiente mensaje:



Al seleccionar la opción “visualizar”, se abre la siguiente ventana:

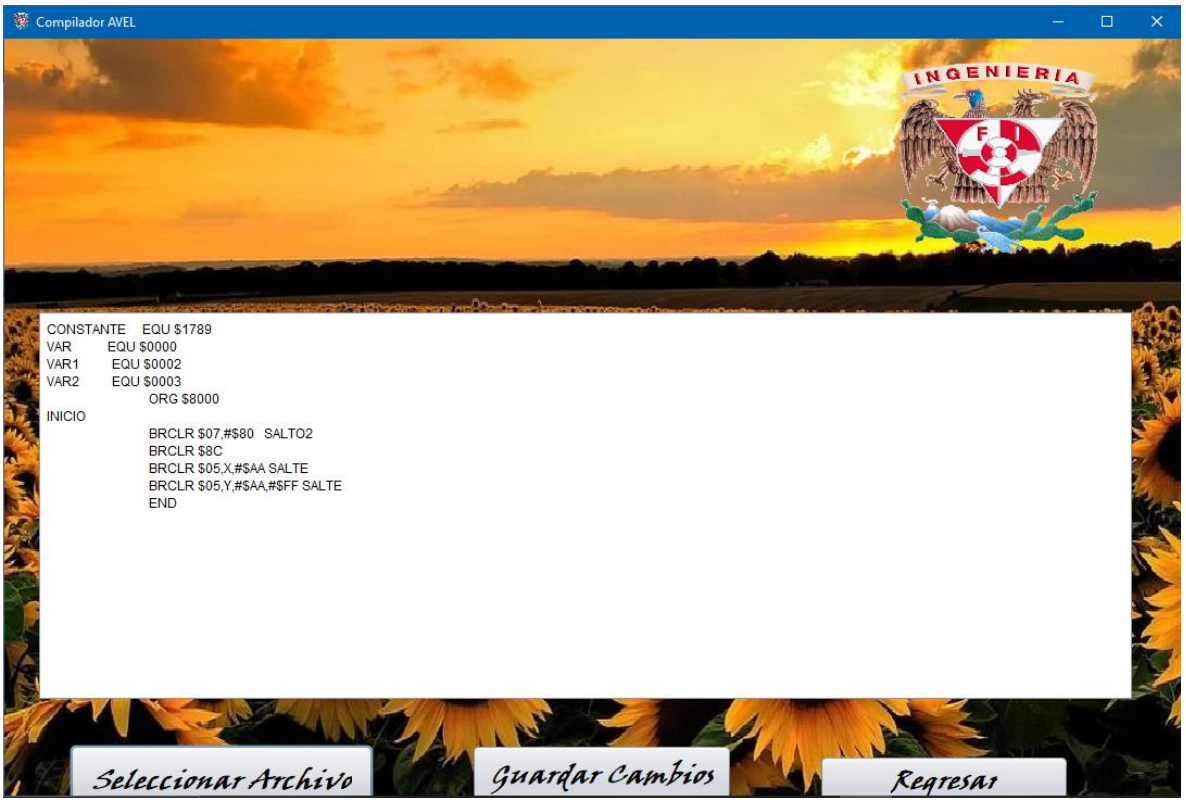


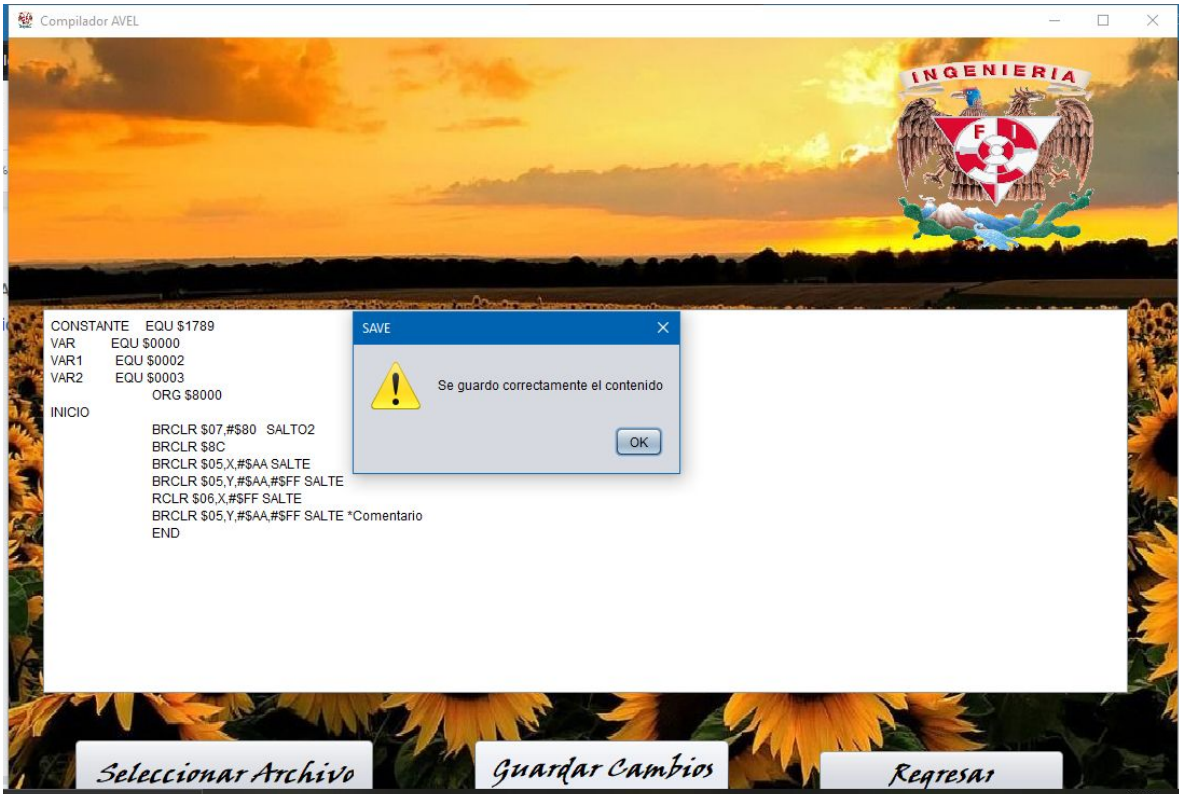
Donde podemos seleccionar un archivo de tipo .asc, .txt, .lst y .s19 para su visualización en la interfaz



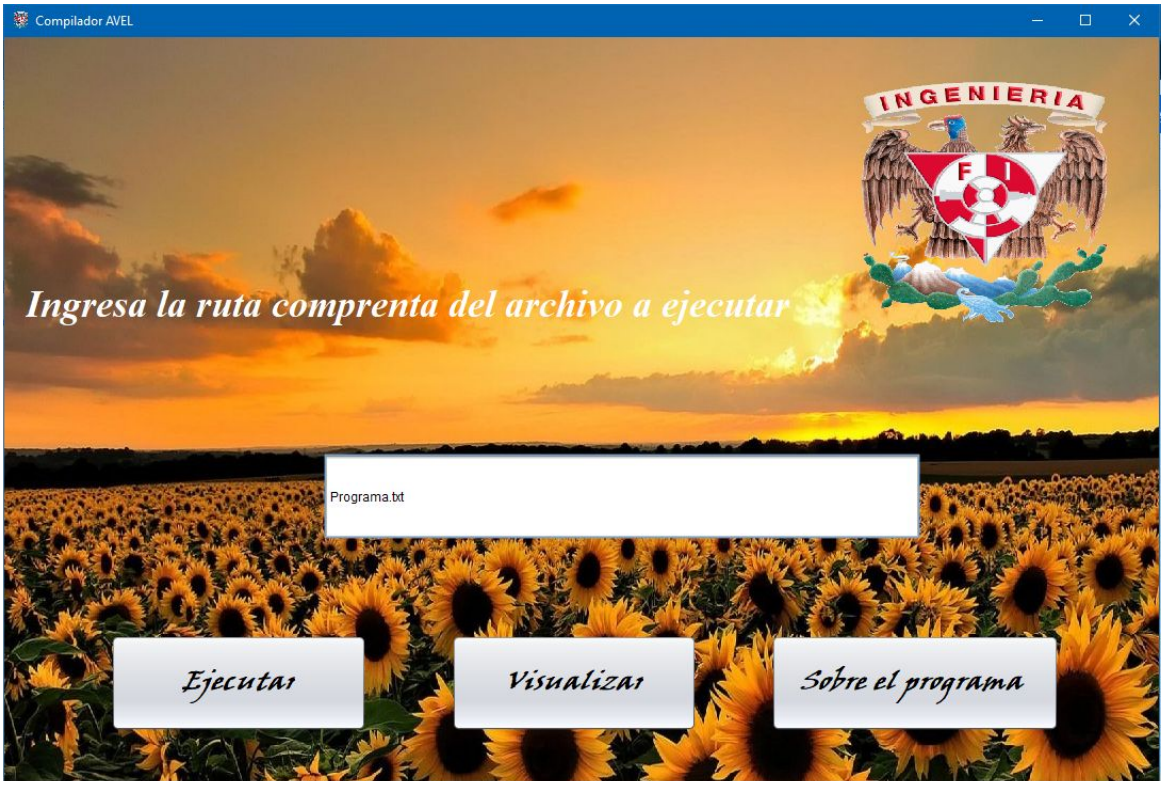
:

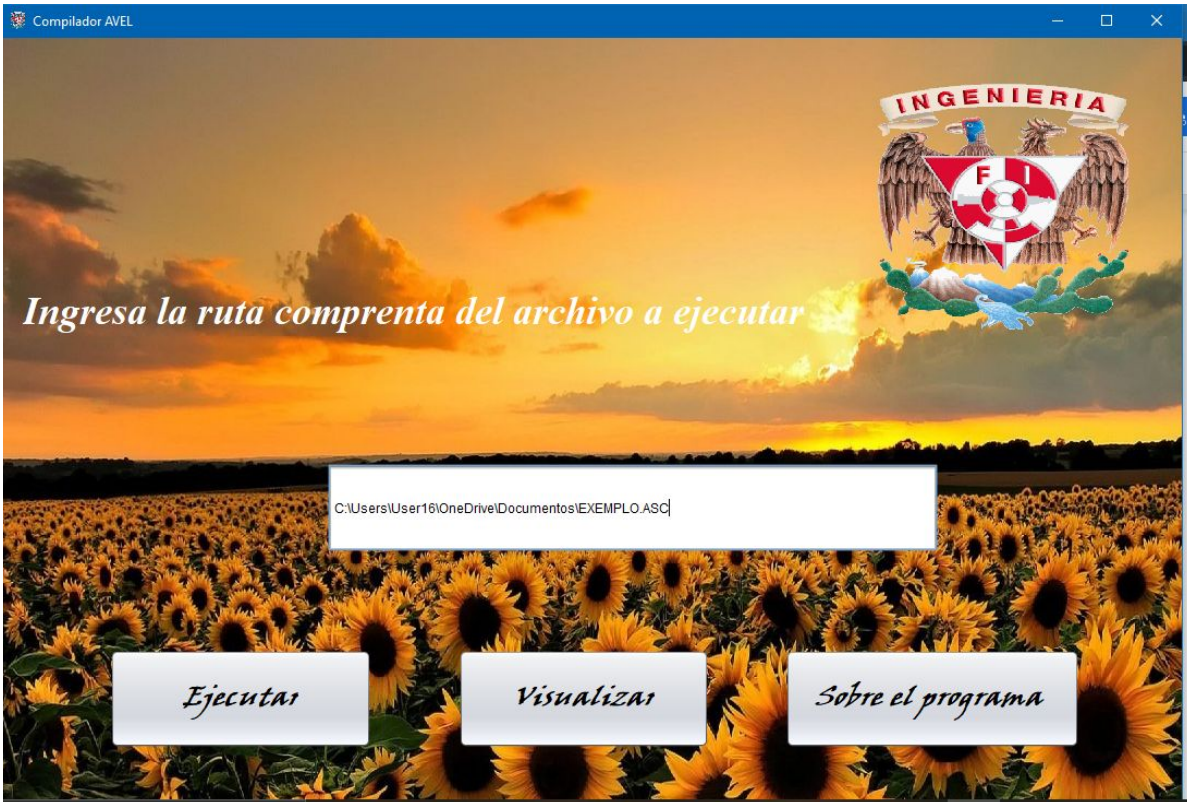
Se pueden modificar y guardar los datos fácilmente los tipos de archivos de tipo .asc, .txt y .lst :





De regreso en la pantalla original, se puede omitir la ruta de nuestro archivo con extensión .ASC si este se encuentra dentro de la carpeta del proyecto, en caso contrario debe especificarse la ruta completa:



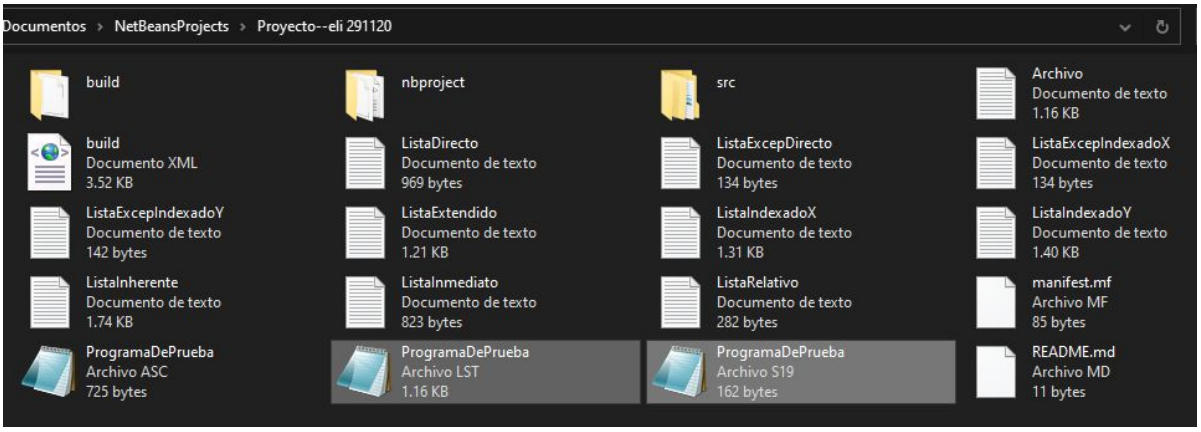


Al ejecutar un archivo sin errores, se muestra lo siguiente dentro de la IDE Netbeans:

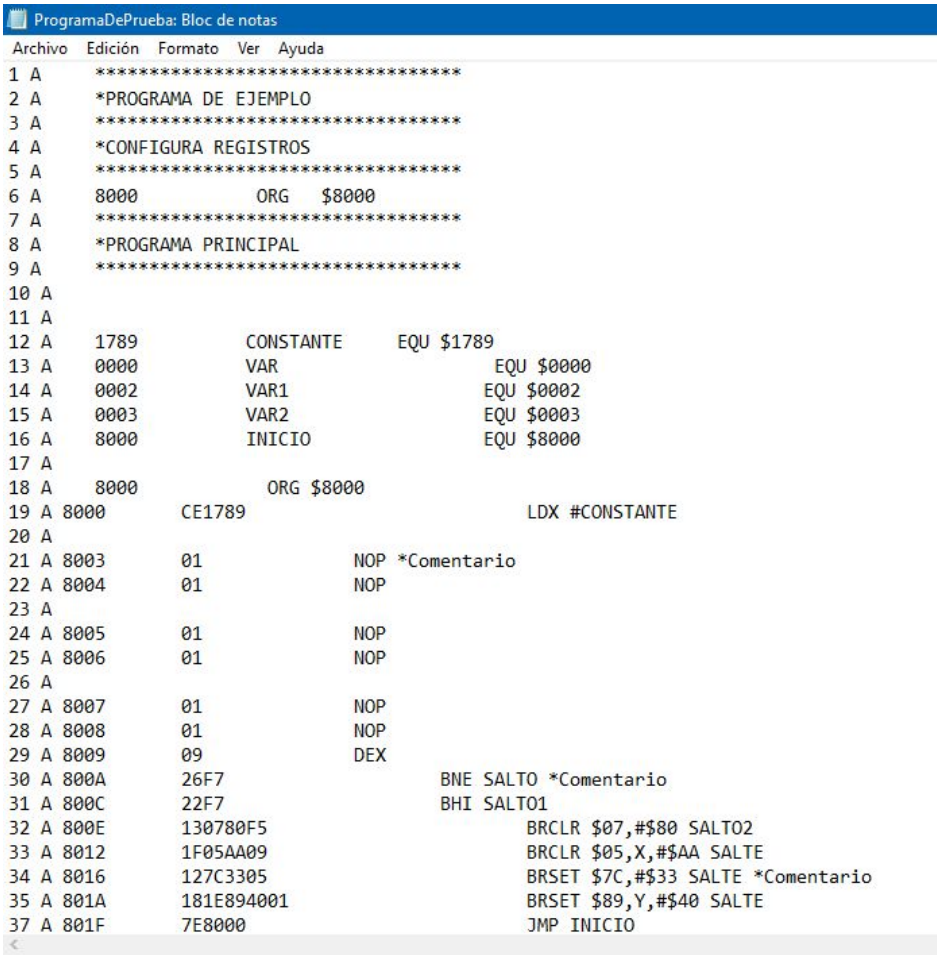
```
CE1789                      LDX #CONSTANTE
01                          NOP *Comentario
01                          NOP
01                          NOP
01                          NOP
01                          NOP
01                          NOP
09                          DEX

26F7                  BNE SALTO *Comentario

22F7                  BHI SALTO1
130780F5                    BRCLR $07,#$80 SALTO2
1F05AA09                    BRCLR $05,X,#$AA SALTE
127C3305                    BRSET $7C,#$33 SALTE *Comentario
181E894001                  BRSET $89,Y,#$40 SALTE

7E8000                      JMP INICIO
```

Se muestra el opcode junto con sus operandos de diferente color y subrayado para cada modo de direccionamiento.

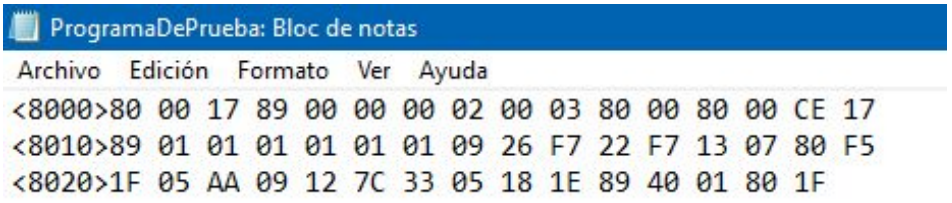
Lo anterior genera 2 archivos en la misma carpeta del proyecto, como se muestra a continuación:



Archivo con extensión .LST



Archivo con extensión .S19



Por el contrario, si se compila un archivo con errores, se muestra lo siguiente dentro de NetBeans:

Lo cual genera un solo archivo en la carpeta del proyecto:

Algunas consideraciones:

Las directivas de ensamblador (ORG, END, EQU y FCB) deben tener un espacio relativo al margen del código o el compilador no las detectará y puede generar errores.

Los archivos que se generan tienen el mismo nombre que el archivo compilado con extensión .asc, pero con diferente extensión.

La interfaz aparece en segundo plano al ejecutar el sistema, por lo que es necesario abrirla manualmente.

El tamaño de la interfaz puede variar dependiendo del dispositivo utilizado, se estableció un tamaño promedio de la pantalla.

El programa genera un archivo adicional llamado Archivo.txt que corresponde al archivo en la primera pasada, el cual nos sirve de utilidad para realizar una segunda pasada cuando se encontraron instrucciones del modo relativo, por lo que se recomienda no borrar el Archivo, se elimina automáticamente con cada ejecución.

Al ejecutar el programa se eliminarán archivos con el mismo nombre del archivo compilado, pero con extensiones .LST y .S19, esto con la finalidad de no generar errores, por lo que se recomienda tener esto en consideración para no perder información.

Algunas de las instrucciones deben llevar un espacio relativo al margen, se recomienda que este espacio sea con tabuladores o espacios, pero no ambos, es decir, que una instrucción contenga solo tabuladores o solo espacios, para evitar errores inesperados.

Al encontrar errores, aún así se muestran las localidades de memoria en el archivo con extensión .LST, sin embargo, estas localidades de memoria no son correctas debido a los errores, se recomienda hacer caso nulo a ellas.

El programa utiliza colores en NETBEANS para diferenciar de cada uno de los modos de direccionamiento el OPCODE correspondiente al mnemónico utilizado y el correspondiente al operando, los colores se detallan a continuación:

- ★ Subrayado blanco con letras grises: OPCODE de las variables o constantes.
- ★ Subrayado amarillo con letras negras: OPCODE de los mnemónicos correspondientes al modo de direccionamiento inmediato.
- ★ Subrayado blanco con letras amarillas: OPCODE de los operandos correspondientes a una instrucción del modo inmediato.
- ★ Subrayado cyan con letras negras: OPCODE de los mnemónicos correspondientes al modo de direccionamiento directo.
- ★ Subrayado blanco con letras cyan: OPCODE de los operandos correspondientes al modo directo.
- ★ Subrayado magenta con letras negras: OPCODE de los mnemónicos correspondientes al modo de direccionamiento extendido.
- ★ Subrayado blanco con letras magentas: OPCODE de los operandos correspondientes al modo extendido.
- ★ Subrayado azul con letras blancas: OPCODE de los mnemónicos correspondientes al modo de direccionamiento indexado, ya sea respecto a X o respecto a Y.

★ Subrayado blanco con letras azules: OPCODE de los operandos correspondientes al modo indexado.
★ Subrayado verde con letras negras: OPCODE de los mnemónicos correspondientes al modo de direccionamiento inherente.
★ Subrayado cyan con letras de color azul: OPCODE de los mnemónicos correspondientes al modo de direccionamiento relativo.
★ Subrayado azul con letras color cyan: OPCODE de los operandos correspondientes al modo relativo.

★ Subrayado amarillo con letras rojas: OPCODE de los mnemónicos “excepcionales”: BSET, BCLR, BRSET y BRCLR.
★ Subrayado rojo con letras amarillas: OPCODE de los operandos correspondientes a los mnemónicos “excepcionales”: BSET, BCLR, BRSET y BRCLR.
★ Subrayado blanco con letra roja: Errores que se identifican durante el programa.