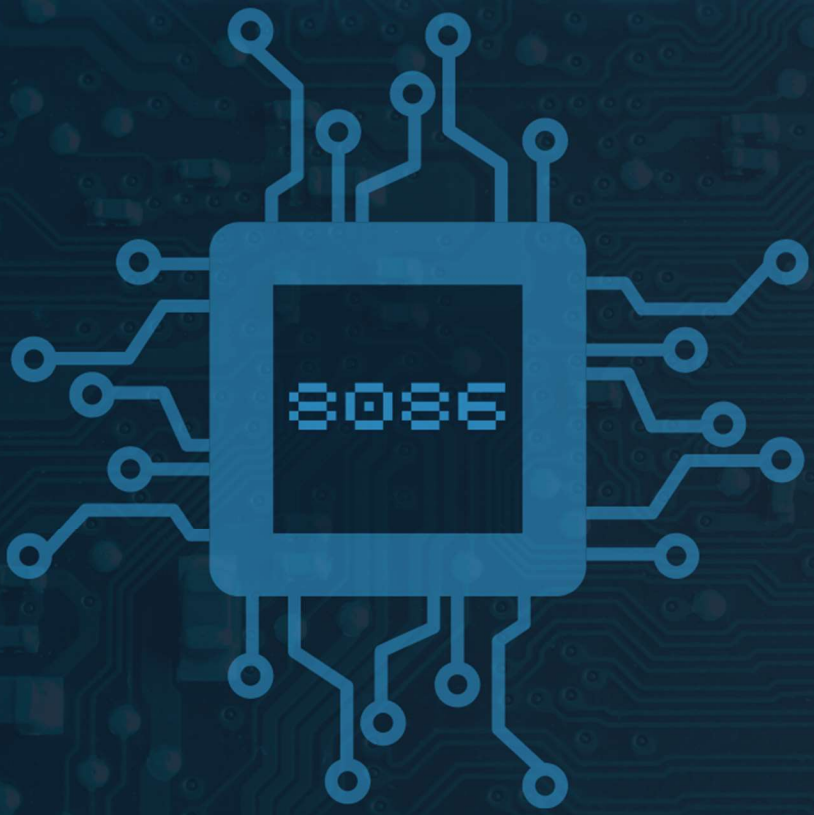
TRADUCTORES DE BAJO NIVEL



PROYECTO FINAL

UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE NEGOCIOS CAMPUS IV

INGENIERÍA EN DESARROLLO Y TECNOLOGÍAS DE SOFTWARE

TRADUCTORES DE BAJO NIVEL

PROYECTO FINAL

INTEGRANTES	MATRÍCULA
PEDRO OCTAVIO CULEBRO PRADO	B200227
PAULINO ENRIQUE NORIEGA VELAZQUEZ	B200150
SAMUEL SANCHEZ GUZMAN	B200079
JOSÉ EDUARDO OROZCO CARDENAS	B200071
EMILIA ZUÑIGA LOSADA	B200152

DOCENTE

MCC. VANESSA BENAVIDES GARCIA

GRADO Y GRUPO

5° “D”

TAPACHULA DE CÓRDOVA Y ORDOÑEZ, CHIAPAS A 11 DE NOVIEMBRE DE 2022

1. INDICE

2. INTRODUCCIÓN

Para la materia de traductores de bajo nivel, comprender cómo trabajan los computadores es una habilidad de interés a desarrollar, por lo que el aprendizaje guiado del funcionamiento de todos los conceptos manejados dentro de los programas con lenguaje ensamblador, contribuyen a concretar ideas concretas con respecto a todos los eventos que suceden en la parte más baja de estos mismos.

Se presenta el ensamblador como el lenguaje de la familia de los lenguajes ensambladores para los procesadores de la familia x86 introducida en abril de 1972, que incluye desde los procesadores Intel 8086 y 8088, pasando por los Pentium de Intel y los Athlon de AMD y llegando hasta los últimos procesadores x86 de estas compañías.

A partir de la asimilación de las características del lenguaje ensamblador, se comienzan a desarrollar diferentes actividades que, con diferentes tipos de complejidad, permiten tener un acercamiento importante hacia todas las capacidades de programar en este nivel.

Posterior a esto, se plantea, con la intención de dar por finalizada la asignatura, un proyecto de software por equipos, en el que se abordarán los temas previamente aprendidos en su totalidad. La elaboración de este programa viene acompañada de recursos audiovisuales y escritos, que complementan y explican todas las funciones implementadas, ya que en el formato de video es posible ver la interacción que existe entre el usuario y el software. Finalmente este trabajo será el encargado de expresar la parte escrita, en la que con la intención de detallar lo que en el código puede no ser comprendido sin un análisis profundo o conocimientos preestablecidos todos los recursos empleados de programación con lenguaje ensamblador x86, concluyendo así con el temario manejado en cuanto a las unidades de competencia correspondientes de la materia de Traductores de Bajo Nivel del quinto semestre grupo “D” llevadas por los integrantes de este equipo en pro de su formación como Ingenieros de Software.

3. SUSTENTO TEÓRICO

3.1 ENSAMBLADOR 8086

Para la realización tanto de las prácticas como del proyecto fue necesario elegir un tipo de ensamblador que fuera compatible con los procesadores, para ello se nos asignó el ensamblador 8086 el cual funciona como cualquier otro lenguaje de programación que cuenta con un conjunto de palabras que ayudan a dar indicaciones al ordenador sobre lo que tiene que hacer, las palabras del lenguaje ensamblador son nemotécnicos que representan el código máquina, lenguaje que entiende el procesador.

Para crear un programa ejecutable en lenguaje ensamblador es necesario realizar la serie de pasos, conocida como ciclo de desarrollo de programas, la cual consta de cuatro pasos edición (archivo del código fuente), ensamble (archivo del código objeto), enlace (archivo del programa ejecutable) y depuración, al final se genera un archivo del programa ejecutable. El ensamblador 8086 utiliza un formato de almacenamiento que se denomina Little envían que el byte menos significativo del dato se guardará en la parte baja de la memoria [imagen 1].

3.2 TRADUCTORES

Un traductor es un mediador entre dos entidades: emisoras y receptoras, los mediadores enmascaran la complejidad y heterogeneidad de los lenguajes. Un traductor convierte un lenguaje de entrada (código fuente) a una de salida (código objeto). Es una herramienta que convierte un lenguaje de alto nivel a código máquina.

3.2.1 INTERPRETES

Intérprete es un programa que convierte el lenguaje de alto nivel al formato de bits que es el lenguaje de máquina, por ello se dice que el intérprete traduce una línea a la vez y la ejecuta, pero no se genera ningún código objeto, por lo que cada vez que se deba ejecutar el programa, se debe interpretar primero.

3.2.1.1 VENTAJA

- Leen cada línea del programa fuente por separado y la ejecutan directamente en la plataforma sin traducir primero el código.
- Es un programa que va leyendo el código fuente de otro programa y lo va ejecutando según lo lee.
- Proceso de desarrollo sencillo (sobre todo en términos de depuración)

3.2.1.2 DESVENTAJA

- A diferencia de un compilador, no leen todo el código primero como un todo.
- Proceso de traducción poco eficiente y velocidad de ejecución lenta.

3.2.2 COMPILADORES

Se encarga de analizar el programa fuente y lo traduce a otro equivalente escrito en otro lenguaje (por ejemplo, en el lenguaje de la máquina) y traduce todo el código, su acción es equivalente a la de un traductor humano, que toma un libro y produce otro equivalente escrito en otra lengua.

3.2.2.1 VENTAJA

- Se compila una vez, se ejecuta n veces.
- En bucle, la compilación genera código equivalente al bucle, pero interpretándolo se traduce tantas veces una línea como veces se repite el bucle.
- El compilador tiene una visión global del programa, por lo que la información de mensajes de error es más detallada.
- Genera un ejecutable.

3.2.2.2 DESVENTAJA

- Un compilador consume bastante memoria.
- En principio eran más abundantes desde que los ordenadores tenían poca memoria.
- Permiten una mayor interactividad con el código en tiempo de desarrollo.

3.2.2.3 TIPOS

- Básicos: son de bajo nivel y su tarea se centra en dar nombres simbólicos a los parámetros o variables que puedan aparecer.
- Modular: Son considerados por algunos expertos como lenguajes de medio nivel al tener ciertas características que los acercan a los lenguajes de bajo nivel, pero teniendo, al mismo tiempo, ciertas cualidades que lo hacen un lenguaje más cercano al humano y, por tanto, de alto nivel.
- Modular de 32 bits (o de alto nivel): Este tipo apareció ante la necesidad que exigía la arquitectura de procesadores de 32 bits, teniendo además una compatibilidad trasera pues se permite el trabajo con arquitecturas de 16 bits.

3.3 ENSAMBLADOR

Ensamblador se refiere a un tipo de programa informático que se encarga de traducir un fichero fuente escrito en un lenguaje ensamblador, a un fichero objeto que contiene código máquina ejecutable directamente por la máquina para la que se ha generado. Ejemplos: MASM

3.3.1 COMPONENTES

- Alfanuméricos: son grupos de caracteres alfanuméricos que simbolizan las órdenes o tareas a realizar.
- Convierte a 0,1:
- Tiene su propio lenguaje
- Nemotécnicos: es un sistema sencillo utilizado para recordar una secuencia de datos, nombres, números, y en general para recordar listas de items que no pueden recordarse fácilmente.
- Subrutinas: son una porción de código que realiza una operación en base a un conjunto de valores dados como parámetros de forma independiente al resto del programa y que puede ser invocado desde cualquier lugar del código, incluso desde dentro de ella misma.
- Ensamblador por arquitectura del procesador

3.3.2 LENGUAJE ENSAMBLADOR

Un lenguaje ensamblador es un lenguaje de programación de bajo nivel que se utiliza para manipular las instrucciones internas de un dispositivo.

Este lenguaje se puede utilizar de esta manera debido fundamentalmente a que siempre el lenguaje ensamblador ofrece una correspondencia uno a uno entre sí y las instrucciones de código de máquina brutas del dispositivo que se está programando.

Cada línea de código escrito del lenguaje ensamblador es equivalente a una instrucción del dispositivo que se está programando. Esto permite que no deba ni interpretarse ni compilarse para que el hardware “entienda”.

En una explicación un poco más técnica, el lenguaje ensamblador, “Assembly” o “ASM” por su nombre en inglés, es una serie de mnemónicos que están diseñados para representar instrucciones básicas en electrónica.

3.3.3 APLICACIONES

El uso del lenguaje ensamblador le permite al programador indicarle al computador exactamente cómo llevar a cabo una tarea específica usando la menor cantidad de instrucciones. Aun cuando el código generado por los compiladores con opción de optimización es eficiente, la optimización manual puede resultar en una mejora sustancial en términos de rendimiento y consumo de memoria. El lenguaje ensamblador es usualmente utilizado en las siguientes circunstancias:

- Mejorar la eficiencia de una rutina específica que se ha transformado en un cuello de botella.
- Obtener acceso a funciones de bajo nivel del procesador para realizar tareas que no son soportadas por los lenguajes de alto nivel.
- Escribir manejadores de dispositivos para comunicarse directamente con hardware especial tales como tarjetas de red.
- Trabajar en ambientes con recursos limitados puede requerir el uso del lenguaje ensamblador pues el código ejecutable puede ser menor que el generado por el compilador.

3.4 CARGADORES

Un cargador es un programa que coloca en la memoria para su ejecución, el programa guardado en algún dispositivo de almacenamiento secundario. Un cargador es un programa del sistema que realiza la función de carga, pero muchos cargadores también incluyen relocalización y ligado. Algunos sistemas tienen un ligador para realizar las operaciones de enlaces y un cargador separado para manejar la relocalización y la carga. Los procesos de ensamblado y carga están íntimamente relacionados.

3.4.1 FUNCIONES

- Coloca el programa objeto en la memoria.
- En el programa fuente se definen las direcciones de memoria donde se va a cargar el programa objeto.
- Comprueba el resultado de cargar el programa objeto.
- Permite subir a memoria el programa objeto para poder ser ejecutado.
- Localiza la dirección de memoria.

3.1.2 TIPOS DE CARGADORES

- Bootstrap: Una vez, situado en la memoria del computador, cargará el programa de aplicación y los datos. Pero, previamente, se ha debido cargar el cargador en la memoria.
- Inicial: Indican a la computadora la forma de poner, dentro de la memoria principal unos datos que están guardados en un periférico de memoria externa (cinta, disco, etc.).
- Absoluto: Pone en memoria las instrucciones guardadas en sistemas externos. Independientemente de que sea un cargador inicial, o no sin dichas instrucciones se almacenan siempre en el mismo espacio de memoria (cada vez que se ejecuta el programa cargador) se dice que es un cargador absoluto.
- Reubicación: En ocasiones un mismo programa necesita ejecutarse en diferentes posiciones de memoria. Para esto la traducción debe estar realizada en forma adecuada, es decir no utilizando referencias absolutas a direcciones en memoria, sino referencias a una dirección especial llamada de reubicación.

- Ligador: Conocidos también como linker. Un linker es un término en inglés que significa montar. Montar un programa consiste en añadir al programa objeto obtenido a la traducción las rutinas externas a las que hace referencia dicho programa.

3.5 MACROPROCESADORES

Los ensambladores y compiladores cuentan con los denominados macroprocesadores, estos permiten definir una abreviatura para definir una parte del programa y utilizar dicha abreviatura cuantas veces sea requerido, lo que soluciona el problema de redundar el propio código al programador, en otras palabras, son las partes repetibles de un programa en ensamblador.

3.5.1 FUNCIÓN

La función principal y general de un macro procesador es la de una notación convencional para el programador, dicha macro representa en el código un grupo de instrucciones en el lenguaje de programación fuente.

Los macro procesadores reemplazan cada macro instrucción con el correspondiente grupo de instrucciones en el código fuente, esto es comúnmente llamado como expandir la macro, la función de las macroinstrucciones es la de permitir escribir al programador versiones cortas de un programa, y dejar que los detalles mecánicos sean controlados por el macroprocesador.

Para utilizar o implementar una macro en el código fuente del lenguaje ensamblador, primero se tiene que declarar antes que cualquier definición de segmento, en la declaración se le establece un nombre que se le asignara a esta y el conjunto de instrucciones que representará cuando vaya a ser empleada en la estructura de código principal. [Imagen 2]

3.5.2 COMPONENTES

Los componentes principales de una macro serian, como primero sería la asignación de un nombre para la macro, como el caso de la imagen anterior se le denomino como “UBICCAD”, posteriormente a eso se coloca la palabra reservada del ensamblador llamada “MACRO”.

Lo que va dentro de una macro seria únicamente las instrucciones de código que se mandaran a llamar y posteriormente a ejecutar en el segmento de código principal, por último, se coloca la palabra “ENDM” para dar por concluida nuestra macro.

3.5.3 TIPOS

Existen dos tipos de macros en ensamblador, las macros que reciben un parámetro del segmento de código y las que no que directamente se ejecutan sin recibir parámetro alguno cuando son llamadas, en el siguiente ejemplo se muestra la diferencia entre los dos tipos: las que reciben parámetros [Imagen 3] y los que no reciben parámetros [Imagen 4].

3.6 REGISTROS

Los registros son espacios físicos que forman parte dentro de un microprocesador con capacidades de 64 bits dependiendo del microprocesador que se use, dichos registros son una porción de memoria ultrarrápida útiles para desempeñar funciones como controlar las funciones que están en ejecución, llevar un buen manejo de redireccionamiento de memoria, etc.

3.6.1 CATEGORÍAS

- Registro de estado: El registro de estado es un registro que contiene información sobre el estado actual del procesador, dependiendo de la arquitectura este será un registro de 64 bits o un registro de 32 bits.
- Puntero: El puntero es el equivalente en C al acceso indirecto del ensamblador. Puede definirse el puntero como una variable que contiene la dirección de otra variable o constante (IP, SP, BP, SI, DI).

- Registro de segmento: Un registro de segmento contiene 16 bits de longitud y facilita un área de memoria para direccionamiento conocida como el segmento actual.
- Los registros de segmentos son los CS, DS, SS y ES.

- Visibles al usuario: Los registros visibles a usuarios serían los registros de estado que contienen una serie de banderas que indican distintas situaciones en las que se encuentra el procesador.
  Cada uno de estos registros se puede direccionar como un registro de 16 bits (AX, BX, CX, DX) o como un registro de 8 bits (AH, AL, BH, BL, CH, CL, DH, DL).
- Cada uno de los registros de 16 bits está formado por la concatenación de dos registros de 8 bits: AX = AH:AL, BX = BH:BL, CX = CH:CL y DX = DH:DL, donde el bit 0 del registro AH es el bit 8 del registro AX.
- REGISTROS EDO (high y low): AL, AH, BL, BH, CL, CH, DL, DH

3.6.2 TIPOS DE REGISTROS

La unidad de proceso central o mejor conocida como CPU tiene 14 registros internos, estos son los únicos registros que pueden usarse de modo dual (en 8 o 16 bits), los registros de la CPU son conocidos por sus nombres propios, que serán nombrados a continuación:

- AX (acumulador): El registro AX se utiliza para almacenar los resultados, ya sean de lectura o escritura desde o hacia los puertos.
- BX (registro base): El registro BX sirve como apuntador base o como un índice.
- CX (registro contador): El registro CX funciona como un contador en las operaciones de iteración, que se incrementa o decrementa automáticamente de acuerdo con el tipo de instrucción empleada.
- DX (registro de datos): El registro DX funge como puente para el acceso de los datos.
- DS (registro del segmento de datos): El registro DS tiene la función de actuar como policía donde se encuentran los datos, cualquier dato ya sea inicializado o no debe estar en este registro.

- ES (registro del segmento extra): El registro tiene el propósito general de permitir operaciones sobre cadenas, pero también puede ser una extensión del registro DS.
- SS (registro del segmento de pila): El registro SS tiene la tarea exclusiva de manejar la posición de memoria donde se encuentra la pila (stack), dicha estructura es usada para almacenar datos en forma temporal, tanto de un programa como de las operaciones internas de la computadora.
- CS (registro del segmento de código): En este registro se encuentra el código ejecutable de cada programa, el cual va directamente ligado a los diferentes modelos de memoria.
- BP (registro de apuntadores base): El registro BP se emplea para manipular la pila sin afectar al registro de segmentos SS.
- SI (registro índice fuente), DI (registro índice destino): Ambos registros son útiles para manejar bloques de cadenas en memoria, siendo el registro SI el índice fuente y el DI como el índice destino.
- SP (registro del apuntador de pila): El registro SP apunta hacia un área específica de memoria que sirve para almacenar datos bajo la estructura LIFO (último en entrar, primero en salir), conocida como pila (stack).
- IP (registro del apuntador de siguiente instrucción): El registro IP apunta a la siguiente instrucción que será ejecutada en el programa.
- F (registro de banderas): Es un registro en el que cada bit indica un estado particular del µP.

3.7 SEGMENTOS

Un segmento es un área especial en un programa que inicia en un límite de párrafo, estas áreas son secciones de código con nombres ya establecidos, los cuales separan el código del programa.

3.7.1 FUNCIONES

La funcion principal de los segmentos es la de separar el código escrito en lenguaje ensamblador para que sea reconocido por la computadora.

3.7.2 EXPLICAR LOS TIPOS DE SEGMENTOS

Existen cinco tipos de segmentos, los cuales son los siguientes:

- Model: Este segmento indica el modelo de memoria que se utilizara, este va dependiendo del tamaño que tendrá el programa, sus opciones son tiny, small, médium, compact, large y huge.

- Stack: En el segmento de pila donde se declara el tamaño de la pila que utilizara el programa en kilobytes.

- Data: En el segmento de datos se declaran variables o constantes que se utilizarán durante la ejecución de todo el programa. [Imagen 5]

- Code: El segmento de código contiene casi todo el código del programa, siendo recursos externos como macros o procedimientos la excepción dado que simplemente se importan o mandan a llamar como extensiones. [Imagen 6]

- Exit: El segmento exit se utiliza para indicar el final de un programa. [Imagen 7]

3.8 INTERRUPCIONES

Las interrupciones en ensamblador son instrucciones que se encargan de detener la ejecución de un programa para darle tiempo de procesador a otro proceso que sea más importante, esto permite que el procesador pueda llevar a cabo funciones especiales que ya se encuentra predefinidas por el ensamblador que se denominan servicios, que tiene varias funciones una de ellas es el despliegue información.

Estas forman parte del mecanismo más importante de la conexión del microprocesador con el exterior ya que es posible sincronizar la ejecución del programa con los acontecimientos externos, un ejemplo de ello es cuando requerimos algún dato del usuario, esto se realiza enviando un pedido de interrupción al procesador para llamar su atención.

3.8.1 INT 21H

La interrupción 21h nos permite leer de teclado, escribir en vídeo, escribir en impresora, leer y escribir de dispositivo auxiliar, además realizar cambios en el vector de interrupciones. Su función principal es detectar si se ha pulsado una tecla, de esta forma el registro AL los almacena en código ASCII.

- 09H: se encarga de imprimir una cadena de caracteres en la pantalla, estos son desplegados uno a uno desde la dirección indicada en el registro DS: DX hasta que encuentra el caracter "$" que sirve para determinar el final de la cadena. [Imagen 8]
- 4CH: esta rutina se encarga de finalizar el programa y devuelve el control del sistema, esto se agrega al final el programa. [Imagen 9]
- 01H: lee un carácter del teclado y también lo puede mostrar en pantalla, su registro de retorno es AL, el registro que lee lo guarda en hexadecimal y se guarda en AL.
- 02H: despliega un carácter a la pantalla. No tiene un registro de retorno, el carácter que despliega tiene un código hexadecimal correspondiente al valor que se ha almacenado en el registro DL.

3.8.2 INT 10H

Es una interrupción que tiene varias funciones y todas ayudan en el control de la entrada y salida de imagen o de video, para tener acceso a todas estas funciones es a través del registro AH. [Imagen 10]

- 02: esta función ayuda a colocar el cursor en la pantalla en la posición que se le indique usando coordenadas a través del registro DH y DL, para ello es importante saber el tamaño de la pantalla y poder asignar una ubicación.
- 06: Desplaza hacia arriba un número determinado de líneas en la ventana especificada mediante los registros CX y DX. Las líneas desplazadas, quedan vacías, rellenándose con blancos. El color utilizado en estas líneas vacías se indica mediante el registro BH.

3.9 PALABRAS RESERVADAS

Ciertas palabras en lenguaje ensamblador se encuentran reservadas únicamente para propósitos propios del lenguaje y son usadas solo bajo condiciones específicas y no se pueden emplear utilizar con oros propósitos como nombre de variables. Algunas de las categorías incluyen: instrucciones, directivas que se emplean para proporcionar comandos al ensamblador, operadores que se utilizan en expresiones y símbolos predefinidos. El uso de una palabra reservada para un propósito equivocado provoca que el ensamblador genere un mensaje de error. Algunas de las palabras reservadas con las que nos podemos encontrar son:

- SEGMENT: contiene instrucciones de máquina que se ejecutan, es la primera instrucción ejecutable, así mismo inicia el segmento. Redirecciona el segmento de código y necesita 64 bits.
- DB (define byte): reserva la memoria para los datos de tipo byte (8 bytes) o bien para variables.
- EDNS: finaliza el programa.
- $: indica que la cadena ha finalizado.
- ASSUME: Le dice al ensamblador que segmento va a direccionar cada uno de los registros de segmento. No inicializa los registros del segmento solo conduce al compilador al segmento de pila, datos.
- MOV: mueve o transfiere un byte o una palabra desde el operando fuente al destino, el dato se copa y no desaparece del operando fuente.
- OFFSET: desplazamiento de la variable o etiqueta de la posición desde el principio del segmento hasta la expresión indicada.
- INT: indica que se está solicitando el uso de las interrupciones.
- LEA: carga la dirección efectiva, transfiere el desplazamiento del operando fuente al destino.
- SUB: se utiliza para la sustracción, resta el operando fuente del destino.
- ADD: suma de operandos guarda el operando destino.
- MUL: multiplicación sin signo, en caso de hacer multiplicaciones de 8 bits el resultado se guarda en AX.

- DIV: división sin signo, si es de 8 bits toma como dividento a 16 bits de AX y el residuo de guarda en AH.

3.10 SALTOS

La definición que corresponde a los saltos, es la de instrucciones que permiten al programador cambiar el orden de ejecución del programa según sea necesario; dentro de ensamblador existen dos tipos de salto principales: condicionales e incondicionales.

3.10.1 TIPOS DE SALTOS

Para estas instrucciones existen subdivisiones que ayudan a comprender el funcionamiento de cada una según el grupo al que pertenecen.

Comenzando por los saltos incondicionales, que se utilizan mediante la orden JMP, la cual transfiere el control a la línea especificada después de la palabra JMP, misma que puede ser un valor directo o una etiqueta. [Imagen 11]

También se puede contar como un salto incondicional la instrucción CALL, la cual llama un procedimiento y al terminar devuelve el control a la línea siguiente de donde se inició la llamada a procedimiento. [Imagen 12]

La siguiente categoría, se refiere a la de los saltos condicionales, que son los que transfieren el control del programa a la ubicación que se les dé como parámetro si al hacer una comparación se cumple la condición establecida en el salto, los saltos condicionales son los siguientes:

- JA (Jump if Above): Salta cuando el valor es superior, su condición es equivalente al salto JNBE (Jump if Not Below or Equal). [Imagen 13]
- JAE (Jump if Above or Equal): Salta cuando el valor es superior o igual, su condición es equivalente al salto JNB (Jump if Not Below). [Imagen 14]
- JB (Jump if Below): Salta cuando el valor es menor, su condición es equivalente al salto JNAE (Jump if Not Above or Equal). [Imagen 15]
- JBE (Jump if Below or Equal): Salta cuando el valor es menor o igual, su condición es equivalente al salto JNA (Jump if Not Above). [Imagen 16]
- JE (Jump if Equal): Salta cuando el valor es igual. [Imagen 17]

- JZ (Jump if Zero): Salta cuando el valor es cero. [Imagen 18]
- JNE (Jump if Not Equal): Salta cuando el valor no es igual. [Imagen 19]
- JNZ (Jump if Not Zero): Salta cuando el valor no es cero. [Imagen 20]
- JG (Jump if Greater): Salta cuando el valor es mayor, su condición es equivalente al salto JNLE (Jump if Not Less or Equal). [Imagen 21]
- JGE (Jump if Greater or Equal): Salta cuando el valor es mayor o igual, su condición es equivalente al salto JNL (Jump if Not Less). [Imagen 22]
- JL (Jump if Less): Salta cuando el valor es menor, su condición es equivalente al salto JNGE (Jump if Not Greater or Equal). [Imagen 23]
- JLE (Jump if Less or Equal): Salta cuando el valor es menor o igual, su condición es equivalente al salto JNG (Jump if Not Greater). [Imagen 24]

3.10.3 COMPARACION

Correspondiente a una de las instrucciones más recurrentes en conjunto con los saltos, el CMP es la instrucción que se incluye entre las instrucciones de dos operandos cuyo objetivo es comparar dos operandos por medio de una resta, antes de usar un salto condicional. [Imagen 25]

3.11 ARCHIVOS

Otra parte importante a destacar tanto dentro de los diferentes ensambles previos como del realizado para el proyecto, es el manejo que se le da a los archivos, por lo que se presenta la siguiente definición:

Un archivo informático está identificado por un nombre y una descripción, el cual almacena información en formato binario (es decir ceros y unos). En lenguajes de alto nivel manejan los grupos de información (archivos), escondiendo la complejidad sobre el manejo y compilación de los mismos.

En lenguajes de alto nivel la manipulación de archivos se reduce a tareas simples, por ejemplo, creación, lectura, escritura. En lenguaje ensamblador, la manipulación de archivos requiere de mayor detalle.

3.11.1 FUNCIONES

Existen dos formas de operar los archivos dentro del lenguaje ensamblador, a continuación, se detallan y se explican las ventajas y desventajas de cada una. La primera y más antigua se llama “bloque de control de archivo” (FCB: File Control Block). La segunda y más nueva se le dice metodología Handles o canales de comunicación (Manejadores de archivo).

3.11.1 3CH CREAR O TRUNCA UN ARCHIVO

- AH=3CH
- CX= Atributo del Archivo
- DS:DX
- Registros de Regreso: La bandera de acarreo (CF):
- CF= 0, todo estuvo bien y AX obtiene el numero Handle para el archivo.
- CF= 1, ocurrio un error y AX obtiene el código de error
- AX=03H, ruta no encontrada
- AX=04H, no hay una handle disponible para asignar
- AX=05H, acceso negado

3.11.2 40H ESCRIBE A UN ARCHIVO

Escribe a un archivo o Dispositivo una cierta cantidad de byte, a partir de un buffer designado

- AH=40H
- BX= Handle Asignado
- CX= Cantidad de byte a ser escritos en el archivo
- DS:DX = Apuntador a buffer de datos
- Registros de Regreso: La bandera de acarreo (CF):
- CF= 0, todo estuvo bien y AX obtiene el número de bytes escritos.
- CF= 1, ocurrió un error y AX obtiene el código de error
- AX=05H, acceso negado
- AX=06H, Handle es errado

3.11.3 3EH CIERRA A UN ARCHIVO DESIGNADO

- AH=3EH
- BX= Handle Asignado
- Registros de Regreso: La bandera de acarreo (CF):
- CF= 0, todo estuvo bien y AX obtiene el número de bytes escritos.
- CF= 1, ocurrio un error y AX obtiene el código de error
- AX=06H, Handle es errado

3.11.2 VENTAJAS DE MANIPULACIÓN DE ARCHIVOS A TRAVÉS DE FCB

- Permiten tener número ilimitados de archivos abiertos.
- Los FCB se usan para crear volumen en los dispositivos de almacenamiento.

3.11.3 VENTAJAS DE MANIPULACIÓN DE ARCHIVOS A TRAVÉS DE HANDLES

- Simplicidad para manejar errores.
- Funciones de Handle pueden permanecer en las versiones actuales de S.O
- Toman ventaja de la estructura de directorio del S.O.
- Permite al programador centrarse en la programación pura sin ocuparse de tantos detalles.

Es importante también mencionar que debido a que FCB no permite más que nombres de archivo de 8 caracteres máximo y no servía para almacenar rutas a archivo incluyendo directorios, fue sustituido entonces por los Maneja de archivos. [Imagen 26]

3.11.3.1 FORMA DE OPERAR EN LOS ARCHIVOS CON LOS HANDLER

- Las funciones Básicas para el manejo de archivos con la Int 21H son:
- Función 3CH: Se utiliza para crear un archivo
- Función 40H: Se utiliza para Escribir sobre un archivo
- Función 3EH: Se utiliza para cerrar un archivo
- Operar con función: 3CH•La Función 3CH Crear o Trunca un archivo

- AH=3CH
- CX= Atributo del Archivo
- DS: DX

- Registros de Regreso: La bandera de acarreo (CF):
  - CF= 0, todo estuvo bien y AX obtiene el número Handle para el archivo.
  - CF= 1, ocurrió un error y AX obtiene el código de error
  - AX=03H, ruta no encontrada
  - AX=04H, no hay una handle disponible para asignar
  - AX=05H, acceso negado [Imagen 27]

3.11.4 ATRIBUTOS DE LOS ARCHIVOS

Para comprender el funcionamiento de cada atributo, se muestra a continuación una tabla con cada uno de los valores. [Imagen 28]

- La Función 40H Escribe a un archivo o Dispositivo una cierta cantidad de byte, a partir de un buffer designado
- AH=40H
- BX= Handle Asignado
- CX= Cantidad de byte a ser escritos en el archivo
- DS: DX = Apuntador a buffer de datos

Registros de Regreso: La bandera de acarreo (CF):

- CF= 0, todo estuvo bien y AX obtiene el número de bytes escritos.
- CF= 1, ocurrió un error y AX obtiene el código de error
- AX=05H, acceso negado
- AX=06H, Handle es errado [Imagen 29]

3.12 CICLOS

Como en cualquier otro lenguaje de programación, hay ocasiones en las que es necesario hacer que el programa no siga una secuencia lineal, sino que repita varias

veces una misma instrucción o bloque de instrucciones antes de continuar con el resto del programa, es para esto que se utilizan los ciclos.

Existen 5 tipos de ciclos predefinidos en ensamblador, aunque también se pueden crear ciclos personalizados por medio de instrucciones de salto (los cuales ya fueron mencionado).

3.12.1 TIPOS DE CICLOS

3.12.1.1 LOOP

Esta función decrementa el valor del registro contador CX, si el valor contenido en CX es cero ejecuta la siguiente instrucción, en caso contrario transfiere el control a la ubicación definida por la etiqueta utilizada al momento de declarar el ciclo.

Ejemplo:

- mov cx,25: número de veces que se repetirá el ciclo, en este caso 25.
- ciclo: Etiqueta que se utilizará como referencia para el ciclo loop.
- int 21h: Instrucción contenida dentro del ciclo (puede contener más de una instrucción).
- loop: Ciclo loop que transferirá el control a la línea de la etiqueta ciclo en caso de que CX no sea cero.

3.12.1.2 LOOPE

Esta función decrementa el valor del registro contador CX, si el valor contenido en CX es cero y ZF es diferente de uno ejecuta la siguiente instrucción, en caso contrario transfiere el control a la ubicación definida por la etiqueta utilizada al momento de declarar el ciclo.

Ejemplo:

- ciclo: Etiqueta que se utilizará como referencia para el ciclo loope.
- int 21h: Instrucción contenida dentro del ciclo (puede contener más de una instrucción).
- loope: Ciclo loope que transferirá el control a la línea de la etiqueta ciclo en caso de que CX no sea cero y ZF sea igual a uno.

3.12.1.3 LOOPNE

Esta función decrementa el valor del registro contador CX, si el valor contenido en CX es cero y ZF es diferente de cero ejecuta la siguiente instrucción, en caso

contrario transfiere el control a la ubicación definida por la etiqueta utilizada al momento de declarar el ciclo, esta es la operación contraria a loope.

Ejemplo:

- ciclo: Etiqueta que se utilizará como referencia para el ciclo loopne.
- int 21h: Instrucción contenida dentro del ciclo (puede contener más de una instrucción).
- loopne: Ciclo loopne que transferirá el control a la línea de la etiqueta ciclo en caso de que CX no sea cero y ZF sea igual a cero.

3.12.1.4 LOOPZ

Esta función decrementa el valor del registro contador CX, si el valor contenido en CX es cero y ZF es diferente de uno ejecuta la siguiente instrucción, en caso contrario transfiere el control a la ubicación definida por la etiqueta utilizada al momento de declarar el ciclo.

Ejemplo:

- ciclo: Etiqueta que se utilizará como referencia para el ciclo loopz.
- int 21h: Instrucción contenida dentro del ciclo (puede contener más de una instrucción).
- loopz: Ciclo loopz que transferirá el control a la línea de la etiqueta ciclo en caso de que CX no sea cero y ZF sea igual a uno.

3.12.1.5 LOOPNZ

Esta función decrementa el valor del registro contador CX, si el valor contenido en CX es cero y ZF es diferente de cero ejecuta la siguiente instrucción, en caso contrario transfiere el control a la ubicación definida por la etiqueta utilizada al momento de declarar el ciclo, esta es la operación contraria a loopz.

Ejemplo:

- ciclo: Etiqueta que se utilizará como referencia para el ciclo loopnz.
- int 21h: Instrucción contenida dentro del ciclo.
- loopnz: Ciclo loopnz que transferirá el control a la línea de la etiqueta ciclo en caso de que CX no sea cero y ZF sea igual a cero.

3.12.1.6 CICLOS CONDICIONALES

En ensamblador no existen de forma predefinida estos ciclos, pero pueden crearse haciendo uso de los saltos incondicionales (que ya fueron mencionados), generando ciclos que se repetirán hasta que se cumpla la condición definida por el programador.

Ejemplo:

- mov al, 0: Asigna el valor cero al registro al.
- ciclo: Etiqueta a la que se hará referencia para el ciclo condicional.
- INC al: Aumenta en 1 el valor del registro al.
- CMP al, bl: Comparación entre el valor almacenado en al y el almacenado en bl.
- JL ciclo: Instrucción que indica que el flujo del programa continuara desde la ubicación de la etiqueta ciclo si el valor de al es menor al de bl.

3.12.1.7 INCREMENTO Y DECREMENTO (USADO EN LOS CICLOS)

En ensamblador existen dos instrucciones que cumplen con el propósito de aumentar o reducir el valor contenido dentro de un registro. [Imagen 30]

- INC: Incrementa en uno el valor contenido dentro del registro que se le dé como parámetro.
- INC al: Aumenta en 1 el valor del registro al.
- DEC: Reduce en uno el valor contenido dentro del registro que se le dé como parámetro.
- DEC al: Reduce en 1 el valor del registro al.

3.13 PROCEDIMIENTOS

Un procedimiento no es más que una colección de instrucciones que realizan una tarea específica: Sumar dos valores, desplegar un carácter en la pantalla, leer un carácter del teclado, inicializar un puerto, etc.

Dependiendo de su extensión y complejidad, un programa puede contener uno, algunos o inclusive cientos de procedimientos. Para emplear un procedimiento en un programa se requiere definir el procedimiento y llamarlo. Al definir a un procedimiento escribimos las instrucciones que contiene.

Al llamar al procedimiento transferimos el control del flujo del programa al procedimiento para que sus instrucciones se ejecuten.

3.13.1 DEFINICIÓN DE UN PROCEDIMIENTO

La sintaxis de la definición de un procedimiento es la siguiente:

nombreProc PROC [near/far]

Instrucciones a ejecutar

[ret]

nombre ENDP

- Las directivas proc y endp marcan el inicio y el final del procedimiento. No generan código.
- nomProc es el nombre del procedimiento y etiqueta la primera instrucción del procedimiento.
- Near/far depende si la operación implica un procedimiento cercano o lejano.
- La instrucción ret regresa al segmento donde fue invocado el procedimiento.

Al menos una de las proposiciones de un procedimiento es la instrucción ret, la cual se describe más adelante.

3.13.2 LLAMADA A UN PROCEDIMIENTO

La llamada a un procedimiento normalmente tiene la siguiente forma:

call nomProc

La instrucción call se describe más adelante. nomProc es el nombre que se le dio al procedimiento al definirlo

La instrucción CALL transfiere el control a un procedimiento llamado, y la instrucción RET regresa del procedimiento llamado al procedimiento original que hizo la llamada. RET debe ser la última instrucción en un procedimiento llamado.

- El código objeto particular que CALL y RET generan depende de si la operación implica un procedimiento NEAR (cercano) o un procedimiento FAR (lejano).

3.13.3 VENTAJAS DE LOS PROCEDIMIENTOS

La organización de un programa en procedimientos proporciona los siguientes beneficios:

1. Reduce la cantidad de código, ya que un procedimiento común puede ser llamado desde cualquier lugar en el segmento de código.
2. Fortalece la mejor organización del programa.
3. Facilita la depuración del programa, ya que los errores pueden ser aislados con mayor claridad.
4. Ayuda en el mantenimiento progresivo de programas, ya que los procedimientos son identificados de forma rápida para su modificación.

4.CONCLUSIONES

Luego de haber realizado los estudios necesarios para el desarrollo del proyecto orientado hacia el lenguaje ensamblador se logró cumplir con el objetivo general. Aunado a eso, como conclusión al presente proyecto, podemos decir que el lenguaje ensamblador es mucho más que un lenguaje de bajo nivel en el cual nosotros empleamos para crear programas informáticos, ya que dicho lenguaje se creó en base a instrucciones para sustituir al lenguaje maquina por uno similar al empleado por las personas, ahí radica su importancia.

La verdadera importancia del lenguaje ensamblador radica fundamentalmente en que en él se pueden llevar a cabo cualquier tipo de programa que en otros lenguajes de alto nivel no podrían, al igual que llegar a optimizar la memoria de nuestro dispositivo.

De esta manera se logró comprender como trabaja, opera y principalmente de cómo está compuesto internamente (a nivel de código) un compilador, igualmente fue posible conocer los fundamentos técnicos para la programación de los lenguajes de alto nivel, en tanto que se pudo también precisar las características operativas y de hardware optimizándolo para un mejor rendimiento.

También mencionar que el lenguaje ensamblador todavía es enseñado en los programas de ciencias de la computación y en ingeniería electrónica, aunque pocos programadores trabajan normalmente con este lenguaje, es útil para estudiar conceptos fundamentales, como aritmética binaria, asignación de memoria, procesamiento de interrupciones y diseño de compiladores, siendo de mucha utilidad debido a que el código de máquina es fácil de trasladar hacia lenguaje ensamblador para luego ser examinado en esta forma, aunque sigue siendo muy difícil trasladarlo a un lenguaje de alto nivel. Finalmente se resalta el trabajo en equipo ya que promovió la generación de ideas a partir de los diferentes puntos de vista y trajo consigo un trabajo que fusiona los conocimientos aprendidos a lo largo del curso y nos proporciona experiencia con respecto a las metodologías empleadas en la realización del mismo.

5.REFERENCIAS

mgonzalez, mgonzalez. (2020, 5 febrero). *Ensamblador 8086*. cs.buap. Recuperado 16 de octubre de 2022, de https://www.cs.buap.mx/~mgonzalez/asm_mododir2.pdf

sylabus, sylabus. (2019, 11 marzo). *Conceptos básicos de Lenguaje Ensamblador 8086*. hopelchen. Recuperado 16 de octubre de 2022, de https://hopelchen.tecnm.mx/principal/sylabus/fpdb/recursos/r87222.PDF

1.4 El concepto de interrupciones. (2018, 12 noviembre). Lenguajes de interfaz. Recuperado 16 de octubre de 2022, de https://ittlenguajesdeinterfaz.wordpress.com/1-4-el-concepto-de-interrupciones/

EL CONCEPTO DE INTERRUPCIONES. (s. f.). lenguaje-ensamblador. Recuperado 16 de octubre de 2022, de https://leo-yac.wixsite.com/lenguaje-ensamblador/el-concepto-de-interrupciones

Interrupciones y manejo de archivos. (s. f.). Recuperado 16 de octubre de 2022, de https://moisesrbb.tripod.com/unidad6.htm

Funciones del DOS. (s. f.). Recuperado 16 de octubre de 2022, de http://arantxa.ii.uam.es/%7Egdrivera/labetcii/int_dos.htm

Palabras reservadas en lenguaje ensamblador. (2013, 28 junio). Cursos gratis. Recuperado 16 de octubre de 2022, de https://conocimientosweb.net/dcmt/ficha15401.html

Registros (lenguaje ensamblador) - 1321 Palabras | Monografías Plus. (s. f.). Recuperado 15 de octubre de 2022, de https://www.monografias.com/docs/Registroslenguaje-ensamblador-PKYXJFYMZ

Registros de Lenguaje Ensamblador. (2022, 15 octubre). Recuperado 15 de octubre de 2022, de http://yesenializbethguerrerogarcia.blogspot.com/2017/03/registros-de-lenguaje-ensamblador.html

Macroprocesadores - 1281 Palabras | Monografías Plus. (s. f.). Recuperado 15 de octubre de 2022, de https://www.monografias.com/docs/Macroprocesadores-P3CNK5VFJ8GNY

Codificar en Lenguaje ensamblador y Registro de estado En este artículo explicamos Leer más. (2022, 14 mayo). Codifica.me | Desarrollo web | Programación. Recuperado

15 de octubre de 2022, de https://www.codifica.me/programacion-en-lenguaje-ensamblador-registro-de-estado/

Just a moment. . . (s. f.). Recuperado 31 de octubre de 2022, de https://underc0de.org/foro/asm/procedimientos-en-assembler-emu8086-(facil)/

Unidad 2. Programación Básica. (2020, 28 abril). NANONBLOGS. https://brandon22esquivel.wixsite.com/misitio/post/unidad-2-programaci%C3%B3n-b%C3%A1sica

Laureano, A. (2019, 13 marzo). Comparaciones, saltos, ciclos condicionales, incrementos y decrementos. https://pythonambrocioisaias.blogspot.com/2019/03/comparaciones-saltos-ciclos.html

Andres, G. C. J. (2021, 17 mayo). Editor y manejo de archivos (CRUD) Ensamblador 8086 [Vídeo]. YouTube. https://www.youtube.com/watch?v=yi5ioQwUnu8&feature=youtu.be

TRADUCTOR Y SU ESTRUCTURA QU ES. (2015, 9 febrero). Recuperado 16 de octubre de 2022, de https://slidetodoc.com/1-4-traductor-y-su-estructura-qu-es/

Compiladores. (s. f.). Recuperado 16 de octubre de 2022, de https://es.slideshare.net/fabianch78/compiladores-103523588

Rivas, A. (2022, 26 marzo). Ensamblador. Recuperado 4 de noviembre de 2022, de https://muytecnologicos.com/diccionario-tecnologico/ensamblador

Marker, G. (2022, 7 octubre). El lenguaje ensamblador. Recuperado 4 de noviembre de 2022, de https://www.tecnologia-informatica.com/el-lenguaje-ensamblador/

Cargadores. (2015, 23 marzo). Recuperado 4 de noviembre de 2022, de https://programaciondesistemas2015.wordpress.com/cargadores/

6.ANEXOS

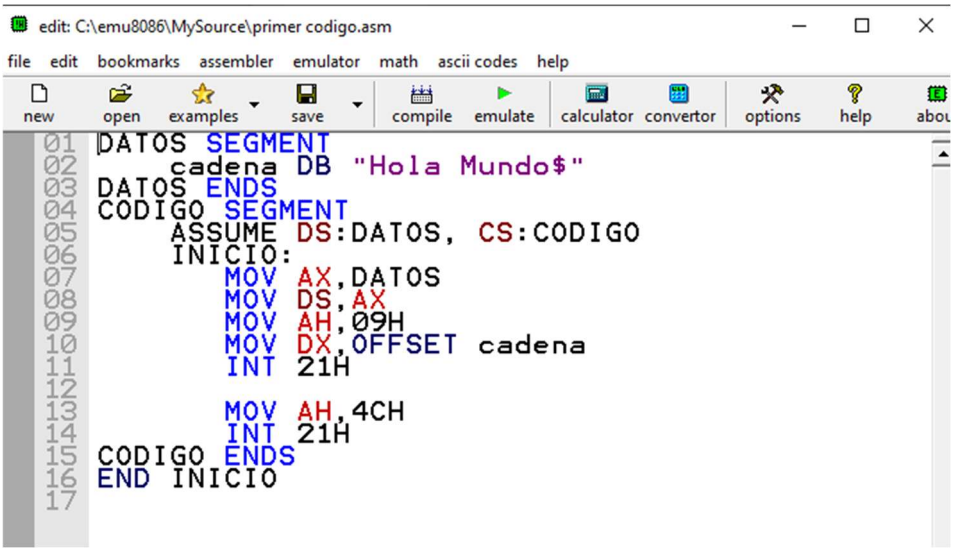


Imagen 1. Código en ensamblador

```
2
3   ;macros para imprimir y unicar cadena
4   UBICCAD MACRO A1, A2, A3
5      MOV AH, 02H
6      MOV BH, 0H   ;indica la pagina
7      MOV DH, A1
8      MOV DL, A2
9      INT 10H
10     MOV AH,09H
11     MOV DX, OFFSET A3
12     INT 21H
13  ENDM ;fin macro
14
```

Imagen 2. Macro declarada

```
17
18    ;Funcion para limpiar pantalla
19    LIMPIAR MACRO LIM1
20        MOV AH, 06H
21        MOV AL, 00H
22        MOV BH, LIM1      ;pantall-letra
23        MOV CX, 0000H
24        MOV DX, 184fH
25        INT 10H
26    ENDM ;fin macro
27
28
```

Imagen 3. Macro que recibe parámetros

```
29
30   COMPARACION MACRO
31      LIMPIAR 10000000B
32        COLOREAR 00001111B, 00D, 00D, 01D, 79D ;negro
33        COLOREAR 01100000B, 02D, 00D, 24D, 19D ;cafe
34        COLOREAR 01110000B, 02D, 16D, 24D, 79D ;blanco
35
36      UBICCAD 01D, 20D, CADENAOP2
37      UBICCAD 03D, 16D, CADENA1OP2
38      PEDIRN DIG1OP2
39        UBICCAD 04D, 16D, CADENA2OP2
40      PEDIRN DIG2OP2
41      ;MOV DIG2OP2, AL
42   ENDM
43
```

Imagen 4. Macro que no recibe parámetros

```
17
18  DATOS SEGMENT
19     ;MENU PRINCIPAL
20     CADENA DB "°  °  ° BIENVENIDO °  °  ° $"
21     CADENA1 DB "1.- PRESENTACION DEL EQUIPO$"
22     CADENA2 DB "2.- COMPARACION DE DOS NUMEROS$"
23     CADENA3 DB "3.- OPERACIONES CON ARCHIVOS$"
24     CADENA4 DB "4.- REGALOS NAVIDENOS$"
25     CADENA5 DB "5.- CASA DE ASTERISCOS$"
26     CADENA6 DB "6.- SALIR DEL PROGRAMA $"
27     CADENA7 DB "OPCION: $"
28     DIGITO DB ?
29     CADENA8 DB "CONTINUAR [ENTER] $"
30
31     ;SUBMENU
32     CADENASUB1 DB "1.- REGRESAR AL MENU PRINCIPAL$"
33     CADENASUB2 DB "2.- SALIR$"
34     CADENASUB3 DB "2$"
35     CADENASUB4 DB ?
36
37  DATOS ENDS
```

Imagen 5. Segmento de datos

```
42
43  CODIGO SEGMENT
44      ASSUME DS:DATOS, CS:CODIGO
45
46  INICIO:
47      MOV AX, DATOS
48      MOV DS, AX
49
50      MENU_PRINCIPAL:
51          ;IMPORTAR MACRO DEL MENU PRINCIPAL
52          MENUPRINCIPAL
53          ;Comprar las opciones ingresadas
54          CMP AL, 01h
55          JE slt_opcion1
56
57          CMP AL, 02h
58          JE slt_opcion2
59
60          CMP AL, 03h
61          JE slt_opcion3
```

Imagen 6. Segmento de código

```
190         ;CERRAR EL PROGRAMA
191     opcion6:
192         MOV AH, 4CH
193         INT 21H
194     jmp MENU_PRINCIPAL
195     ret
196
197     FIN:
198         MOV AH, 4CH
199         INT 21H
200
201 CODIGO ENDS
202 END INICIO
```

Imagen 7. Salir del programa

```
MOV AH, 09H
MOV DX, OFFSET CADENA
INT 21H
```

Imagen 8. Interrupción 21H función 09

```
MOV AH, 4CH
INT 21H
```

Imagen 9. Interrupción 21H función 4CH

```
MOV AH, 02H
MOV BH, 0H
MOV DH, 08H
MOV DL, 32H
INT 10H
```

Imagen 10. Interrupción 10H función 02

```
org 100h
mov ax,5
mov bx,2

jmp calc

back: jmp stop

calc:
add ax,bx
jmp back

stop:
ret
```

Imagen 11. Saltos incondicionales

```
org 100h
 ; utilizaremos un loop para i
 mov cx,5  ;movemos el valor 5
 repetir:
 call programita
  loop repetir
ret
programita proc; creamos un pr
    mov ax,2 ; movemos a ax el
    call pthis ;
    db 13,10,'ora ora ora',0 ;
    ret
    programita endp ; terminam
```

Imagen 12. Instrucción CALL

```
restart4:mov bl,0

mov al,10
cmp al,2
ja salto5      ;ja,jmp
jmp restart5
```

Imagen 13. JA (Jump if Above)

```
sextoC:
    mov ah, 02h
    mov dl,asterisco
    int 21h

    inc cl
    cmp cl,cant3
    jb sextoC
mov cx, bx

inc y
inc cl
cmp cl, aux3
jb quintoC
```

Imagen 14. JB (Jump if Below)

```
; read character in al:
mov ah, 1
int 21h

cmp al, '0'
jbe stop
```

Imagen 16. JBE (Jump if Below or Equal)

```
evaluar:
cmp al, dato1
JE sumar


sumar:
mov ah, 09h
lea dx, msjop
int 21h

mov ah, 01h
int 21h
sub al, 30h
mov num1, al
```

Imagen 17. JE (Jump if Equal)

```
mov al,'*'
mov bh,0
mov bl,1
mov cx,1
mov ah,09h
inc pos ;Imprimimos una x
int 10h
mov ah, 1
int 16h
jz wait_for_key
```

Imagen 18. JZ (Jump if Zero)

```
mov si, 0 ;ponemos si en 0
comienzo:
mov al, msg2[0] ;copiar la primera letra de la palabra a al
cmp msg[si],"$" ;si es el fin de la cadena mandar a final
jz final
cmp msg[si], al ;comparar si encuentra la primera letra de la cadena
jne seguir
```

Imagen 19. JNE (Jump if Not Equal)

```
final:
cmp ah, 0
jnz final
mov oct[si], al
```

Imagen 20. JNZ (Jump if Not Zero)

```
repeticion:
  mov ax,lista[bx]
  cmp ax,mayor
  jg cambiar
  jmp continuar
  cambiar:
    mov mayor,ax


  continuar:
  add suma,ax
  add bx,2
```

Imagen 21. JG (Jump if Greater)

```
cmp al,-5
jge salto1
mov bl,1
```

Imagen 22. JGE (Jump if Greater or Equal)

```
Etiqueta4:      mov ah,Z
                inc al
                cmp al,ah ;Compara al con ah
                jl Etiqueta4    ;salta a Etiqueta3 si  al < ah
                mov bl,0FFh
```

Imagen 23. JL (Jump if Less)

```
mov ah, 1
int 21h

cmp al, '0'
JNG stop
```

Imagen 24. JLE (Jump if Less or Equal)

```
; (unsigned)
; 255 is above 1
mov ah, 255
mov al, 1
cmp ah, al
nop
```

Imagen 25. Comparador (cmp)

```
MOV AH, 0FH ; Apertura del Archivo
MOV DX, OFFSET ARCHIVO ; Direccion del Archivo
INT 21h ; Llamado de la interrupcion
CMP AL, 0FH
JE ERROR
ERROR:
```

Imagen 26. Abrir un archivo en FCB

```
mov ah,3ch ; Funcion utilizada para crear archivo
mov cx,06H ; Atributo del archivo
; 06H= Escondido y de sistemaö
mov dx,offset reporte
int 21h ;Llama de la interrupcion
```

Imagen 27. Ejemplo de la función 3CH

Valor	Tipo de Atributo
00H	Normal
02H	Escondido
04H	Sistema
06H	Escondido y de Sistema

Imagen 28. Atributos de los archivos

```
mov ah,40h ; funcion utilizada para escribir en el archivo
mov bx,handle ; Handle asignado al archivo
mov dx, offset escribe archivo ; Direcci¾n de datos a escribir
mov cx, tamano Texto ; Cantidad de Byte a escribir
int 21h
```

Imagen 29. Ejemplo de la función 40H

```
primero:
    mov ah, 02h
    mov bh, 00h
    mov dh, y          ; coordenadas de las filas
    mov dl, x          ; coordenadas de las columnas
    int 10h

    mov bx, cx         ; Almaceno temporalmente el contador principal

    xor cx, cx         ; Reseteo del contador

 segundo:            ; Ciclo para el pintado de los asteriscos
        mov ah, 02h  ; Aca imprimimos el * el numero de veces que se le indique
        mov dl,asterisco
        int 21h

        inc cl
        cmp cl, cant    ; compara si la cantidad de * es igual a la cantidad que se debe pintar
        jb segundo     ; si es menor continuara pintando

    mov cx, bx          ; Recupero el contador principal
    inc y               ; aca actualizamos la fila donde se va a posicionar ahora el cursor
    inc cl              ; incrementa en 1 el contador cl
    cmp cl, aux1
    jb primero
```

Imagen 30. Ejemplo de ciclo en asm

7.CAPTURAS DEL CÓDIGO EN EJECUCIÓN

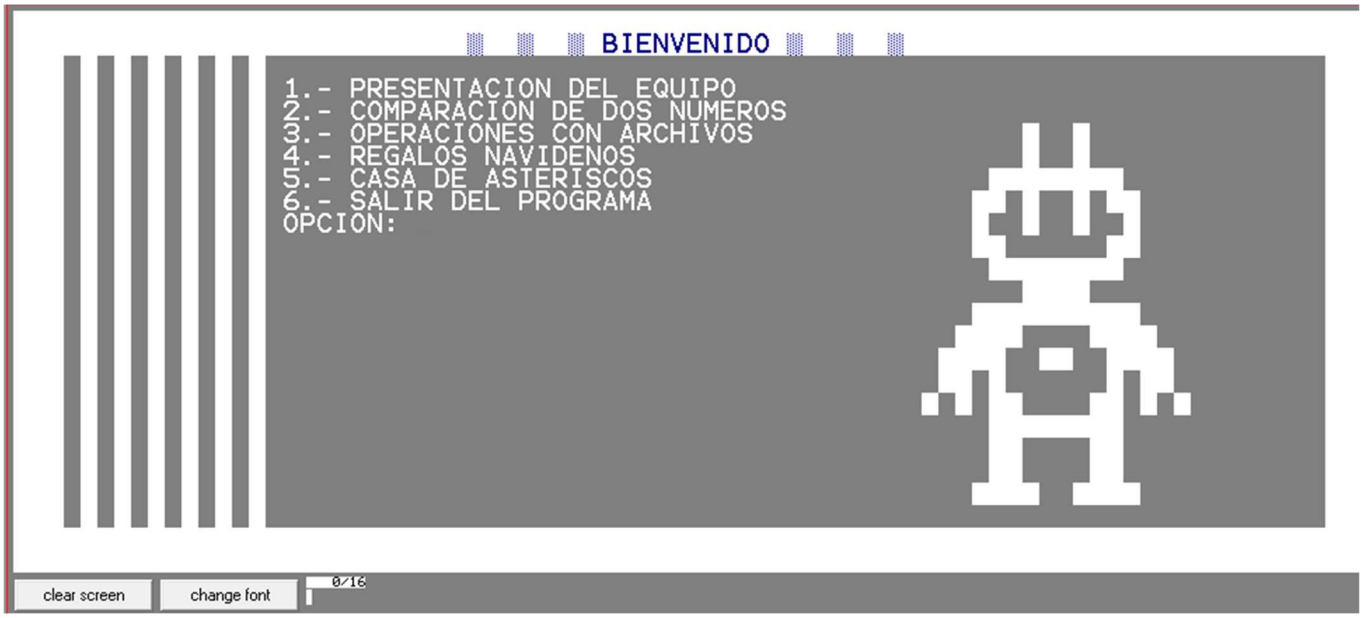


Imagen 31. Menú principal en ejecución

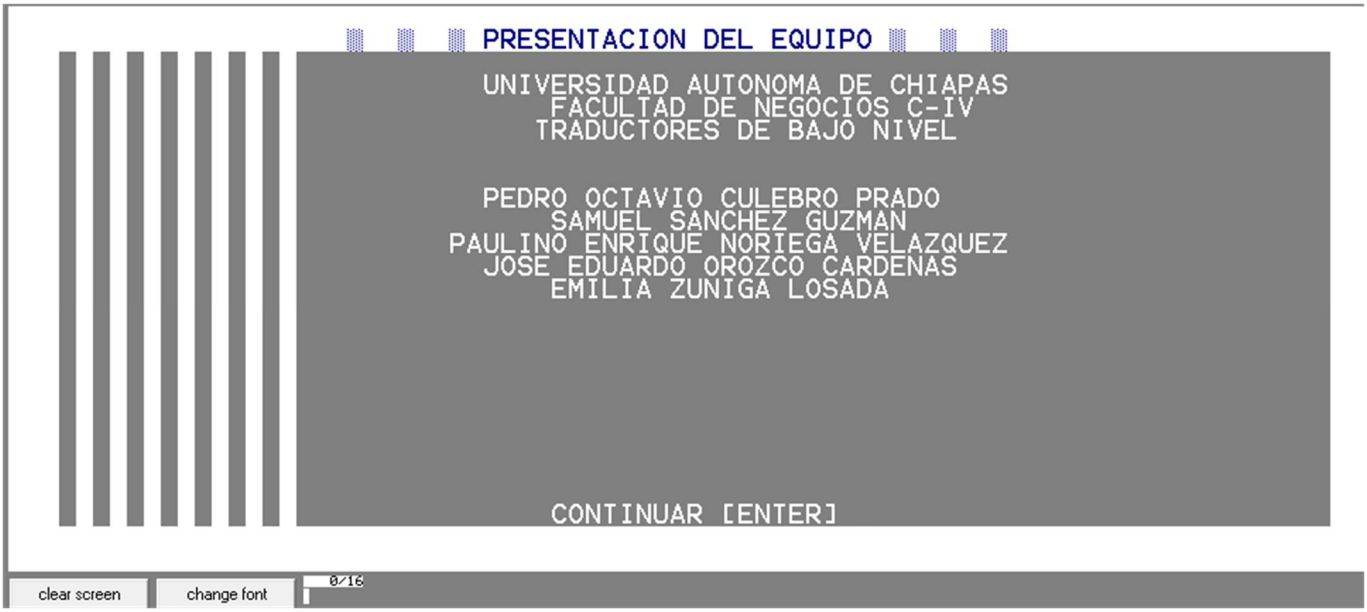


Imagen 32. Opción 1 del menú principal en ejecución, Presentación

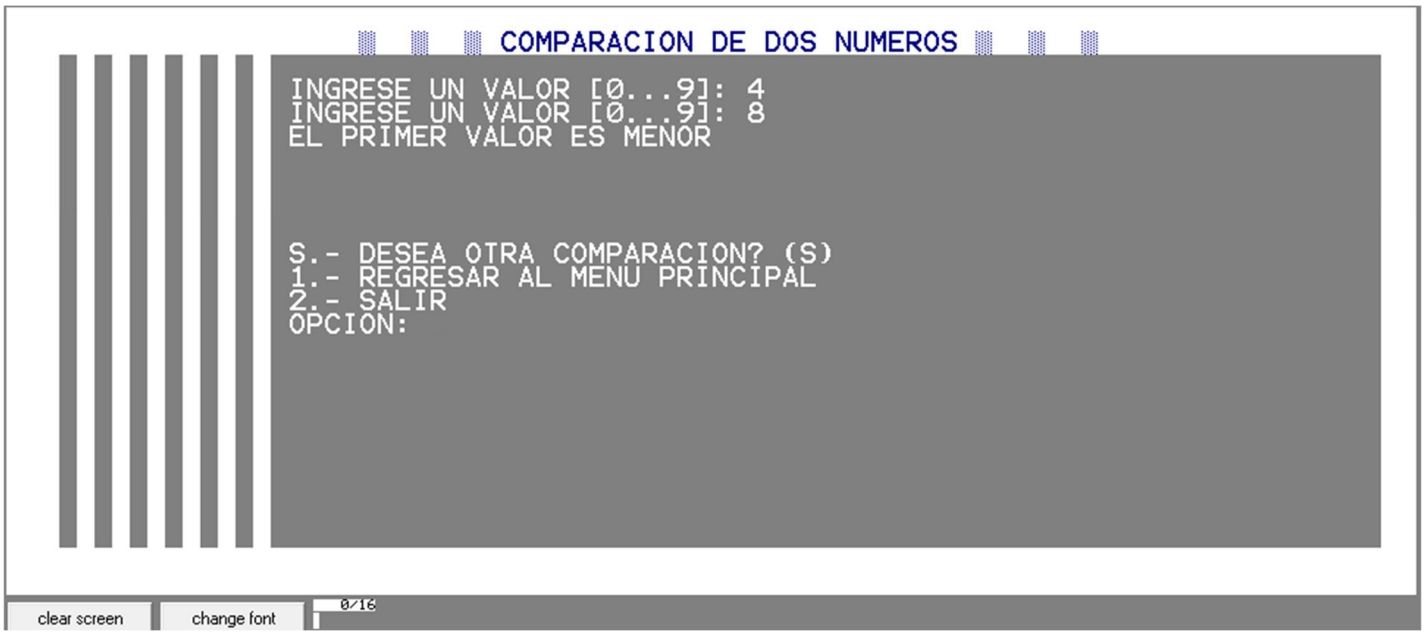


Imagen 33. Opción 2 del menú principal en ejecución, Comparación entre dos números

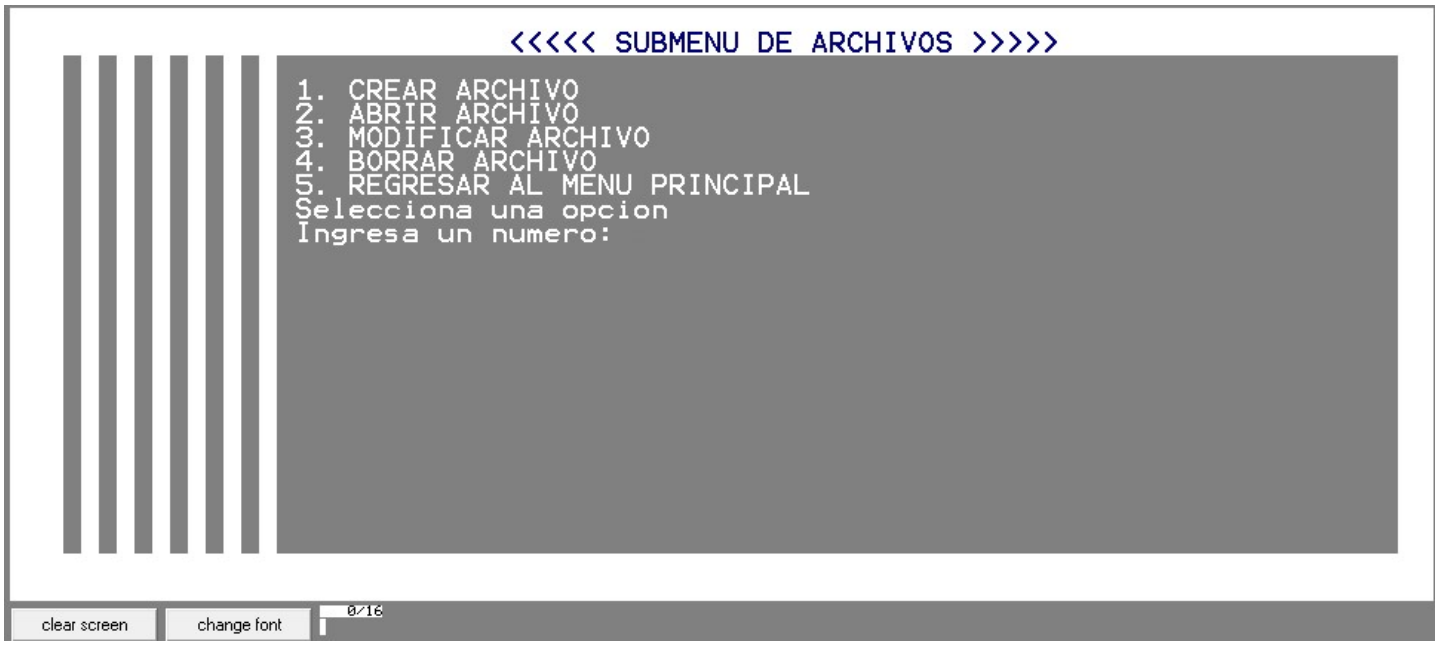


Imagen 34. Opción 3 del menú principal en ejecución, Operaciones con archivos

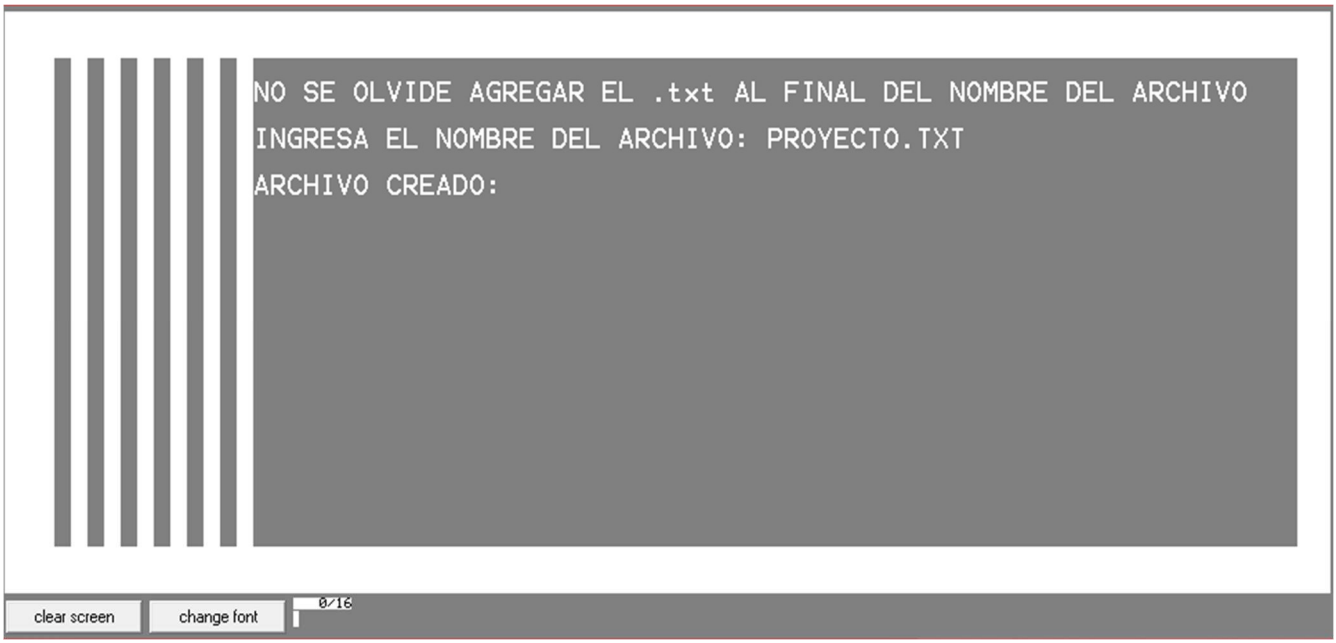


Imagen 35. Opción 1 submenú con archivos en ejecución, Crear archivo

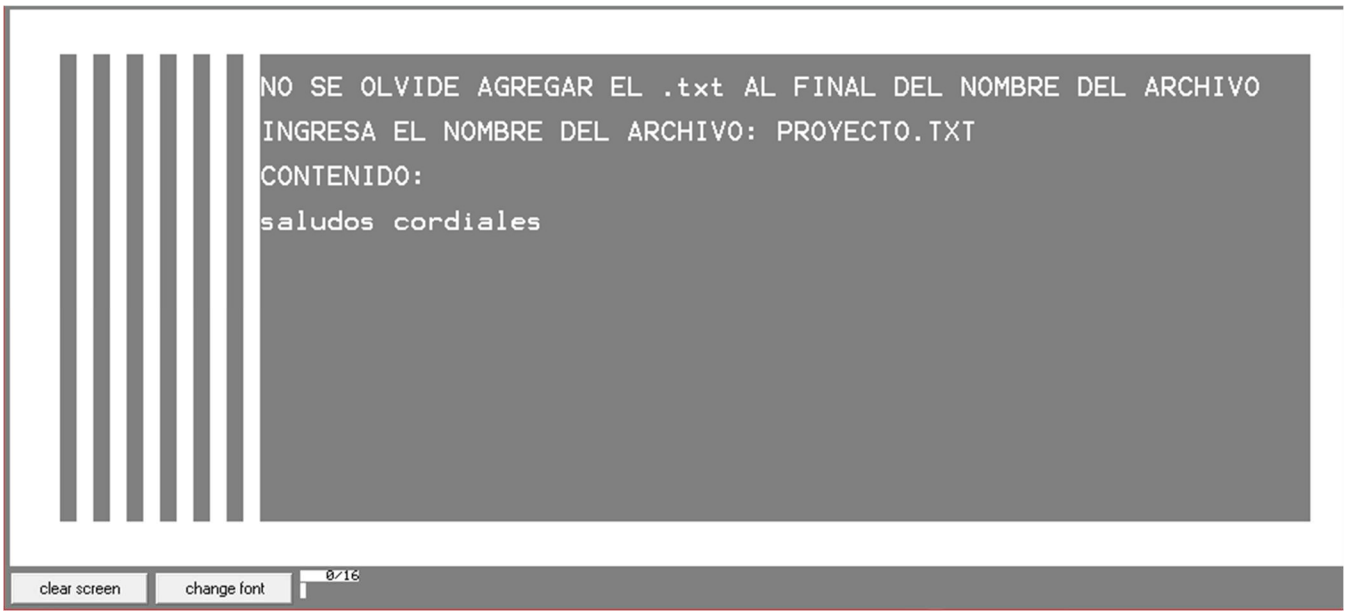


Imagen 36. Opción 2 submenú con archivos en ejecución, Leer archivo

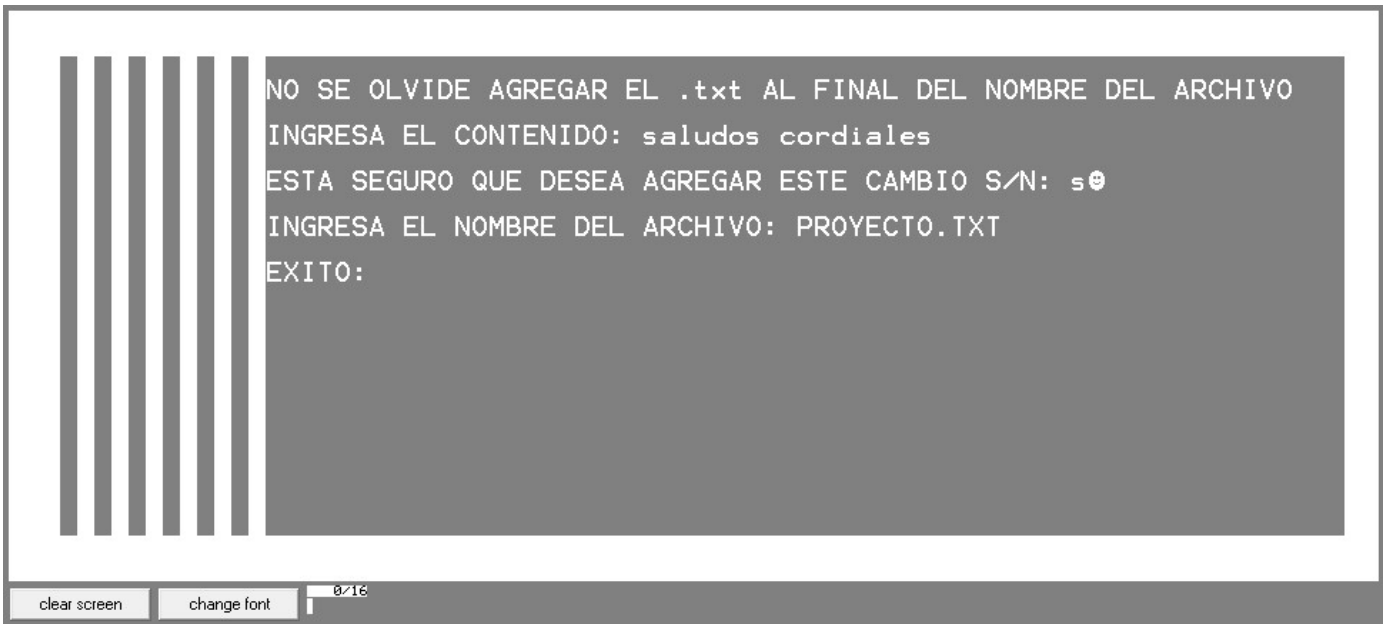


Imagen 37. Opción 3 submenú con archivos en ejecución, Escribir en archivo

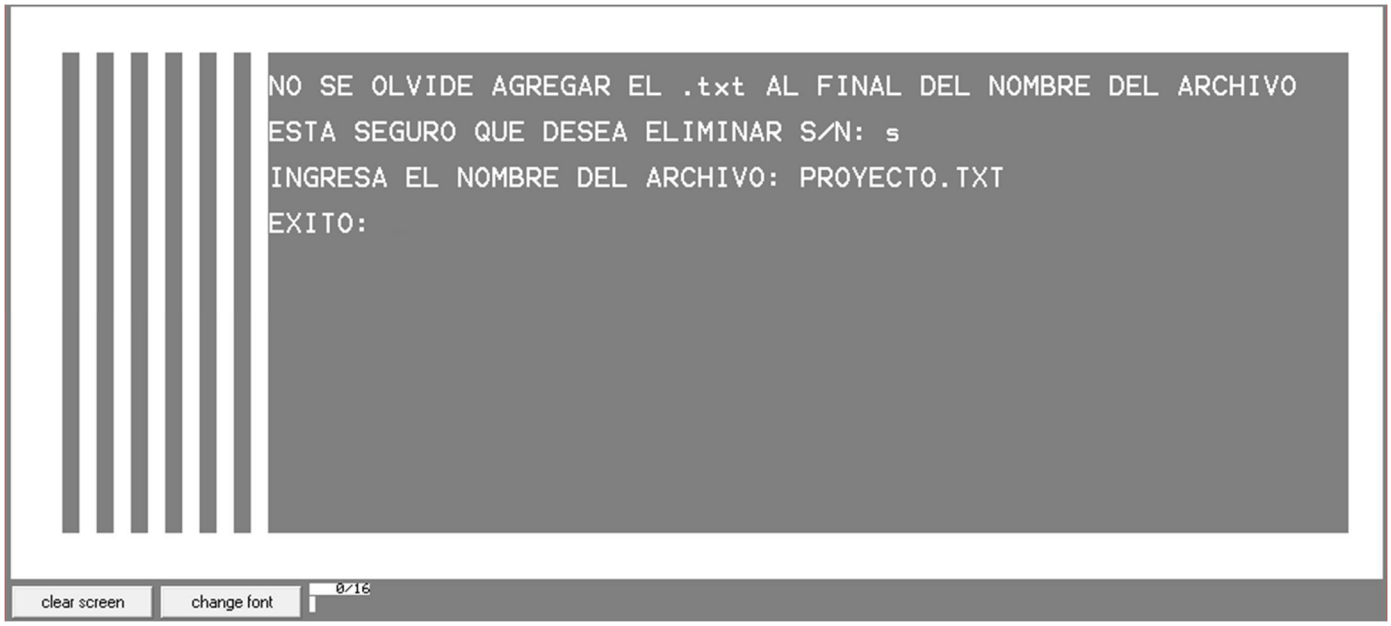


Imagen 38. Opción 4 submenú con archivos en ejecución, Eliminar archivo

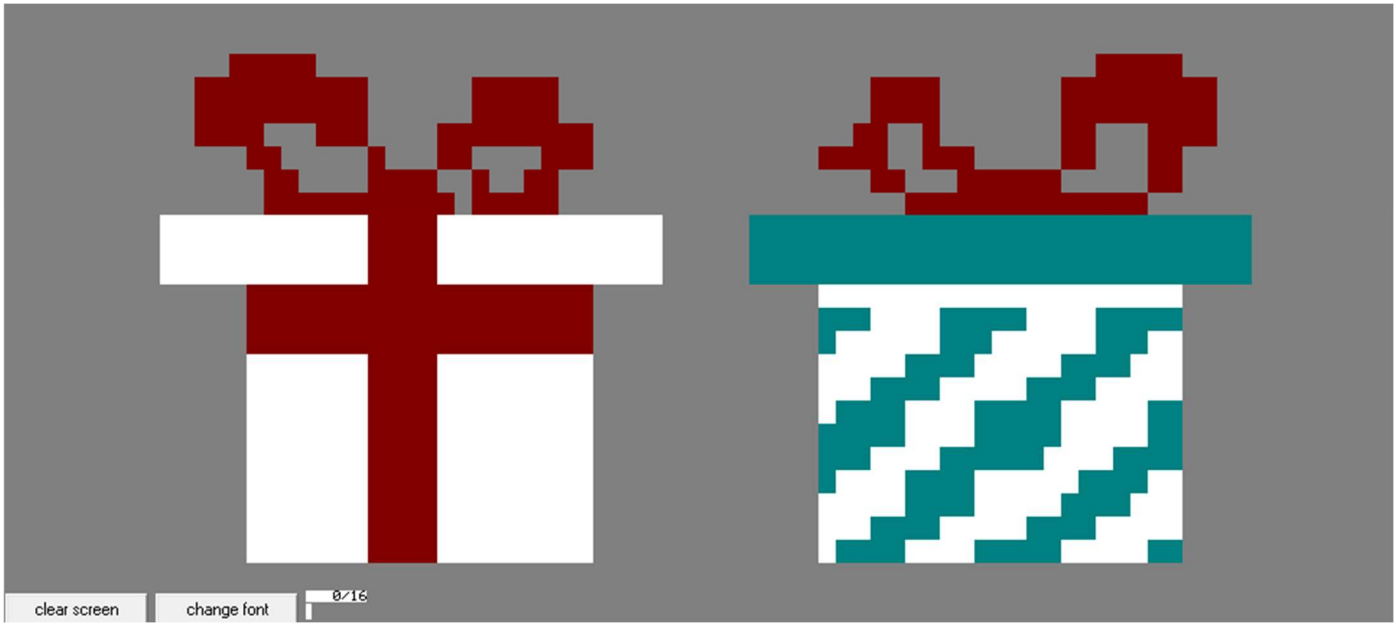


Imagen 39. Opción 4 del menú principal en ejecución, Regalos navideños

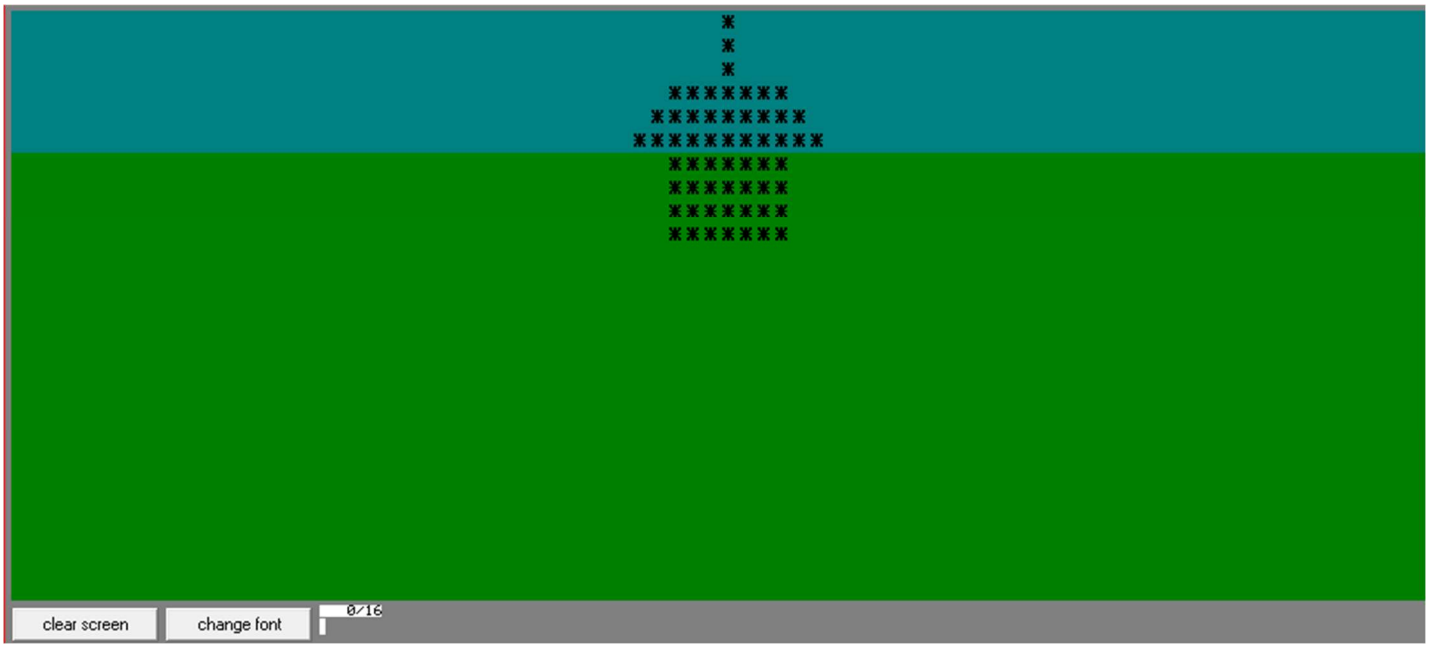


Imagen 40. Opción 5 del menú principal en ejecución, Casa de asteriscos

Imagen 41. Opción 6 del menú principal en ejecución, Salir del programa