



TRADUCTORES DE BAJO NIVEL

PROYECTO FINAL
Noviembre 2022



Universidad Autónoma de Chiapas

Integrantes:

- | | |
|---------------------------------|---------|
| 1. Culebro Ovando Paola Jadziry | B200065 |
| 2. Gómez Alvarado Alejandro | B200070 |
| 3. Guillen Gerardo Noé | B200100 |
| 4. Nájera Verdugo Álvaro | B200076 |
| 5. Sánchez Hernández Jeannette | B200080 |

5° "D"

Licenciatura:

Licenciatura en ingeniería desarrollo y tecnologías de software

Docente:

Mtra. Vanessa Benavides García

Materia:

Traductores de bajo nivel

Actividad:

Proyecto final

Tapachula, Chiapas a 11 de
noviembre del 2022

ÍNDICE

1. INTRODUCCIÓN	1
2. SUSTENTO TEÓRICO	2
2.1. Lenguaje ensamblador.....	2
2.2. Emulador 8086.....	4
2.3. Sistemas numéricos	4
2.3.1. Binario.....	5
2.3.2. Hexadecimal	5
2.3.3. Decimal.....	6
2.4. Código ASCII.....	6
2.5. Tipos de ensamblador.....	7
2.5.1. Ensambladores Cruzados:	7
2.5.2. Ensambladores Residentes	7
2.5.3. Microensambladores	7
2.5.4. Macroensambladores	7
2.5.5. Ensambladores de una fase.....	8
2.5.6. Ensambladores de dos fases	8
2.6. Segmentos	8
2.6.1. Segmento de Pila	8
2.6.2. Segmento de datos	9
2.6.3. Segmento de código.....	9
2.7. Registros	9
2.7.1. Registro de segmento	10
▪ CS.....	10
▪ DS.....	10
▪ SS.....	10
▪ ES.....	10
2.7.2. Registro de Índice.....	10
▪ SI.....	10
2.7.3. Registro de propósito general	11
▪ AX.....	11
▪ BX.....	11
▪ CX.....	11

▪	DX.....	11
2.8.	Comentarios.....	12
2.9.	Variables.....	12
2.10.	Palabras reservadas	13
2.11.	Tipos de datos	14
2.12.	Identificador	15
2.13.	Etiquetas	16
2.14.	Instrucciones	17
2.14.1.	Instrucción ADD.....	17
2.14.2.	Instrucción INC	17
2.14.3.	Instrucción LEA	18
2.14.4.	Instrucción MOV	18
2.14.5.	Instrucción SUB.....	19
2.15.	Interrupciones	19
2.15.1.	Interrupción 10H	19
▪	Función 0H – Establece modo de video	19
▪	Función 02H – Selección de posición del cursor	20
▪	Función 06H – Cambio a la página anterior activa	20
2.15.2.	Interrupción 21H	21
▪	Función 01H – Entrada desde teclado.....	21
▪	Función 02H – Exhibe salida.....	21
▪	Función 09H – Impresión de cadena (Video)	22
▪	Función 0AH – Entrada desde el teclado usando buffer	23
▪	Función 3CH – Crear archivo	23
▪	Función 3DH – Abre archivo	24
▪	Función 3EH – Cierra manejador de archivo	24
▪	Función 3FH – Lectura desde dispositivo/Archivo	25
▪	Función 40H – Escritura en dispositivo/ Archivo	25
▪	Función 41H – Borrar Fichero	25
▪	Función 4CH -Terminación de programa con Código de retorno	26
2.16.	Directivas.....	27
2.16.1.	Directiva ASSUME	27
2.16.2.	Directiva CALL.....	27
2.16.3.	Directiva de definición de datos	28
▪	DB o Byte	28
▪	DW o Dword.....	28
2.16.4.	Directiva END	29

2.16.5.	Directiva ENDS.....	29
2.16.6.	Directiva ENDP	29
2.16.7.	Directiva INCLUDE	30
2.16.8.	Directiva PROC	30
2.16.9.	Directiva SEGMENT	30
2.16.10.	Directiva RET	31
2.17.	Operadores	31
2.17.1.	Operador OFFSET	32
2.17.2.	Operador \$	32
2.18.	Salto.....	32
2.18.1.	Salto condicionales.....	32
▪	CMP	32
▪	JA.....	33
▪	JB.....	33
▪	JE.....	33
▪	JC	33
2.18.2.	Salto incondicionales	34
▪	Instrucción JMP	34
2.19.	Ciclos.....	34
2.20.	Macros	35
2.21.	Procedimientos.....	37
2.22.	Mnemotécnicos	37
3.	CONCLUSIÓN.....	38
4.	REFERENCIAS BIBLIOGRÁFICAS.....	39
5.	ANEXOS	43
6.	CAPTURAS DE LA EJECUCIÓN DEL CÓDIGO	49

1. INTRODUCCIÓN

En este trabajo se plantearán y expondrán los diferentes conceptos y temas que se emplean a lo largo de la resolución de las diferentes problemáticas planteadas en el proyecto final de este semestre (agosto - diciembre 2022).

Por lo tanto, se explica desde lo que es el lenguaje ensamblador, el cual es el lenguaje con el cual se desarrollará la codificación necesaria, a la par que se menciona lo que es el Emulador 8086, el cual es el Emulador en el cual se usa la sintaxis establecida para la codificación del mismo lenguaje. Además de todo eso, se plantean las definiciones de cada uno de los conceptos utilizados, por ejemplo, interrupciones, etiquetas, macros, etcétera.

Esto se realiza con la intención de poner en práctica todo el conocimiento recabado a lo largo de este semestre, con el objetivo de demostrar que se efectuó con el 100% de aprovechamiento. Además, que el hecho de trabajar con este lenguaje de bajo nivel nos ayudará a entender como es el procedimiento que se lleva a cabo cuando se desarrolla en lenguajes de alto nivel, dado que este lenguaje es el que trabaja directamente con el hardware, esto hace que el código sea extenso, pero ciertamente existen técnicas las cuales hacen que nuestro código reduzca considerablemente, estas mismas se explican en este mismo trabajo formal.

2. SUSTENTO TEÓRICO

2.1.Lenguaje ensamblador

El lenguaje ensamblador es el lenguaje de programación utilizado para escribir programas informáticos de bajo nivel, y constituye la representación más directa del Código máquina específico para cada arquitectura de computadoras legible por un programador. Aun hoy se utiliza en la programación de handler o manipuladores de dispositivos de hardware.

Características

- El código escrito en lenguaje ensamblador posee una cierta dificultad de ser entendido directamente por un ser humano ya que su estructura se acerca más bien al lenguaje máquina, es decir, lenguaje de bajo nivel.
- El lenguaje ensamblador es difícilmente portable, es decir, un código escrito para un Microprocesador, suele necesitar ser modificado, muchas veces en su totalidad para poder ser usado en otra máquina distinta, aun con el mismo Microprocesador, solo pueden ser reutilizados secciones especiales del código programado.
- Los programas hechos en lenguaje ensamblador, al ser programado directamente sobre Hardware, son generalmente más rápidos y consumen menos recursos del sistema (memoria RAM y ROM). Al programar cuidadosamente en lenguaje ensamblador se pueden crear programas que se ejecutan más rápidamente y ocupan menos espacio que con lenguajes de alto nivel.
- Con el lenguaje ensamblador se tiene un control muy preciso de las tareas realizadas por un Microprocesador por lo que se pueden crear segmentos de código difíciles de programar en un lenguaje de alto nivel.
- También se puede controlar el tiempo en que tarda una Rutina en ejecutarse, e impedir que se interrumpa durante su ejecución.

- El lenguaje ensamblador es un código estructurado y gravitatorio desarrollado sobre un archivo de programación (.ASM), en el cual pueden existir varios programas, macros o rutinas que pueden ser llamados entre sí.

Un programa escrito en lenguaje ensamblador consiste en una serie de Instrucciones que corresponden al flujo de órdenes ejecutables que pueden ser cargadas en la Memoria de un sistema basado en Microprocesador. Por ejemplo, un Procesador x86 puede ejecutar la siguiente instrucción Binaria como se expresa en código de máquina:

Binario: 10110000 01100001 (Hexadecimal: 0xb061)

La representación equivalente en lenguaje ensamblador es más fácil de recordar:

MOV al, 061h

Esta instrucción significa:

Asigna el valor Hexadecimal 61 (97 Decimal) al registro "al".

El mnemónico "mov" es un código de operación u "opcode", elegido por los diseñadores de la colección de instrucciones para abreviar "move" (mover, pero en el sentido de copiar valores de un sitio a otro). El opcode es seguido por una lista de argumentos o parámetros, completando una instrucción de ensamblador típica. A diferencia de los lenguajes de alto nivel, aquí hay usualmente una correspondencia 1 a 1 entre las instrucciones simples del ensamblador y el lenguaje de máquina. Sin embargo, en algunos casos, un ensamblador puede proveer "pseudo instrucciones" que se expanden en un código de máquina más extenso a fin de proveer la funcionalidad necesaria. Por ejemplo, para un código máquina condicional como "si X mayor o igual que", un ensamblador puede utilizar una pseudo instrucción al grupo "haga si menor que", y "si = 0" sobre el resultado de la condición anterior. Los Ensambladores más completos también proveen un rico lenguaje de macros que se utiliza para generar código más complejo y secuencias de datos. Cada arquitectura de microprocesadores tiene su propio lenguaje de máquina, y en consecuencia su propio lenguaje ensamblador ya que este se encuentra muy ligado a la estructura del hardware para el cual se programa. Los microprocesadores difieren en el tipo y número de operaciones que soportan;

también pueden tener diferente cantidad de registros, y distinta representación de los tipos de datos en memoria. Aunque la mayoría de los microprocesadores son capaces de cumplir esencialmente las mismas funciones, la forma en que lo hacen difiere y los respectivos lenguajes ensamblador reflejan tal diferencia. Pueden existir múltiples conjuntos de mnemónicos o sintaxis de lenguaje ensamblador para un mismo conjunto de instrucciones, instanciados típicamente en diferentes programas en ensamblador. En estos casos, la alternativa más popular es la provista por los fabricantes, y usada en los manuales del programa.

2.2.Emulador 8086

El emu8086 es un emulador del microprocesador 8086 (Intel o AMD compatible) con assembler integrado. Este entorno corre sobre Windows y cuenta con una interfaz gráfica muy amigable e intuitiva que facilita el aprendizaje el lenguaje de programación en assembler.

Dado que en un entorno emulado de microprocesador no es posible implementar una interfaz real de entrada/salida, el emu8086 permite interfacear con dispositivos virtuales y emular una comunicación con el espacio de E/S. Para esto, el emu8086 cuenta con una serie de dispositivos virtuales preexistentes en el software base, listos para ser utilizados.

Emulador de microprocesador emula principalmente el procesador, no las otras funciones que tendría un microordenador que ejecuta un procesador 8086. (Anexo-1)

2.3.Sistemas numéricos

Los sistemas numéricos son como tal un conjunto de grupos de reglas, normas, convenios y estándares los cuales nos permiten realizar una representación de

todos los números naturales, por medio un grupo amplio de símbolos básicos los cuales están definidos por la base que se utiliza.

Estos sistemas tienen como objetivo principal, lograr realizar el conteo de los diferentes elementos que contiene un conjunto, por lo tanto, la finalidad de los sistemas numéricos es la representación de los números.

Características de los sistemas numéricos:

- Cada uno de los sistemas numéricos se caracteriza por su base.
- Los sistemas numéricos tienen un base o como tal un conjunto de símbolos que permiten representar las diferentes y amplias cantidades numéricas.
- Son sistemas posicionales.
- Están compuestas por dígitos.
- Cada elemento del sistema tiene un valor ponderado.
- El número cero expresa o denota la ausencia de una cantidad determinada.

2.3.1. Binario

Este sistema es un sistema de numeración el cual usa un conjunto de dos símbolos, los cuales son el cero (0) y el uno (1), a los cuales se les denomina como símbolos binarios. Este sistema es usado para la representación de textos, datos y programas ejecutables en dispositivos informáticos.

Cada dígito binario (0's y 1's) constituye un bit, siguiendo esta lógica, 8 bits constituyen un byte y cada byte constituyen un carácter, letra o número.

Ejemplo: $(01011)_2$

2.3.2. Hexadecimal

Hexadecimal (del griego "Hexa" que significa seis y del latino "decem" que significa diez).

El sistema hexadecimal es un sistema de valor de posición el cual representa números sobre una base de 16 dígitos. Este sistema usa el siguiente conjunto de símbolos:

- Números del 0 al 9 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- Letras de la A a la F (A, B, C, D, E, F)

Ejemplo: $(4F)_{16}$

2.3.3. Decimal

También conocido sistema numérico posicional, este sistema es un conjunto de símbolos y reglas las cuales nos permiten formar todos los números que existen y los cuales son válidos. En dicho sistema mencionado las cantidades se pueden representar usando como bases aritméticas potenciadas por el número diez. Es decir, los números que componen este sistema numérico son 10, los cuales son del 0 al 9.

Ejemplo: $(777)_{10}$

2.4. Código ASCII

Para uniformar la representación de caracteres, los fabricantes de microcomputadoras han adoptado el código ASCII (American Standard Code for Information Interchange). Un código uniforme facilita la transferencia de información entre los diferentes dispositivos de la computadora. El código ASCII extendido de 8 bits que utiliza la PC proporciona 256 caracteres, incluyendo símbolos para alfabetos extranjeros. Por ejemplo, la combinación de bits 01000001 (41 hex) indica la letra A.

Las categorías son:

00-1FH

Códigos de control para la pantalla, impresoras y transmisión de datos, que son utilizados para provocar una acción

20-7FH

Códigos de caracteres para números, letras, y puntuación. Observe que 20H es el espacio o blanco estándar.

80-FFH

Códigos ASCII ampliados, caracteres de otras escrituras, griegos y símbolos matemáticos y caracteres gráficos para dibujar cajas.

En la siguiente imagen están los códigos de control desde la 00H hasta la 1FH; los que están en paréntesis no se imprimen. (Anexo-2)

2.5.Tipos de ensamblador

Los ensambladores hacen casi la misma tarea, pero se clasifica de acuerdo a algunas características que los hacen diferentes unos de los otros, y son los siguientes:

2.5.1.Ensambladores Cruzados:

Se denominan así a los ensambladores que se utilizan en una computadora que posee el procesador diferente al que tendrán las computadoras donde se va a ejecutar el programa objeto producido. El empleo de este tipo permite aprovechar el soporte de medios físicos y de programación que ofrecen las máquinas potentes para desarrollar programas que luego los van a ejecutar sistemas muy especializados en determinados tipos de tareas.

2.5.2.Ensambladores Residentes

Son aquellas que permanecen en la memoria principal de la computadora y cargan para su ejecución al programa objeto producido. Este tipo de ensamblador tiene la ventaja de que se puede comprobar inmediatamente el programa sin necesidad de transportarlo de un lugar a otro, como se hacía en crossassembler, y sin necesidad de programas simuladores. Sin embargo, puede presentar problemas de espacio de memoria, ya que el traductor ocupa espacio que no puede ser utilizado por el programador.

2.5.3.Microensambladores

Al programa que indica al intérprete de instrucciones de la CPU como debe actuar se le denomina microprograma. El programa que ayuda a realizar este microprograma se llama micro ensamblador. Existen procesadores que permiten la modificación de sus microprogramas, para lo cual se utilizan microensambladores.

2.5.4.Macroensambladores

Son ensambladores que permiten el uso de macroinstrucciones. Debido a su potencia, normalmente son programas robustos que no permanecen en memoria

una vez generado el programa objeto. Puede variar la complejidad de los mismos, dependiendo de las posibilidades de definición y manipulación de las macroinstrucciones, pero normalmente son programas bastante complejos.

2.5.5. Ensambladores de una fase

Lee una línea y la traduce directamente para producir una instrucción de lenguaje máquina o la ejecuta si se trata de una pseudoinstrucción. Se construye la tabla de símbolos a medida que aparecen las definiciones de variables, etiquetas, etc. Debido a su forma de traducción estos ensambladores obligan a definir los símbolos antes de ser empleados para que, cuando aparezca una referencia a un determinado símbolo en una instrucción, se conozca la dirección de dicho símbolo y se pueda traducir de forma correcta.

2.5.6. Ensambladores de dos fases

Realiza la traducción en dos etapas: 1º Fase lee el programa fuente y construyen la tabla de símbolos, 2º Fase vuelve a leer el programa fuente y pueden ir traduciendo totalmente pues reconocen la totalidad de los símbolos. Estos ensambladores son más utilizados en la actualidad.

2.6. Segmentos

Los segmentos en el lenguaje ensamblador son un área especial designada en el programa que inicia en un límite de un párrafo, esto es en una localidad regularmente divisible entre 16 o 10 hex. Los segmentos principales o bien, los más usados son el segmento de datos, segmento de código y segmento de pila, de los cuales se hablarán a continuación:

2.6.1. Segmento de Pila

El segmento de pila es aquel que tiene la función de contener direcciones, constantes, funciones, etc. Gracias a este segmento se facilita el uso de las funciones, debido que no hay necesidad de estar declarando las constantes, o creando las funciones, es tan simple como mandarlas a llamar. El registro del

segmento de pila (SS) es el que se encarga de direccionar el segmento de pila, además de que este registro tiene una capacidad de 64K y se puede definir más de un segmento de pila, sí en dado caso se llegase a requerir.

2.6.2.Segmento de datos

Este segmento es el encargado de contener o almacenar las variables, constantes y áreas de trabajo definidas por el programa y el programador. En este segmento se almacenan todas aquellas variables necesarias para el correcto funcionamiento del código, El registro del segmento datos (DS) es el que se encarga de direccionar el segmento de datos, además de que este registro tiene una capacidad de 64K y se puede definir más de un segmento de datos, sí en dado caso se llegase a requerir.

2.6.3.Segmento de código

Este segmento es el encargado de almacenar todas aquellas instrucciones que son definidas por el programador, con la intención de realizar alguna operación o un conjunto de ellas, para así lograr la resolución de una problemática. El registro de código (CS) direcciona el segmento de código, además de que este registro tiene una capacidad de 64K y se puede definir más de un segmento de código, sí en dado caso se llegase a requerir.

2.7.Registros

Los registros del procesador se emplean para controlar instrucciones en ejecución, manejar direccionamiento de memoria y proporcionar capacidad aritmética. Los registros son direccionables por medio de un nombre. Los bits, por convención, se numeran de derecha a izquierda, como en:

... 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

2.7.1.Registro de segmento

Conjunto de registros que son de 16 bits, y contienen el valor de segmento. Este nos facilita un área de memoria para el direccionamiento conocida como el segmento actual.

- CS

El registro de segmentos más importante es el **CS** o segmento de código. Es aquí donde se encuentra el código ejecutable de cada programa, el cual está directamente ligado a los diferentes modelos de memoria. Ejemplo de su uso : (Anexo-3)

- DS

El **DS** es un registro de segmento cuya función es actuar como policía donde se encuentran los datos. Cualquier dato, ya sea una variable inicializada o no, debe estar dentro de este segmento. Ejemplo de su uso: (Anexo-3)

- SS

El **SS** tiene la tarea exclusiva de manejar la posición de memoria donde se encuentra la pila (stack) Esta es una estructura usada para almacenar datos en forma temporal, tanto de un programa como de las operaciones internas

- ES

El registro **ES** tiene el propósito general de permitir operaciones sobre cadenas, pero también puede ser una extensión del DS.

2.7.2.Registro de Índice

Registros que son usados como índices por algunas instrucciones, que están disponibles para direccionamiento indexado y para sumas y restas

- SI

El registro SI es útil para manejar bloques de cadenas en memoria, siendo el primero la índice fuente, en otras palabras, SI representa la dirección donde se encuentra la cadena, por lo tanto, se puede decir que, SI es un registro de índice, en el que sirve para indicar al OFFSET dentro de un segmento. Trabaja juntamente con el registro DI. Ejemplo de su uso: (Anexo-4) en este

es donde llevamos el contador de las veces que se ingresa un carácter, llevando su registro con SI.

2.7.3.Registro de propósito general

Los registros **AX**, **BX**, **CX** Y **DX** son los caballos de batalla del sistema. Son únicos en el sentido de que se puede direccionarlos como una palabra o como una parte de un byte. Los registros a continuación son de propósito general, aunque todos tiene alguna función por defecto que se dará a conocer en su respectivo concepto.

- **AX**

El registro **AX** se usa para almacenar resultados, lectura o escritura desde o hacia los puertos. Puede ser accedido en 8 bits como AH para la parte alta (HIGH) y AL (LOW) para la parte baja.

AX: AH / AL

- **BX**

El **BX** sirve como apuntador base o índice. Podemos decir que se usa para apuntar posiciones en memoria con él. También puede accederse como BH y BL, parte alta y baja respectivamente.

BX: BH / BL

- **CX**

El **CX** se utiliza en operaciones de iteración, como un contador que automáticamente se incrementa o decrementa de acuerdo con el tipo de instrucción usada, por lo que es conocido como contador. CL (parte baja del registro) almacena el desplazamiento en las operaciones de desplazamiento y rotación de múltiples bits.

CX: CH/CL

- **DX**

El **DX** o registros de datos se usa como puente para el acceso de datos. Algunas operaciones de entrada/salida requieren su uso, y las operaciones de multiplicación y división con cifras grandes suponen al **DX** y al **AX** trabajando juntos

2.8.Comentarios

Los comentarios son utilizados para mejorar la claridad, más si está orientado al lenguaje ensamblador, conocido por ser difícil, nos sirve de mucha ayuda los comentarios debido a que su propósito es junto con un punto y coma, donde sea codificado este signo, le da a entender al ensamblador que todos los caracteres a la derecha de esa línea son comentarios.

Un comentario puede aparecer solo en una línea o a continuación de una instrucción en la misma línea, un ejemplo claro de estos comentarios, es el siguiente:

1. ; Toda esta línea es un comentario. Ejemplo (Anexo-5)
2. MOV AX, DATOS; Esto es un comentario
 MOV DS, AXEste ya no

2.9.Variables

La declaración de variables en un programa en ensamblador se puede incluir en la sección .data o en la sección. bss, según el uso de cada una.

Sección .data, variables inicializadas

Las variables de esta sección se definen utilizando las siguientes directivas:

- **db:** define una variable de tipo byte, 8 bits. Ejemplo (Anexo-6)
- **dw:** define una variable de tipo palabra (word), 2 bytes = 16 bits. Ejemplo (Anexo-7)
- **dd:** define una variable de tipo doble palabra (double word), 2 palabras = 4 bytes = 32 bits.
- **dq:** define una variable de tipo cuádruple palabra (quad word), 4 palabras = 8 bytes = 64 bits. El formato utilizado para definir una variable empleando cualquiera de las directivas anteriores es el mismo:

nombre_variable directiva valor_inicial

Ejemplo

```
var1 db 255 ; define una variable con el valor FFh
Var2 dw 65535 ; en hexadecimal FFFFh
var4 dd 4294967295 ; en hexadecimal FFFFFFFFh
var8 dq 18446744073709551615 ; en hexadecimal
FFFFFFFFFFFFFFFFh
```

Se puede utilizar una constante para inicializar variables. Solo es necesario que la constante se haya definido antes de su primera utilización.

```
tamañoVec equ 5
indexVec db tamañoVec
```

Los valores iniciales de las variables y constantes se pueden expresar en bases diferentes como decimal, hexadecimal, octal y binario, o también como caracteres y cadenas de caracteres.

Si se quieren separar los dígitos de los valores numéricos iniciales para facilitar la lectura, se puede utilizar el símbolo '_', sea cual sea la base en la que esté expresado el número.

```
var4 dd 4_294_967_295
var8 dq FF_FF_FF_FF_FF_FF_FF_FFh
```

Los valores numéricos se consideran por defecto en decimal, pero también se puede indicar explícitamente que se trata de un valor decimal finalizando el número con el carácter d.

```
var db 67 ;el valor 67 decimal
var db 67d ;el mismo valor
```

2.10. Palabras reservadas

Las palabras reservadas tienen un significado especial en MASM, y sólo pueden usarse dentro de su contexto correcto. Hay distintos tipos de palabras reservadas:

- Nemónicos de instrucciones, como MOV, ADD y MUL.
- Directivas, las cuales indican a MASM cómo ensamblar programas.
- Atributos, que proporcionan información acerca del tamaño y uso de las variables y operandos. Dos ejemplos son BYTE y WORD.
- Operadores, que se utilizan en expresiones constantes. Símbolos predefinidos como @data, que devuelven valores enteros constantes en tiempo de ensamblado.

2.11. Tipos de datos

El ensamblador reconoce un conjunto de tipos de datos internos básicos (tipos de datos intrínsecos), de acuerdo con el tamaño de los datos (bytes, palabras, palabras dobles, etc.), ya sea con signo, entero o número real para describir su tipo. Estos tipos tienen un grado considerable de superposición, por ejemplo, el tipo DWORD (entero sin signo de 32 bits) se puede intercambiar con el tipo SDWORD (entero con signo de 32 bits).

Algunas personas pueden decir que el programador usa SDWORD para decirle al lector que este valor está firmado, pero no es obligatorio para el ensamblador. El ensamblador solo evalúa el tamaño del operando. Entonces, por ejemplo, los programadores solo pueden especificar enteros de 32 bits como tipos DWORD, SDWORD o REAL4.

La siguiente tabla ofrece una lista de todos los tipos de datos internos. Los símbolos IEEE en algunas entradas se refieren al formato de número real estándar publicado por IEEE Computer Society.

Tipos de datos	Uso
BYTE	Entero sin signo de 8 bits, B representa el byte
SBYTE	Entero de 8 bits con signo, S significa con signo
WORD	Entero sin signo de 16 bits
SWORD	Entero de 16 bits con signo

DWORD	Entero sin signo de 32 bits, D significa doble (palabra)
SDWORD	Entero de 32 bits con signo, SD significa doble con signo (palabra)
FWORD	Entero de 48 bits (puntero lejano en modo protegido)
QWORD	Entero de 64 bits, Q representa cuatro (palabra)
TBYTE	Entero de 80 bits (10 bytes), T representa 10 bytes
REAL4	Número real corto IEEE de 32 bits (4 bytes)
REAL8	Número real largo IEEE de 64 bits (8 bytes)
REAL10	80 bits (10 bytes) Número real extendido IEEE

2.12. Identificador

Un identificador es un nombre que se aplica a elementos en el programa. Los dos tipos de identificadores son: nombre, que se refiere a la dirección de un elemento de dato. y etiqueta, que se refiere a la dirección de una instrucción. Las mismas reglas se aplican tanto para los nombres como para las etiquetas.

Un identificador puede usar los siguientes caracteres:

- 1.- Letras del alfabeto: Desde la A hasta la Z
- 2.- Dígitos: Desde el 0 al 9 (no puede ser el primer carácter)
- 3.- Caracteres especiales Signo de interrogación (?)

Subrayado (_)

Signo de pesos (\$)

Arroba (@)

Punto (.) (no puede ser el primer carácter)

El primer carácter de un identificador debe ser una letra o un carácter especial, excepto el punto. Ya que el ensamblador utiliza algunos símbolos especiales en palabras que inician con el símbolo @, debe evitar usarlo en sus definiciones.

Ejemplos de nombres validos son COUNT, PAGE25 y \$E10. Se recomienda que los nombres sean descriptivos y con significado. Los nombres de registros, como

AX, DI y AL, están reservados para hacer referencia a esos mismos registros. En consecuencia, en una instrucción tal como:

ADD AX, BX

el ensamblador sabe de forma automática que AX y BX se refieren a los registros. Sin embargo, en una instrucción como:

2.13.Etiquetas

Las etiquetas son nombres usados para referenciar a números, cadenas de caracteres o en dado caso localizaciones de memoria dentro de un programa. Las etiquetas son definidas por el programador, estas etiquetas, pueden ser un conjunto de los siguientes caracteres:

- Letras del alfabeto (A – Z, a - z)
- Dígitos (0 - 9)
- Caracteres de subrayado (_)
- Signo de pesos (\$)
- Signo de interrogación (¿ ?)

Estas etiquetas solo son declaradas una sola vez dentro de nuestro segmento de código, y a la hora de hacer uso de ellas, solo se debe de hacer un salto hacia nuestra etiqueta, y con ello, el emulador se dirigirá hacia ese segmento de instrucciones y se hará lo que dicha etiqueta contenga.

Sintaxis:

Nombre_etiqueta:

Insert your code here

Ejemplo: (Anexo-8)

2.14.Instrucciones

Una instrucción es un enunciado que se vuelve ejecutable cuando se ensambla un programa. El ensamblador traduce las instrucciones en bytes de lenguaje máquina, para que la CPU los cargue y los lleve a cabo en tiempo en ejecución. Una instrucción contiene cuatro partes básicas:

- Etiquetas (Opcional)
- Nemónico de instrucción (Requerido)
- Operando (s) (por lo general, son requeridos)
- Comentario(opcional)

2.14.1. Instrucción ADD

Esta instrucción tiene la función de adicionar operandos introducidos por el teclado, es decir suma los dos operandos y guarda el resultado en el operando destino.

Sintaxis:

ADD operando_destino, operando_fuente

2.14.2.Instrucción INC

Esta instrucción efectúa un incremento a una variable o a un registro establecido por el programador, es decir, tiene la función de efectuar una suma a un operando.

Sintaxis:

INC destino

Destino es el operando al cual se le debe de efectuar el incremento. Estas instrucciones son muy usadas en lo que son los ciclos.

Ejemplos:

1) Ejemplo sin ciclo:

INC DX

2) Ejemplo con ciclo:

Ciclo_ar:

MOV AH,01H

INT 21H

CMP AL,0dh

JE FinCiclo

CMP SI,33

JE FinCiclo

MOV ruta [si],al

INC si

JB Ciclo_ar

2.14.3.Instrucción LEA

De las siglas en inglés (Load Effective Address) es una forma de obtener la dirección que surge de cualquiera de los modos de direccionamiento de memoria del procesador.

Es decir, esta directiva tiene la función de leer y cargar lo que se ingresa por teclado, para que esa entrada se pueda trabajar con un conjunto de operaciones, con la intención de resolver una problemática, así lo establezca el desarrollador.

2.14.4.Instrucción MOV

Es una instrucción la cual, como su nombre traducido del inglés (Mover), es la encargada de mover de mover un dato entre los registros de memoria del ensamblador. Aunque no solo mueve, sino también almacena una copia del dato operando fuente y destino, esto lo hace de manera consciente.

Sintaxis:

MOV destino_fuente, dato_a_mover

2.14.5.Instrucción SUB

Esta instrucción tiene la función de una resta, es decir, ejecuta la operación aritmética equivalente a una resta. Su sintaxis es la siguiente:

SUB destino, fuente

La operación se lleva de la siguiente manera, el dígito u operando fuente es restado del valor que contenga el dígito u operando destino, seguido se lleva a cabo la sobreescritura del resultado en la variable destino.

Ejemplo:

SUB AL, BL

2.15.Interrupciones

Una interrupción es el rompimiento en la secuencia de un programa para ejecutar un programa especial llamando una rutina de servicio cuya característica principal es que al finalizar regresa al punto donde se interrumpió el programa.

2.15.1.Interrupción 10H

Llamar a diversas funciones de video del BIOS, donde esta interrupción tiene diversas funciones, todas aquellas que nos permiten controlar la entrada y salida de video, la forma de acceso a cada una de las opciones es por medio del registro AH.

- Función 0H – Establece modo de video

Objetivo: establece el modo de video que se va a utilizar

Forma de llamada:

AH = 01H

Bits 0-4 de CH = Línea inicial del Cursor

Bits 0-4 de CL = Línea final del Cursor.

Registro de retorno: ninguno

Solo selecciona un nuevo tamaño de Cursor en modo texto.

- Función 02H – Selección de posición del cursor

Objetivo: Posiciona el cursor en la pantalla dentro de las coordenadas válidas de texto.

Forma de llamada:

AH = 02H

BH = Página de video en la que se posicionará el cursor.

DH = Fila

DL = Columna

Registros de retorno: Ninguno.

Las posiciones de localización del cursor son definidas por coordenadas iniciando en 0,0, que corresponde a la esquina superior izquierda hasta 79,24 correspondientes a la esquina inferior derecha. Por lo que los valores que pueden tomar los registros DH y DL en modo de texto de 80 x 25 son de 0 hasta 24 y de 0 hasta 79.

- Función 06H – Cambio a la página anterior activa

Objetivo: Desplaza las líneas de texto.

Forma de llamada:

AH = 06H

AL= Número de líneas que se va a desplazar. Si AL=, se borra toda la ventana seleccionada mediante los registros CX Y DX

BH = Atributo a usar en las líneas borradas

CH= Fila donde comienza la ventana de texto

CL= Columna donde comienza la ventana de texto

DH = Fila donde se acaba la ventana de texto

DL = Columna donde acaba la columna de texto

Registro de retorno: Ninguno

Este 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.

2.15.2.Interrupción 21H

Interrupción del dos para mostrar salidas en pantalla y aceptar entradas desde el teclado, por lo que decimos la función de esta interrupción provee los servicios funcionales especiales para el manejo de problemas de tipo input/ output.

La llamada a la INT 21H se realizará como sigue:

- Introducimos en (AH) el número de función a la que deseamos acceder.
- En caso de que deseemos acceder a una subfunción dentro de una función, debemos indicarlo introduciendo en (AL) el número de esa subfunción.
- Llamar a la INT 21H

- Función 01H – Entrada desde teclado

Objetivo: carácter del teclado y desplegarlo.

Forma de llamada: AH = 01H

Registros de retorno: AL = Carácter leído

Con esta función es muy sencillo leer un carácter del teclado, el código hexadecimal del carácter leído se guarda en el registro AL. En caso de que sea un carácter extendido el registro AL contendrá el valor de 0 y será necesario llamar de nuevo a la función para obtener el código de este carácter.

- Función 02H – Exhibe salida

Objetivo: Posiciona el cursor en la pantalla dentro de las coordenadas válidas de texto.

Forma de llamada:

AH = 02H

BH = Página de video en la que se posicionará el cursor.

DH = Fila

DL = Columna

Registros de retorno:Ninguno.

Las posiciones de localización del cursor son definidas por coordenadas iniciando en 0,0, que corresponde a la esquina superior izquierda hasta 79,24 correspondientes a la esquina inferior derecha. Por lo que los valores que pueden tomar los registros DH y DL en modo de texto de 80 x 25 son de 0 hasta 24 y de 0 hasta 79.

- Función 09H – Impresión de cadena (Video)

Objetivo: Desplegar un carácter un determinado número de veces con un atributo definido empezando en la posición actual del cursor.

Forma de llamada:

AH = 09H

AL = Carácter a desplegar

BH = Página de video en donde se desplegará

BL = Atributo a usar

CX = Número de repeticiones

Registros de retorno: Ninguno

Esta función despliega un carácter el número de veces especificado en CX, pero sin cambiar la posición del cursor en la pantalla.

- Función 0AH – Entrada desde el teclado usando buffer

Objetivo: Leer caracteres del teclado y almacenarlos en un buffer.

Forma de llamada:

AH = 0AH

DS: DX = Dirección del área de almacenamiento

BYTE 0 = Cantidad de bytes en el área

BYTE 1 = Cantidad de bytes leídos

desde BYTE 2 hasta BYTE 0 + 2 = caracteres leídos

Registros de retorno:Ninguno

Los caracteres son leídos y almacenados en un espacio predefinido de memoria. La estructura de este espacio le indica que en el primer byte del mismo se indican cuantos caracteres serán leídos. En el segundo byte se almacena el número de caracteres que ya se leyeron, y del tercer byte en adelante se escriben los caracteres leídos.

- Función 3CH – Crear archivo

Objetivo: Crear un archivo si no existe o dejarlo en longitud 0 si existe

Forma de llamada:

AH = 3CH

CH= Atributo de archivo

DS: DX = Apuntador a una especificación ASCIIZ

Registros de retorno:

CF = 0 y AX el número asignado al handle si no hay error, en caso de haberlo CF será 1 y AX contendrá el código de error: 3 ruta no encontrada, 4 no hay handles disponibles para asignar y 5 acceso negado.

Esta función sustituye a la 16H. El nombre del archivo es especificado en una cadena ASCIIZ, la cual tiene como característica la de ser una cadena de bytes convencional terminada con un carácter 0.

- Función 3DH – Abre archivo

Objetivo: Abre un archivo y regresa un handle

Forma de llamada:

AH = 3DH

AL= modo de acceso

DS: DX = Apuntador a una especificación ASCIIZ

Registros de retorno:

CF = 0 y AX = número de handle si no hay errores, de lo contrario CF = 1 y AX = código de error: 01H si no es válida la función, 02H si no se encontró el archivo, 03H si no se encontró la ruta, 04H si no hay handles disponibles, 05H en caso de acceso negado, y 0CH si el código de acceso no es válido. El handle regresado es de 16 bits.

- Función 3EH – Cierra manejador de archivo

Objetivo: cierra el archivo (Handle)

Forma de llamada:

AH = 3EH

BX= Handle asignado

Registros de retorno:

CF = 0 si no hubo errores, en caso contrario CF será 1 y AX contendrá el código de error: 06H si el handle es inválido.

Esta función actualiza el archivo y libera o deja disponible el handle que estaba utilizando.

- Función 3FH – Lectura desde dispositivo/Archivo

Objetivo: Leer información de un dispositivo o archivo.

Forma de llamada:

AH = 3FH

BX = Número asignado al dispositivo

CX = Número de bytes a procesar

DS: DX = Dirección del área de almacenamiento

Registros de retorno:

CF = 0 si no hay error y AX = número de bytes leídos.

CF = 1 si hay error y AX contendrá el código del error.

- Función 40H – Escritura en dispositivo/ Archivo

Objetivo: escribir a un dispositivo o archivo.

Forma de llamada:

AH = 40H

BX = vía comunicación

CX=cantidad de bytes

DS: DX = dirección del inicio de datos a escribir.

Registro con retornos:

CF = 0 si no hubo error -> AX = número de bytes escritos

CF= 1 si hubo error-> AX = Código de error

- Función 41H – Borrar Fichero

Objetivo: La eliminación de un archivo en nuestro disco duro

Forma de llamada:

AH = 41H

DS: DX = Segmento: Desplazamiento de la cadena
ASCII con el nombre del fichero a borrar.

Registro con retornos:

Si se ejecutó correctamente: Flag de acarreo (Cf) = 0

Si NO se ejecutó correctamente: Flag de acarreo (Cf) = 1

AX = Código de error.

Se borra el archivo mediante la cadena ASCII

- Función 4CH -Terminación de programa con Código de retorno

Objetivo: La finalización de la ejecución de un programa en ejecución y se devuelve un código de retorno al programa padre.

Forma de llamada:

AH = 4CH

AL= Código de retorno para el programa padre

Registro de retorno: ninguno

Finaliza la ejecución del programa en curso, y se devuelve un código de retorno al programa padre. Mediante este código de retorno, se puede ofrecer información al programa padre acerca de la ejecución del programa (si se ha producido error, etc....) La terminación del programa conlleva:

- Liberación de toda la memoria asignada al programa.
- Todos los buffers de fichero son vaciados.
- Se cierra cualquier fichero abierto por el programa.
- Se restauran los tres vectores de interrupción (INT 22H, INT 23H, INT 24H) cuyo contenido original fue almacenado en la pila.

Este es el método idóneo de terminación de programas, ya que no necesita que el registro CS tenga ningún contenido especial. Y aparte, devuelve información al programa padre.

2.16.Directivas

El lenguaje ensamblador permite usar diferentes enunciados que permiten controlar la manera en que un programa ensambla y lista. Estos enunciados llamados directivas, actúan solo durante el ensamblado de un programa y no generan código ejecutable de máquina. Las directivas pueden definir variables, macros y procedimientos. Pueden asignar nombres a los segmentos de memoria y realizar muchas otras tareas de mantenimiento relacionadas con el ensamblador

2.16.1.Directiva ASSUME

Este es una directiva que es la que se encarga de inicializar un registro de segmento asignado, así como su direccionamiento de cada uno de los registros del segmento dado. ASSUME utiliza al registro SS para direccionar a la pila, al registro DS para direccionar el segmento de datos y al registro CS para direccionar el segmento de código, para que después verifique cada acceso a una variable de memoria con el nombre valido, y es como sigue su sintaxis:

OPERACIÓN	OPERANDO
------------------	-----------------

ASSUME	SS:nom_SegPila, DS:nom_SegDatos, CS:nom_SegCodigo
--------	---

ASSUME es solo un mensaje que ayuda al ensamblador a convertir código simbólico a código maquina; aún puede tener que codificar instrucciones que físicamente cargan direcciones en registros de segmentos en el momento de su ejecución Ejemplo: (Anexo -9)

2.16.2.Directiva CALL

Es una instrucción de transferencia de control, en la cual hace una llamada a una subrutina con retorno, en la que esa llamada se encuentra la dirección de memoria

indicada por la etiqueta, este nos permite efectuar la transferencia de la ejecución a un subprograma, indicando las zonas de memoria compartida.

CALL puede llamar a un procedimiento, en la que consiste en dirigir al procesador para que empiece la ejecución en una nueva ubicación de la memoria, en la que mete su dirección de retorno de la pila y copia la dirección del procedimiento al que se llamó en el apuntador de instrucciones. Esta instrucción funciona de la mano en la instrucción RET. Ejemplo (Anexo-10)

2.16.3.Directiva de definición de datos

Las directivas convencionales usadas para definir datos, junto con los otros nombres introducidos por MASM 6.0 es la siguiente forma:

DESCRIPCIÓN	DIRECTIVAS CONVENCIONALES	DIRECTIVAS MASM 6.0
Definir byte(s)	DB	Byte
Definir una palabra	DW	Word

- **DB o Byte**

De las directivas que definen elementos de datos, una de las más útiles es DB (Definir Byte). Esta expresión numérica DB puede definir una o mas constantes de un byte, e. El máximo de un byte significa dos dígitos hexadecimales. Con el bit de más a la izquierda actuando como el de signo, el número hexadecimal más grande positivo de un byte es 7F; todos los números "superiores", del 80 al FF (en donde el bit de signo es 1), representan valores negativos. En términos de números decimales, estos límites son +127 y -128. E.

Una expresión de carácter DB puede contener una cadena de cualquier longitud, hasta el final de la línea.

- **DW o Dword**

La directiva DW define elementos con una longitud de una palabra (dos bytes). Una expresión numérica DW (o WORD) puede definir una o más

constantes de una palabra. El número hexadecimal positivo de una palabra es 7FFF; todos los números "superiores", desde 8000 hasta FFFF (donde el bit de signo es 1), representan valores negativos. En términos de números decimales, los límites son +32,767 y -32,768.

El ensamblador convierte constantes numéricas DW a código objeto binario (representado en hexadecimal), pero almacena los bytes en orden inverso. En consecuencia, un valor decimal definido como 12345 lo convierte a 3039 hex, pero es almacenado como 3930.

Una expresión de caracteres DW está limitada a dos caracteres, que el ensamblador invierte en el código objeto, así que 'P C' se convertiría en 'CP'. Si piensa que DW es de uso limitado para la definición de cadenas de caracteres, está en lo correcto.

2.16.4.Directiva END

Esta directiva es la encargada de señalar la parte en la que termina un segmento, es decir, es la directiva encargada de marcar el fin de un segmento previamente inicializado, debido que todo segmento inicializado debe de ser finalizado. Este debe ser colocada al final del programa fuente. Ejemplo (Anexo-11)

2.16.5.Directiva ENDS

Esta directiva es la encargada de señalar la parte en la que termina un segmento, es decir, es la directiva encargada de marcar el fin de un segmento previamente inicializado, debido que todo segmento inicializado debe de ser finalizado. Ejemplo (Anexo-12)

2.16.6.Directiva ENDP

Es la directiva que indica el fin de un procedimiento y contiene el mismo nombre que el enunciado PROC para permitir que es ensamblador relaciones a los dos, ya que los procedimientos deben estar por completo dentro de un segmento, y se

define al final de un procedimiento antes que ENDS defina el final del segmento.
Ejemplo:(Anexo-13)

2.16.7.Directiva INCLUDE

Esta directiva desempeña una función muy importante, al cual es permitirnos añadir librerías, funciones o archivos (.txt) que se encuentran en otros ficheros a nuestro programa principal.

Sintaxis:

INCLUDE nombre_archivo_o_libreria

Ejemplo. (Anexo-14)

2.16.8.Directiva PROC

El segmento de código contiene el código ejecutable de un programa. También tiene uno o más procedimientos, definidos con la directiva PROC. Un segmento que tiene sólo un procedimiento puede aparecer como sigue:

NOMBRE	OPERACIÓN	OPERANDO
nombreProc	PROC	FAR

El nombre del procedimiento debe estar presente, ser único y seguir las reglas para la formación de nombres del lenguaje. El operando FAR en este caso está relacionado con la ejecución del programa. Ejemplo (Anexo-13)

2.16.9.Directiva SEGMENT

Este es una directiva que define el inicio de un segmento o más segmentos. Este puede definir el segmento de pila que define el almacén de la pila, un segmento de datos, en donde se define los elementos de datos, y el segmento de código en el que se define y proporciona un código ejecutable, en general es un bloque de

NOMBRE	OPERACIÓN	OPERANDO	COMENTARIO
nombre	SEGMENT	[opciones];	inicia el segmento

sentencias que puede contener definiciones de variables o instrucciones. La etiqueta precediendo a la directiva SEGMENT es el nombre del segmento, y tiene el siguiente formato:

El nombre del segmento debe de estar presente, ser único y cumplir con las convenciones para los nombres del lenguaje. Esta directiva puede especificar los atributos que son asignados al ese segmento, para que después se le den instrucciones al ligador y al ensamblador de como establecer y combinar los segmentos. Ejemplo: (Anexo-15)

2.16.10.Directiva RET

Es una instrucción de transferencia de control, en la cual hace un retorno de una subrutina, en la que retorna una dirección cargada de la pila, que fue llamada previamente con la instrucción CALL. Su ejecución el programa continuo desde la dirección formada al extraer 2 bytes de la pila. En el primer lugar de la pila se saca el byte más significativo. El Stack Pointer (puntero de pila) utiliza durante el RET un esquema de post-decremento. Ejemplo: (Anexo-16)

2.17.Operadores

Un operador proporciona una facilidad para cambiar o analizar operandos durante un ensamblador. Los operandos están divididos en varias categorías:

- Operadores de cálculo: Aritméticos, índice, lógicos, desplazamiento y nombre de escritura de campo
- Operadores de registro: MASK Y WIDTH con la directiva RECORD
- Operadores relacionales: EQ, GE, GT, LE, LT Y NE
- Operadores de segmento: OFFSET, SEG y pasar por alto el segmento
- Operadores de tipo (o atributo): HIGH, HIGHWORD, LENGTH, LOW, LOWWORD, PTR, SHORT, SIZE, THIS y TYPE

Los que usamos en la realización de este proyecto son los siguientes:

2.17.1. Operador OFFSET

El operador OFFSET regresa la dirección de desplazamiento (esto es, la dirección relativa dentro del segmento de datos o del segmento de código) de una variable o etiqueta. El formato general es el siguiente:

OFFSET variable o etiqueta

El desplazamiento representa la distancia (en bytes) de la etiqueta, a partir del inicio del segmento de datos, en la siguiente figura se muestra una variable llamada mybytes dentro del segmento de datos. En el modo protegido, los desplazamientos son de 32 bits. (Anexo-17) En el modo de direccionamiento real, los desplazamientos son de 16 bits. (Anexo-18)

2.17.2. Operador \$

Indica la posición del contador de posiciones («Location Counter») utilizado por el ensamblador dentro del segmento para llevar la cuenta de por dónde se llega ensamblando. En general indica el fin de una cadena. Ejemplo: (Anexo-19)

2.18. Saltos

Los saltos son aquellas instrucciones que permiten al programador cambiar el orden de ejecución del programa según sea conveniente, dentro del ensamblador existen dos saltos principales, tanto condicionales como incondicionales.

2.18.1. Saltos condicionales

Como su nombre lo menciona, son saltos que transfieren el control del programa a la ubicación que se les dé como parámetro, al hacer una comparación y se cumple tal condición establecida en el salto.

- **CMP**

Esta instrucción por lo común es utilizada para comparar dos campos de datos, uno o ambos de los cuales están contenidos en un registro. Su sintaxis es la siguiente:

CMP [registro/memoria], [registro/memoria/inmediato]

Si los operandos son iguales entonces pone a la bandera ZF en cero ($ZF=0$), si los operandos son diferentes pone a ZF en uno ($ZF=1$). Ejemplo: (Anexo-20)

- JA

Propósito: Salto condicional

Sintaxis: JA Etiqueta

Después de una comparación este comando salta si es mayor o salta si no es menor o igual. Lo que significa que el salto se realiza solo si la bandera CF esta desactivada o si la bandera ZF esta desactivada (que alguna de las dos sea igual a cero). Ejemplo: (Anexo-21)

- JB

Propósito: salto condicional

Sintaxis: JB etiqueta

Salta si está abajo o salta si no está arriba o si no es igual. Se efectúa el salto si CF esta activada. Ejemplo: (Anexo-21)

- JE

Propósito: salto condicional

Sintaxis: JE etiqueta

Salta si es igual o salta si es cero. El salto se realiza si ZF está activada. Ejemplo: (Anexo-20)

- JC

Propósito: Salto condicional, se toma en cuenta el estado de las banderas

Sintaxis: JC etiqueta

Salta si hay acarreo. El salto se realiza si $CF = 1$. Ejemplo: (Anexo-22)

2.18.2.Saltos incondicionales

Son utilizados mediante la instrucción JMP, la cual transfiere el control a la línea especificada después de escribir tal palabra, la cual puede ser un valor directo o una etiqueta.

- Instrucción JMP

Propósito: Salto incondicional

Sintaxis: JMP destino

Esta instrucción se utiliza para desviar el flujo de un programa sin tomar en cuenta las condiciones actuales de las banderas ni de los datos, usada comúnmente para la transferencia de control. Cuando la CPU ejecuta una transferencia incondicional, el desplazamiento destino (a partir del segmento de código) se mueve hacia el apuntador de instrucciones, lo cual provoca que la ejecución continúe en su nueva ejecución. Bajo circunstancias normales, solo se puede saltar a una etiqueta dentro del procedimiento actual. Ejemplo: (Anexo-23)

2.19.Ciclos

Instrucción JMP (JuMP)

Sintaxis:

JMP direc

JMP etiqueta

- Realiza un salto de ejecución incondicional hacia la dirección o etiqueta especificada.

- Ejemplos:

JMP 100H; Salta a CX: 100h

JMP 55AAH:100H; Salto lejano a otro segmento

JMP WORD PTR [BX]; Salto a la dirección contenida en
; La dirección de memoria especificada
; Por BX (salto indirecto)

JMP REPITE; Salto a la etiqueta REPITE

Instrucciones CALL Y RET (RETurn)

Sintaxis:

```
CALL    direc ; Saltar a direc
        ...; Dirección de retorno
        ...
direc   ... ; Dirección de salto
        ...
RET
```

Ejemplo: (Anexo-24)

- Realiza un salto incondicional hacia la dirección, etiqueta o procedimiento especificado. A diferencia de la instrucción JMP, la instrucción CALL realiza un salto a una subrutina con retorno. El salto puede ser cercano o lejano.
- En el primer caso, la dirección a la que salta corresponde al offset dentro del segmento de código actual, por lo que, antes de realizar el salto, CALL guarda en la pila el contenido de IP, el cual apunta a la instrucción inmediatamente después de la instrucción CALL.
- En el segundo caso, la dirección a la que salta corresponde a un offset dentro de otro segmento de código, por lo que, antes de realizar el salto, CALL guarda en la pila el contenido de CS e IP (CS: IP), el cual apunta a la instrucción inmediatamente después de la instrucción CALL.
- Una vez realizado el salto, se ejecutarán las instrucciones que allí hubiera hasta encontrar la sentencia RET, la cual extrae de la pila la dirección de retorno almacenada con CALL.

2.20.Macros

Una macro es un nombre que define un conjunto de instrucciones que serán sustituidas por la macro cuando el nombre de ésta aparezca en un programa (proceso denominado expansión de macros) en el momento de ensamblar el programa. Los macros (Procedimientos) se definen de manera directa al principio de un programa de código fuente, en donde las instrucciones de macros se pueden

guardar en el programa mismo o en un archivo separado que el programa pueda identificar, y se copian en un programa mediante una directiva INCLUDE.

Una macroinstrucción es una instrucción compleja, formada por otras instrucciones más sencillas. Esto permite la automatización de tareas repetitivas. Además, tiene que estar almacenada, el término no se aplica a una serie de instrucciones escritas en la línea de comandos enlazadas unas con otras por redirección de sus resultados o para su ejecución consecutiva. En cada punto en el que se hace una llamada al macro, el ensamblador inserta una copia de su código fuente en el programa.

Definición de macros

Un macro se define usando las directivas MACRO y ENDM la sintaxis en la siguiente (Anexo-25):

```
Nombre_macro MACRO parámetro-1, parámetro-2...  
Lista-de-instrucciones  
ENDM
```

Parámetros: Los parámetros de las macros son contenedores para los argumentos de texto que se pasan a la macro que hace la llamada. De hecho, los argumentos pueden ser enteros, nombres de variables y otros valores, pero el preprocesador los trata como texto.

Invocación de macros:

Para llamar a una macro, se inserta su nombre en el programa, posiblemente seguido de los argumentos de una macro, su sintaxis es la siguiente: (Anexo-26)

```
Nombre_macro argumento-1, argumento-2
```

Ventajas:

- Menor posibilidad de cometer errores por repetición.
- Mayor flexibilidad en la programación al permitir el uso de parámetros.
- Código fuente más compacto.
- Al ser más pequeño el código fuente, también es más fácil de leer por otros.

Desventajas:

- El código ejecutable se vuelve más grande con cada llamada a la macro.
- Las macros deben ser bien planeadas para evitar la redundancia de código.

2.21.Procedimientos

El procedimiento es un conjunto de instrucciones que realizan una tarea y preferentemente solo una que se ha de utilizar en más de una ocasión, pero se declara una sola vez en el código fuente.

Un procedimiento puede llamar a otro, y este a su vez a otro y así sucesivamente.

El procedimiento se guarda en memoria cuando se ensambla y ejecuta y entonces puede ser llamado tantas veces como sea necesario, ahorrando espacio y facilitando el desarrollo de software gracias a que permite organizarlo. Los procedimientos deben ser cortos, de no más de una o dos páginas.

Los procedimientos en ensamblador se declaran mediante la sintaxis `nombreprocedimiento Proc [far/near]` dependiendo de si es un procedimiento cercano o lejano. Ejemplo: (Anexo-13)

2.22.Mnemotécnicos

Son aquellos que se utilizan principalmente para recordar una secuencia de datos, nombre, números, y en general para recordar listas de ítems que no pueden recordarse fácilmente, con esto se obtiene una reducción de líneas en nuestro código.

A la par que los mnemotécnicos se utilizan para especificar un código de operación que representa una instrucción de lenguaje máquina completa y operativa, posteriormente el ensamblador es el que lo traduce para generar el código objeto.

3. CONCLUSIÓN

En conclusión, este trabajo menciona cada uno de los conceptos implicados en el desarrollo de nuestro proyecto, con el objetivo de conseguir una retroalimentación de los temas, o en dado caso si se desconocen los conceptos, se lleguen a comprender los mismos. Este trabajo formal se elaboró con la intención de tener un sustento teórico de todas aquellas técnicas, conceptos y elementos del lenguaje ensamblador, las cuales, a lo largo del desarrollo o la codificación del mismo proyecto, se usaron e implementaron. A la par que se plateó implementar operaciones con archivos, lo que tuvo como consecuencias la indagación sobre cómo llevar a cabo las operaciones CRUD en archivos los cuales tienen extensión .txt, consiguiendo así el aprendizaje necesario para aplicarlo en cualquier momento que sea requerido. Además, que se puede resaltar el importante trabajo en equipo, ya que cada uno de los integrantes aportó conocimiento, ideas, esfuerzo para así poder entregar un trabajo con un buen nivel de profesionalismo, esto mismo nos da un claro ejemplo de cómo es el ambiente del campo laboral, permitiéndonos coleccionar experiencia, la cual en un futuro será de gran ayuda cuando cada uno de los integrantes del equipo, ejerza esta maravillosa profesión.

4. REFERENCIAS BIBLIOGRÁFICAS

- EcuRed. (s. f.). *Lenguaje ensamblador - EcuRed*. Recuperado 18 de octubre de 2022, de https://www.ecured.cu/Lenguaje_ensamblador
- Juárez Fuentes, J. (s. f.). *Lenguaje ensamblador. APUNTES DE LENGUAJE ENSAMBLADOR*. Recuperado 18 de octubre de 2022, de https://www.utm.mx/~jjf/le/LE_APENDICE_D.pdf
- Alegrechi, D., & Almiron, E. (2014, October). *Introducción al entorno emu8086*. Retrieved October 24, 2022, from https://www.dsi.fceia.unr.edu.ar/images/downloads/digital_II/Introduccion_emu8086_v1.4.pdf
- V., G. (2021, December 02). *Sistemas numéricos: Qué Son, para qué sirven, características, Tipos*. Retrieved October 30, 2022, from <https://www.euston96.com/sistemas-numericos/#:~:text=%C2%BFQu%C3%A9%20son%20los%20sistemas%20num%C3%A9ricos%3F%20Los%20sistemas%20num%C3%A9ricos,que%20est%C3%A1%20definido%20por%20la%20base%20que%20utiliza>.
- Desconocido. (s. f.). *Significado de Sistema Binario*. Retrieved October 30, 2022, from <https://www.significados.com/sistema-binario/#:~:text=El%20sistema%20binario%20es%20un%20sistema%20de%20numeraci%C3%B3n,textos%2C%20datos%20y%20programas%20ejecutables%20en%20dispositivos%20inform%C3%A1ticos>
- Briceño V., G. (2021, December 02). *Sistema decimal: Qué Es, para qué sirve, Características, ejemplos*. Retrieved October 30, 2022, from <https://www.euston96.com/sistema-decimal/>

- OKDIARIO. (2022, September 29). Sistema hexadecimal: Qué Es y funcionamiento. Retrieved October 30, 2022, from <https://okdiario.com/curiosidades/como-funciona-sistema-hexadecimal-3556445#:~:text=El%20sistema%20hexadecimal%2C%20o%20sistema%20num%C3%A9rico%20hexadecimal%2C%20es,16%20s%C3%ADmbolos%20para%20marcar%20un%20n%C3%BAmero%2C%20que%20son%3A>
- Abel, P. (1996). Lenguaje Ensamblador y programación para IBM, PC y compatibles (3a. ed.). México: Pearson.
- Edith Gomez, A. D. (2013). Tipos de ensambladores. Retrieved November 2, 2022, from <https://informatica4194.webnode.mx/contactanos/tipos-de-ensambladores/>
- Desconocido. (2013, May 23). Comentarios en Lenguaje ENSAMBLADOR. Retrieved November 5, 2022, from <https://conocimientosweb.net/dcmt/ficha1524.html>
- Desconocido. (s.f.). Programador CLIC. Retrieved November 5, 2022, from <https://programmerclick.com/article/31591260164/>
- Desconocido. (2013, June 04). Identificadores en Lenguaje ENSAMBLADOR. Retrieved November 5, 2022, from <https://conocimientosweb.net/dcmt/ficha10062.html#:~:text=Un%20identificador%20es%20un%20nombre,la%20direcci%C3%B3n%20de%20una%20instrucci%C3%B3n.> (Accessed: October 30, 2022).
- Mac, L. S. (2022, July 30). ¿Qué es una etiqueta en lenguaje ensamblador? Retrieved October 23, 2022, from [https://www.centrobanamex.com.mx/que-es-una-etiqueta-en-lenguaje-ensamblador#:~:text=%C2%BFQu%C3%A9%20significa%20lea%20en%](https://www.centrobanamex.com.mx/que-es-una-etiqueta-en-lenguaje-ensamblador#:~:text=%C2%BFQu%C3%A9%20significa%20lea%20en%20)

20ensamblador%3F%20La%20instrucci%C3%B3n%20LEA%28Load,de%20memoria%20designada%20en%20el%20registro%20de%20destino.

- Las instrucciones del ensamblador. (s. f.-a). Recuperado 24 de octubre de 2022, de <https://moisesrbb.tripod.com/unidad5.htm>
- INT 21H. (s. f.). Recuperado 24 de octubre de 2022, de http://ict.udlap.mx/people/oleg/docencia/ASSEMBLER/asm_interrup_21.html
- Int 21h y 10h. (2022, 22 septiembre). pdfslide.tips. <https://pdfslide.tips/documents/int-21h-y-10h-563b87818f4d6.html?page=9>
- Vidal, S. E. R. (s. f.). Interrupci3n 21h y 10h en lenguaje ensamblador. prezi.com. Recuperado 24 de octubre de 2022, de <https://prezi.com/p/5znpqcjwhj-c/interrupcion-21h-y-10h-en-lenguaje-ensamblador/>
- Desconocido. (2018, November 12). 2.6 saltos. Retrieved October 24, 2022, from <https://ittlenguajesdeinterfaz.wordpress.com/2-6-saltos/>
- Author. (2020, August 24). ¿Qué utilidad tiene la directiva include? Retrieved October 23, 2022, from <https://respuestasrapidas.com.mx/que-utilidad-tiene-la-directiva-include/#:~:text=%C2%BFQu%C3%A9%20utilidad%20tiene%20la%20directiva%20include%3F%20Para%20ello,de%20dos%20maneras%20%28siempre%20antes%20de%20las%20declaraciones%29.>
- Desconocido. (n.d.). 5.2 constantes y operadores. - el lenguaje ensamblador del 8. Retrieved October 30, 2022, from <https://1library.co/article/constantes-operadores-lenguaje-ensamblador.q7xjl3vy#:~:text=%C2%84%20Operador%20%27%24%27%3>

A%20indica%20la%20posici%C3%B3n%20del%20contador,de%20posicion
es%20%28%C2%ABLocation%20Counter%C2%BB%29%20utilizado%20p
or%20el%20ensamblador

- Santin Seguir Estudiante, C. (2020, July 29). Call ENSAMBLADOR 3. Retrieved October 30, 2022, from <https://es.slideshare.net/CesarSantrn/call-ensamblador-3>
- Unknown. (1970, January 01). Mnemónicos de Lenguaje ENSAMBLADOR. Retrieved October 23, 2022, from <https://guillermomirelesg.blogspot.com/2016/10/mnemonicos-de-lenguaje-ensamblador.html>
- Tijerina, I. (2016, May 04). Macros en ensamblador. Retrieved October 24, 2022, from <https://isaacantonio95.wixsite.com/tiai95/single-post/2016/05/03/macros-en-ensamblador>
- Desconocido, (n.d.). Procedimientos y macros en Lenguaje Ensamblador. Retrieved October 24, 2022, from <https://www.monografias.com/docs/Procedimientos-Y-Macros-En-Lenguaje-Ensamblador-FKAL9CYMZ>

5. ANEXOS

Anexo-1

Emulador utilizado para la realizar el proyecto integrador.



Anexo-2

Código ASCII.

HEX	CARÁCTER	HEX	CARÁCTER	HEX	CARÁCTER
00	(Nulo)	01	Carita sonriente	02	Carita sonriente
03	Corazón	04	Diamante	05	Club
06	Espada	07	(Beep)	08	(Retroceso)
09	(Tabulador)	0A	(Avance de línea)	0B	(Tab vertical)
0C	(Avanza página)	0D	(Return)	0E	(Shift out)
0F	(Shift in)	10	(Línea de datos esc)	11	(Dev ctl 1)
12	(Dev ctl 2)	13	(Dev ctl 3)	14	(Dev ctl 4)
15	(Reconocimiento neg)	16	(Sincr. ociosa)	17	(Fin de trans. de bloque)

Anexo-3

Registro DS que representa el segmento de datos, y el registro CS que representa el segmento de código.

```
DS:DATA, CS: CODE ;
```

Anexo-4

Registro SI utilizado como el contador y llevar el registro de índice.

```
MOV ruta [si],al
```


Anexo-5

Comentario en ensamblador.

```
;Fin del segmento de datos
```

Anexo-6

Definición de variables DB en el segmento de datos.

```
;cadenas para el submenu  
cadeSub0 DB 'Que desea hacer?','$'  
cadeSub1 DB '1.- Retroceder','$'  
cadeSub2 db '2.- Salir','$'  
cadePedSub DB 'R=','$'  
cadeNumSub DB
```

Anexo-7

Definición de variables DW en el segmento de datos.

```
nombre DB 13  
handle DW 0D  
aux DW 0d ;  
var DB 50d ;
```

Anexo-8

Ejemplo de una Etiqueta.

```
;Etiqueta para la seccion del menu principal  
Menu_principal:  
    macroColor 01101111B, 00D, 00D, 25D, 79D  
    macroPosiciones 01D, 21D, CADE  
    macroPosiciones 04D, 01D, CADE2  
    macroPedirDatos 09d, 00D, cadePedirNumIndice, numIndic  
    JMP Comparaciones_principales
```

Anexo-9

Directiva ASSUME que inicializa los segmentos de datos y de código.

```
ASSUME DS:DATA, CS: CODE
```

Anexo-10

Directiva CALL que llama a un procedimiento cercano.

```
CALL Ciclo_ar
```

Anexo-11

Directiva END que finaliza un segmento.

```
END INICIO ;Fin del segmento de código
```

Anexo-12

Directiva ENDS que finaliza el segmento de código.

```
CODE ENDS
```

Anexo-13

Directiva PROC y ENDS que definen el inicio y el final del procedimiento

```
-----  
; PROCESO CERCANO  
;proceso que realiza la llamada al ciclo que guarda el nombre del archivo  
CICLOP PROC NEAR  
    MOV SI,3d;mueve el puntero a la posición 3  
    CALL Ciclo_ar ;llamamos a la etiqueta Ciclo_Ar  
    RET  
CICLOP ENDP  
-----
```

Anexo-14

Directiva Include que llama a un archivo .txt en la cual se tienen las macros a utilizar

```
include macrosProyecto.txt
```

Anexo-15

Directiva SEGMENT que inicializa el segmento de datos

```
;Inicio del segmento de datos  
DATA SEGMENT
```

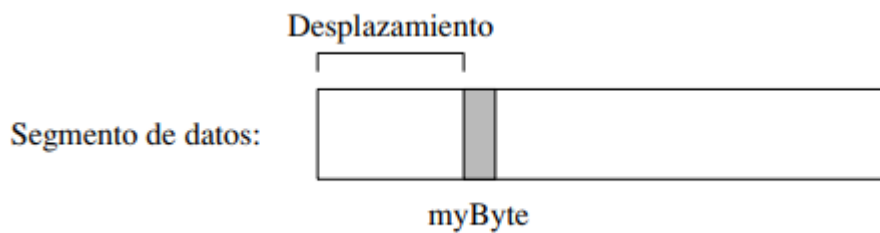
Anexo-16

Directiva Ret que retorna un registro

```
FinCiclo:
    MOV ruta[si],0d
    ret ; nos mueve
```

Anexo-17

Como se desplaza el offset al momento de utilizarlo



Anexo-18

Operador OFFSET que imprime una cadena deseada

```
MOV AH, 09H
MOV DX, OFFSET cadena
```

Anexo-19

Operador \$ que indica el fin de una cadena

```
;Cadena de mensaje para la bota
Mensaje4 db 'Bota de navidad :D $'
```

Anexo-20

Salto CMP que realiza la comparación entre un registro y un dato.

```
CMP numArchi, 0D
JE Opcion_incorrecta
```

Anexo-21

Uso de los saltos condicionales JA, JB y JE en la primera opción del proyecto

```
macroDefinirDatos 020, 020, code4, no
CMP num1, a1 ; compara el numero
JA EsMayor ; es mayor el num 1
JB EsMenor ; es menor el num2
JE EsIgual ; son iguales
```

Anexo-22

Uso del salto JC para ver si hay acarreos

```
JC Salir ;si hubo error
```

Anexo-23

Salto incondicional que nos dirige al submenu

A screenshot of the JMP software interface showing the 'JMP Submenu' with various options like 'Open', 'Save', 'Print', etc.

Anexo-24

Ciclo implementado en las operaciones de archivos

```

;Etiqueta para el ciclo utilizado en los archivos
Ciclo_ar:
    MOV AH,1H    ; Se mueve 1h para realizar la entrada del teclado
    INT 21H
    CMP AL,0dh   ;si ingresa enter finaliza el ciclo
    JE FinCiclo ; llamamos a finalizar ciclo
    CMP SI,16
    JE FinCiclo

    MOV ruta[si],al ;ingresamos un caracter a la vez
    INC si
    JB Ciclo_ar

```

Anexo-25

Estructura de una macro

```
;MACRO PARA DAR COLOR
macroColor MACRO colorFondo,a, x, b, y ;a=inicio colum, b=fin colum, x = inicio fila, y= fin fila
    MOV AX, 0600H
    MOV BH, colorFondo
    MOV CH, a
    MOV CL, x
    MOV DH, b
    MOV DL, y
    INT 10H
ENDM
```

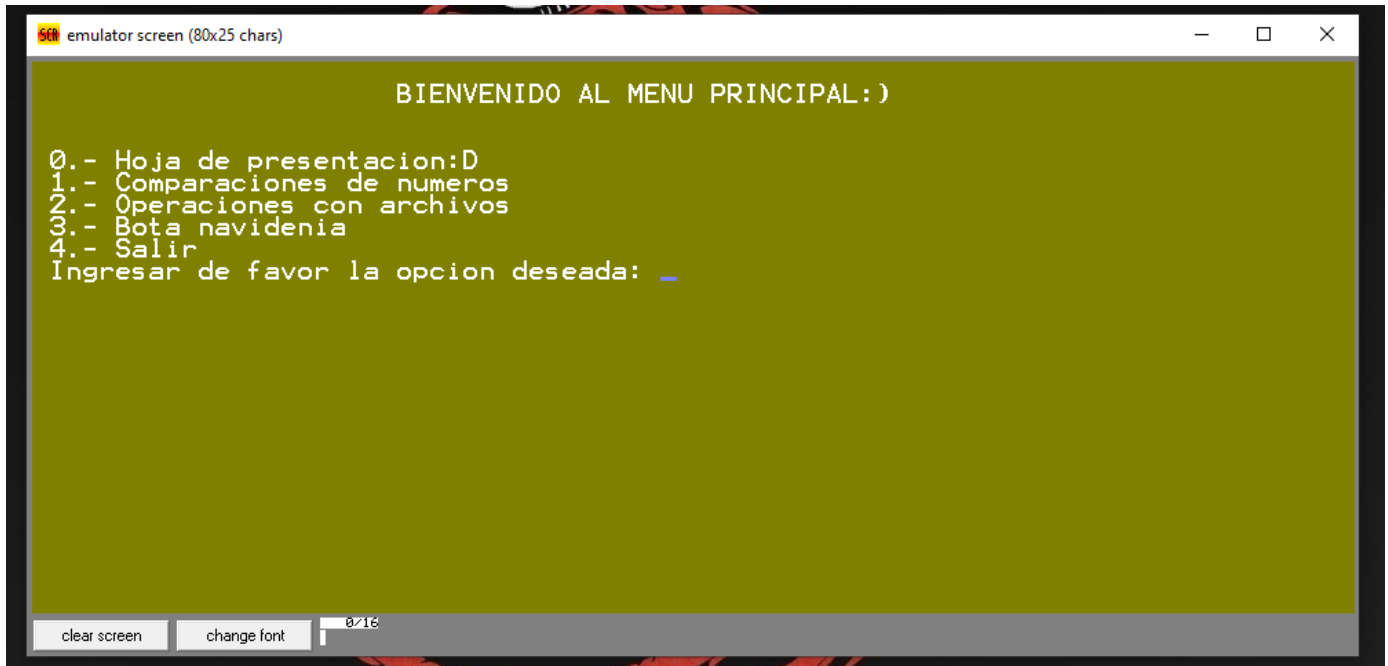
Anexo-26

Llamada a una macro

```
macroColor 00001111B, 00D, 62D, 25D, 79D
```

6. CAPTURAS DE LA EJECUCIÓN DEL CÓDIGO

Pantalla que muestra el menú principal, con las opciones principales



The screenshot shows a window titled "emulator screen (80x25 chars)". The main area has a green background with white text. The text reads: "BIENVENIDO AL MENU PRINCIPAL:)", followed by a list of options: "0.- Hoja de presentacion:D", "1.- Comparaciones de numeros", "2.- Operaciones con archivos", "3.- Bota navidenia", and "4.- Salir". Below the list, it says "Ingresar de favor la opcion deseada: _". At the bottom of the window, there are two buttons: "clear screen" and "change font", and a small status bar showing "0/16".

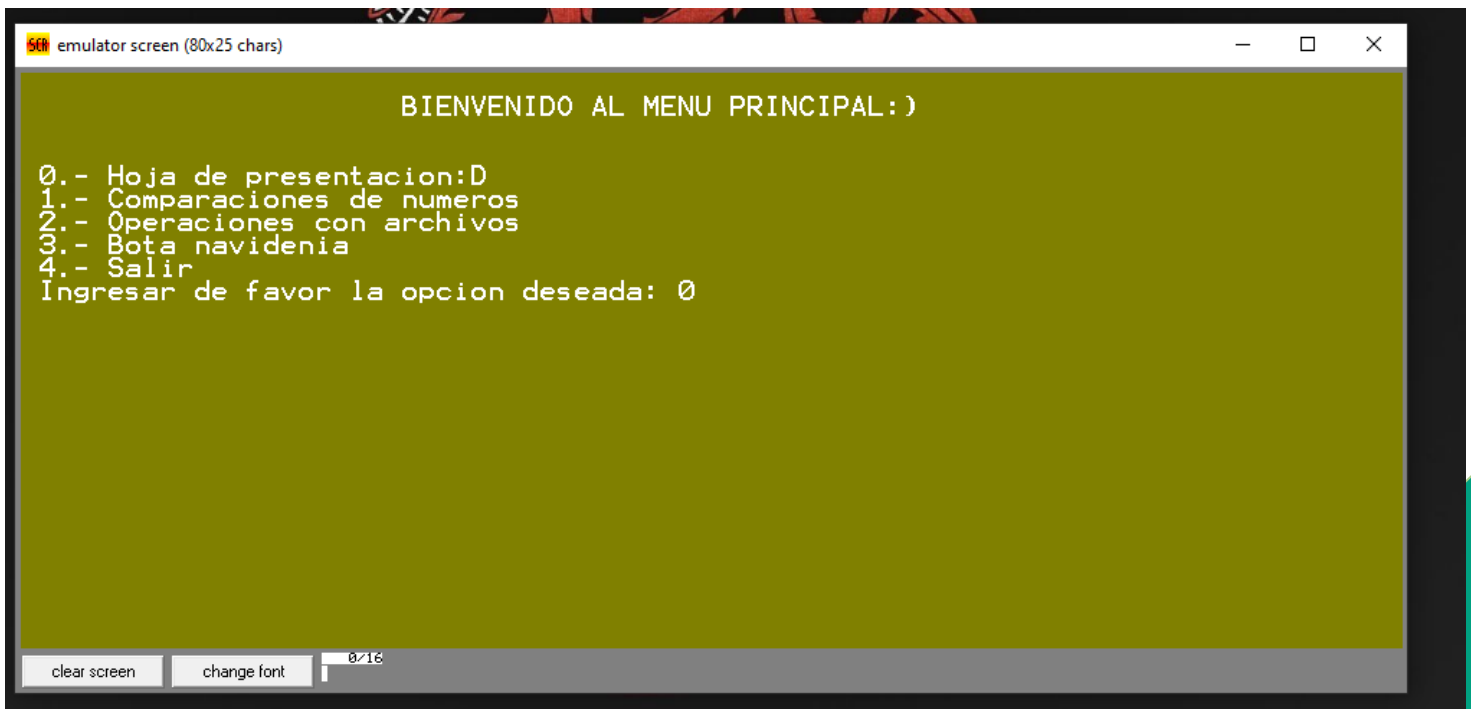
```
emulator screen (80x25 chars)

BIENVENIDO AL MENU PRINCIPAL:)

0.- Hoja de presentacion:D
1.- Comparaciones de numeros
2.- Operaciones con archivos
3.- Bota navidenia
4.- Salir
Ingresar de favor la opcion deseada: _

clear screen  change font  0/16
```

Se ingresa a la opción número cero del menú principal



The screenshot shows the same emulator window as before, but the text "Ingresar de favor la opcion deseada:" is now followed by "0". The rest of the menu is unchanged. The status bar at the bottom still shows "0/16".

```
emulator screen (80x25 chars)

BIENVENIDO AL MENU PRINCIPAL:)

0.- Hoja de presentacion:D
1.- Comparaciones de numeros
2.- Operaciones con archivos
3.- Bota navidenia
4.- Salir
Ingresar de favor la opcion deseada: 0

clear screen  change font  0/16
```

Se muestra la hoja de presentación, con los nombres de los integrantes del equipo



Regresamos al menú principal



Ingresamos la opción número uno, que es la comparación de dos números:



A screenshot of a terminal window titled "scm emulator screen (80x25 chars)". The background is olive green. The text is white and reads: "BIENVENIDO AL MENU PRINCIPAL:)", followed by a list of options: "0.- Hoja de presentacion:D", "1.- Comparaciones de numeros", "2.- Operaciones con archivos", "3.- Bota navidenia", "4.- Salir", and "Ingresar de favor la opcion deseada: 1". At the bottom, there are buttons for "clear screen" and "change font", and a font size indicator "0/16".

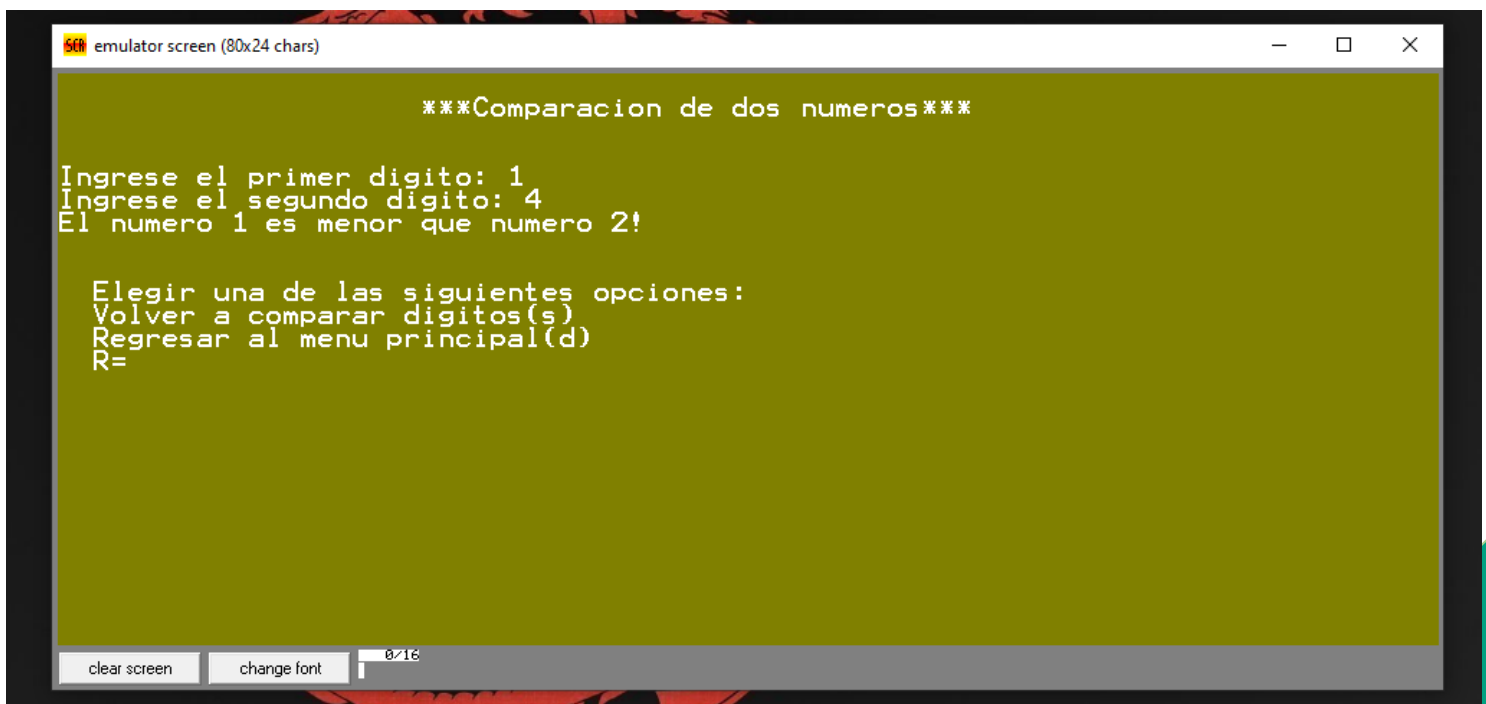
```
scm emulator screen (80x25 chars)

BIENVENIDO AL MENU PRINCIPAL:)

0.- Hoja de presentacion:D
1.- Comparaciones de numeros
2.- Operaciones con archivos
3.- Bota navidenia
4.- Salir
Ingresar de favor la opcion deseada: 1

clear screen  change font  0/16
```

Se realiza la comparación de dos números, con la elección de los números del uno-nueve



A screenshot of a terminal window titled "scm emulator screen (80x24 chars)". The background is olive green. The text is white and reads: "***Comparacion de dos numeros***", "Ingresa el primer digito: 1", "Ingresa el segundo digito: 4", "El numero 1 es menor que numero 2!", "Elegir una de las siguientes opciones:", "Volver a comparar digitos(s)", "Regresar al menu principal(d)", and "R=". At the bottom, there are buttons for "clear screen" and "change font", and a font size indicator "0/16".

```
scm emulator screen (80x24 chars)

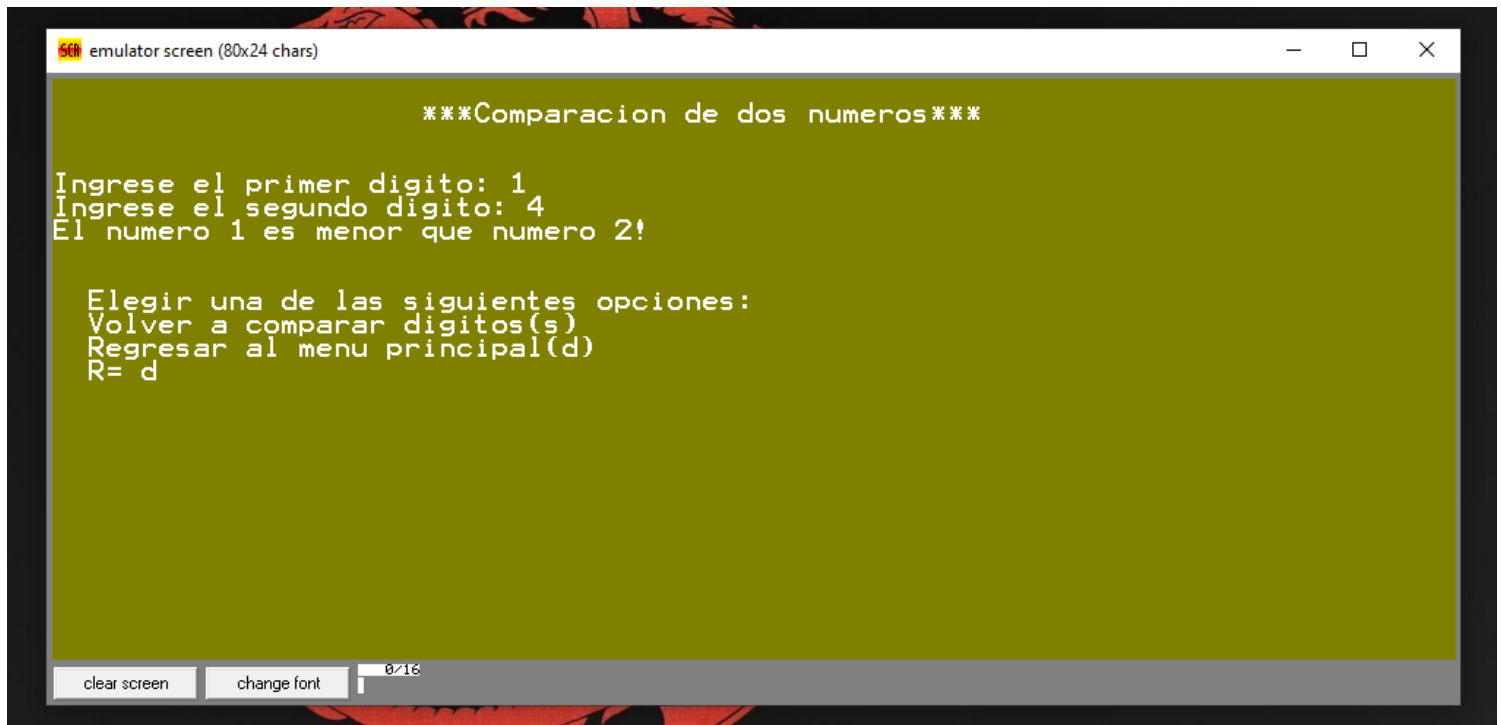
***Comparacion de dos numeros***

Ingresa el primer digito: 1
Ingresa el segundo digito: 4
El numero 1 es menor que numero 2!

Elegir una de las siguientes opciones:
Volver a comparar digitos(s)
Regresar al menu principal(d)
R=

clear screen  change font  0/16
```


Se digita d para regresar al menú principal

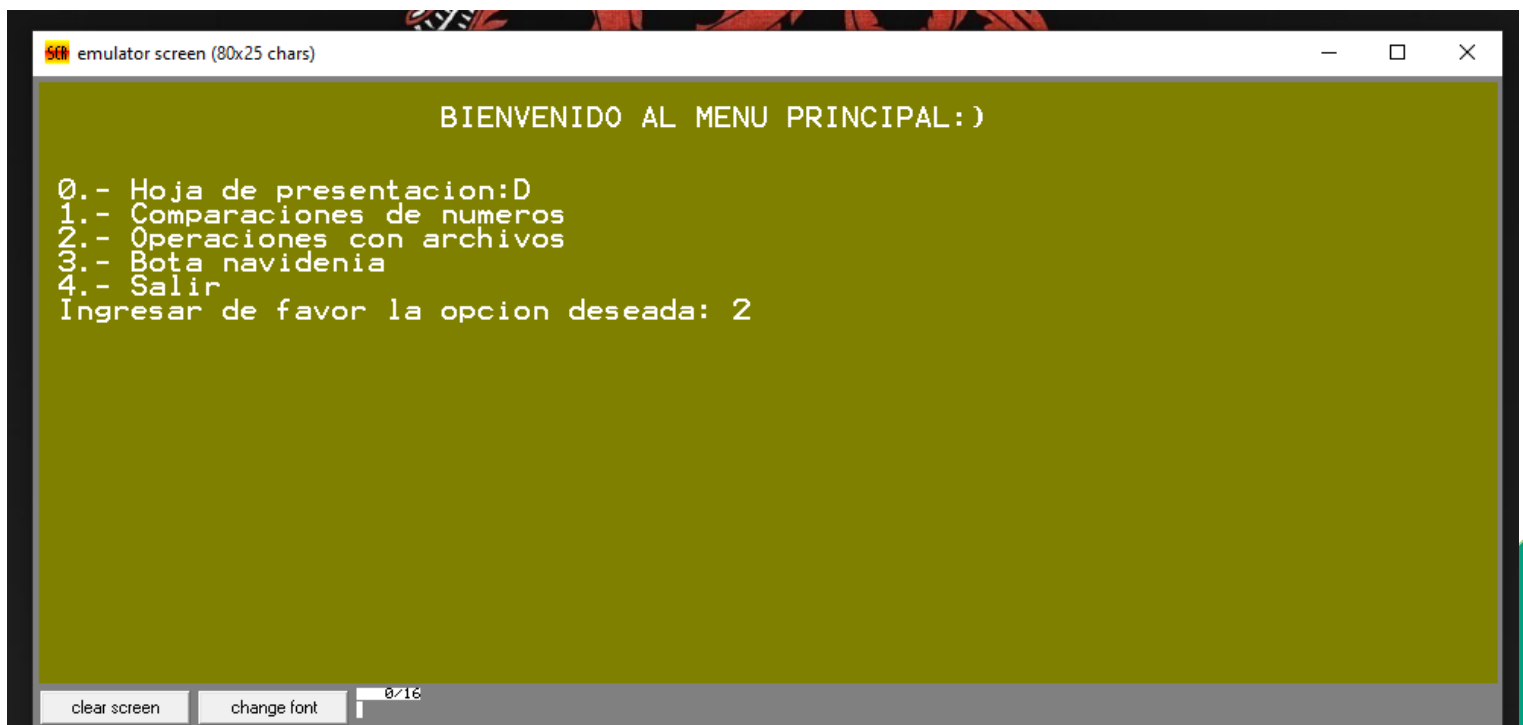


The screenshot shows a terminal window titled "emulator screen (80x24 chars)". The background is olive green. The text is as follows:

```
***Comparacion de dos numeros***  
  
Ingrese el primer digito: 1  
Ingrese el segundo digito: 4  
El numero 1 es menor que numero 2!  
  
Elegir una de las siguientes opciones:  
Volver a comparar digitos(s)  
Regresar al menu principal(d)  
R= d
```

At the bottom, there are two buttons: "clear screen" and "change font", followed by a font size indicator "0/16".

Se ingresa a la opción numero dos




The screenshot shows a terminal window titled "emulator screen (80x25 chars)". The background is olive green. The text is as follows:

```
BIENVENIDO AL MENU PRINCIPAL:)  
  
0.- Hoja de presentacion:D  
1.- Comparaciones de numeros  
2.- Operaciones con archivos  
3.- Bota navidenia  
4.- Salir  
Ingresar de favor la opcion deseada: 2
```

At the bottom, there are two buttons: "clear screen" and "change font", followed by a font size indicator "0/16".

Se muestra el submenú de las operaciones con archivos



The screenshot shows a terminal window titled "emulator screen (80x25 chars)". The background is olive green. The text is as follows:

```
Menu de archivos xD

1.- Creacion de un archivo.
2.- Lectura de un archivo.
3.- Escritura de un archivo.
4.- Eliminacion de un archivo.
5.- Regresar al menu principal***

Ingresar de favor la opcion deseada:
```

At the bottom of the terminal, there are two buttons: "clear screen" and "change font", followed by a small status bar showing "0/16".

Se regresa al menú principal

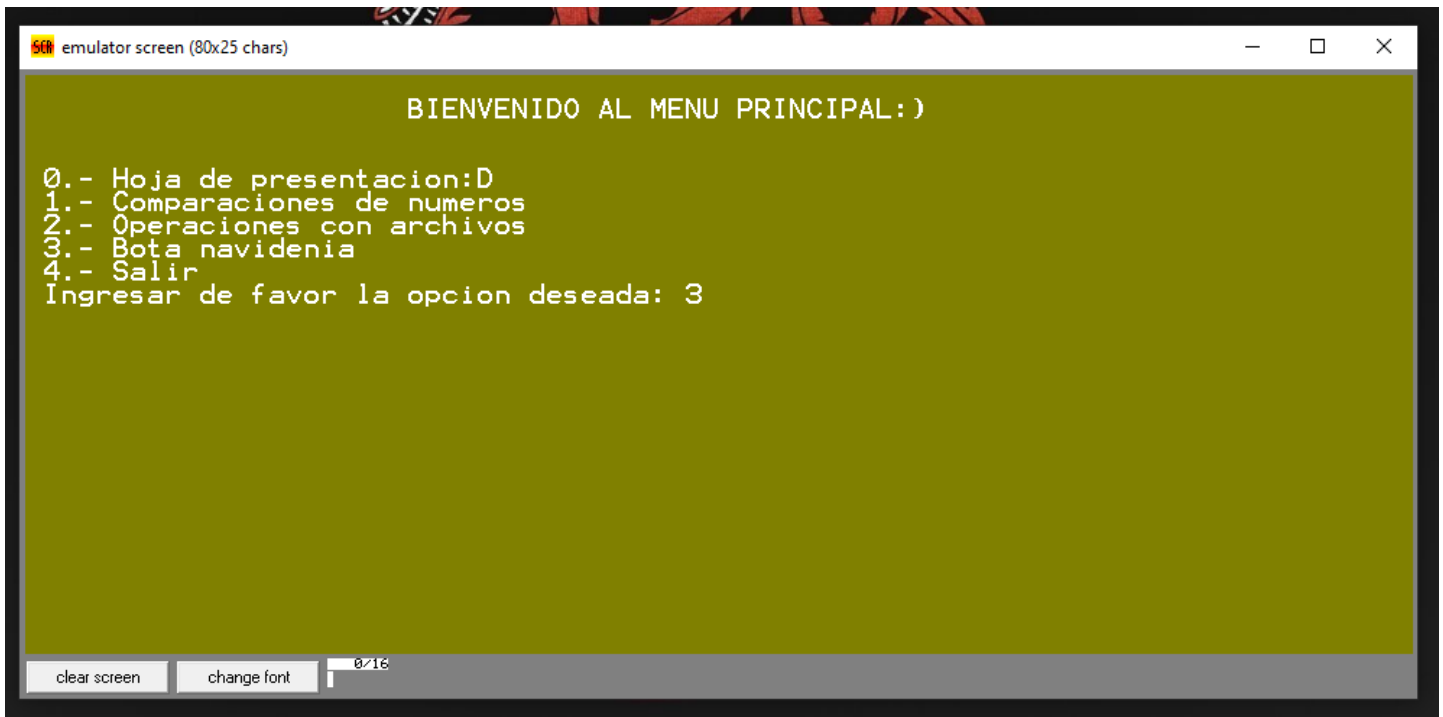


This screenshot is identical to the previous one, but with the number "5" entered after the prompt "Ingresar de favor la opcion deseada:". The text now reads:

```
Ingresar de favor la opcion deseada: 5
```

The rest of the interface, including the menu options and the bottom controls, remains the same.

Se elige la opción número tres, para mostrar la bota navideña



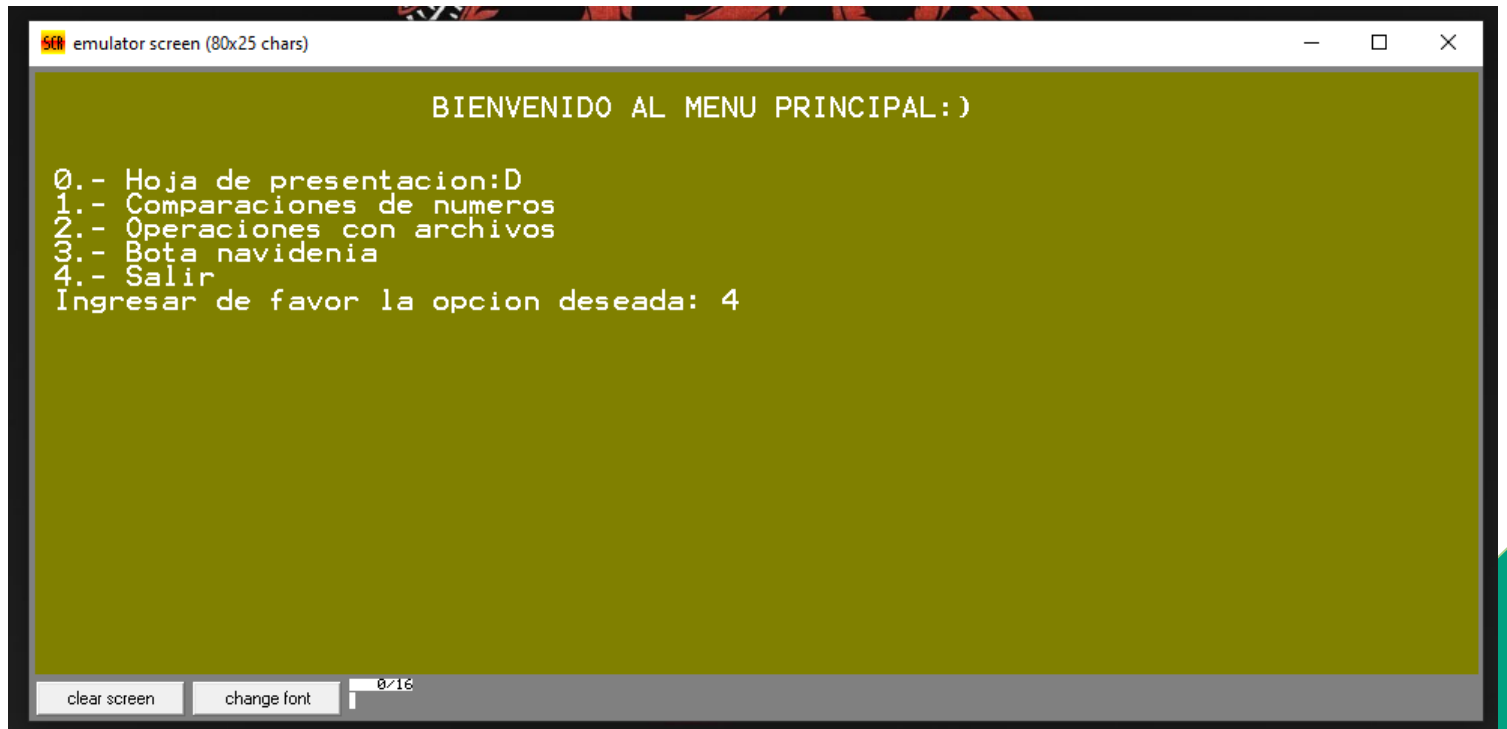
Se muestra la opción de la bota navideña



Se regresa al menú principal



Se elige la opción numero cuatro para salir del menú principal



Salimos exitosamente del menú

