

TECNOLÓGICO NACIONAL DE MEXICO INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

Carrera: Sistemas Computacionales

Unidad 2. Programación Básica.

Alumno:

Reyes Villar Luis Ricardo | 21070343

Profesora: Guadalupe Martínez Jauregui

Materia: Lenguajes de Interfaz

Hora: 09:00 – 10:00 hrs

Grupo: 6502-C

Semestre: Agosto 2023 – Diciembre 2023

2.1 Ensamblador (y ligador) a utilizar.

Ensamblador.

El lenguaje ensamblador es un lenguaje de bajo nivel que se comunica directamente con el hardware de la máquina. El termino 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 el microprocesador. El programa lee el fichero escrito en lenguaje ensamblador y sustituye cada uno de los códigos nemotécnicos que aparecen por su código de operación correspondiente en sistema binario para la plataforma que se eligió como destino en las opciones específicas del ensamblador.

Características.

- El código escrito en lenguaje ensamblador posee una cierta dificultad de ser entendido ya que su estructura se acerca al lenguaje máquina, es decir, es un lenguaje de bajo nivel.
- El lenguaje ensamblador es difícilmente portable, es decir, un código escrito para un microprocesador, puede necesitar ser modificado, para poder ser usado en otra máquina distinta. Al cambiar a una máquina con arquitectura diferente, generalmente es necesario reescribirlo completamente.
- 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 y/o muy ineficientes de programar en un lenguaje de alto nivel, ya que, entre otras cosas, en el lenguaje ensamblador se dispone de instrucciones del CPU que generalmente no están disponibles en los lenguajes de alto nivel. Podemos distinguir entre dos tipos de ensambladores:
 - Ensambladores modulares 32 bits o de alto nivel:
 - son ensambladores que aparecieron como respuesta a una nueva arquitectura de procesadores de 32 bits, muchos de ellos teniendo compatibilidad hacia atrás pudiendo trabajar con programas con estructuras de 16 bits.
 - Ensambladores básicos:
 - Son de muy bajo nivel, y su tarea consiste básicamente en ofrecer nombres simbólicos a las distintas instrucciones, parámetros y cosas.

2.2 Ciclos numéricos.

Un ciclo, conocido también como interacción, es la repetición de un proceso un cierto número de veces hasta que alguna condición se cumpla. En estos ciclos se utilizan los brincos condicionales basados en el estado de la bandera.

Los ciclos numéricos que se utilizan son los siguientes: instrucción *jmp*, instrucción *loop*, instrucción *cmp*, instrucción *cmps* e instrucción de conteo.

Instrucción JMP.

Es una instrucción basada comúnmente para la transferencia de control, un salto es incondicional ya que la operación transfiere el control bajo cualquier circunstancia. También vacía el resultado de la instrucción previamente procesada: por lo que un programa con muchas operaciones de saltos puede perder velocidad de procesamiento, el formato general para la instrucción JMP es: [etiqueta] |jmp| dirección corta, cercana o lejana|

Instrucción LOOP

La instrucción Loop requiere un valor inicial en el registro CX, en cada interacción, Loop de forma automática disminuye 1 de CX. Si el valor en el CX es cero, el control pasa a la instrucción que sigue; si el valor en el CX no es cero, el control pasa a la dirección del operando.

La distancia debe ser un salto corto, desde -128 hasta +127 bites. Para una operación que exceda este límite, el ensamblador envía un mensaje como un salto relativo fuera de rango.

Instrucción CMP.

La instrucción CMP por lo común es utilizada para comparar dos campos de datos, uno de los cuales están contenidos en un registro. El formato general para el CMP es: |[etiqueta]|CMP |{registro/memoria}, {registro/memoria/inmediato}|

La arquitectura de los procesadores x86 obliga al uso de segmentos de memoria para manejar la información, el tamaño de estos segmentos es de 64kb. La razón de ser de estos segmentos es que, considerando que el tamaño máximo de un número que puede manejar el procesador esta dado por una palabra de 16 bits o registro, no sería posible acceder a más de 65536 localidades de memoria utilizando uno solo de estos registros, ahora, si se divide la memoria de la pc en grupos o segmentos, cada uno de 65536 localidades, y utilizamos una dirección en un registro exclusivo para localizar cada segmento, y entonces cada dirección de una casilla específica la formamos con dos registros, nos es posible acceder a una

cantidad de 4294967296 bytes de memoria, lo cual es, en la actualidad, más memoria de la que veremos instalada en una PC.

Para que el ensamblador pueda manejar los datos es necesario que cada dato o instrucción se encuentren localizados en el área que corresponde a sus respectivos segmentos. El ensamblador accesa a esta información tomando en cuenta la localización del segmento, dada por los registros DS, ES, SS y CS, y dentro de dicho registro la dirección del dato específico.

2.3 Captura básica de cadenas.

En el lenguaje ensamblador el tipo de dato cadena(string) no está definido, pero para fines de programación, una cadena es definida como un conjunto de localidades de memoria consecutivas que se reservan bajo el nombre de una variable.

Instrucciones para el manejo de strings.

El lenguaje ensamblador cuenta con cinco instrucciones para el manejo de cadenas:

LODS (cargar un byte o palabra): carga el registro acumulador (AX o AL) con el valor de la localidad de memoria determinada por DS:SI se incrementa tras la transferencia.

En el siguiente ejemplo simplemente se captura una cadena en un vector y se imprime cuantas veces quiera el usuario, en este programa se aprende a usar vectores en MASM.

Los vectores se pueden declarar de cualquier tipo de dato que una variable cualquiera, solo se especifica la longitud del vector y con qué tipo de dato inicia al crearse para ello se utiliza la palabra reservada dup.

Ejemplo de declaracion de un vector:

```
vector db 100 dup('$')
```

En este caso se declara un vector de tipo db con una longitud de 100 posiciones y se inicia con cadena vacia ('\$').

Para manejar las posiciones de los vectores se utilizan los registros SI y DI.

Ejemplo del programa usando un vector para almacenar una cadena digitada por el usuario:

```
.model small
```

```

.stack 64

.data
inicio db 'Repetir Texto..',10,13,'$'
ingresar db 10,13,'Ingresa tu nombre', 10,13,'$'
nombre db 'Su nombre es: $'
repetir db 10,13,'Quiere repetir s/n?',10,13,'$'
;Declaracion del vector
vtext db 100 dup('$')

.code
mov ax,@data
MOV DS,AX
lea dx,inicio
mov ah,09
int 21h

lea dx,ingresar
mov ah,09
int 21h

;Iniciamos nuestro conteo de si en la posicion 0.
mov si,00h
leer:
mov ax,0000
mov ah,01h
int 21h

;Guardamos el valor tecleado por el usuario en la posicion si del vector.

```

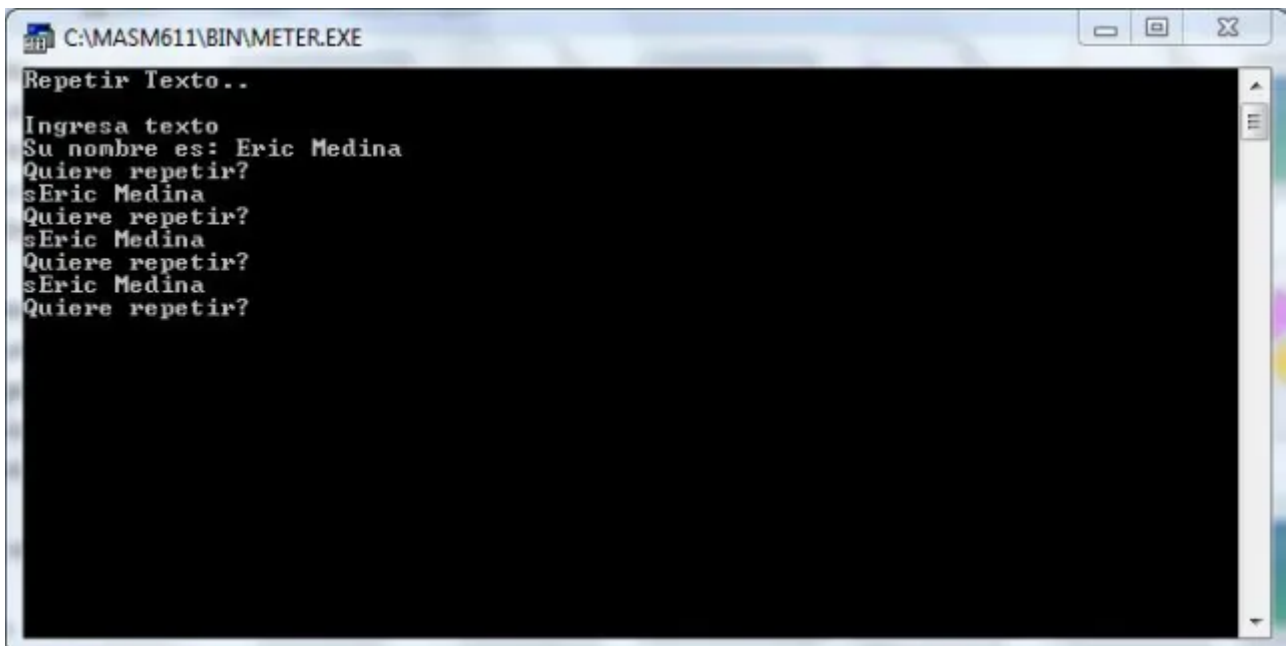
```
mov vtext[si],al
inc si ;Se incrementa el contador
cmp al,0dh ;Se repite el ingreso de datos hasta que se teclee un Enter.
ja leer
jb leer
lea dx,nombre
mov ah,09
int 21h

ver:
mov dx,offset vtext ;Imprime el contenido del vector con la misma instrucción de
una cadena
mov ah,09h
int 21h
lea dx,repetic ;imprime si quiere ver de nuevo el texto escrito.
mov ah,09
int 21h
mov ah,01h
int 21h
cmp al,73h ;Si la tecla presionada es una s, se repite la impresión del vector.
je ver

salir:
mov ah,4ch
int 21h

end
```

Programa en ejecución:



```
C:\MASM611\BIN\METER.EXE
Repetir Texto..
Ingresa texto
Su nombre es: Eric Medina
Quiere repetir?
sEric Medina
Quiere repetir?
sEric Medina
Quiere repetir?
sEric Medina
Quiere repetir?
```

2.4 Comparación y prueba.

Dentro del lenguaje ensamblador no existe el tipo de dato cadena (string en otros lenguajes), por lo que para utilizarla es necesario tratar a las cadenas como un conjunto de caracteres reservados bajo el nombre de una sola variable.

El lenguaje ensamblador cuenta con instrucciones que por su naturaleza sirven para el manejo de cadenas, estas son:

MOVSB:

Mueve un byte desde una localidad de memoria hasta otra.

MOVSW:

Mueve una palabra desde una localidad de memoria hasta otra.

LODSB:

Carga en la parte baja del registro acumulador (AL) el valor de la localidad de memoria determinada por DS:SI.

LODSW:

Carga en el registro acumulador (AX) el valor de la localidad de memoria determinada por DS:SI.

Ejemplo:

```
.DATA
CADENA DB 20 DUP()
CADENA2 DB 'ESCRIBRE LA CADENA','$'
.STACK
.CODE
.STARTUP

mov ax,@data
mov ds,ax

mov dx,OFFSET CADENA2
MOV AH,9H
INT 21H

mov ah,02h
mov dl,10
int 21h

mov ah,02h
mov dl,13
int 21h

MOV AH,0AH
MOV DX,OFFSET CADENA
INT 21H

mov ah,10h
int 16h

.EXIT
END
```


2.5 Saltos

La mayoría de los programas constan de varios ciclos en los que una serie de pasos se repite hasta alcanzar un requisito específico y varias pruebas para determinar qué acción se realiza de entre varias posibles.

Una instrucción usada comúnmente para la transferencia de control es la instrucción JMP (jump, salto, bifurcación).

Un salto es incondicional, ya que la operación transfiere el control bajo cualquier circunstancia. También JMP vacía el resultado de la instrucción previamente procesada; por lo que, un programa con muchas operaciones de salto puede perder velocidad de procesamiento.

La instrucción LOOP, requiere un valor inicial en el registro CX. En cada iteración, LOOP de forma automática disminuye 1 de CX. Si el valor en el CX es cero, el control pasa a la instrucción que sigue; si el valor en el CX no es cero, el control pasa a la dirección del operando. La distancia debe ser un salto corto, desde -128 hasta +127 bytes. Para una operación que exceda este límite, el ensamblador envía un mensaje como "salto relativo fuera de rango".

2.6 Ciclos condicionales.

Dentro de la programación existen ocasiones en la que es necesario ejecutar una misma instrucción un cierto número de veces, el cual no siempre es conocido por el programador o puede cambiar durante la ejecución del programa, para lo que existen los ciclos condicionales, los cuales una vez se cumpla la condición que tienen establecida, dejan de ejecutarse como ciclo y permitirán que el programa continúe con su flujo normal.

En ensamblador no existen de forma predefinida estos ciclos, pero pueden crearse haciendo uso de los saltos incondicionales, 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.

2.7 Incremento y decremento.

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

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.

2.8 Captura de cadenas con formato.

El capturar cadenas con formato permite el movimiento, comparación o búsqueda rápida entre bloques de datos, las instrucciones son las siguientes:

MOVC: Esta instrucción permite transferir un carácter de una cadena.

MOVW: Esta instrucción permite transferir una palabra de una cadena.

CMPC: Este comando es utilizado para comparar un carácter de una cadena.

CMPW: Esta instrucción es utilizada para comparar una palabra de una cadena.

SCAC: Esta instrucción permite buscar un carácter de una cadena.

SCAW: Esta instrucción se utiliza para buscar una palabra de una cadena.

LODC: Esta instrucción permite cargar un carácter de una cadena.

LODW: Esta instrucción es utilizada para cargar una palabra de una cadena.

STOC: Esta instrucción permite guardar un carácter de una cadena.

STOW: Esta instrucción es utilizada para guardar una palabra de una cadena.

2.9 Instrucciones aritméticas.

Dentro de ensamblador se pueden llevar a cabo las 4 instrucciones aritméticas básicas, cada una de ellas cuenta con su propia función:

Instrucción de Suma ADD:

Suma los operandos que se le dan y guarda el resultado en el primer operando.

Ejemplo:

- ADD al, bl: Suma los valores guardados en los registros al y bl, almacenando el resultado en al.

Instrucción de Resta SUB:

Resta el primer operando al segundo y almacena el resultado en el primero.

Ejemplo:

- SUB al, bl: Resta el valor de AL al de BL y almacena el resultado en AL.

Instrucción de multiplicación MUL:

Multiplica el contenido del acumulador por el operando, a diferencia de los métodos anteriores, solo es necesario indicar el valor por el que se multiplicará, ya que el resultado siempre es almacenado en el registro AX.

Ejemplo:

- MUL DX: Multiplica el valor del registro acumulador (AX) por el de DX.

Instrucción de división DIV:

Divide un numero contenido en el acumulador entre el operando fuente, el cociente se guarda en AL o AX y el resto en AH o DX según el operando sea byte o palabra respectivamente. Es necesario que DX o AH sean cero antes de la operación por lo que es necesario utilizar el ajuste de división antes de la instrucción DIV.

Ejemplo:

- AAM: Ajuste ASCII para la división.
- DIV bl: Instrucción que divide los valores en ax y bl.

2.10 Manipulación de la pila.

La pila es un grupo de localidades de memoria que se reservan para contar con un espacio de almacenamiento temporal cuando el programa se está ejecutando.

La pila es una estructura de datos del tipo LIFO (Last In First Out), esto quiere decir que el último dato que es introducido en ella, es el primero que saldrá al sacar datos de la pila.

Para la manipulación de la pila ensamblador cuenta con dos instrucciones específicas, las cuales son las siguientes:

Push: Esta instrucción permite almacenar el contenido del operando dentro de la última posición de la pila.

Ejemplo:

Push ax El valor contenido en ax es almacenado en el último espacio de la pila.

Pop: Esta instrucción toma el último dato almacenado en la pila y lo carga al operando.

Ejemplo:

Pop bx El valor contenido en el último espacio de la pila se almacena en el registro

El siguiente ejemplo muestra como implementar la instrucción XCHG por medio de las instrucciones Push y Pop. Recuerde que la instrucción XCHG intercambia el contenido de sus dos operandos.

.COMMENT

Programa: PushPop.ASM

Descripción: Este programa demuestra el uso de las instrucciones para el manejo de la pila, implementando la instrucción XCHG con Push y Pop

MODEL tiny

.CODE

Inicio: ;Punto de entrada al programa

Mov AX,5 ;AX=5

Mov BX,10 ;BX=10

Push AX ;Pila=5

Mov AX,BX ;AX=10

Pop BX ;BX=5

Mov AX,4C00h ;Terminar programa y salir al DOS

Int 21h ;

END Inicio

END

2.11 Obtención de cadena con representación decimal.

En este modo, los datos son proporcionados directamente como parte de la instrucción.

Ejemplo:

Mov AX,34h ;

Copia en AX el número 34h hexadecimal Mov CX,10 ;

Copia en CX el número 10 en decimal

.COMMENT

Programa: PushPop.ASM

Descripción: Este programa demuestra el uso de las instrucciones para el manejo de la pila, implementando la instrucción XCHG con Push y Pop

MODEL tiny

.COD

Inicio: ;Punto de entrada al programa

Mov AX,5 ;AX=5

Mov BX,10 ;BX=10

Push AX ;Pila=5

Mov AX,BX ;AX=10

Pop BX ;BX=5

Mov AX,4C00h ;Terminar programa y salir al DOS

Int 21h ;

END Inicio

END

2.12 Instrucciones Lógicas.

Ensamblador cuenta con un grupo de cuatro instrucciones lógicas a nivel de bit, las cuales con excepción de la primera requieren de dos operandos, estas son las siguientes:

NOT: La instrucción NOT o negación requiere un solo operando y su función es cambiar el estado de los bits del mismo, es decir, cambiar los ceros por unos y los unos por ceros.

Ejemplo:

NOT ax Se aplica la negación al valor del registro ax.

AND: Esta instrucción también conocida como producto lógico requiere de dos operandos y su valor será igual a uno cuando los bits que se comparen ambos sean uno.

Ejemplo:

AND ax,bx Se aplica el producto lógico a los valores de ax y bx.

OR: La instrucción OR también conocida como suma lógica requiere de dos operandos y su valor será uno si alguno de los bits que compara es uno.

Ejemplo:

OR ax,bx Se aplica la suma lógica a los valores de ax y bx.

XOR: La instrucción XOR o suma lógica exclusiva requiere dos operandos, los cuales se comparan y el resultado obtenido es uno cuando uno de los bits es uno y el otro cero.

Ejemplo:

XOR ax,bx Se aplica la instrucción XOR a los valores de ax y bx.

2.13 Desplazamiento y Rotación.

Las instrucciones de desplazamiento son cuatro: shl, shr, sar y sal; y su objetivo es desplazar los bits de un operando un determinado número de posiciones a la izquierda o a la derecha. La estructura de los operandos manejados por estas instrucciones y su significado es idéntico para las cuatro instrucciones.

- **SHL (Shift Left = desplazamiento a la izquierda)**

Se desplazan a la izquierda los bits del operando destino tantas posiciones como indique el operando fuente. El desplazamiento de una posición se realiza de la siguiente forma: el bit de mayor peso del operando se desplaza al bit CF del registro de estado, el resto de los bits se desplazan una posición hacia la izquierda, y la posición de menor peso se rellena con un 0

- **SAL (Shift Arithmetic Left = desplazamiento aritmético a la izquierda)**

El objetivo de un desplazamiento aritmético a la izquierda es multiplicar un operando, interpretado con signo, por una potencia de 2.

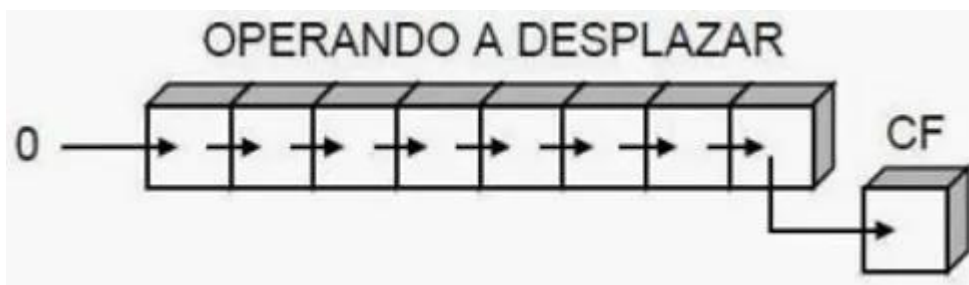
Para llevar a cabo este tipo de desplazamiento, hay que desplazar los bits del operando hacia la izquierda introduciendo ceros por su derecha.

- **SHR (Shift Right = desplazamiento a la derecha)**

La instrucción shr funciona de la misma forma que shl, pero desplazando los bits a la derecha en lugar de a la izquierda.

- **SAR (Shift Arithmetic Right = desplazamiento aritmético a la derecha)**

Esta instrucción desplaza los bits del operando destino a la derecha tantos bits como indique el operando fuente. Esta forma de funcionamiento es similar a la de la instrucción shr; sin embargo, ambas instrucciones se diferencian en que sar, en vez introducir ceros por la izquierda del operando, replica el bit de mayor peso (bit de signo) en cada desplazamiento.



2.14 Obtención de una cadena con la representación hexadecimal.

Obtención de una cadena con la representación hexadecimal La conversión entre numeración binaria y hexadecimal es sencilla. Lo primero que se hace para una conversión de un número binario a hexadecimal es dividirlo en grupos de 4 bits, empezando de derecha a izquierda. En caso de que el último grupo (el que quede más a la izquierda) sea menor de 4 bits se rellenan los faltantes con ceros. Tomando como ejemplo el número binario 101011 lo dividimos en grupos de 4 bits y nos queda: 10; 1011 Rellenando con ceros el último grupo (el de la izquierda): 0010; 1011 después tomamos cada grupo como un número independiente y consideramos su valor en decimal: 0010 = 2; 1011 = 11 Pero como no podemos representar este número hexadecimal como 211 porque sería un error, tenemos que sustituir todos los valores mayores a 9 por su respectiva representación en hexadecimal, con lo que obtenemos: 2BH (Donde la H representa la base hexadecimal) Para convertir un número de hexadecimal a binario solo es necesario invertir estos pasos: se toma el primer dígito hexadecimal y se convierte a binario, y luego el segundo, y así sucesivamente hasta completar el número.

La conversión entre numeración binaria y hexadecimal es sencilla. Lo primero que se hace para una conversión de un número binario a hexadecimal es dividirlo en grupos de 4 bits, empezando de derecha a izquierda. En caso de que el último grupo (el que quede más a la izquierda) sea menor de 4 bits se rellenan los faltantes con ceros.

Ejemplos:

<code>.model Small</code>	→	Declaracion del tamaño del programa.
<code>.Stack 64</code>	→	Declaracion del tamaño de pila.
<code>.Data</code>	→	Inicio del segmento de datos.
<code>Nombre db 'Datos de cadena\$'</code>	→	Declaración de variable y sus datos iniciales.
<code>.Code</code>	→	Inicio del segmento de código.
<code>mov ax,@data</code>	→	Se carga la dirección del segmento de datos al registro ax.
<code>mov ds,ax</code>	→	Se carga la dirección almacenada en ax al segmento de datos.
<code>Mov ah,09h</code>	→	Se carga 09h que es el servicio a utilizar a ah.
<code>lea dx,Nombre</code>	→	Instrucción para obtener el dato almacenado en la variable nombre, para utilizar offset la sintaxis seria <code>mov dx, offset Nombre</code> .
<code>int 21h</code>	→	Llamada al servicio 09h de la interrupción 21h para desplegar la cadena.
<code>.Exit</code>	→	Inicio del segmento final.
<code>End</code>	→	Fin del programa.

2.15 Captura y almacenamiento de datos numéricos

Las variables numéricas son muy útiles en ensamblador de la misma forma que en otros lenguajes de programación, ya que permiten al programador hacer operaciones aritméticas con datos que se desconocen al momento de la compilación.

La utilización de datos numéricos es similar a la de cadenas, con la diferencia de que en vez de declarar las variables como db, se declaran como dw, lo cual significa que son variables numéricas.

Nombre	Número	Uso
zero	0	Constante 0 (valor cableado)
at	1	Reservado para el ensamblador
v0, v1	2, 3	Evaluación de expresión y resultado de una función
a0, ..., a3	4, ..., 7	Argumentos a rutina (resto de argumentos, a pila)
t0, ..., t7	8, ..., 15	Temporales (no preservados a través de llamada, guardar invocador)
s0, ..., s7	16, ..., 23	Guardado temporalmente (preservado a través de llamada, guardar invocador)
t8, t9	24, 25	Temporales (no preservados a través de llamada, guardar invocador)
k0, k1	26, 27	Reservados para el núcleo del S.O.
gp	28	Puntero global, apunta a la mitad de un bloque de 64K en seg. datos estáticos
sp	29	Puntero de pila, apunta la primera posición libre en la pila
fp	30	Puntero de encuadre
ra	31	Dirección de retorno (usada por llamada de procedimiento)

Ejemplo:

.model tiny	→	Declaración del tamaño del programa.
.Stack 64	→	Declaración del tamaño de la pila.
.Data	→	Inicio del segmento de datos.
num1 dw 20	→	Declaración de la variable numérica num1 como 20.
num2 dw 30	→	Declaración de la variable numérica num2 como 30.
num3 dw ?	→	Declaración de la variable numérica num3.
.Code	→	Inicio del segmento de código.
mov ax,@data	→	Se asigna la localización del segmento de datos a ax.
mov ds,ax	→	Se asigna el valor de ax al segmento de datos.
mov ax,num1	→	Se asigna el valor de num1 a ax.
add ax,num2	→	Se suma el valor de ax con el de la variable num2.
mov num3,ax	→	Se almacena el valor obtenido a la variable num3.
.Exit	→	Inicio del segmento final.
End	→	Fin del programa.

2.16 Operaciones básicas sobre archivos.

Un archivo está identificado por un nombre y contiene datos en formato binario.

En lenguajes de alto nivel existen funciones especializadas que permiten el manejo de archivos de forma directa, con instrucciones como crear, leer o escribir archivo. En ensamblador el manipular un archivo es más complejo, pero también es posible.

Hay dos formas diferentes de utilizar archivos en ensamblador, la primera y más antigua es mediante bloques de control de archivo o FCB por sus siglas en inglés.

La segunda es mediante handles o canales de comunicación.

FCB (File Control Block):

Permite tener un gran número de archivos abiertos y crear volúmenes en dispositivos de almacenamiento.

El manejo de archivos con FCB utiliza el servicio 0FH de la instrucción 21h para abrir un archivo.

Ejemplo:

mov ah, 0FH	→	Servicio para abrir un archivo.
mov dx, OFFSET Archivo es una archivo al que se	→	Se carga la dirección del archivo a dx, 'Archivo' variable que contiene la dirección del desea acceder.
Int 21h el servicio	→	Se llama la interrupción 21h, la cual ejecutara 0FH.

Handles:

Permite un manejo de errores más simple, utiliza la estructura de directorio del sistema operativo y es más sencillo para el programador.

El manejo de archivos por Handles utiliza 3 servicios principales para ello:

3CH: Crea un nuevo archivo.

40H: Escribe sobre un archivo existente.

3EH: Cierra un archivo que se encuentre abierto.

Referencias.

<https://lenguajesdeinterfazitsncg.blogspot.com/2015/05/ensamblador-y-ligador-utilizar.html>

<https://ittlenguajesdeinterfaz.wordpress.com/2-1-ensamblador-y-ligador-a-utilizar/>

<https://lenguajesdeinterfaz.wordpress.com/2017/11/10/2-2-ciclos-numericos/>

<https://lenguajesdeinterfaz.wordpress.com/2017/11/20/2-3-captura-basica-de-cadenas/>

<https://ittlenguajesdeinterfaz.wordpress.com/2-7-ciclos-condicionales/>

<https://brandon22esquivel.wixsite.com/misitio/post/unidad-2-programaci%C3%B3n-b%C3%A1sica>