Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

**INVESTIGACIONES**

**UNIDAD IV**

Integrantes: **Martinez Reyes Fernando**

**Gonzalez Saldívar Luis Roberto**

Profesor: **Dr. García Ruiz Alejandro Humberto**

Asignatura: **Programación de Microprocesadores**

**8v**o. Semestre – Grupo “**I**”

2022-3

ÍNDICE

[1.- Cadenas de caracteres en MASM 3](#_Toc121243753)

[2.- Dup 4](#_Toc121243754)

[3.- Movsb 5](#_Toc121243755)

[4.- Cmpsb 5](#_Toc121243756)

[5.- Scasb 6](#_Toc121243757)

[6.- Stosb 6](#_Toc121243758)

[7.- lodsb 7](#_Toc121243759)

[8.- Registros Esi y Edi 7](#_Toc121243760)

[9.- Prefijos de repetición 7](#_Toc121243761)

[10.- Arreglos Unidimensionales en MASM 8](#_Toc121243762)

[11.- Arreglos Bidimensionales en MASM 9](#_Toc121243763)

[12.- Bandera de Dirección : CLD y STD 9](#_Toc121243764)

# 1.- Cadenas de caracteres en MASM

Permite leer una cadena desde el teclado de 10 caracteres. Luego se muestra en una coordenada especifica.

;Autor Hector Torres

;URL

.model small

.stack

.data

cadena db ' $'

.code

.startup

;limpiar pantalla

mov ah,00h

mov al,03h

int 10h

;leo un caracter sin eco

mov cx,10

mov si,0

leer:

mov ah,07h

int 21h

;lee 10 caracteres y los gurda el cadena

mov dl,al

mov ah,02h

int 21h

mov cadena[si],al

inc si

loop leer

;simulando el gotoxy

mov ah,02h

mov dh,10h

mov dl,10h

mov bx,00h

int 10h

mov ah,09h

mov dx,offset[cadena]

int 21h

.exit

End

# 2.- Dup

Este operador asigna almacenamiento para varios elementos de datos, usando una expresión constante como contador. En especial, es útil cuando se asigna espacio para una cadena o arreglo, y puede utilizarse con datos

inicializados o sin inicializar:

BYTE 20 DUP(0) ; 20 bytes, todos iguales a cero

BYTE 20 DUP(?) ; 20 bytes, sin inicializar

BYTE 5 DUP("PILA") ; 20 bytes: "PILAPILAPILAPILAPILA"

# 3.- Movsb

Las instrucciones MOVSB, MOVSW y MOVSD copian datos de la ubicación de memoria a la que apunta ESI, hasta la ubicación de memoria a la que apunta EDI. Las dos reformas automáticas (según el valor de la bandera Dirección):

MOVSB Mueve (copia) bytes

MOVSW Mueve (copia) palabras

MOVSD Mueve (copia) dobles palabras

# 4.- Cmpsb

Las instrucciones CMPSB, CMPSW y CMPSD comparan un operando de memoria al que apunta ESI, con un operando de memoria al que apunta EDI:

CMPSB Compara bytes

CMPSW Compara palabras

CMPSD Compara dobles palabras

Puede usar un preﬁjo de repetición con CMPSB, CMPSW y CMPSD. La bandera Dirección determina el incremento o decremento de ESI y EDI.

Forma explícita de CMPSB: en otra forma de la instrucción de comparación de cadenas llamada forma explícita, se suministran dos operandos indirectos. El operando PTR aclara los tamaños de los operandos. Por ejemplo,

cmps DWORD PTR [esi],[edi]

Pero CMPS es engañoso, ya que el ensamblador nos permite suministrar operandos erróneos:

cmps DWORD PTR [eax],[ebx]

Sin importar qué operandos se utilicen, CMPS compara el contenido de la memoria a la que apunta ESI con la memoria a la que apunta EDI. Observe que el orden de los operandos en CMPS es opuesto a la instrucción CMPS, más conocida:

CMP destino,origen

CMPS origen,destino

He aquí otra forma de recordar la diferencia: CMP implica restar el origen del destino. CMPS implica restar el destino del origen. C Se sugiere evitar el uso de CMPS y utilizar las versiones especíﬁ cas (CMPSB, CMPSW, CMPSD).

# 5.- Scasb

Las instrucciones SCASB, SCASW y SCASD comparan un valor en AL/AX/EAX con un byte, palabra o doble palabra, respectivamente, la cual está direccionada por EDI. Las instrucciones son útiles cuando se busca un valor individual en una cadena o arreglo. Si se combinan con el preﬁjo REPE (o REPZ), la cadena o arreglo se explora mientras ECX 0, y el valor en AL/AX/EAX coincida con cada valor subsiguiente en memoria. El preﬁjo REPNE explora hasta que AL/AX/EAX coincida con un valor en memoria, o cuando ECX = 0.

Explorar en busca de un carácter que coincida En el siguiente ejemplo buscamos la letra F en la cadena alfa. Si se encuentra la letra, EDI apunta una posición más allá del carácter que coincidió. Si no se encuentra la letra, JNZ termina el programa:

.data

alfa BYTE "ABCDEFGH",0

.code

mov edi,OFFSET alfa ; EDI apunta a la cadena

mov al,'F' ; busca la letra F

mov ecx,LENGTHOF alfa ; establece la cuenta de búsqueda

cld ; dirección = avance

repne scasb ; repite mientras no sea igual

jnz salir ; termina si no se encontró la letra

dec edi ; se encontró: retrocede EDI

JNZ se agregó después del ciclo para evaluar la posibilidad de que el ciclo se detuviera debido a ECX = 0, y que no se encontrara el carácter en AL.

# 6.- Stosb

Las instrucciones STOSB, STOSW y STOSD almacenan en memoria el contenido de AL/AX/EAX, respectivamente, en el desplazamiento al que apunta EDI. EDI se incrementa o decrementa con base en el estado de la bandera Dirección. Cuando se utilizan con el preﬁjo REP, estas instrucciones son útiles para rellenar todos los elementos de una cadena o arreglo con un solo valor. Por ejemplo, el siguiente código inicializa cada byte en cadena1 con 0FFh:

.data

Cuenta = 100

Cadena1 BYTE Cuenta DUP(?)

.code

# 7.- lodsb

Las instrucciones LODSB, LODSW y LODSD cargan un byte o palabra de la memoria en ESI, hacia AL/AX/EAX, respectivamente. ESI se incrementa o decrementa según el estado de la bandera Dirección. El preﬁjo REP se utiliza raras veces con LODS, ya que cada nuevo valor que se carga en el acumulador sobrescribe su contenido anterior. En vez de ello, LODS se utiliza para cargar un solo valor. En el siguiente ejemplo, LODSB sustituye a las dos instrucciones siguientes (suponiendo que la bandera Dirección esté en cero):

mov al,[esi] ; mueve byte hacia AL

inc esi ; apunta al siguiente byte

# 8.- Registros Esi y Edi

Hay algunas operaciones que puede realizar sólo hacer con EDI/ESI (o sus homólogos extendidos, si no aprendiste ASM en 1985). Entre ellos se encuentran **REP STOSB, REP MOVSB, REP SCASB**

Que son, respectivamente, operaciones de almacenamiento repetido (masivo), carga y exploración. Lo que se hace es configurar ESI y/o EDI para que apunten a uno o ambos operandos, tal vez poner un conteo en CX y luego dejarlo correr. Estas son operaciones que trabajan en un montón de bytes a la vez, y como que ponen a la CPU en automático. Como no estás codificando explícitamente los bucles, hacen su trabajo más eficientemente (normalmente) que un bucle codificado a mano.

En caso de que se lo pregunten: Dependiendo de cómo configures la operación, el almacenamiento repetido puede ser algo simple como perforar el valor 0 en un gran bloque contiguo de memoria; MOVSB se utiliza, creo, para copiar datos de un buffer (cualquier montón de bytes) a otro; y SCASB se utiliza para buscar un byte que coincida con algún criterio de búsqueda

# 9.- Prefijos de repetición

Los prefijos de repetición trabajan en conjunto con instrucciones de cadena de caracteres, ya que, al realizar operaciones con cadenas de caracteres. Antes de ver las instrucciones que manipulan cadenas, es necesario comentar el uso de los **prefijos de repetición**, modificadores que sólo se pueden usar con las instrucciones de manipulación de cadenas. REP este modificador repite la instrucción a la que acompaña mientras que CX sea distinto de 0 (decrementa CX cada vez). Las instrucciones con las que se puede usar son MOVS, MOVDSW o STOS.

**MOVS destino, fuente**

**REP MOVS destino, fuente**

REPE/REPZ este modificador repite la instrucción a la que acompaña mientras que CX sea distinto de 0 y ZF = 1 (decrementa CX cada vez). Las instrucciones con las que se puede usar son CMPS o SCAS

**CMPS destino, fuente**

**REPE/REPZ CMPS destino, fuente**

REPNE/REPNZ este modificador repite la instrucción a la que acompaña si CX es distinto de 0 y ZF = 0 (decrementa CX cada vez). Las instrucciones con las que se puede usar son CMPS o SCAS.

**REPNE/REPN CMPS destino, fuente**

MOVS/MOVSW copia un byte o un WORD de una parte a otra de la memoria.

MOVS destino, fuente

Donde destino es ES:DI y fuente es DS:SI, lo que quiere decir que antes de utilizar la instrucción hay que cargar en SI y DI los valores apropiados.

# 10.- Arreglos Unidimensionales en MASM

Un array es un conjunto de datos del mismo tipo ordenados de forman líneal uno después de otro. Los componentes de un array se han de referenciar por medio del nombre del array y un índice de desplazamiento para indicar el componente deseado.

Los arrays son usados extensamente por los programadores para contener listas de datos en la memoria, por ejemplo, los datos almacenados en un disco suelen leerse y ponerse dentro de un array con el objetivo de facilitar la manipulación de dichos datos, ya que los datos en memoria pueden ser modificados, clasificados, marcados para su eliminación, etc. para luego ser reescritos al disco. Otro ejemplo podría ser el de un menú de opciones que se desplegarán dentro de una ventana para que el usuario pueda elegir una de éstas, en tales casos y cuando las opciones son numerosas, solamente se ponen unas cuantas, de ellas dentro de la ventana, pero se le da al usuario la oportunidad de poder subir y bajar a su antojo para ver el resto de opciones que, aunque no se vean en la ventana, forman parte del menú o array de opciones.

**Array unidimensional**

Una array uni-dimensional es aquel en donde los componentes son accesibles por medio de uno y solamente un índice que apunte al componente requerido. Los arrays de este tipo son conocidos también con el nombre de vectores. Conceptualmente, podemos pensar en un array unidimensional como una lista compuesta de datos, donde para referirnos a uno de ellos emplearemos un número para indicar la posición del mismo dentro de la lista. Por ejemplo, consideremos el caso de la tabla o array VentaSemanal, la cual está pensada para registrar las ventas de cada uno de los días de la semana.

# 11.- Arreglos Bidimensionales en MASM

**Array bidimensional**

Un array bi-dimensional es aquel en donde los componentes son accesibles por medio de una pareja de índices que apunten a la fila y a la columna del componente requerido. Los arrays de este tipo son conocidos también con el nombre de matrices. Conceptualmente, podemos pensar en un array bidimensional como en una lista compuesta de filas y columnas, en donde para referirnos a una de ellas emplearemos un número para indicar la posición de fila y otro número para indicar la posición de la columna del componente deseado. Por ejemplo, consideremos el caso de la tabla o array VentaSemanaQ, la cual está pensada para registrar las ventas de cada uno de los días de la semana por cuatro semanas, o sea, una tabla de 7 x 4 elementos.

# 12.- Bandera de Dirección : CLD y STD

La bandera de dirección es una bandera que controla la dirección de izquierda a derecha o de derecha a izquierda del procesamiento de cadenas, almacenada en el registro FLAGS en todas las CPU compatibles con x86. Es el bit número 10.

Esta bandera se utiliza para determinar la dirección ('hacia adelante' o 'hacia atrás') en la que se copiarán varios bytes de datos de un lugar en la memoria a otro. La dirección es importante principalmente cuando la posición de los datos originales en la memoria y la posición de los datos de destino se superponen.

Si se establece en 0 (utilizando la dirección de bandera clara instrucción CLD) - significa que la cadena se procesa empezando de menor a mayor dirección; este modo de instrucciones se denomina modo de incremento automático. Tanto el índice de la fuente y el índice de destino (como MOVS) aumentará ellos;

En caso de que se establezca en 1 (utilizando la instrucción set-direction-flag STD), la cadena se procesa de la dirección más alta a la más baja. A esto se le llama modo de decremento automático.