Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

**INVESTIGACIONES**

**UNIDAD III**

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.- ¿Qué es un macro en ensamblador? 3](#_Toc121242810)

[2.- ¿Qué es un procedimiento en ensamblador? 3](#_Toc121242811)

[3.- ¿En qué consiste la programación modular? 4](#_Toc121242812)

[4.- Diferencia entre programación Estructurada y Programación Modular 4](#_Toc121242813)

[5- Liberia IRVINE32 4](#_Toc121242814)

[6.- MsgBox 5](#_Toc121242815)

[7.- MsgBoxAsk 5](#_Toc121242816)

[8.- Random 32 6](#_Toc121242817)

[9.- Randomize 6](#_Toc121242818)

[10.- RandomRange 7](#_Toc121242819)

[11.- ReadChar 7](#_Toc121242820)

[12.- ReadDec 8](#_Toc121242821)

[13.- ReadInt 8](#_Toc121242822)

[14.- ReadString 9](#_Toc121242823)

[15.- Delay 9](#_Toc121242824)

[16.- Strlength 10](#_Toc121242825)

[17.- waitMsg 10](#_Toc121242826)

[18.- writeChar 10](#_Toc121242827)

[20.- writeDec 11](#_Toc121242828)

[21.- writeString 11](#_Toc121242829)

[22.- Offset 12](#_Toc121242830)

[23.- SizeOf 12](#_Toc121242831)

[24.- isDigit 13](#_Toc121242832)

[25.- clrscr 13](#_Toc121242833)

[26.- Delay 13](#_Toc121242834)

[27.- dumpRegs 13](#_Toc121242835)

[28.- Dumpmem 14](#_Toc121242836)

[29.- Crlf 15](#_Toc121242837)

[30.- Directiva Local 15](#_Toc121242838)

# 1.- ¿Qué es un macro en ensamblador?

Una macro es un nombre que define un grupo de indicaciones que van a ser sustituidas por la macro una vez que el nombre de ésta aparezca en un programa (proceso nombrado extensión de macros) en el instante de ensamblar el programa. Las indicaciones de macros tienen la posibilidad de guardar en el programa mismo o en un documento separado que el programa logre detectar.

Una macroinstrucción es una instrucción compleja, formada por otras normas más sencillas. Esto posibilita la automatización de labores repetitivas. Además, tiene que estar almacenada, el concepto no se aplica a una secuencia de normas escritas en la línea de comandos vinculadas unas con otras por redirección de sus resultados o para su ejecución consecutiva.

Las macros acostumbran a almacenarse en el campo del propio programa que las usa y se ejecutan pulsando una mezcla particular de teclas o un botón en especial desarrollado y asignado para tal impacto. La diferencia entre una macroinstrucción y un programa es que en las macroinstrucciones la ejecución es secuencial y no existe otro criterio del flujo de programa que, por consiguiente, no puede bifurcarse.

Si bien es cierto que las macros dan más grande flexibilidad en el momento de planear, además es cierto que poseen varias desventajas.

# 2.- ¿Qué es un procedimiento en ensamblador?

Un método es una serie de normas que en general conducen a cabo una labor específica.

En programación un método es un segmento de código que cuenta con normas a las cuales se puede entrar a partir de cualquier parte del programa y una vez se acaba la ejecución de estas, el programa continuo con su ejecución habitual, tomando el control la siguiente línea luego de la llamada al método.

Los métodos tienden a ser conjuntos de normas que se necesitara realizar bastante más de una vez en un programa, debido a que un método podría ser denominado en cualquier instante a lo largo de la ejecución del programa primordial, la proporción de veces que sea primordial sin necesidad de reescribir el código.

# 3.- ¿En qué consiste la programación modular?

La programación modular está basada en la técnica de diseño descendente, que como ya vimos se basa en dividir el problema original en varios subproblemas que tienen la posibilidad de solucionar por separado, para luego recomponer los resultados y obtener la solución al problema.

Un subproblema se llama módulo y es una sección del problema que se puede solucionar de forma libre. Que un módulo sea sin dependencia nos posibilita, por un lado, concentrarnos en su resolución olvidándonos del resto del problema, y sin embargo, posibilita reutilizar la solución obtenida para otra parte del programa u otro programa diferente.

Una subrutina es una parte de código separada del programa primordial que podría ser llamada en un rato dado (llamada a subrutina) y que una vez acabada su ejecución

vuelve al punto donde se hizo la llamada. Un subprograma hace el papel de un programa. Puede tener una parte de declaraciones (variables, constantes, etc) y tiene además unos datos de ingreso y de salida.

Esto posibilita, como ya observaremos, que el subprograma sea plenamente libre del programa primordial.

Cada módulo se codifica dentro del programa como un subprograma, o sea, una parte de código libre que ejecuta una labor específica dentro del programa. El término de subprograma es una evolución del antiguo criterio de subrutina, presente en idiomas como ensamblador, Basic o primeras variantes de Fortran.

# 4.- Diferencia entre programación Estructurada y Programación Modular

El paradigma modular se basa en la construcción de funciones del programador.

El paradigma estructurado se basa en diseñar los algoritmos utilizando únicamente 3 estructuras básicas, llamadas estructuras de control.

# 5- Liberia IRVINE32

para la programación en modo de direccionamiento real y modo protegido, con código fuente completo.

# 6.- MsgBox

El procedimiento MsgBox muestra un cuadro de mensaje contextual gráfico con una leyenda opcional. Se le pasa el desplazamiento de una cadena en EDX, la cual aparece dentro del cuadro. De manera opcional, se le pasa el desplazamiento de una cadena en EBX para el título del cuadro. Para dejar el título en blanco, EBX se establece en cero. He aquí una llamada de ejemplo:

.data

leyenda db "Titulo del cuadro de dialogo", 0

MsjHola BYTE "Este es un cuadro de mensaje contextual.", 0dh,0ah

BYTE "Haga clic en Aceptar para continuar...", 0

. code

mov ebx, OFFSET leyenda

mov edx, OFFSET MsjHola

call MsgBox

# 7.- MsgBoxAsk

El procedimiento MsgBoxAsk muestra un cuadro de mensaje contextual gráfico con botones Sí y No. Se le pasa el desplazamiento de una cadena de preguntas en EDX, la cual aparece dentro del cuadro. De manera opcional, se le pasa el desplazamiento de una cadena en EBX para el título del cuadro. Para dejar el título en blanco, EBX se establece en cero. MsgBoxAsk devuelve un entero en EAX que nos indica qué botón seleccionó el usuario: IDYES (igual a 6) o IDNO (igual a 7). He aquí una llamada de ejemplo:

.data

leyenda BYTE "Encuesta terminada",0

pregunta BYTE "Gracias por completar la encuesta."

BYTE 0dh,0ah

¿BYTE “Desea recibir los resultados?",0

resultados BYTE "Los resultados se enviarán via correo electronico.",0dh,0ah,0

.code

mov ebx,OFFSET leyenda

mov edx,OFFSET pregunta

call MsgBoxAsk

;(comprobar el valor de retorno en EAX)

# 8.- Random 32

El procedimiento Random32 genera y devuelve un entero aleatorio de 32 bits en EAX. Si se llama repetidas veces, Random32 genera una secuencia aleatoria simulada, en la que cada número se conoce como entero pseudoaleatorio. Los números se crean utilizando una función simple, que tiene una entrada conocida como semilla. Esta función utiliza la semilla en una fórmula que genera el valor aleatorio. Los valores aleatorios subsiguientes se generan utilizando cada valor aleatorio generado anteriormente como sus semillas. De este punto en adelante, el término aleatorio significara pseudoaleatorio. He aquí una

llamada de ejemplo:

.data

val Aleatorio DWORD?

. code

call Random32

mov valAleatorio, eax

El procedimiento Random32 también está disponible en la biblioteca Irvine16, y devuelve su valor en EAX.

# 9.- Randomize

El procedimiento Randomize inicializa el valor de la semilla inicial de los procedimientos Random32 y RandomRange. La semilla es igual a la hora del día, con una precisión de 1/100 de un segundo. Cada vez que se ejecute un programa que llama a Random32 y a RandomRange, la secuencia generada será

distinta y cualquier secuencia de números aleatorios también será única. Sólo necesitamos llamar a Randomize una vez al principio de un programa. En el siguiente ejemplo, producimos 10 enteros aleatorios:

call Randomize

mov ecx,10

L1: call Random32

; aquí se utiliza o se muestra el valor aleatorio en EAX...

loop L1

# 10.- RandomRange

El procedimiento RandomRange produce un entero aleatorio dentro del rango de 0 a n - 1, en donde n es un parámetro de entrada que se pasa en el registro EAX. El entero aleatorio se devuelve en EAX. El siguiente ejemplo genera un entero aleatorio individual entre 0 y 4999, y lo coloca en EAX:

.data

valAleat DWORD?

.code

mov eax,5000

call RandomRange

mov valAleat,eax

# 11.- ReadChar

El procedimiento ReadChar lee un solo carácter del teclado y devuelve ese carácter en el registro AL. El carácter no se imprime en la ventana de consola. He aquí una llamada de ejemplo:

.data

car BYTE?

. code

call ReadChar

mov car,al

Si el usuario oprime una tecla extendida, como una tecla de función, tecla de ﬂ echa de cursor, Insert o Supr, el procedimiento establece AL en cero y AH contiene un código de exploración del teclado. En las guardas de este libro (a la vuelta de la portada) se presenta una lista de códigos de exploración. La mitad superior de EAX no se preserva.

# 12.- ReadDec

El procedimiento ReadDec lee un entero decimal sin signo de 32 bits del teclado y devuelve el valor en EAX. Los espacios a la izquierda se ignoran. El valor de retorno se calcula en base a todos los dígitos válidos que aparezcan, hasta encontrarse con un carácter que no sea dígito. Por ejemplo, si el usuario escribe 123ABC, el valor devuelto en EAX es 123. He aquí una llamada de ejemplo:

.data

valEntero DWORD?

. code

call ReadDec

mov valEntero,eax

ReadDec afecta a la bandera Acarreo en lo siguiente:

• Si el entero está en blanco, EAX = 0 y CF = 1.

• Si el entero sólo contiene espacios, EAX = 0 y CF = 1.

• Si el entero es mayor que 2

32

=1, EAX = 0 y CF = 1.

• En caso contrario, EAX = el entero convertido y CF = 0.

# 13.- ReadInt

El procedimiento ReadInt lee un entero con signo de 32 bits desde el teclado, y devuelve el valor en EAX. El usuario puede escribir un signo positivo o negativo opcional a la izquierda, y el resto del número sólo puede consistir en dígitos. ReadInt activa la bandera Desbordamiento y muestra un mensaje de error si el valor introducido no puede representarse como entero con signo de 32 bits (rango: -2,147,483,648 a +2,147,483,647). El valor de retorno se calcula a partir de todos los dígitos válidos encontrados, hasta que se encuentra un carácter que no sea dígito. Por ejemplo, si el usuario escribe +123ABC, el valor devuelto es +123. He aquí una llamada de ejemplo:

.data

valEntero SDWORD?

. code

call ReadInt

mov valEntero,eax

# 14.- ReadString

El procedimiento ReadString lee una cadena del teclado, y se detiene cuando el usuario oprime Intro. Recibe el desplazamiento de un búfer en EDX y establece ECX al máximo número de caracteres que puede introducir el usuario, más 1 (para guardar espacio para el byte de terminación nulo). El procedimiento devuelve la cuenta del número de caracteres escritos por el usuario en EAX. He aquí una llamada de ejemplo:

.data

bufer BYTE 21 DUP (0) ; búfer de entrada

cuentaBytes DWORD? ; guarda el contador

. code

mov edx,OFFSET bufer ; apunta al búfer

mov ecx,SIZEOF bufer ; especifica el máximo de caracteres

call ReadString ; recibe la cadena de entrada

mov cuentaBytes,eax ; número de caracteres

ReadString inserta en forma automática un terminador nulo en memoria, al ﬁ nal de la cadena. A continuación, se muestra un vaciado hexadecimal y ASCII de los primeros 8 bytes de bufer, después de que el usuario introduce la cadena “ABCDEFG”:

# 15.- Delay

El procedimiento Delay detiene la ejecución del programa durante cierto número de milisegundos. Antes de llamar a Delay, asigne a EAX el intervalo deseado. He aquí una llamada de ejemplo:

mov eax,1000 ; 1 segundo

call Delay

(La versión Irvine16.lib no funciona bajo Windows NT, 2000 o XP).

# 16.- Strlength

El procedimiento StrLength devuelve la longitud de una cadena con terminación nula. Recibe el desplazamiento de la cadena en EDX. El procedimiento devuelve la longitud de la cadena en EAX. He aquí una llamada de ejemplo:

.data

bufer BYTE "abcde",0

longBufer DWORD?

. code

mov edx,OFFSET bufer ; apunta a la cadena

call StrLength ; EAX = 5

mov longBufer,eax ; guarda la longitud

# 17.- waitMsg

El procedimiento WaitMsg muestra el mensaje “Press any key to continue…” (“Oprima cualquier tecla para continuar”) y espera a que el usuario oprima una tecla. Este procedimiento es útil cuando deseamos detener la visualización de la pantalla antes de que se desplacen los datos y desaparezcan. No tiene parámetros de entrada. He aquí una llamada de ejemplo: **call WaitMsg**

# 18.- writeChar

El procedimiento WriteChar escribe un solo carácter en la ventana de consola. Pasa el carácter (o su código ASCII) en AL. He aquí una llamada de ejemplo:

mov al,'A'

call WriteChar ; muestra "A"

**19.- writeInt**

El procedimiento WriteInt escribe un entero con signo de 32 bits en la ventana de consola, en formato decimal con un signo a la izquierda y sin ceros a la izquierda. El entero se pasa en EAX. He aquí

una llamada de ejemplo:

mov eax,216543

call WriteInt ; muestra: "+216543"

# 20.- writeDec

El procedimiento WriteDec escribe un entero sin signo de 32 bits en la ventana de consola, en formato decimal sin ceros a la izquierda. El entero se pasa en EAX. He aquí una llamada de ejemplo:

mov eax,295

call WriteDec ; muestra "295"

# 21.- writeString

El procedimiento WriteString escribe una cadena con terminación nula en la ventana de consola. El desplazamiento de la cadena se pasa en EDX. He aquí una llamada de ejemplo:

.data

indicador BYTE "Escriba su nombre: ",0

. code

mov edx,OFFSET indicador

call WriteString

# 22.- Offset

El operador OFFSET devuelve el desplazamiento de una etiqueta de datos. El desplazamiento representa la distancia (en bytes) de la etiqueta, a partir del inicio del segmento de datos. En el siguiente ejemplo, declaramos tres tipos distintos de variables:

.data

valB BYTE?

valW WORD?

valD DWORD?

valD2 DWORD?

Si valB se encontrara en el desplazamiento 00404000 (hexadecimal), el operador OFFSET devolvería los siguientes valores:

mov esi,OFFSET valB ; ESI = 00404000

mov esi,OFFSET valW ; ESI = 00404001

mov esi OFFSET valD ; ESI = 00404003

mov esi,OFFSET valD2 ; ESI = 00404007

OFFSET también puede aplicarse a un operando de desplazamiento directo. Suponga que miArreglo contiene cinco palabras de 16 bits. La siguiente instrucción MOV obtiene el desplazamiento de miArreglo, le suma 4 y mueve la suma a ESI:

.data

miArreglo WORD 1,2,3,4,5

. code

mov esi,OFFSET miArreglo + 4

# 23.- SizeOf

El operador SIZEOF devuelve un valor que equivale a multiplicar LENGTHOF por TYPE. Por ejemplo, arregloInt tiene los valores TYPE = 2 y LENGTHOF = 32. Por lo tanto, SIZEOF arregloInt es igual a 64:

.data

arregloInt WORD 32 DUP (0)

. code

mov eax,SIZEOF arregloInt ; EAX = 64

# 24.- isDigit

El procedimiento IsDigit determina si el carácter en AL es un dígito decimal válido. Al llamarlo, se pasa un carácter ASCII en AL. El procedimiento activa la bandera Cero si AL contiene un dígito decimal válido; en caso contrario, la bandera Cero se borra. He aquí una llamada de ejemplo:

mov AL,uncaracter

call IsDigit

jz digito\_encontrado

fefw

# 25.- clrscr

El procedimiento Clrscr borra la ventana de consola. Por lo general, a este procedimiento se le llama al principio y al ﬁ nal de un programa. Si lo llamamos otras veces, tal vez sea necesario detener la ejecución del programa llamando a WaitMsg. Esto permite al usuario ver la información que ya se encuentra en la pantalla, antes de que se borre. He aquí una llamada de ejemplo:

call WaitMsg ; "Oprima cualquier tecla..."

call Clrscr

# 26.- Delay

El procedimiento Delay detiene la ejecución del programa durante cierto número de milisegundos. Antes de llamar a Delay, asigne a EAX el intervalo deseado. He aquí una llamada de ejemplo:

mov eax,1000 ; 1 segundo

# 27.- dumpRegs

El procedimiento DumpRegs muestra los registros EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EIP y EFL (EFLAGS) en hexadecimal. También muestra los valores de las banderas Acarreo, Signo, Cero, Desbordamiento, Acarreo auxiliar y Paridad. He aquí una llamada de ejemplo:

call DumpRegs

Resultados de ejemplo:

EAX=00000613 EBX=00000000 ECX=000000FF EDX=00000000

ESI=00000000 EDI=00000100 EBP=0000091E ESP=000000F6

EIP=00401026 EFL=00000286 CF=0 SF=1 ZF=0 OF=0 AF=0 PF=1

El valor mostrado de EIP es el desplazamiento de la instrucción que va después de la llamada a DumpRegs.

Este procedimiento puede ser útil al depurar programas, ya que muestra una instantánea de la CPU. No tiene parámetros de entrada ni valor de retorno.

# 28.- Dumpmem

El procedimiento DumpMem escribe un rango de memoria a la ventana de consola en hexa- decimal. Se le pasa la dirección inicial en ESI, el número de unidades en ECX y el tamaño de la unidad en EBX (1 = byte, 2 = palabra, 4 = doble palabra). La siguiente llamada de ejemplo muestra un arreglo de 11 dobles palabras en hexadecimal:

.data

arreglo DWORD 1,2,3,4,5,6,7,8,9,0,0Ah,0Bh

.code

main PROC

mov esi,OFFSET arreglo ; desplazamiento inicial

mov ecx,LENGTHOF arreglo ; número de unidades

mov ebx,TYPE arreglo ; formato de doble palabra

call DumpMem

Se produce la siguiente salida:

00000001 00000002 00000003 00000004 00000005 00000006 00000007

00000008 00000009 0000000A 0000000B

# 29.- Crlf

El procedimiento Crlf desplaza el cursor al principio de la siguiente línea en la ventana de consola.

Escribe una cadena que contiene los valores 0Dh y 0Ah. He aquí una llamada de ejemplo:

call Crlf

# 30.- Directiva Local

Podemos suponer que Microsoft creó la directiva LOCAL como un sustituto de alto nivel para la instrucción ENTER. LOCAL declara sólo una o más variables locales por nombre, y les asigna atributos de tamaño. Por otra parte, ENTER sólo reserva un solo bloque sin nombre de espacio en la pila para las variables locales. Si se utiliza, LOCAL debe aparecer en la línea que va justo después de la directiva PROC. Su sintaxis es:

LOCAL listavars

listavars es una lista de deﬁ niciones de variables, separadas por comas, que de manera opcional abarcan varias líneas. Cada deﬁ nición de variable toma la siguiente forma:

etiqueta:tipo

La etiqueta puede ser cualquier identiﬁ cador válido, y el tipo puede ser cualquier tipo estándar (WORD, DWORD, etc.) o un tipo deﬁ nido por el usuario. En el capítulo 10 describiremos las estructuras y otros tipos deﬁnidos por el usuario.

Ejemplos El procedimiento MiSub contiene una variable local llamada var1, de tipo BYTE.

MiSub PROC

LOCAL var1:BYTE

El procedimiento OrdenaBurbuja contiene una variable tipo doble palabra llamada temp y una variable llamada Intercambia Bandera de tipo BYTE:

OrdenaBurbuja PROC

LOCAL temp:DWORD, IntercambiaBandera:BYTE

El procedimiento Mezclar contiene una variable local PTR WORD llamada pArreglo, la cual es un apuntador a un entero de 16 bits:

Mezcla PROC

LOCAL pArreglo:PTR WORD

La variable local ArregloTemp es un arreglo de 10 dobles palabras. Observe el uso de los corchetes para mostrar el tamaño del arreglo:

LOCAL ArregloTemp[10]:DWORD