Lenguaje ensamblador

Instrucciones

Formato de instrucciones para lenguaje ensamblador:

Directivas de segmento

```

.model small
.data
.stack
.code
  
```

Nombre_procedimiento PROC

fuente

Instrucción operando **destino, operando**

Nombre_procedimiento ENDP

END

; comentarios

MOV

Transfiere datos entre celdas de memoria y registros.

Sintaxis: **MOV Destino,Fuente**

Ejemplo:

```
MOV AX,0006h
```

```
MOV DX,AX
```

MOVS (MOVSB) (MOVSW)

Mueve cadenas de bytes o palabras desde la fuente, direccionada por SI, hasta el destino direccionado por DI.

Sintaxis: **MOVS**

Este comando no necesita parametros ya que toma como dirección fuente el contenido del registro SI y como destino el contenido de DI.

Ejemplo:

```
MOV SI, OFFSET VARIABLE1  
MOV DI, OFFSET VARIABLE2  
MOVS
```

Primero se inicializan los valores de SI y DI con las direcciones de las variables VARIABLE1 y VARIABLE2 respectivamente, después al ejecutar MOVS se copia el contenido de VARIABLE1 a VARIABLE2.

Los comandos MOVSB y MOVSW se utilizan de la misma forma que MOVS, el primero mueve un byte y el segundo una palabra.

LODS (LODSB) (LODSW)

Carga cadenas de un byte o palabra al acumulador.

Sintaxis: **LODS**

Toma la cadena que se encuentre en la dirección especificada por SI, la carga al registro AL (o AX) y suma o resta 1 (según el estado de DF) a SI si la transferencia es de bytes o 2 si la transferencia es de palabras.

Ejemplo:

```
MOV SI, OFFSET VARIABLE1  
LODS
```

La primera línea carga la dirección de VARIABLE1 en SI y la segunda línea lleva el contenido de esa localidad al registro AL.

Los comandos LODSB y LODSW se utilizan de la misma forma, el primero carga un byte y el segundo una palabra (utiliza el registro completo AX).

LAHF

Transfiere al registro AH el contenido de las banderas

Sintaxis: **LAHF**

Se utiliza para verificar el estado de las banderas durante la ejecución de un programa.

Las banderas quedan en el siguiente orden dentro del registro:

SF ZF __ AF __ PF __ CF

LEA

Carga la dirección del operando fuente.

Sintaxis: **LEA destino, fuente**

El operando fuente debe estar ubicado en memoria, y se coloca su desplazamiento en el registro índice o apuntador especificado en destino.

Ejemplo:

MOV SI, OFFSET VAR1

Que es equivalente a:

LEA SI, VAR1

POP

Recupera un dato de la pila

Sintaxis: **POP destino**

Transfiere el último valor almacenado en la pila al operando destino y después incrementa en dos el registro SP.

Este incremento se debe a que la pila va creciendo desde la dirección más alta de memoria del segmento hacia la más baja, y la pila solo trabaja con palabras (2 bytes), entonces al incrementar en dos el registro SP realmente se le está restando dos al tamaño real de la pila.

POPF

Extrae las banderas almacenadas en la pila.

Sintaxis: **POPF**

Transfiere bits de la palabra almacenada en la parte superior de la pila hacia el registro de banderas.

La forma de transferencia es la siguiente:

BIT BANDERA 0 CF ___ 2 PF ___ 4 AF ___ 6 ZF 7 SF 8 TF 9 IF 10 DF 11 OF

Estas localizaciones son las mismas para el comando PUSHF

Una vez hecha la transferencia se incrementa en 2 el registro SP disminuyendo así el tamaño de la pila.

PUSH

Coloca una palabra en la pila.

Sintaxis: **PUSH fuente**

La instrucción PUSH decrementa en dos el valor de SP y luego transfiere el contenido del operando fuente a la nueva dirección resultante en el registro recién modificado.

El decremento en la dirección se debe a que al agregar valores a la pila ésta crece de la dirección mayor a la dirección menor del segmento, por lo tanto al restarle 2 al valor del registro SP lo que hacemos es aumentar el tamaño de la pila en dos bytes, que es la única cantidad de información que puede manejar la pila en cada entrada y salida de datos.

PUSHF

Coloca el valor de las banderas en la pila

Sintaxis: **PUSHF**

Decrementa en 2 el valor del registro SP y luego se transfiere el contenido del registro de banderas a la pila, en la dirección indicada por SP.

Las banderas quedan almacenadas en memoria en los mismos bits indicados en el comando POPF

Instrucciones lógicas

AND

Realiza la conjunción de los operandos bit por bit.

Sintaxis: **AND destino, fuente**

Con esta instrucción se lleva a cabo la operación "y" lógica de los dos operandos:

Fuente	Destino	Resultado en operando destino
1	1	1
1	0	0
0	1	0
0	0	0

NEG

Genera el complemento a 2

Sintaxis: **NEG destino**

Genera el complemento a 2 del operando destino y lo almacena en este mismo operando.

Ejemplo, si AX guarda el valor de -2 (FFFE), entonces:

NEG AX

Dejaría como resultado en AX el valor 0002.

NOT

Lleva a cabo la negación bit por bit del operando destino.

Sintaxis: **NOT destino**

El resultado se guarda en el mismo operando destino.

OR

OR inclusivo lógico

Sintaxis: **OR destino, fuente**

La instrucción OR lleva a cabo, bit por bit, la disyunción inclusiva lógica de los dos operandos:

Fuente	Destino	Resultado en operando destino
1	1	1
1	0	1
0	1	1
0	0	0

TEST

Compara logicamente los operandos

Sintaxis: **TEST destino, fuente**

Realiza una conjunción, bit por bit, de los operandos, pero a diferencia de AND esta instrucción no coloca el resultado en el operando destino, solo tiene efecto sobre el estado de las banderas.

XOR

OR exclusivo

Sintaxis: **XOR destino, fuente**

Su función es efectuar bit por bit la disyunción exclusiva lógica de los dos operandos.

Fuente	Destino	Resultado en operando destino
1	1	0
1	0	1
0	1	1
0	0	0

Instrucciones aritméticas

ADC

Adición con acarreo.

Sintaxis: **ADC destino, fuente**

Lleva a cabo la suma de dos operandos y suma uno al resultado en caso de que la bandera CF esté activada, esto es, en caso de que exista acarreo.

El resultado se guarda en el operando destino.

ADD

Adición de los operandos.

Sintaxis: **ADD destino, fuente**

Suma los dos operandos y guarda el resultado en el operando destino.

DIV

División sin signo

Sintaxis: **DIV fuente**

El divisor puede ser un byte o palabra y es el operando que se le da a la instrucción.

Si el divisor es de 8 bits se toma como dividendo el registro de 16 bits AX y si el divisor es de 16 bits se tomara como dividendo el registro par DX:AX, tomando como palabra alta DX y como baja AX.

Si el divisor fué un byte el cociente se almacena en el registro AL y el residuo en AH, si fué una palabra el cociente se guarda en AX y el residuo en DX.

IDIV

División con signo

Sintaxis: **IDIV fuente**

Consiste basicamente en lo mismo que la instrucción DIV, solo que esta última realiza la operación con signo.

MUL

Multiplicación sin signo

Sintaxis: **MUL fuente**

El ensamblador asume que el multiplicando será del mismo tamaño que el del multiplicador, por lo tanto multiplica el valor almacenado en el registro que se le da como operando por el que se encuentre contenido en AH si el multiplicador es de 8 bits o por AX si el multiplicador es de 16 bits.

Cuando se realiza una multiplicación con valores de 8 bits el resultado se almacena en el registro AX y cuando la multiplicación es con valores de 16 bits el resultado se almacena en el registro par DX:AX.

IMUL

Multiplicación de dos enteros con signo.

Sintaxis: **IMUL fuente**

Este comando hace lo mismo que el anterior, solo que si toma en cuenta los signos de las cantidades que se multiplican.

Los resultados se guardan en los mismos registros que en la instrucción MUL.

SBB

Substracción con acarreo

Sintaxis: **SBB destino, fuente**

Esta instrucción resta los operandos y resta uno al resultado si CF está activada. El operando fuente siempre se resta del destino.

Este tipo de substracción se utiliza cuando se trabaja con cantidades de 32 bits.

SUB

Substracción

Sintaxis: **SUB destino, fuente**

Resta el operando fuente del destino.

Instrucciones de salto

JMP

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.

JA (JNBE)

Sintaxis: **JA Etiqueta**

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).

JAЕ (JNB)

Sintaxis: **JAЕ etiqueta**

El salto se efectua si CF esta desactivada.

JB (JNAE)

Sintaxis: **JB etiqueta**

Se efectúa el salto si CF esta activada.

JBE (JNA)

Sintaxis: **JBE etiqueta**

El salto se efectúa si CF está activado o si ZF está activado (que cualquiera sea igual a 1).

JE (JZ)

Sintaxis: **JE etiqueta**

El salto se realiza si ZF está activada.

JNE (JNZ)

Sintaxis: **JNE etiqueta**

El salto se efectua si ZF está desactivada.

JG (JNLE)

Sintaxis: **JG etiqueta**

El salto ocurre si $ZF = 0$ u $OF = SF$.

JGE (JNL)

Sintaxis: **JGE etiqueta**

El salto se realiza si $SF = OF$

JL (JNGE)

Sintaxis: **JL etiqueta**

El salto se efectúa si SF es diferente a OF.

JLE (JNG)

Sintaxis: **JLE etiqueta**

El salto se realiza si $ZF = 1$ o si SF es diferente a OF

JC

Sintaxis: **JC etiqueta**

Salta si hay acarreo. El salto se realiza si $CF = 1$

JNC

Sintaxis: **JNC etiqueta**

Salta si no hay acarreo. El salto se efectúa si $CF = 0$.

JNO

Sintaxis: **JNO etiqueta**

Salta si no hay desbordamiento. El salto se efectua si $OF = 0$.

JNP (JPO)

Sintaxis: **JNP etiqueta**

Salta si no hay paridad o salta si la paridad es non. El salto ocurre si $PF = 0$.

JNS

Sintaxis: **JNP etiqueta**

El salto se efectúa si $SF = 0$.

JO

Sintaxis: **JO etiqueta**

Salta si hay desbordamiento (overflow). El salto se realiza si $OF = 1$.

JP (JPE)

Sintaxis: **JP etiqueta**

Salta si hay paridad o salta si la paridad es par. El salto se efectúa si $PF = 1$.

JS

Sintaxis: **JS etiqueta**

El salto se efectúa si $SF = 1$.

Instrucciones de bucle

LOOP

Sintaxis: **LOOP etiqueta**

La instrucción loop decrementa CX en 1, y transfiere el flujo del programa a la etiqueta dada como operando si CX es diferente a 1.

LOOPE

Genera un ciclo en el programa considerando el estado de ZF

Sintaxis: **LOOPE etiqueta**

Esta instrucción decrementa CX en 1. Si CX es diferente a cero y ZF es igual a 1, entonces el flujo del programa se transfiere a la etiqueta indicada como operando.

LOOPNE

Genera un ciclo en el programa, considerando el estado de ZF

Sintaxis: **LOOPNE etiqueta**

Esta instrucción decrementa en uno a CX y transfiere el flujo del programa solo si ZF es diferente a 0.

DEC

Sintaxis: **DEC destino**

Esta operación resta 1 al operando destino y almacena el nuevo valor en el mismo operando.

INC

Sintaxis: **INC destino**

La instrucción suma 1 al operando destino y guarda el resultado en el mismo operando destino.

CMP

Sintaxis: **CMP destino, fuente**

Esta instrucción resta el operando fuente al operando destino pero sin que éste almacene el resultado de la operación, solo se afecta el estado de las banderas.

CMPS (CMPSB) (CMPSW)

Compara cadenas de un byte o palabra.

Sintaxis: **CMPS destino, fuente**

Con esta instrucción la cadena de caracteres fuente se resta de la cadena destino.

Se utilizan DI como índice para el segmento extra de la cadena fuente y SI como índice de la cadena destino.

Solo se afecta el contenido de las banderas y tanto DI como SI se incrementan.

Instrucciones de bandera

CLC

Limpia la bandera de acarreo.

Sintaxis: **CLC**

Esta instrucción apaga el bit correspondiente a la bandera de acarreo, o sea, lo pone en cero.

CLD

Limpia la bandera de dirección

Sintaxis: **CLD**

La instrucción CLD pone en cero el bit correspondiente a la bandera de dirección.

CLI

Limpia la bandera de interrupción

Sintaxis: **CLI**

CLI pone en cero la bandera de interrupciones, desabilitando así aquellas interrupciones enmascarables.

Una interrupción enmascarable es aquella cuyas funciones son desactivadas cuando $IF = 0$.

CMC

Complementar la bandera de acarreo.

Sintaxis: **CMC**

Esta instrucción complementa el estado de la bandera CF, si $CF = 0$ la instrucción la iguala a 1, y si es 1 la instrucción la iguala a 0, es decir, únicamente "invierte" el valor de la bandera.

STC

Activa la bandera de acarreo.

Sintaxis: **STC**

Esta instrucción pone la bandera CF en 1.

STD

Activa la bandera de dirección.

Sintaxis: **STD**

La instrucción STD pone la bandera DF en 1.

STI

Activa la bandera de interrupción.

Sintaxis: **STI**

La instrucción activa la bandera IF, esto habilita las interrupciones externas enmascarables (las que funcionan unicamente cuando $IF = 1$).

Interrupciones de software

Las interrupciones de software pueden ser activadas directamente por el ensamblador invocando al número de interrupción deseada con la instrucción INT.

Interrupción 21H

Llama a diversas funciones del DOS.

Sintaxis: **Int 21H**

Para acceder a cada una de ellas es necesario que en el registro AH se encuentre el número de función que se requiera al momento de llamar a la interrupción.

Funciones para desplegar información al video.

02H Exhibe salida

09H Impresión de cadena (video)

40H Escritura en dispositivo/Archivo

Funciones para leer información del teclado.

01H Entrada desde teclado

0AH Entrada desde teclado usando buffer

3FH Lectura desde dispositivo/archivo

Funciones para trabajar con archivos.

Método FCB (File Control Blocks)

0FH Abrir archivo
14H Lectura secuencial
15H Escritura secuencial
16H Crear archivo
21H Lectura aleatoria
22H Escritura aleatoria

Handles

3CH Crear archivo
3DH Abrir archivo
3EH Cierra manejador de archivo
3FH Lectura desde archivo/dispositivo
40H Escritura en archivo/dispositivo
42H Mover apuntador de lectura/escritura en archivo

Función 02H Despliega un caracter a la pantalla.

Registros de entrada: AH = 02H y DL = Valor del caracter a desplegar.

Registros de retorno: Ninguno

Función 09H Despliega una cadena de caracteres en la pantalla.

Registros de entrada: Ah = 09H y DS:DX = Dirección de inicio de una cadena de caracteres

Registros de retorno: Ninguno.

Esta función despliega los caracteres, uno a uno, desde la dirección indicada en el registro DS:DX hasta encontrar un caracter \$, que es interpretado como el final de la cadena.

Función 40H Se utiliza para escribir a un dispositivo o a un archivo.

Registros de entrada:

AH = 40H

BX = Vía de comunicación

CX = Cantidad de bytes a escribir

DS:DX = Dirección del inicio de los datos a escribir

Registros de retorno:

CF = 0 si no hubo error

AX = Número de bytes escritos

CF = 1 si hubo error

AX = Código de error

El uso de esta función para desplegar información en pantalla se realiza dándole al registro BX el valor de 1 que es el valor preasignado al video por el sistema operativo MS-DOS.

Función 01H Leer un caracter del teclado y desplegarlo.**Registros de entrada:** AH = 01H**Registros de retorno:** AL = Caracter leído

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

Función 0AH Leer caracteres del teclado y almacenarlos en un buffer.**Registros de entrada:**

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.

Cuando se han almacenado todos los caracteres indicados menos uno la bocina suena y cualquier caracter adicional es ignorado. Para terminar la captura de la cadena es necesario darle [ENTER].

Función 3FH Se utiliza para leer de un dispositivo o archivo.

Registros de entrada:

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 0FH

Abrir archivo FCB

Registros de entrada:

AH = 0FH

DS:DX = Apuntador a un FCB

Registros de retorno:

AL = 00H si no hubo problema, de lo contrario regresa 0FFH

Función 14H Leer secuencialmente un archivo FCB.

Registros de entrada:

AH = 14H

DS:DX = Apuntador a un FCB ya abierto.

Registros de retorno:

AL = 0 si no hubo errores, de lo contrario se regresará el código correspondiente de error: 1 error al final del archivo, 2 error en la estructura del FCB y 3 error de lectura parcial.

Esta función lo que hace es que lee el siguiente bloque de información a partir de la dirección dada por DS:DX, y actualiza este registro.

Función 15H Escribir secuencialmente a un archivo FCB**Registros de entrada:**

AH = 15H

DS:DX = Apuntador a un FCB ya abierto

Registros de retorno:

AL = 00H si no hubo errores, de lo contrario contendrá el código del error: 1 disco lleno o archivo de solo lectura, 2 error en la formación o especificación del FCB.

La función 15H después de escribir el registro al bloque actual actualiza el FCB.

Función 16H Crear un archivo FCB.**Registros de entrada:**

AH = 16H

DS:DX = Apuntador a un FCB ya abierto.

Registros de retorno:

AL = 00H si no hubo errores, de lo contrario contendrá el valor 0FFH

Se basa en la información proveída en un FCB para crear un archivo en el disco.

Función 21H Leer en forma aleatoria un archivo FCB.**Registros de entrada:**

AH = 21H

DS:DX = Apuntador a un FCB ya abierto.

Registros de retorno:

A = 00H si no hubo error, de lo contrario AH contendrá el código del error: 1 si es fin de archivo, 2 si existe error de especificación de FCB y 3 si se leyó un registro parcial o el apuntador del archivo se encuentra al final del mismo.

Esta función lee el registro especificado por los campos del bloque actual y registro actual de un FCB abierto y coloca la información en el DTA (área de transferencia de disco o Disk Transfer Area).

Función 22H Escribir en forma aleatoria en un archivo FCB.

Registros de entrada:

AH = 22H

DS:DX = Apuntador a un FCB abierto.

Registros de retorno:

AL = 00H si no hubo error, de lo contrario contendrá el código del error: 1 si el disco está lleno o es archivo de solo lectura y 2 si hay error en la especificación de FCB.

Escribe el registro especificado por los campos del bloque actual y registro actual de un FCB abierto. Escribe dicha información a partir del contenido del DTA (área de transferencia de disco).

Función 3CH Crear un archivo si no existe o dejarlo en longitud 0 si existe.

(Handle)

Registros de entrada:

AH = 3CH

CH = Atributo de archivo

DS:DX = Apuntador a una especificación ASCIIIZ

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 ASCIIIZ, la cual tiene como característica la de ser una cadena de bytes convencional terminada con un carácter 0.

El archivo creado contendrá los atributos definidos en el registro CX en la siguiente forma:

Valor Atributos

00H Normal

02H Escondido

04H Sistema

06H Escondido y de sistema

El archivo se crea con los permisos de lectura y escritura. No es posible crear directorios utilizando esta función.

Función 3DH Abre un archivo y regrese un handle

Registros de entrada:

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.

El código de acceso se especifica en la siguiente forma:

BITS

7 6 5 4 3 2 1

... 0 0 0 Solo lectura

... 0 0 1 Solo escritura

... 0 1 0 Lectura/Escritura

... X ... RESERVADO

Función 3EH Cerrar archivo (Handle).

Registros de entrada:

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 Leer de un archivo abierto una cantdad definida de bytes y los

almacena en un buffer específico.

Registros de entrada:

AH = 3FH

BX = Handle asignado

CX = Cantidad de bytes a leer

DS:DX = Apuntador a un área de trabajo.

Registros de retorno:

CF = 0 y AX = número de bytes leídos si no hubo error, en caso contrario CF = 1 y AX = código de error: 05H si acceso negado y 06H si no es válido el handle.

Función 40H Escribe a un archivo ya abierto una cierta cantidad de bytes a partir del buffer designado.

Registros de entrada:

AH = 40H

BX = Handle asignado

CX = Cantidad de bytes a escribir.

DS:DX = Apuntador al buffer de datos.

Registros de retorno:

CF = 0 y AX = número de bytes escritos si no hay errores, en caso de existir CF = 1 y AX = código del error: 05H si el acceso es negado y 06H si el handle es inválido.

Función 42H Mover apuntador al archivo (Handle)

Registros de entrada:

AH = 42H

AL = método utilizado

BX = Handle asignado

CX = La parte más significativa del offset

DX = La parte menos significativa del offset

Registros de retorno:

CF = 0 y DX:AX = la nueva posición del apuntador. En caso de error CF será 1 y AX = código de error: 01H si la función no es válida y 06H si el handle no es válido.

El método utilizado se configura como sigue:

Valor de AL Método

00H A partir del principio del archivo

01H A partir de la posición actual

02H A partir del final del archivo

Interrupción 10H

Su función es llamar a diversas funciones de video de la BIOS.

Sintaxis: **Int 10H**

Esta interrupción tiene diversas funciones, todas ellas nos sirven para controlar la entrada y salida de video, la forma de acceso a cada una de las opciones es por medio del registro AH.

Funciones comunes de la interrupción 10H.

02H Selección de posición del cursor

09H Escribe atributo y caracter en el cursor

0AH Escribe caracter en la posición del cursor

0EH Escritura de caracteres en modo alfanumérico

Función 02H Posiciona el cursor en la pantalla dentro de las coordenadas válidas de texto.

Registros de entrada:

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. Tenemos entonces 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 respectivamente.

Función 09H Desplegar un caracter un determinado número de veces con un atributo definido empezando en la posición actual del cursor.

Registros de entrada:

AH = 09H

AL = Caracter a desplegar

BH = Página de video en donde se desplegará

BL = Atributo a usar

Número de repeticiones.

Registros de retorno:

Ninguno

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

Función 0AH Desplegar un caracter en la posición actual del cursor.**Registros de entrada:**

AH = 0AH

AL = Caracter a desplegar

BH = Página en donde desplegar

BL = Color a usar (sólo en gráficos).

CX = Número de repeticiones

Registros de retorno: Ninguno.

La única diferencia entre esta función y la anterior es que ésta no permite modificar los atributos, simplemente usa los atributos actuales.

Tampoco se altera la posición del cursor con esta función.

Función 0EH Deplegar un caracter en la pantalla actualizando la posición del cursor.**Registros de entrada:**

AH = 0EH

AL = Caracter a desplegar

BH = Página donde se desplegara el caracter

BL = Color a usar (solo en gráficos)

Registros de retorno:

Ninguno

Interrupción 16H

Maneja la entrada/salida del teclado.

Sintaxis: **Int 16H**

Existen dos opciones de la interrupción 16H y son llamadas utilizando el registro AH.

Funciones de la interrupción 16H

00H Lee un caracter de teclado

01H Lee estado del teclado

Función 00H Leer un caracter del teclado.

Registros de entrada:

AH = 00H

Registros de retorno:

AH = código de barrido (scan code) del teclado

AL = Valor ASCII del caracter.

Cuando se utiliza esta interrupción se detiene la ejecución del programa hasta que se introduzca un caracter desde el teclado, si la tecla presionada es un caracter ASCII su valor será guardado en el registro AH, de lo contrario el código de barrido será guardado en AL y AH contendrá el valor 00H.

El código de barrido fué creado para manejar las teclas que no tienen una representación ASCII como [ALT], [CONTROL], las teclas de función, etc.

Función 01H Leer estado del teclado.

Registros de entrada:

AH = 01H

Registros de retorno:

Si la bandera de cero, ZF, está apagada significa que hay información en el buffer, si se encuentra prendida es que no hay teclas pendientes.

En caso de existir información el registro AH contendrá el código de la tecla guardada en el buffer.

Interrupción 17H

Maneja la entrada/salida de la impresora.

Sintaxis: **Int 17H**

Esta interrupción es utilizada para escribir caracteres a la impresora, inicializarla y leer su estado.

Funciones de la interrupción 16H

00H Imprime un caracter ASCII

01H Inicializa la impresora

02H Proporciona el estado de la impresora

Función 00H Escribir un caracter a la impresora.

Registros de entrada:

AH = 00H

AL = Caracter a imprimir

DX = Puerto a utilizar

Registros de retorno:

AH = Estado de la impresora.

El puerto a utilizar, definido en DX, se especifica así: LPT1 = 0, LPT2 = 1, LPT3 = 2 ...

El estado de la impresora se codifica bit por bit como sigue:

BIT 1/0 SIGNIFICADO

0 1 Se agotó el tiempo de espera

1 -

2 -

3 1 Error de entrada/salida

4 1 Impresora seleccionada

5 1 Papel agotado

6 1 Reconocimiento de comunicación

7 1 La impresora se encuentra libre

Los bits 1 y 2 no son relevantes.

Función 01H Inicializar un puerto de impresión.

Registros de entrada:

AH = 01H

DX = Puerto a utilizar

Registros de retorno:

AH = Status de la impresora

El puerto a utilizar, definido en DX, se especifica así: LPT1 = 0, LPT2 = 1, etc.

El estado de la impresora se codifica bit por bit como sigue:

BIT 1/0 SIGNIFICADO

0 1 Se agotó el tiempo de espera

1 -

2 -

3 1 Error de entrada/salida

4 1 Impresora seleccionada

5 1 Papel agotado

6 1 Reconocimiento de comunicación

7 1 La impresora se encuentra libre

Los bits 1 y 2 no son relevantes.

La mayoría de los BIOS únicamente soportan 3 puertos paralelos aunque existen algunos que soportan 4.

Función 02H Obtener el estado de la impresora.

Registros de entrada:

AH = 01H

DX = Puerto a utilizar

Registros de retorno:

AH = Status de la impresora.

El puerto a utilizar, definido en DX, se especifica así: LPT1 = 0, LPT2 = 1, etc.

El estado de la impresora se codifica bit por bit como sigue:

BIT 1/0 SIGNIFICADO

 0 1 Se agotó el tiempo de espera
 1 -
 2 -
 3 1 Error de entrada/salida
 4 1 Impresora seleccionada
 5 1 Papel agotado
 6 1 Reconocimiento de comunicación
 7 1 La impresora se encuentra libre

Sintaxis de un procedimiento

CALL NombreProc

NombreProc PROC

RET

NombreProc ENDP

Definición de una macro

Una macro es un grupo de instrucciones repetitivas en un programa que se codifican solo una vez y pueden utilizarse cuantas veces sea necesario.

La principal diferencia entre una macro y un procedimiento es que en la macro se hace posible el paso de parámetros y en el procedimiento no. Al momento de ejecutarse la macro cada parámetro es sustituido por el nombre o valor especificado al momento de llamarla.

Podemos decir entonces que un procedimiento es una extensión de un determinado programa, mientras que la macro es un módulo con funciones específicas que puede ser utilizado por

diferentes programas.

Otra diferencia entre una macro y un procedimiento es la forma de llamar a cada uno, para llamar a un procedimiento se requiere el uso de una directiva, en cambio la llamada a las macros se realiza como si se tratara de una instrucción del ensamblador.

Sintaxis: **NombreMac MACRO1 [parametro1, parametro2...]**

Aunque se tiene la funcionalidad de los parametros es posible crear una macro que no los necesite.

La directiva de terminación de la macro es: **ENDM**

Ejemplo:

Posicion MACRO Fila, Columna

PUSH AX

PUSH BX

PUSH DX

MOV AH, 02H

MOV DH, Fila

MOV DL, Columna

MOV BH, 0

INT 10H

POP DX

POP BX

POP AX

ENDM

Ejemplo de llamada a la macro:

Posicion 8, 10