

INDICE

	Página
Presentación	
 UNIDAD I: INTRODUCCION AL LENGUAJE ENSAMBLADOR	
Capítulo I: Introducción.	4
Capítulo II: Lenguaje de programación ensamblador.	25
Capítulo III: Técnicas de direccionamiento.	40
Capítulo IV: Juego de Instrucciones.	55
Capítulo V: Diseño de software.	69
 UNIDAD II: PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR	
Capítulo VI: Instrucciones del ensamblador.	78
Capítulo VII: Herramientas de programación.	114
Capítulo VIII: Interrupciones.	143
Capítulo IX: Operadores y pseudo-operadores.	149
 UNIDAD III: ADMINISTRACIÓN DE ARCHIVOS, PERIFÉRICOS Y MEMORIA	
Capítulo X: Administración de archivos.	152
Capítulo XI: Manejo de ficheros.	161
Capítulo XII: El BIOS.	172
Capítulo XIII: Programas residentes.	185
 REFERENCIAS BIBLIOGRAFICAS.	200

PRESENTACION

El propósito del presente manual es servir como apoyo a la enseñanza del lenguaje ensamblador de las computadoras personales. El material presentado es apropiado para un curso intermedio de programación en ensamblador, describiendo en forma detallada la interfaz software-hardware.

La evolución de la tecnología de computadoras es impresionante, y lo será aun más en los próximos años.

¿Por que aprender a programar en lenguaje ensamblador?

Esta pregunta es de gran debate entre los programadores y la diferencia de puntos de vista siempre gira alrededor de las aplicaciones. En general, los programadores que emplean lenguajes de alto nivel para desarrollar aplicaciones donde el tiempo no es factor critico o que hacen uso de un dispositivo estándar de entrada/salida, rara vez necesitan llamar rutinas que no formen parte de la librería del compilador. En otras palabras, las necesidades de programar en lenguaje ensamblador en este tipo de aplicaciones, es mínima.

Sin embargo, alguien debe escribir las rutinas de librerías que estos programadores emplean, con el fin de obtener la interfaz estándar. Estas rutinas forman la parte no transportable del lenguaje que utilizan y están escritas en lenguaje ensamblador. Asimismo, si parte del código de un programa escrito en lenguaje de alto nivel tarda mucho en ejecutarse, entonces la mejor opción para el programador es escribir una rutina en lenguaje ensamblador que haga lo mismo que el código en cuestión.

La programación en lenguaje ensamblador es la base para evaluar cualquier actividad profesional de programación.

El presente manual esta estructurado en tres partes, cada uno de los cuales se detallan a continuación:



UNIDAD I, describe el marco introductorio al lenguaje ensamblador, describiendo temas como: direccionamiento, el juego de instrucciones y diseño de software.

UNIDAD II, describe las Instrucciones del ensamblador, herramientas de programación, las interrupciones del BIOS – DOS y los operadores y pseudo-operadores.

UNIDAD III, describe la administración de archivos, Manejo de ficheros, el BIOS y programas residentes.

Finalmente se incluyen las referencias bibliográficas.



INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR



CAPITULO I

Introducción, definiciones, temas preliminares, sistemas de numeración, registros, buses, bytes, microprocesadores

1.1 Introducción

¿Tiene sentido dedicar nuestro tiempo a aprender a programar en un lenguaje, como es el ensamblador, de tan bajo nivel, teniendo a nuestra disposición sofisticadas herramientas de desarrollo rápido que, en minutos, son capaces de generar las aplicaciones más complejas que podamos imaginar?

Programando en ensamblador nos convertiremos prácticamente en artesanos del desarrollo de programas, ocupándonos de todas las tareas en las que deseemos intervenir personalmente.

El tiempo empleado en crear cualquier programa será superior pero, a cambio, obtendremos programas muchísimo mas pequeños e infinitamente mas rápidos que los que podamos crear en cualquier lenguaje de alto nivel.

Importancia del lenguaje ensamblador

La importancia del lenguaje ensamblador radica principalmente que se trabaja directamente con el microprocesador; por lo cual se debe de conocer el funcionamiento interno de este, tiene la ventaja de que en el se puede realizar cualquier tipo de programas que en los lenguajes de alto nivel tal vez no lo pueden realizar.

Otro punto sería que los programas en ensamblador ocupan menos espacio en memoria.

Ventajas del lenguaje ensamblador

1. Velocidad.

Como trabaja directamente con el microprocesador al ejecutar un programa, pues como este lenguaje es el más cercano a la máquina la computadora lo procesa más rápido.



2. Eficiencia de tamaño.

Un programa en ensamblador no ocupa mucho espacio en memoria porque no tiene que cargar librerías y demás como son los lenguajes de alto nivel.

3. Flexibilidad.

Es flexible porque todo lo que puede hacerse con una máquina, puede hacerse en el lenguaje ensamblador de esta máquina; los lenguajes de alto nivel tienen en una u otra forma limitante para explotar al máximo los recursos de la máquina.

Desventajas del lenguaje ensamblador

1. Tiempo de programación.

Como es un lenguaje de bajo nivel requiere más instrucciones para realizar el mismo proceso, en comparación con un lenguaje de alto nivel.

2. Programas fuentes grandes.

Por las mismas razones que aumenta el tiempo, crecen los programas fuentes; simplemente requerimos más instrucciones primitivas para describir procesos equivalentes.

3. Falta de portabilidad.

Porque para cada máquina existe un lenguaje ensamblador; por ello, evidentemente no es una selección apropiada de lenguaje cuando deseamos codificar en una máquina y luego llevar los programas a otros sistemas operativos o modelos de computadoras.

1.2 Definiciones

Ensambladores y procesadores

Los ordenadores personales, los grandes servidores, los pequeños Palm y Pocket Pc, los teléfonos móviles y hasta la mayoría de los electrodomésticos tienen en su interior uno o mas microprocesadores.



Cada microprocesador reconoce un cierto conjunto de instrucciones, cuenta con un determinado conjunto de registros y tiene una capacidad de direccionamiento concreta. Usando el lenguaje c, por mencionar uno de los mas conocidos, puede desarrollarse un programa con el mismo código fuente y funcionaria en un antiguo Msx, un Power Mac o un Pc. Utilizando ensamblador, por el contrario, esto no es posible porque los microprocesadores z80, PowerPc y Pentium son totalmente distintos.

Ensambladores y sistemas

Al programar en ensamblador no solo se utiliza el conjunto de instrucciones y registros de un cierto microprocesador sino que, además, se usaran dichas instrucciones para acceder a elementos hardware, como el adaptador de video, el teclado o los buses de comunicaciones de una cierta arquitectura de ordenador.

De igual manera, para efectuar ciertas tareas se utilizaran servicios puestos a disposición de las aplicaciones por el sistema operativo.

Cuando las tareas a efectuar son algo más complejas, como es recuperar información de un archivo en disco o abrir una ventana en un entorno grafico, suele recurrirse a los servicios que ofrece el sistema operativo. Este es otro punto en el cual se crean fuertes dependencias entre programa y plataforma. Son distintos, lógicamente, los servicios ofrecidos por Linux, DOS, Mac OS y Windows, no estando disponible las mismas funciones en cada una de ellas.

1.3 Temas preliminares

Herramientas

Para poder programar en ensamblador precisara algunas herramientas básicas como un editor para introducir código, un ensamblador y un enlazador. Adicionalmente se puede requerir un depurador.

Aparte también se puede necesitar módulos con declaraciones de macros, estructuras de datos y funciones, utilidades para generar archivos de recursos.



Editores

Es la primera pieza para poder crear un programa y guardarlo en un archivo, recuperarlo y, en general, efectuar todas las tareas habituales de edición.

Editores para:

- Dos: Edit, Assembler Editor, etc.
- Windows: bloc de notas, visual assembler.
- Linux: Emacs

Ensambladores

De poco nos servirá haber editado un programa sino tenemos un ensamblador, es decir la herramienta que traduzca ese código fuente al lenguaje de maquina (código objeto no ejecutable).

Variantes:

- TASM: usado bajo DOS.
- MASM: Usado bajo DOS y Windows.
- NASM: Usado bajo DOS, Windows y Linux.

Enlazadores

Un archivo ejecutable debe contar con uno o varios encabezados con información para el sistema operativo, no instrucciones para el procesador.

Los encabezados indican:

- El tipo de ejecutable,
- La memoria que necesita,
- Los datos que debe inicializarse en memoria,
- El punto de entrada, etc.



La herramienta encargada de tomar el código objeto generado por el ensamblador, añadir los encabezados apropiados y producir un archivo ya ejecutable es el conocido como linker o enlazador.

Variantes para:

- MASM: LINK,
- TASM: TLINK,
- NASM: No tiene un enlazador propio pero puede utilizar el ALINK.

Depuradores

Una de las fases más importantes del desarrollo de cualquier programa es el proceso de depuración. Dicho proceso adquiere aun mas importancia al programar en ensamblador, dado que las operaciones efectuadas son de muy bajo nivel y cualquier fallo puede provocar un funcionamiento erróneo o, incluso el bloqueo del sistema.

Variantes para:

- DOS: Debug, GRDBDL09.
- WINDOWS: Turbo Debugger, CodeView.
- LINUX: GDB

1.4 Sistemas de numeración informáticos

A la hora de programar en ensamblador necesitamos conocer además del sistema de numeración decimal que todos conocemos, otro que resulta imprescindible: El Sistema Binario.

Hay dos sistemas adicionales, el Octal y el Hexadecimal, que facilitan en ciertas situaciones, la codificación de valores sin tener que escribir una larga secuencia de ceros y unos.



Operar con números binarios

La base 2 o binaria es particularmente interesante a la hora de programar en ensamblador, ya que un dígito binario puede tener tan solo dos estados y, consecuentemente, ser usado para representar estados de puertas lógicas como el registro de indicadores que tiene el procesador.

Los dígitos válidos son 0 y 1.

Bits, Nibbles y Bytes

Hacen referencia a conjuntos formados por 1, 4 y 8 dígitos binarios, respectivamente.

0000 → 0	1000 → 8
0001 → 1	1001 → 9
0010 → 2	1010 → 10
0011 → 3	1011 → 11
0100 → 4	1100 → 12
0101 → 5	1101 → 13
0110 → 6	1110 → 14
0111 → 7	1111 → 15

Números con signo

Para representar números negativos en la base decimal, la que utilizamos a diario, basta con anteponer un signo – al número en cuestión. Esta notación, sin embargo, no es habitual al operar con números binarios.

En ensamblador no puede ponerse un signo – delante de un número binario y esperar que se interprete como negativo, ya que el número completo solo puede contener ceros y unos, ningún otro símbolo.

Utilizando el complemento a dos se asume que el séptimo bit del byte, el que aparece más a la izquierda, actúa como bit de signo. Si está a cero, el número es positivo, mientras que si está a 1 se interpretará como negativo.



Pasos:

1. Invertir todos los bits del numero positivo
2. A continuación sumar 1 al numero invertido

Ejemplo:

Representar el número -16 en binario

Solución:

Convertir 16 a binario : 00010000

Invertir todos los bits : 11101111

Sumar 1 : 00000001

El resultado es : 11110000

Por lo tanto -16 en binario es 11110000

Operar con números Hexadecimales

Operar con números binarios en ocasiones puede ser algo complicado porque se necesitan muchos dígitos para representar números relativamente pequeños.

La base hexadecimal o dieciséis emplea seis símbolos mas que la base decimal, las letras a A la F, para representar los números del 10 al 15 respectivamente.

0000 → 0	1000 → 8
0001 → 1	1001 → 9
0010 → 2	1010 → A
0011 → 3	1011 → B
0100 → 4	1100 → C
0101 → 5	1101 → D
0110 → 6	1110 → E
0111 → 7	1111 → F

Identificación de la base de un número en ensamblador

segment pila stack

resb 256



iniciopila:

segment codigo

```
mov ax, pila
mov ss, ax
mov sp, iniciopila
mov cx,10      ; número decimal
mov cx,10q     ; número octal
mov cx,10b     ; número binario
mov cx,10h     ; número hexadecimal
mov ah,4ch
int 21h
```

1.5 Registros, buses, bytes y microprocesadores

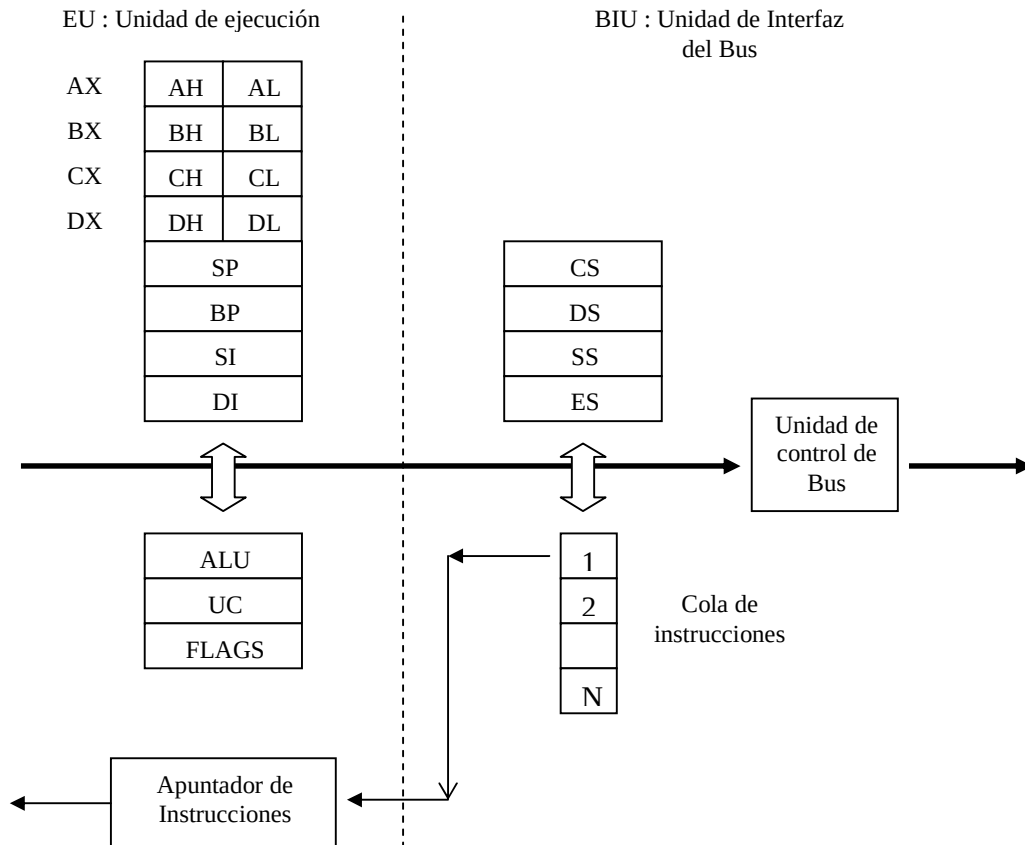
El procesador

Un elemento importante del hardware de la Pc es la unidad del sistema, que contiene una tarjeta de sistema, fuente de poder y ranuras de expansión para tarjetas opcionales. Los elementos de la tarjeta son un microprocesador Intel (o equivalente), memoria de solo lectura (Rom) y memoria de acceso aleatorio (Ram). El cerebro de la Pc y compatibles es un microprocesador basado en la familia 8086 de Intel, que realiza todo el procesamiento de datos e instrucciones. Los procesadores varían en velocidad y capacidad de memoria, registros y bus de datos.

PROCESADORES INTEL

PROCESADOR	REGISTRO DE...	BUS DE DATOS ...	DIRECCIONA (BYTE)
8088/80188	16 BITS	8 BITS	1 MILLON
8086/80186	16 BITS	16 BITS	1 MILLON
80286	16 BITS	16 BITS	16 MILLONES
80386	32 BITS	32 BITS	4,000 MILLONES
80486	32 BITS	32 BITS	MAS...
PENTIUM (*)	32 BITS	64 BITS	MAS...

(*) Intel patentó el nombre Pentium



Memoria Interna

Los microprocesadores poseen dos tipos de memoria interna: Ram y Rom. Los Bytes en memoria se numeran en forma consecutiva, iniciando con 00.

INICIO	DIRECCION	USO	
1024K	FFFFF	64K SISTEMA BASE DE ROM	MEMORIA SUPERIOR
960K	F0000	192K AREA DE EXPANSION DE MEMORIA (ROM)	
768K	C0000	128K AREA DE DESPLIEGUE DE VIDEO (RAM)	
640K	A0000	640K MEMORIA (RAM)	MEMORIA CONVENCIONAL
CERO	00000		

MAPA DE MEMORIA FISICA

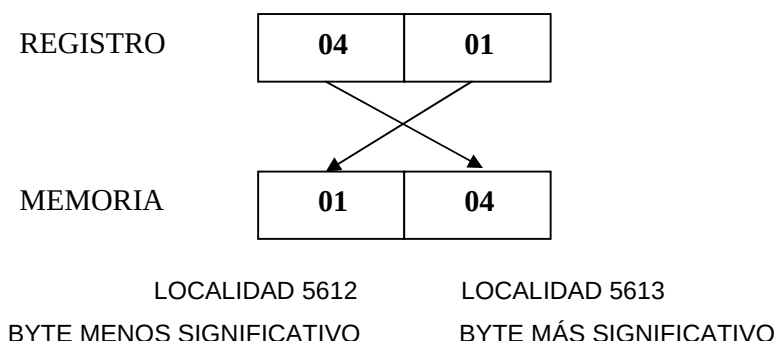


Direccionamiento de localidades de memoria

Dependiendo del modelo, el procesador puede acceder a uno más Bytes de memoria a la vez.

Considere el número decimal 1025. La representación hexadecimal de esta cifra es 0401h. Requiere de dos Bytes (1 palabra). Consta de 1 Byte de orden alto 04 y un Byte de orden bajo 01. El sistema almacena en memoria estos Bytes en secuencia inversa de Bytes.

Por ejemplo, el procesador transferirá 0401h de un registro a las localidades de memoria 5612 y 5613 como:



Segmentos y direccionamiento

Un segmento es un área especial en un programa que inicia en un límite de un párrafo. Esto es en una localidad regularmente divisible entre 16 o 10h.

Un segmento en modo real puede ser de hasta 64k.

Se puede tener cualquier número de segmentos; para direccionar un segmento en particular basta con cambiar la dirección en el registro del segmento apropiado.

Los tres segmentos principales son los segmentos de código, de datos y de la pila.



- Segmento de código (CS), contiene las instrucciones de maquina que son ejecutables.
- Segmento de datos (DS), contiene datos, constantes y áreas de trabajo definidas por el programador.
- Segmento de pila (SS), contiene los datos y direcciones que Ud. necesita para guardar temporalmente o para uso de sus "llamadas" rutinas.

Limite de los segmentos

Los registros de segmentos contienen la dirección inicial de cada segmento. Otros registros de segmentos son el es (segmento extra) y, en los procesadores 80386 y posteriores, los registros FS y GS, que tienen uso especializado.

Suponga que un DS inicia en la localidad de memoria 045f0H. Ya que en este caso y todos los demás casos el ultimo digito hexadecimal de la derecha es cero. Por acuerdo de los diseñadotes este último digito cero no se almacena, obteniéndose entonces 045fh.

Desplazamiento de registros

En un programa, todas las localidades de memoria están referidas a una dirección inicial de segmento. La distancia en bytes desde la dirección de segmento se define como el desplazamiento (offset).

Así el primer byte del segmento tiene un desplazamiento 00, el segundo un desplazamiento 01, etc., hasta el desplazamiento 65535.

Suponiendo que el registro DS tiene la dirección de segmento del segmento de datos en 045fh y una instrucción hace referencia a una localidad con un desplazamiento de 0032h bytes dentro del segmento de datos.

Por lo tanto, la dirección real será 04622h.

- DIRECCION SEGMENTO DS: 045F0H
- DESPLAZAMIENTO : 0032H



➤ DIRECCION REAL : 04622H

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.

Registro de segmento

Los registros de segmento tienen 16 bits de longitud, y facilita un área de memoria para direccionamiento.

- Registro CS
Esta dirección mas un valor de desplazamiento en el apuntador de instrucción, indica la dirección de una instrucción.
- Registro DS
Dirección inicial segmento datos.
- Registro SS
Permite la colocación de la pila en memoria.
- Registro ES
Es un registro extra
- Registro FS y GS
80386 y posteriores

Registro de apuntador de instrucciones

El registro apuntador de instrucciones (IP) de 16 bits contiene la dirección de la siguiente instrucción a ejecutar.

El IP esta asociado con el registro CS.

En el ejemplo siguiente CS contiene 25a4h y el IP contiene 412h, entonces:



- Dirección en DS : 25A40H
- Desplazamiento registro IP : 412H
- Dirección siguiente instrucción : 25E52H

Registros apuntadores

Los registros SP (apuntador de pila) y BP (apuntador base) están asociados con el registro SS y permiten acceder datos en el segmento de pila.

El registro SP es de 16 bits, y proporciona un valor de desplazamiento que se refiere a la palabra actual que esta siendo procesada en la pila.

El registro BP de 16 bits facilita la referencia de parámetros, los cuales son datos y direcciones transmitidos vía la pila. En los 80386 y posteriores es el EBP de 32 bits.

Registros de propósito general

Los registros de propósito general son AX, BX, CX, DX. Los procesadores 80386 y posteriores permiten el uso de registros de propósito general: EAX, EBX, ECX, EDX de 32 bits.

- Registro AX: Acumulador principal
- Registro BX: Registro base
- Registro CX: Registro contador
- Registro DX: Registro de datos

Registros índices

Los registros SI y DI están disponibles para direccionamiento indexado y para sumas y restas.

- Registro SI: el registro índice fuente es requerido por algunas operaciones con cadenas. Esta asociado al registro DS.
- Registro DI: el registro índice destino también es requerido por algunas operaciones con cadena. En este contexto esta asociado el registro ES.



Registro de banderas

De los 16 registros de bandera, nueve son comunes a toda la familia de procesadores 8086 e indican el estado actual de la maquina y el resultado del procesamiento.

- OF: Overflow
- DF: Dirección (1 izquierda, 0 derecha) al mover o comparar cadenas.
- IF: Indica una interrupción externa
- TF: Permite la depuración en modo de un paso
- SF: Signo (0 positivo, 1 negativo)
- ZF: Resultado operación aritmética (1 = resultado cero, 0 <> de cero)
- AF: acarreo auxiliar para aritmética especializada.
- PF: paridad. indica paridad par o impar de una operación en datos de 8 bits de orden bajo.
- CF: contiene el acarreo de orden más alto después de una operación aritmética.

Las banderas más importantes son OF, SF, ZF y CF para operaciones de comparación y DF para operaciones de cadena.

1.6 Asignación de valores

La necesidad mas urgente en el desarrollo de cualquier aplicación es la de asignar valores ya sea a registros o a celdas de memoria.

La instrucción ensamblador que en este caso nos interesa es: Mov

El formato de la instrucción es el siguiente:

MOV destino , origen

Un principio básico, es que tanto destino como origen deben ser del mismo tamaño



Valores inmediatos

El destino de un valor inmediato puede ser un registro o bien una posición de memoria, pero no otro valor inmediato.

- Sentencia invalida: MOV 10 , 24

- Sentencias validas: MOV AH , 24
MOV [10] , 24

La segunda sentencia valida introduce un valor en la memoria correspondiente a la dirección DS:10

Hay que tener en cuenta que este tipo de asignación no esta permitida en algunos casos.

Por ejemplo:

MOV DS , 0b800h

Es una sentencia inválida

lo correcto seria:

MOV AX , 0b800h

MOV DS , AX

Asignación entre registros

Siempre que los registros destino y origen sean del mismo tamaño, es posible prácticamente cualquier asignación. Podemos asignar un registro de propósito general a uno de segmento.

Sentencia valida: MOV DS , AX

Sentencia inválida: MOV CX, AL



Lectura de datos de la memoria

En el interior del microprocesador solo es posible mantener una pequeña cantidad de información, alojada en los registros de propósito general, por lo que es indispensable almacenar todos los datos en memoria, generalmente en una sección o segmento específico, recuperándolo cuando sea necesario.

En este caso el destino sería un registro, mientras que el origen sería la dirección de la celda de memoria.

Si escribimos: `MOV AH, 100`

Realmente estaríamos asignando a AH el valor 100.

Por el contrario,

Si escribimos: `MOV AH, [100]`

Realmente estaríamos asignando a AH el valor almacenado en la posición de memoria 100.

También es válido: `MOV BX, 100`

`MOV AH, [BX]`

Escritura de datos de la memoria

Como puede suponer, para escribir datos en la memoria hay, básicamente que intercambiar destino y origen. El destino sería una dirección de memoria, ya sea constante o alojada en un registro, mientras que el origen podría ser tanto un valor inmediato como el contenido de un registro.

Si el origen es un registro, este determinaría el tamaño del operando y, por tanto, la cantidad de información a transferir a la memoria

En el caso de los valores inmediatos es necesario indicar de manera explícita la cantidad de memoria a transferir.



En el ejemplo: `MOV [BX] , 24`

La cantidad de información no esta disponible.

Pero en: `MOV WORD PTR [BX] , 24`

La palabra Word es la que indica el tamaño del operador.

Lo anterior es valido en MASM, en NASM el modificar PTR no es valido

En el caso de que el valor este almacenado en un registro, por ejemplo AX, seria redundante indicar el tamaño del valor a transferir.

Por ejemplo:

`MOV WORD [BX] , AX`

`MOV [BX] , AX`

Son equivalentes

Definición de datos en el programa

Leer y escribir en celdas de memoria aleatoriamente, sin saber exactamente que es lo que hay en las celdas y para que use el sistema su contenido, no es una practica aconsejable.

De hecho podemos alterar el funcionamiento del programa o del ordenador e incluso bloquearlo.

En la mayoría de los casos, las posiciones de memoria empleadas por los programas corresponden a su propio segmento de datos en el que se han definido previamente.



Campos simples

La sintaxis para definir campos en el segmento de datos depende en parte del ensamblador usado.

Sintaxis:

IDENTIFICADOR INDICADOR VALOR

Ejemplos:

```
ASTERISCO DB "*" ; DB DEFINE BYTE  
POSICION DW 3280 ; DW DEFINE WORD  
VALOR32 DD 0 ; DD DEFINE DOBLE PALABRA
```

Conjunto de campos

Cuando se quiere múltiples campos de un mismo tamaño y asociados entre si, lo que en otros lenguajes de programación se conoce como matrices, arreglos o vectores, tenemos varias opciones.

Sintaxis:

[NOMBRE] INDICADOR Contador_de_repeticiones

Ejemplos:

```
ARR1 DW 10 DUP(?) ; Diez palabras no inicializadas  
ARR2 DB 5 DUP (14) ; OEOEOEOEOE  
DB 3 DUP(4 DUP (8)) ; 888888888888
```

Referencias al segmento de datos

Una vez definidos ya los campos que se precisen, para acceder a ellos desde el código del programa es necesario componer una dirección. El primer paso será asignar a un registro de segmento, generalmente DS, la dirección del segmento de datos.



Ejemplo 1:

MOV AX, SEG Caracteres

MOV DS , AX

Ejemplo 2:

MOV AX , DATOS ; Datos es el nombre del segmento

MOV DS , AX

1.7 Ejemplo de un programa en ensamblador

El objetivo del programa es mostrar en la pantalla de texto dos caracteres, en diferentes posiciones y con colores distintos. Para ello es necesario saber en que dirección comienza el área de memoria dedicada al contenido de la pantalla, así como la estructura con la que se almacena dicho contenido.

Todos los ordenadores que cuentan con un adaptador de gráficos que permite textos en color, y ejecutan el sistema operativo DOS, colocan el contenido de la pantalla, caracteres y atributos, en el segmento B800h. La notación hexadecimal es la más habitual a la hora de hablar de segmentos. A partir de la dirección 0 de dicho segmento, existe una palabra por cada posición de pantalla, conteniendo uno de los bytes el código de carácter y el otro el atributo asociado: color de fondo y de letra (tinta).

Operando en el habitual modo de 25 líneas por 80 columnas cada una, obtendríamos que cada línea ocuparía 160 bytes, 2 por posición, y que la pantalla completa serían 4000 bytes. De esos bytes, el primero, en la posición 0 del segmento, contendría el código del carácter correspondiente a la columna 1 de la línea 1, mientras que el byte siguiente alojaría el atributo de dicho carácter. En el byte 2 tendríamos el código del carácter que aparece en la segunda columna de la misma fila, en el byte 4 el del carácter de la columna 3 y así sucesivamente.

Sabiendo que cada línea ocupa 160 bytes y cada columna 2 bytes, es fácil calcular donde hay que introducir un valor para hacer aparecer un carácter en



una cierta posición de la pantalla. La formula seria fila * 160 + columna * 2 si fila y columna se cuentan a partir de cero.

En cuanto byte de atributo asociado a cada carácter, esta estructurado en tres partes: color del carácter, en los bits 0 a 3; color de fondo, en los bits 4 a 6, e indicador de parpadeo, en el ultimo bit. Codificándolo en hexadecimal, el digito de la derecha indicaría el color del carácter, de 0 a F, y el de la izquierda el de fondo y el parpadeo.

Programa:

```
                                ; Definimos el segmento de datos
segment Datos

                                ; Definimos varios campos

Asterisco db "*"
Blanco   db 0f0h                ; Fondo blanco
Posición dw 3280                 ; 20 * 160 + 40 * 2 ; línea 20 columna 40
                                ; Segmento para la pila

segment Pila stack
        db 256

InicioPila:

                                ; Segmento de código
segment Codigo
..start:

                                ; inicializamos DS
MOV AX , Datos                 ; Para acceder a los datos
MOV DS , AX                    ; preparamos el registro ES para acceder al
                                ; segmento donde esta el contenido de la
                                ; pantalla

mov ax , 0b800h
mov es , ax

                                ; recuperamos en AL el valor que hay en
                                ; asterisco

mov al , [Asterisco]

                                ; En AH el color
```



```
mov ah , [Blanco]
                                ; y en BX la posición donde queremos poner
                                ; el asterisco

mov bx , [Posicion]
                                ; Transferimos el contenido de AX a la
                                ; direccion ES:BX

mov [ES:BX] , ax
                                ; escribimos directamente en la pantalla un
                                ; valor inmediato

mov word ptr [es:5*160+35*2] , 00a41h
                                ; salimos al sistema

mov ah , 4ch
Int 21h
```



CAPITULO II

Lenguaje de programación ensamblador, contenido de un módulo fuente.

2.1 Lenguaje de programación ensamblador.

Comentarios en el lenguaje ensamblador

El uso de comentarios a lo largo del programa puede mejorar su claridad, en especial en el lenguaje ensamblador.

Un comentario empieza con punto y coma (;)

1. ; Toda la línea es un comentario
2. ADD AX,BX ; Comentario en la misma línea

Los comentarios no generan código de maquina

Palabras reservadas

Ciertas palabras en lenguaje ensamblador están reservadas para sus propósitos propios, y son usadas solo bajo condiciones especiales.

- Instrucciones : MOV , ADD
- Directivas : END , SEGMENT
- Operadores : FAR , SIZE
- Símbolos predefinidos : @DATA , @MODEL

El uso de una palabra reservada para un propósito equivocado provoca que el ensamblador genere un mensaje de error

Identificadores

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.



Un identificador puede utilizar los siguientes caracteres:

- Letras del alfabeto : A - Z
- Dígito : 0 – 9
- Caracteres especiales: ? , _ , \$, @ , .

Instrucciones

Un programa en lenguaje ensamblador consiste en un conjunto de enunciados.

Los dos tipos de enunciados son:

1. Instrucciones tal como MOV y ADD, que el ensamblador traduce a código objeto.
2. Directivas que indican al ensamblador que realice una acción específica, como definir un elemento de dato.

Sintaxis:

[IDENTIFICADOR] OPERACIÓN [OPERANDO(S)] ; Comentario

Ejemplo:

[IDENTIFICADOR] OPERACION [OPERANDO(S)] ; Comentario

```
COUNTER DB 1 ; Directiva
MOV AX , 0 ; Instrucción
INC CX ; Incrementa CX en 1
RET ; Retorna
```

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 genera código ejecutable de maquina.

Directiva: PAGE Y TITLE

La directiva PAGE designa el número máximo de líneas para listar en una página y el número máximo de caracteres por línea.

Se utiliza la directiva TITLE para hacer que se imprima un titulo en la línea 2 de cada página.

Sintaxis: PAGE [LONGITUD] [, ANCHO]

Ejemplo: PAGE 60 , 132 ; 60 LINEAS POR PAGINA
; Y 132 CARACTERES POR LINEA

Sintaxis: TITLE NOMBRE Comentario

Ejemplo: TITLE EJERCICIO1 Mi primer programa

Directiva: SEGMENT

Un programa ensamblado en formato .EXE consiste en uno o más segmentos. Un segmento de pila define el almacén de la pila, un segmento de datos define los elementos de datos y un segmento de código proporciona un código ejecutable.

Formato:

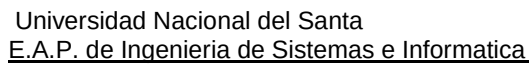
NOMBRE OPERACIÓN OPERANDO COMENTARIO

NOMBRE SEGMENT [OPCIONES] ; Inicia segmento

.
.

NOMBRE ENDS ; Fin del segmento

El operando de un enunciado segment puede tener tres tipos de opciones: alineación, combinar y clase, codificadas en este formato.



NOMBRE	SEGMENT	Alineación	Combinar	'CLASE'
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4
5	5	5	5	5
6	6	6	6	6
7	7	7	7	7
8	8	8	8	8
9	9	9	9	9
10	10	10	10	10
11	11	11	11	11
12	12	12	12	12
13	13	13	13	13
14	14	14	14	14
15	15	15	15	15
16	16	16	16	16
17	17	17	17	17
18	18	18	18	18
19	19	19	19	19
20	20	20	20	20
21	21	21	21	21
22	22	22	22	22
23	23	23	23	23
24	24	24	24	24
25	25	25	25	25
26	26	26	26	26
27	27	27	27	27
28	28	28	28	28
29	29	29	29	29
30	30	30	30	30
31	31	31	31	31
32	32	32	32	32
33	33	33	33	33
34	34	34	34	34
35	35	35	35	35
36	36	36	36	36
37	37	37	37	37
38	38	38	38	38
39	39	39	39	39
40	40	40	40	40
41	41	41	41	41
42	42	42	42	42
43	43	43	43	43
44	44	44	44	44
45	45	45	45	45
46	46	46	46	46
47	47	47	47	47
48	48	48	48	48
49	49	49	49	49
50	50	50	50	50
51	51	51	51	51
52	52	52	52	52
53	53	53	53	53
54	54	54	54	54
55	55	55	55	55
56	56	56	56	56
57	57	57	57	57
58	58	58	58	58
59	59	59	59	59
60	60	60	60	60
61	61	61	61	61
62	62	62	62	62
63	63	63	63	63
64	64	64	64	64
65	65	65	65	65
66	66	66	66	66
67	67	67	67	67
68	68	68	68	68
69	69	69	69	69
70	70	70	70	70
71	71	71	71	71
72	72	72	72	72
73	73	73	73	73
74	74	74	74	74
75	75	75	75	75
76	76	76	76	76
77	77	77	77	77
78	78	78	78	78
79	79	79	79	79
80	80	80	80	80
81	81	81	81	81
82	82	82	82	82
83	83	83	83	83
84	84	84	84	84
85	85	85	85	85
86	86	86	86	86
87	87	87	87	

PARA. Alinea el segmento con el límite de un párrafo, de manera que la posición inicial es múltiplo de 16. Se considera por defecto.

STACK, COMMON, PUBLIC, AT. Indica si el segmento se combina con otros segmentos cuando son enlazados después de ensamblar.

CODE, DATA, STACK. La entrada clase, encerrada entre apostrofes, es utilizada para agrupar segmentos cuando se enlazan.

NOMBRE	SEGMENT	PARA	STACK	'STACK'
--------	---------	------	-------	---------

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 solo un procedimiento puede aparecer como sigue:

NOMSEGMENTO	SEGMENT	PARA	
NOMPROC	PROC	FAR	; Un procedimiento dentro del
	.		; segmento de código
NOMPROC	ENDP		
NOMSEGMENTO	ENDS		

El operando FAR en este caso esta relacionado con la ejecución del programa. Cuando se solicita la ejecución de un programa, el cargador de



programas del DOS utiliza este nombre de procedimiento como el punto de entrada para la primera instrucción a ejecutar.

La directiva ENDP, indica el fin del procedimiento.

La directiva ENDS define el final de un segmento.

Directiva: ASSUME

Un programa utiliza el registro SS para direccionar la pila, al registro DS para direccionar el segmento de datos y al registro CS para direccionar el segmento de código. Para tal fin se tiene que indicar al ensamblador el propósito de cada segmento en el programa:

Formato:

OPERACION	OPERANDO
------------------	-----------------

ASSUME SS:Nomsegpila, DS:Nomsegdato, CS:Nomsegcodigo

También puede contener una entrada para ES.

ASSUME ES:Nomsegextra

Directiva: END

Así como ENDS finaliza un segmento y la directiva ENDP finaliza un procedimiento. Una directiva END finaliza todo el programa.

Formato:

OPERACION	OPERANDO
------------------	-----------------

END [Nombreprocedimiento]



Ejemplo de un programa EXE

PAGE 60,132

TITLE P04ASM1 (EXE) OPERACIONES MOVER Y SUMAR

STACKSG SEGMENT PARA STACK 'STACK'

DW 32 DUP(0)

STACKSG ENDS

DATASG SEGMENT PARA 'DATA'

FLDA DW 250

FLDB DW 125

FLDC DW ?

DATASG ENDS

CODESG SEGMENT PARA 'CODE'

BEGIN PROC FAR

ASSUME SS:STACKSG,DS:DATASG,CS:CODESG

MOV AX, DATASG ; SE ASIGNA DIRECCION DE DATASG

MOV DS, AX ; EN REGISTRO DS

MOV AX, FLDA ; MOVER 0250 A AX

ADD AX, FLDB ; SUMAR 0125 A AX

MOV FLDC, AX ; ALMACENAR SUMA EN FLDC

MOV AX, 4C00H ; PETICION

INT 21H ; SALIDA A DOS

BEGIN ENDP ; FIN DE PROCEDIMIENTO

CODESG ENDS ; FIN DE SEGMENTO

END BEGIN ; FIN DE PROGRAMA

Como inicializar en modo protegido

En modo protegido bajo el 80386 y procesadores posteriores, un programa puede direccionar hasta 16 megabytes de memoria. El uso de DWORD para



alineando segmentos en direcciones de palabras dobles incrementa la velocidad de acceso a memoria para buses de 32 bits.

La directiva `.386` indica al ensamblador que acepte instrucciones que son solo para estos procesadores; el operando `USE32` indica que genere código apropiado para el modo protegido de 32 bits.

`.386`

`NOMSEG SEGMENT DWORD USE32`

La inicialización del registro del segmento de datos podría parecerse a esto, ya que en estos procesadores el registro DS tiene un tamaño de 16 bits.

```
MOV EAX, DATASEG ; Obtiene la dirección del segmento de datos
MOV DS, AX
```

Las instrucciones STI, CLI, IN y OUT no están disponibles en modo protegido

Directivas simplificadas de segmentos

Los ensambladores de Microsoft y de Borland proporcionan algunas formas abreviadas para definir segmentos. Para usar estas abreviaturas, inicialice el modelo de memoria antes de definir algún segmento.

El formato general incluyendo el punto inicial es:

`.MODEL modelo de memoria`

El modelo de memoria puede ser: TINY, SMALL, MEDIUM, COMPACT o LARGE.

Directivas simplificadas de segmentos

Los requisitos de cada modelo son:



MODELO	Nº SEGMENTOS CODIGO	Nº SEGMENTOS DATOS
--------	---------------------	--------------------

TINY	*	*
SMALL	1	1
MEDIUM	MAS DE 1	1
COMPACT	1	MAS DE 1
LARGE	MAS DE 1	MAS DE 1

Puede utilizar cualquiera de estos modelos para un programa autónomo (un programa que no este enlazado a otro).

El modelo TINY esta destinado para uso exclusivo de programas .COM, los cuales tienen sus datos, código y pila en un segmento. Los formatos generales (incluyendo el punto inicial) para las directivas que define los segmentos de la pila, de datos y de código son:

```
. STACK [Tamaño]
. DATA
. CODE [Nombre]
```

Cada una de estas directivas hace que el ensamblador genere el enunciado SEGMENT necesario y su correspondiente ENDS.

Ejemplo programa EXE con directivas simplificadas

```
        PAGE    60,132
        TITLE   P04ASM1 (EXE) OPERACIONES MOVER Y SUMAR
;-----
        . MODEL  SMALL
        . STACK  64          ; SE DEFINE LA PILA
        . DATA
FLDA    DW    250
FLDB    DW    125
FLDC    DW    ?
;-----
```



```
. CODE ; Se define el segmento de código
BEGIN PROC FAR
    MOV AX , DATASG ; Se asigna dirección de DATASG
    MOV DS , AX ; en registro DS

    MOV AX , FLDA ; MOVER 0250 A AX
    ADD AX , FLDB ; SUMAR 0125 A AX
    MOV FLDC , AX ; Almacenar suma en FLDC

    MOV AX , 4C00H ; Petición
    INT 21H ; salida a DOS
BEGIN ENDP ; Fin de procedimiento
END BEGIN ; Fin de programa
```

Directivas .STARTUP y .EXIT

Masm 6.0 introdujo las directivas .STARTUP y .EXIT para simplificar la inicialización y terminación del programa.

.STARTUP genera las instrucciones para inicializar los registros de segmentos.

.EXIT genera las instrucciones de la int 21h, función 4ch para la salida del programa.

Definición de datos

Sintaxis:

[NOMBRE] Dn EXPRESION

NOMBRE: Hace referencia a un elemento de dato

DIRECTIVAS: DB (Byte), DW (Palabra), DD (Doble palabra), DF (Palabra larga), DQ (Palabra cuádruple), DT (Diez bytes)



EXPRESION: Es un operando que puede contener un signo de interrogación para indicar un elemento no utilizado.

Sintaxis:

[NOMBRE] Dn Contador_de_repeticiones DUP [EXPRESION]

DW 6 DUP (?) ; 6 PALABRAS NO INICIALIZADAS

DB 5 DUP (10) ; 5 PALABRAS CON AAAAA

DB 'PRIMERA PRACTICA DE ASM'

Otros:

DD Define una palabra doble

DF Define una palabra larga (6 bytes)

DQ Define una palabra cuádruple

DT Define diez bytes

Directiva EQU

No define elementos de datos. En lugar de eso, define un valor que el ensamblador puede usar para sustituir en otras instrucciones:

Lo siguiente:

TIMES EQU 10

FIELDA DB TIMES DUP(?)

Es equivalente a,

FIELDB DB 10 DUP(?)

2.2 Contenido de un modulo fuente.

Ejemplo: Determinar si un número es primo

TITLE PRIMO

DATOS SEGMENT



```
NUM    DW    24H
MENSAJE1 DB "ES PRIMO",0AH,0DH,"$"
MENSAJE2 DB "NO ES PRIMO",0AH,0DH,"$"
;
DATOS  ENDS
;
PROGRAMA  SEGMENT
;
        ASSUME CS:PROGRAMA; DS:DATOS
INICIO: MOV AX,DATOS
        MOV DS,AX
;
        MOV BX, 02H
SIGTE:  CMP BX,DS:NUM
        JE  PRIMO          ; NUM ES PRIMO SI BX=NUM
        MOV DX, 0H        ; PONER NUMERO EN DX AX, CON DX=0
        MOV AX,DS:NUM
        DIV BX             ; DIVISION DE 16 BITS, RESTO EN DX
        CMP DX, 0H
        JE  NO_PRIMO       ; SI RESTO ES CERO: SALTAR A NO_PRIMO
        INC BX
        JMP SIGTE          ; VOLVER A PROBAR EL SIGUIENTE
;
PRIMO:  MOV DX,OFFSET MENSAJE1      ; IMPRIME EMNSAJE 1
        MOV AH, 09H
        INT 21H
        JMP FIN
;
NO_PRIMO: MOV DX, OFFSET MENSAJE2 ; IMPRIME MENSAJE 2
        MOV AH,09H
        INT 21H
;
FIN:    MOV AH,4CH                ; FIN DEL PROCESO
        INT 21H
;
PROGRAMA  ENDS
;
        END INICIO
```

Como ensamblar un programa fuente

El programa ensamblador de Microsoft (hasta la versión 5.x) es Masm.exe, mientras que el programa de Borland es Tasm.exe. El ensamblador de Microsoft por lo general utiliza el comando ML, pero también acepta Masm por compatibilidad de versiones anteriores.

MASM/TASM [Opciones] Fuente [,Objeto] [,Listado] [,Ref. Cruzadas]



Opciones: estipula características como configuración del nivel de mensajes.

- /A Acomoda los segmentos en orden alfabético
- /C Crea un archivo de referencias cruzadas
- /D MASM: Produce archivo de listado de la pasada 1 y de la pasada 2 para localizar errores. Para TASM, /D símbolo significa define un símbolo.
- /E Acepta instrucciones del coprocesador 80x87 y genera un enlace a BASIC, C o FORTRAN para emular instrucciones de punto flotante
- /H Muestra opciones de ensamblador con una breve explicación. Ingrese /H (help, ayuda) sin nombre de archivo u otras opciones.
- /L Crea un listado de archivo normal
- /ML hace todos los nombres sensibles a mayúsculas y minúsculas
- /MU Convierte todos los nombres a mayúsculas
- /N Suprime la generación de la tabla de símbolos
- /R Proporciona soporte para coprocesador matemático
- /S Deja los segmentos fuente en la secuencia original
- /T (Breve) Muestran diagnostico al final del ensamblador solo si se encuentra un error
- /V (En extenso) Al final del ensamblado, muestra el número de líneas y símbolos procesados. (No para OPTASM)
- /Wn Establece el nivel de mensajes de emergencia: 0 = muestra solo errores críticos; 1 = muestra errores críticos y advertencias graves (por omisión); 2 = muestra errores críticos, advertencias graves y advertencias de consulta.
- /Z Muestra líneas fuente en la pantalla para errores
- /ZD Incluye información de números de líneas en archivo objeto para CodeView, TurboDebugger o SYMDEB
- /ZI Incluye información acerca de números de línea e información simbólica e el archivo objeto para CODEVIEW, TurboDebugger o SYMDEB



Referencias cruzadas: un archivo .CRF o .XRF es utilizado para producir un listado de referencias cruzadas de etiquetas, símbolos y variables de un programa. Sin embargo, tiene que usar CREF para Microsoft o TCREF para Borland a fin de convertir el listado a un archivo de referencias cruzadas ordenado.

Puede teclear CREF o TCREF con una línea de comando o utilizar peticiones.

Uso de peticiones:

CREF FILENAME [.CRF]

LIST FILENAME [CROSS-REF.REF]

- Para la primera petición, teclee el nombre del archivo sin la extensión .CRF,
- Para la segunda petición, puede teclear la unidad y/o la ruta y aceptar el nombre del archivo por omisión.

Como enlazar un programa

El enlazador de Microsoft es LINK y el de Borland TLINK. LINK y TLINK aceptan líneas de comando para solicitar un enlace. LINK también proporciona peticiones.

Enlace con una línea de comandos

LINK/TLINK [Opciones] Objfile, Exefile [,Mapfile] [,Libraryfile]

Opciones para depurar:

- /CO para DOS LINK
- /V de TLINK



Objfile:

Describe el archivo objeto generado por el ensamblador. El enlazador supone la extensión .obj, de modo que no tiene que ingresarla. También puede teclear la unidad o ruta.

Exefile:

Proporciona la generación de un archivo exe. El nombre de archivo y la unidad o ruta puede ser la misma o diferente del fuente.

Mapfile:

Proporciona la generación de un archivo con extensión .map que indica la localidad relativa y el tamaño de cada segmento y cualquier error que link ha encontrado. Ingresando con LE indica al ensamblador que muestre el mapa en la pantalla.

Libraryfile:

Proporciona la opción de bibliotecas.

Para enlazar mas de un archivo

```
LINK D:PROGA+D:PROGB+D:PROGC
```

Enlace por medio de indicaciones

Puede teclear solo el nombre del indicador sin línea de comando. Las indicaciones que hay que responder son:

```
LINK [ENTER]
```

```
OBJECT MODULE [.OBJ]: D:ASMPROG [ENTER]
```

```
RUN FILE [ASMPROG.EXE]: D: [ENTER]
```

```
LIST FILE [NUL.MAP]: CON [ENTER]
```

```
LIBRARIES [.LIB]: [ENTER]
```



Opciones de exe2bin

El programa exe2bin del dos convierte módulos .exe generados por Masm en módulos .com, dado que el programa fuente fue originalmente codificado de acuerdo con los requisitos .com.

Teclee el siguiente comando:

```
EXE2BIN D:Nomarch D:Nomarch.com
```



CAPITULO III

Técnicas de direccionamiento y aspecto de un programa ensamblador.

3.1 Técnicas de direccionamiento

Modos de direccionamiento

MODO	OPERADOR	REGISTRO BASE	EJEMPLO
REGISTRO	Registro	--	MOV AX , BX
VALOR	Valor Inmediato	--	MOV AX , 2604
VARIABLE	Offset Inmediato	DS	MOV AX , [1432]
INDIRECTO MEDIANTE REGISTRO	[BX]	DS	MOV DX , DS:[BX]
	[BP]	SS	MOV DX , SS:[BP]
	[DI]	DS	MOV DX , DS:[DI]
	[SI]	DS	MOV DX , SS:[DI]
RELATIVO A BASE	[BX] + despl.	DS	MOV CX , DS:[BX + 40000]
	[BP] + despl.	SS	MOV CX , SS:[BP + 40000]
DIRECTO INDEXADO	[DI] + despl.	DS	MOV CX , DS:[DI + 40000]
	[SI] + despl.	DS	MOV CX , DS:[SI + 40000]
INDEXADO A BASE	[BX] + [SI] + despl.	DS	MOV AX , DS:[BX][SI] + 300
	[BX] + [DI] + despl.	DS	MOV AX , DS:[BX][DI] + 300
	[BP] + [SI] + despl.	SS	MOV AX , SS:[BP][SI] + 300
	[BP] + [DI] + despl.	SS	MOV AX , SS:[BP][DI] + 300

Ejemplo: Demostración de los modos de direccionamiento

PAGE 40, 132

TITLE DEMODIR - Programa de demostración de direccionamiento

COMMENT *

DESCRIPCION: Esta rutina ilustra varios de los modos de direccionamiento disponibles en macro ensamblador

*

STACK SEGMENT PARA STACK 'STACK'

;

DB 64 DUP('STACK ')

;

STACK ENDS

;



```
DATA SEGMENT PARA PUBLIC 'DATA'
;
    DDDD DW 0
    DDDW DW 300
    DDDX DW 200
    DDDY DW 150
    DDDZ DW 125
    DDDQ DW 100
    DDDR DW 80
    DDDS DW 70
    DDDJ DW 60
    DDDU DW 50
;
DATA ENDS
;
CSEG SEGMENT PARA PUBLIC 'CODE'
;
START PROC FAR
    ASSUME CS:CSEG, DS:DATA, SS:STACK
    PUSH DS
    SUB AX,AX
    PUSH AX
;
    MOV AX, SEG DATA ; Localiza dirección del segmento DATA
    MOV DS, AX ; Carga en DS la dirección del segmento
;
    MOV AX, 10 ; Direcccionamiento modo inmediato (valor)
;
    MOV AX, DS:DDDW ; Direcccionamiento directo
;
    MOV BX, OFFSET DDDX ; Direcccionamiento indirecto de registro
    MOV AX, [BX]
;
    MOV AX, [BX+2] ; Direcccionamiento relativo a base
;
    MOV SI, 2 ; Direcccionamiento indexado directo
    MOV AX, DS:DDDZ[SI]
;
    MOV BX, OFFSET DDDW ; Direcccionamiento indexado de base
    MOV SI, 8
    MOV AX, [BX][SI+2]
;
    RET
;
START ENDP
;
CSEG ENDS
;
END START
```



3.2 El Programa Debug.

Ejecución de instrucciones

El DOS viene con un programa llamado Debug que es utilizado para probar y depurar programas ejecutables. Debug permite ejecutar un programa en modo de paso sencillo (un paso a la vez), de manera que se pueda ver el efecto de cada instrucción sobre las localidades de memoria y los registros.

Debug proporciona un conjunto de comandos que permiten realizar diferentes operaciones útiles.

Comandos de Debug.

- A Ensamblar instrucciones simbólicas y pasarlas a código de maquina
- D Mostrar el contenido de un área de memoria
- E Introducir datos en memoria, iniciando en una localidad específica
- G Correr el programa ejecutable que se encuentra en memoria
- N Nombrar un programa
- P Proceder a ejecutar un conjunto de instrucciones relacionadas
- Q Salir de la sesión con DEBUG
- R Mostrar el contenido de uno o mas registros
- T Rastrear la ejecución de una instrucción
- U “Desensamblar” código de maquina y pasarlo a código simbólico.
- W Escribir o grabar un programa en disco.

Reglas de los comandos de Debug

Para sus propósitos, Debug no distingue entre mayúsculas y minúsculas. También se introduce un espacio solo donde sea necesario separar parámetros en un comando.

Los tres ejemplos siguientes son equivalentes:

D DS:200
DDS:200



dds:200

El despliegue de Debug

Consiste en tres partes.

- A la izquierda esta la dirección hexadecimal del ultimo byte de la izquierda que se despliega en la forma de segmento:desplazamiento
- El área amplia del centro es la representación hexadecimal del área desplegada
- A la derecha esta la representación en ASCII de los bytes que contienen los caracteres desplegables.

DIRECCION |-REPRESENTACION- HEX - |--ASCII--|

```
xxxx.xx10  xx.....XX-XX.....XX  X.....X
xxxx.xx20  xx.....XX-XX.....XX  X.....X
xxxx.xx30  xx.....XX-XX.....XX  X.....X
```

Cada línea despliega 16 bytes de memoria. La dirección de la izquierda solo se refiere al último byte de la derecha.

Inicio con Debug:

Teclear DEBUG [Enter]

Visualización de las localidades de memoria

El comando D (display), lista 8 líneas de 16 bytes cada una.

Verificación de datos del sistema:

Una palabra de estado del equipo en el área de datos del Bios, ofrece una indicación rudimentaria de los dispositivos instalados. Esta palabra esta en las localidades 410h-411h, que puede ver desde Debug por medio de una dirección



de dos pares: 40 para la dirección del segmento y 10 para el desplazamiento desde la dirección del segmento.

D 40:10 [y presione la tecla enter]

El despliegue debe empezar con algo como esto:

0040:0010 63 44

En este ejemplo, los dos bytes en la palabra del estado del equipo contienen los valores hexadecimales 63 y 44. Invierta los bytes (44 63) y convierta a binario.

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

BITS DISPOSITIVO

- 15-14 nº de puertos paralelo para impresora conectados = 1 (bin 01)
- 11-9 nº de puertos serie conectados = 2 (bin 010)
- 7-6 nº de dispositivos de disco flexible = 2 (00=1, 01=2, 10=3, 11=4)
- 5-4 modo inicial de video = 10 (01=40x25 color, 10=80x12 25 en color)
- 1 1 = coprocesador matemático presente
- 0 1 = unidad de disco flexible esta presente

Verificación del tamaño de la memoria

El área del Bios que registra la cantidad de memoria instalada son las 413h y 414h.

D 40:13 [y presione la tecla Enter]

El despliegue debe empezar con algo como esto:

0040:0013



HEX INVERSO	HEX CORREGIDO	DECIMAL (K)
00 02	02 00	512
80 02	02 80	640

Verificación del n° de serie y de la nota de derechos reservados

El n° de serie de la computadora esta alojado en el Rom Bios en la localidad FE000H.

D FE00:0 [y presione la tecla Enter]

La pantalla debe mostrar un numero de serie de siete dígitos seguido, en maquinas convencionales de una nota de derechos reservados

Verificación de la fecha en el Rom Bios

La fecha esta en la localidad FFFF5H.

D FFFF:5 [y presione la tecla Enter]

Verificación de la identificación del modelo

La identificación esta en la localidad FFFFEH.

D FFFF:E [y presione la tecla Enter]

CODIGO	MODELO
F8	PS/2 MODELOS 70 Y 80
F9	PX COMPATIBLE
FA	PS/2 MODELO 30
FB	PC-XT (1986)
FC	PC-AT (1984), PC-XT MODELO 286, PS/2 MODELOS 50 Y 60, ETC
FE	PC-XT (1982), PORTATIL (1982)
FF	PRIMERA IBM PC



Ejemplo I de lenguaje de maquina: datos inmediatos

Este programa ejemplo utiliza datos inmediatos, datos definidos como parte de una instrucción.

INSTRUCCIÓN MAQUINA	CODIGO SIMBOLICO	EXPLICACION
B82301	MOV AX,0123H	MOVER EL VALOR 0123H A AX
052500	ADD AX,0025	SUMAR EL VALOR 0025H A AX
8BD8	MOV BX,AX	MOVER EL CONTENIDO AX A BX
03D8	ADD BX,AX	SUMAR CONTENIDO AX A BX
8BCB	MOV CX,BX	MOVER CONTENIDO CX, BX
2BC8	SUB CX,AX	RESTAR CONTENIDO AX DE CX
2BC0	SUB AX,AX	RESTAR AX DE AX
90	NOP	NO OPERACIÓN (NO HACE NADA)

Iniciar sesión con Debug.

(Ingresar Enter al final de cada línea)

```
-D CS:100
-E CS:100 B8 23 01 05 25 00
-E CS:106 8B D8 03 D8 8B CB
-E CS:10C 2B C8 2B C0 90
-R
-T
-T
-T
-T
-T
-T
-T
-T
-D CS:100
-Q
```



Ejemplo II de lenguaje de maquina: datos definidos

Este programa ejemplo utiliza datos como elementos dentro del programa.

DESPLAZAMIENTO DS	CONTENIDO HEXADECIMAL
-------------------	-----------------------

0200H	2301H
0202H	2500H
0204H	0000H
0206H	2A2A2A2AH

INSTRUCCIÓN	EXPLICACION
-------------	-------------

A10002	Mover la palabra que inicia en DS con el desplazamiento 0200h al registro AX
03060202	Sumar el contenido de la palabra que inicia en DS con desplazamiento 0202h al registro AX
a30402	Mover el contenido del registro AX a la palabra que inicia en DS con desplazamiento
90	No operación

Iniciar sesión con Debug.

(Ingresar Enter al final de cada línea)

```
-D DS:200
-E DS:200 23 01 25 00 00 00
-E DS:206 2A 2A 2A
-E CS:100 A1 00 02 03 06 02 02
-E CS:107 A3 04 02 90
-D DS:200,208
-D CS:100,10A
-R
-T
-T
-T
-D DS:0200,0208
-Q
```



Como introducir un programa simbólico en ensamblador

El comando A. (ensamblar)

Este comando le dice a Debug que acepte instrucciones simbólicas en ensamblador y las convierta en lenguaje de maquina.

Inicialice la dirección de inicio en el segmento de código con desplazamiento 100h para sus instrucciones como:

```
A 100      [ENTER]
MOV AL,25  [ENTER]
MOV BL,32  [ENTER]
ADD AL,BL  [ENTER]
NOP        [ENTER, ENTER]
```

El comando U. (desensamblar)

Este comando muestra el código de maquina para sus instrucciones en lenguaje ensamblador.

```
U 100,106 [ENTER]
```

La pantalla debe mostrar para la localidad, el código de maquina y el código simbólico.

El comando P. (proceder)

Ejecuta toda una rutina de interrupción del Dos.

```
-A 100 [ENTER]
  MOV AH,30 [ENTER]
  INT 21 [ENTER]
  NOP [ENTER,ENTER]
-R [ENTER]
-T [ENTER]
-P [ENTER]
```



En AL se muestra el número principal de la versión del DOS y en AH el número secundario de la versión.

La instrucción INT

Como obtener la fecha actual

```
-A 100 [ENTER]
MOV AH,2A [ENTER]
INT 21 [ENTER]
NOP [ENTER,ENTER]
-R [ENTER]
-T [ENTER]
-P [ENTER]
```

Los registros muestran esta información

AL: DIA DE LA SEMANA (0=DOMINGO)

CX: AÑO (EN HEXADECIMAL)

DH: MES (01 A 12)

DL: DIA DEL MES (01 A 31)

Como guardar un programa desde Debug

Puede utilizar el Debug para guardar un programa en disco bajo dos circunstancias:

1. Para leer el programa, modificarlo y después guardarlo, siga estos pasos:
 - Lea el programa bajo su nombre: Debug n:nombreachivo
 - Utilice el comando **d** para ver el programa en lenguaje de maquina y use **e** para ingresar los cambios
 - Utilice el comando **w** para grabar el programa revisado
2. Usar Debug para escribir un pequeño programa en lenguaje de maquina que ahora quiera guardar: siga estos pasos:
 - Solicite el programa Debug
 - Utilice el comando **a** (ensamblar) y **e** (ingresar) para crear el programa



- Ponga nombre al programa: **n** nombearchivo.com. la extensión del programa debe ser .com
- Ya que solo usted, sabe donde termina en realidad el programa, indique a Debug el tamaño del programa en bytes. examine este ejemplo:
xxxx:0100 mov al,25
xxxx:0102 mov bl, 32
xxxx:0104 add al,bl
xxxx:0106 nop
- Puesto que la ultima instrucción NOP, es de un byte, el tamaño del programa es de 7 bytes, desde 100h hasta 106h, inclusive.
- Primero utilice **r bx** para mostrar el **bx**, e ingrese 0 para limpiarlo
- Ahora use **r cx** para mostrar el registro **cx**. debug responde cx:0000 (valor cero) y usted contesta con el tamaño del programa, 7.
- grabe el programa revisado: **w** [enter]

Ejemplo: DEBUG
 -N LUIS.COM
 -A 100
 MOV AL,25
 MOV BL,32
 ADD AL,BL
 NOP
 -R BX
 0
 -R CX
 7
 -W
 -Q

La razón para limpiar el BX es porque la longitud del programa esta en la pareja
BX: CX



Ejemplo de lenguaje ensamblador: el operador ptr

```
100    MOV AX, [11A]
103    ADD AX , [11C]
107    ADD AX , 25
10A    MOV [11E], AX
10D    MOV WORD PTR [120] , 25
113    MOV BYTE PTR [122] , 30
118    NOP
119    NOP
11A    DB  14 23
11C    DB  05 00
11E    DB  00 00
120    DB  00 00 00
```

Para ejecutar este programa, primero teclee **a 100 [enter]** y luego ingrese las instrucciones. Al salir introduzca **r** para mostrar los registros, después en forma sucesiva **t**. Salga de la ejecución cuando llegue a nop en 118, teclee **d 110** para mostrar los registros cambiados de AX y de las localidades.

3.3 Escritura de programas .COM.

De forma automática el enlazador genera un formato particular para un programa .EXE.

También puede generar un programa .COM para ejecución. Un ejemplo de uso común de programa .COM es el command.com. Las ventajas de programas .COM están en que son mas pequeños que programas .EXE comparables y son mas fáciles de adaptar para actuar como programas residentes de memoria.

Diferencias entre programas .COM y .EXE

Tamaño del programa.



Un .exe puede ser de cualquier tamaño, mientras que un .com esta restringido a un segmento y a un máximo de 64k, incluyendo el PSP.

El PSP es un bloque de 256 bytes que el Dos inserta antes de los programas .com y .exe cuando los carga en memoria.

Segmentos.

De pila: en un .exe se define un segmento de pila, el .com lo genera en forma automática.

De datos: un .exe define un segmento de datos y lo inicializa en el registro DS. En el .com esta definido dentro del mismo segmento de código.

Inicialización.

Cuando el dos carga un programa .com para ejecución, inicializa en forma automática todos los registros de segmento del PSP.

Puesto que el direccionamiento comienza en un desplazamiento de 100h bytes desde el inicio del PSP, codifique una directiva ORG como ORG 100h inmediatamente después de segment (código) o el enunciado .CODE.

La directiva ORG le indica al ensamblador que empiece la generación del código objeto en un desplazamiento de 100h bytes pasando el inicio del PSP, en donde el programa .com real inicia.

Conversión a formato .com

Si el programa fuente ya esta escrito en formato .exe, puede utilizar un editor para convertir las instrucciones a formato .com.

Los formatos de codificación de Masm y Tasm para programas .com son idénticos, aunque sus métodos de conversión difieren.



Cuando la conversión a formato .com esta completa, puede borrar los archivos .obj y .exe.

Conversión con Microsoft

Para ambos programas, .exe y .com, con Masm de Microsoft se ensambla y produce un archivo .obj y después se enlaza para producir un programa .exe.

Un programa con nombre exe2bin convierte programas .exe a .com

```
EXE2BIN D:CALC D:CALC.COM
```

Conversión con Borland

```
TLINK /T D:CALC
```

3.4 Aspecto de un programa en ensamblador

Ejemplo de un programa .com

```

PAGE    60,132
TITLE   P07COM1 (COM) OPERACIONES MOVER Y SUMAR
;-----
CODSEG  SEGMENT PARA 'CODE'
        ASSUME SS:CODSEG,DS:CODSEG,CS:CODESG,ES:CODSEG
        ORG 100H
BEGIN:  JMP MAIN
;-----
FLDA    DW  250
FLDB    DW  125
FLDC    DW  ?
;-----
MAIN    PROC  NEAR
        MOV AX , FLDA      ; MOVER 0250 A AX
```



```
ADD AX , FLDB      ; SUMAR 0125 A AX
MOV FLDC , AX      ; ALMACENAR SUMA EN FLDC
MOV AX , 4C00H     ; PETICION
INT 21H            ; SALIDA A DOS
MAIN ENDP          ; FIN DE PROCEDIMIENTO
CODSEG ENDS        ; FIN DE SEGMENTO
END BEGIN          ; FIN DE PROGRAMA
```

La pila de .com

Para un programa .com el dos define de manera automática una pila y establece la misma dirección de segmento en los cuatro registros de segmento. Si el segmento de 64k para el programa es suficientemente grande, el Dos establece la pila al final del segmento y carga el registro SP con FFFEh, la parte superior de la pila.

Si el segmento de 64k no contiene espacio suficiente para una pila, el dos establece la pila al final de la memoria.

El comando DIR indica el tamaño de un archivo y le dará una idea del espacio disponible para una pila.



CAPITULO IV

Juego de instrucciones.

4.1 Operandos.

Un operando es una fuente de datos para una instrucción. Algunas instrucciones, como CLC y RET, no necesitan un operando, mientras que otras pueden tener uno o dos operandos.

Sintaxis:

OPERACIÓN OPERANDO1 , OPERANDO2

Operandos: REGISTRO

WORDX DW ?

...

MOV CX, WORDX ; Registro en el primer operando

MOV WORDX , BX ; Registro en el segundo operando

MOV CL , AH ; Registro en ambos operandos

Operandos: INMEDIATO

SAVE DB ?

...

ADD CX, 12 ; Suma 12 al CX

MOV SAVE,25 ; Mueve 25 a SAVE

Operandos: MEMORIA DIRECTA

WORD1 DW 0

BYTE DB 0

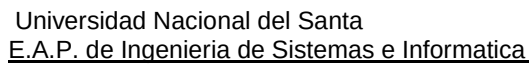
TABLEX DB 25 DUP(?)

...

MOV AX,WORD1 ; Carga word1 en AX

ADD BYTE1,CL ; Suma CL a Byte

MOV BX,DS:[38B0H] ; Mueve una palabra desde la memoria al



MOV AL, TABLE+4 ; La misma operación



Mueve registros:

```
MOV EAX , ECX      ; A REGISTRO
MOV DS,AX           ; A REGISTRO SEGMENTO
MOV BYTEVAL , BH    ; A MEMORIA DIRECTO
MOV [SI] , AX        ; A MEMORIA INDIRECTO
```

Mueve memoria directa:

```
MOV BH , BYTEVAL    ; A REGISTRO DIRECTO
MOV AX,WORDVAL [BX]  ; A REGISTRO INDIRECTO
```

Mueve registro de segmento:

```
MOV BH , DS          ; A REGISTRO
MOV WORDVAL , DS      ; A MEMORIA
```

Instrucciones para mover y llenar

Una limitación de la instrucción MOV es que el destino debe ser de la misma longitud que el fuente. En el 80386 y posteriores, las instrucciones MOVSX y MOVZX (mover y llenar) facilitan la transferencia de datos de un byte o palabra fuente a una palabra o palabra doble de destino.

Sintaxis:

[ETIQUETA:] MOVSX/MOVZX (REG/MEM) , (REG/MEM/INM)

REG = Registro MEM = Memoria INM = Inmediato

Considere mover un byte con 10110000 a una palabra; el resultado en la palabra destino depende de la elección de la instrucción:

```
MOVSX : 1111 1111 1011 0000
MOVZX : 0000 0000 1011 0000
```

Ejemplo:



BYTEVAL DB ?

WORDVAL DW ?

...

MOVSX AX,BYTEVAL

MOVSX EAX,WORDVAL

La instrucción XCHG

Intercambia datos

Sintaxis:

[ETIQUETA:] XCHG (REG/MEM) , (REG/INM)

WORD DW ?

...

XCHG AL , AH ; Intercambia los contenidos de los dos registros

XCHG AX , WORDX ; Intercambia los contenidos del registro y la memoria

La instrucción LEA

Es útil para inicializar un registro con una dirección de desplazamiento.

Sintaxis:

[ETIQUETA:] LEA (REG/MEM)

DATABLK 20 DB (?)

SAVBYTE DB ?

...

LEA BX, DATABLK ; carga la dirección de desplazamiento

; su equivalente es **MOV BX, OFFSET DATABLK**

MOV SAVBYTE, [BX] ; Mueve el primer byte de **DATABLK**

REG = Registro MEM = Memoria



Ejemplo de un programa en ensamblador.

```
PAGE    60,132
TITLE   P06MOVE (EXE) operaciones de movimientos extendidos
;-----
.MODEL  SMALL
.STACK  64
;-----
.DATA
NAME1   DB  'ABCDEFGHI'
NAME2   DB  'JKLMNOPRQ'
;-----
.CODE
BEGIN   PROC  FAR
        MOV  AX , @DATA    ; Inicia registro
        MOV  DS , AX       ; de segmento
        MOV  ES , AX
;
        MOV  CX , 9        ; Iniciación para mover 9 caracteres
        LEA  SI , NAME1    ; Inicialización de direcciones de name1
        LEA  DI , NAME2    ; y name2
B20:    MOV  AL , [SI]      ; Obtener carácter de name1
        MOV  [DI] , AL     ; moverlo a name2
        INC  SI            ; incrementar siguiente carácter en name1
        INC  DI            ; incrementar siguiente carácter en name2
        DEC  CX            ; Decrementar contador de iteraciones
        JNZ  B20           ; ¿contador diferente de cero ? SI, Iterar
        MOV  AX , 4C00H    ; Petición
        INT  21H           ; Salida a DOS
BEGIN   ENDP
        END  BEGIN
```

Direcciones cercanas y lejanas

En un programa una dirección puede ser cercana o lejana. Una dirección cercana solo consiste en parte de desplazamiento de una dirección. Una



instrucción que hace referencia a una dirección cercana supone al segmento actual – a saber, el DS para el segmento de datos y CS para el segmento de código.

Una dirección lejana consta de dos partes, la del segmento y la del desplazamiento, en la forma **segmento:desplazamiento**

4.3 Funciones del DOS de entrada y salida.

Función de impresión de carácter en pantalla: 02h

Se requiere:

Código 02h en AH

Código de carácter a imprimir en DL

Y se ejecuta con:

INT 21h

Función de lectura de carácter de teclado con eco: 01h

Se requiere:

Código 01h en AH

Y se ejecuta con:

INT 21h

Resultado: Código de carácter leído se almacena en AL

Función de impresión de carácter en impresora: 05h

Se requiere:

Código 05h en AH

Código de carácter a imprimir en DL

Y se ejecuta con:

INT 21h



Función de impresión de cadenas en pantalla: 09h

Se requiere:

Código 09h en AH

Dirección de comienzo de la cadena en DS:DX

Y se ejecuta con:

INT 21h

Se imprimen los caracteres hasta encontrar el signo \$

Función de lectura de cadenas por teclado: 0ah

Se requiere:

Código 0Ah en AH.

Dirección de área recepción o buffer en DS:DX.

Numero máximo N de caracteres a ser leídos en la dirección inicial del área de recepción.

Y se ejecuta con:

INT 21h

Resultado:

A partir de la dirección DS:DX+2 se almacenan consecutivamente los caracteres ingresados incluyendo al final 0dh. El numero de caracteres ingresados sin contar 0dh se pone en la dirección DS:DX+1

Ejemplo.

Si se ingresa la cadena: ABC B [enter]

En respuesta al conjunto de instrucciones:

MOV DX,150H ; Buffer empieza en DS:DX = DS:0150

MOV BYTE [0150],12h ; Numero máximo de caracteres = 18-1

MOV AH,0Ah ; Función de lectura

INT 21h ; Ejecución

A partir de la dirección DS:150 se tendrá:



		A	B	C			B	Retorno carro
12	05	41	42	43	20	42	0D	
Nº dado		Nº de		Caracteres leídos				

4.4 Operadores.

Operadores aritméticos

Operador: +

Formato: valor1 + valor2

Función: suma valor1 y valor2

Ejemplo:

tabla_mas_2 dw tabla + 2 ; (desplazamiento de tabla) +2

Operador: -

Formato: valor1 - valor2

Función: resta valor2 de valor1

Ejemplo:

difer tabla1 - tabla2 ; (desp. tabla1) – (desp. de tabla2)

Operador: *

Formato: valor1 * valor2

Función: multiplica valor2 por valor1

Ejemplo:

mins_dia equ 60 * 24 ; minutos de un dia

Operador: /

Formato: valor1 / valor2

Función: divide valor1 entre valor2 y retorna el cociente

Ejemplo:

pi_cocien equ 31416 / 10000 ; el valor es 3



Operador: mod

Formato: valor1 mod valor2

Función: divide valor1 entre valor2 y retorna el resto

Ejemplo:

pi_resto equ 31416 mod 10000 ; el valor es 1416

Operador: shl

Formato: valor shl expresion

Función: desplaza a la izquierda “valor” el número de bits que indica “expresion”

Ejemplo:

mascara equ 110010b

mascara_izq_2 equ mascara shl 2 ; 11001000b

Operador: shr

Formato: valor shr expresion

Función: desplaza a la derecha “valor” el número de bits que indica “expresion”

Ejemplo:

mascara equ 110010b

mascara_der_2 equ mascara shr 2 ; 1100b

Operadores lógicos

Operador: and

Formato: valor1 and valor2

Función: calcula el valor logico “y” de valor1 y valor2

Ejemplo:

binario equ 00110100b and 11010111b ; 00010100b

Operador: or

Formato: valor1 or valor2

Función: calcula el valor lógico “o” exclusivo de valor1 y valor2



Ejemplo:

binario equ 00110100b or 11010111b ; 11110111b

Operador: not

Formato: not valor

Función: obtiene el opuesto de cada bit

Ejemplo:

binario equ not 00110100b ; 11001011b

Operador: xor

Formato: valor1 xor valor2

Función: calcula el valor logico "o" inclusivo de valor1 y valor2

Ejemplo:

binario equ 00110100b xor 11010111b ; 11100011b

Operadores relacionales

Operador: eq

Formato: operando1 eq operando2

Función: verdad si los dos operandos son iguales

Ejemplo:

valor equ 20

mov ax,valor eq 20 ; equivale a mov ax,0ffffh

Operador: ne

Formato: operando1 ne operando2

Función: verdad si los dos operandos son distintos

Ejemplo:

valor equ 20

mov ax,valor ne 20 ; equivale a mov ax,0

Operador: lt

Formato: operando1 lt operando2

Función: verdad si operando1 < operando2



Ejemplo:

```
valor equ 20  
mov ax,valor lt 20 ; equivale a mov ax,0
```

Operador: gt

Formato: operando1 gt operando2

Función: verdad si operando1 > operando2

Ejemplo:

```
valor equ 20  
mov ax,valor gt 20 ; equivale a mov ax,0
```

Operador: le

Formato: operando1 le operando2

Función: verdad si operando1 <= operando2

Ejemplo:

```
valor equ 20  
mov ax,valor le 20 ; equivale a mov ax,0ffffh
```

Operador: ge

Formato: operando1 ge operando2

Función: verdad si operando1 >= operando2

Ejemplo:

```
valor equ 20  
mov ax,valor ge 20 ; equivale a mov ax,0ffffh
```

Operadores de retorno de valores

Operador: seg

Formato: seg variable ó seg etiqueta

Función: devuelve el valor del segmento de la variable o de la etiqueta

Ejemplo:

```
mov ax,seg tabla ; ax = segmento de tabla
```

Operador: offset

Formato: offset variable ú offset etiqueta



Función: devuelve el valor del desplazamiento de la variable o de la etiqueta

Ejemplo:

`mov ax, offset tabla ; ax = desplazamiento de tabla`

Operador: `type`

Formato: `type variable` ó `type etiqueta`

Función: si el operando es una variable, devuelve

- 1 – si byte
- 2 – si palabra
- 4 – si doble palabra
- 8 – si cuádruple palabra
- 10 – si variable definida con `dt`

Si el operando es una etiqueta, devuelve

-1 = 0ffffh – si `near`

-2 = 0ffffh – si `far`

Ejemplo:

`numero dw 20`

`mov ax, type numero ; equivale a mov ax, 2`

Operador: `size`

Formato: `size variable`

Función: devuelve el número de bytes reservados de “variable”. Solo se aplica a variables que se definen con `dup`.

Ejemplo:

`numero dw 20 dup (0)`

`mov ax, size numero ; equivale a mov ax, 40`

Operador: `length`

Formato: `length variable`

Función: devuelve el número de unidades (bytes o palabras) reservadas en “variable”. Solo se aplica a variables que se definen con `dup`.

Ejemplo:

`numero dw 20 dup (0)`



mov ax,length numero ; equivale a mov ax,20

Operadores de atributos

Operador: pointer

Formato: tipo ptr expresion

Función: redefine el atributo de tipo (byte, word, dword, qword, tbyte) o el atributo de distancia (near o far) de un operando de memoria.

tipo: nuevo atributo

expresion: identificador cuyo atributo se va a sustituir

Ejemplo:

tabla dw 100 dup (?) ; reserva 100 palabras

quinto_byte equ byte ptr tabla + 4 ; asigna nombre al quinto byte

Operador: \$

Formato: \$

Función: Este operador es otra manera de referirse al valor del contador de posiciones dentro del segmento. con \$ nos referimos al valor del contador de posiciones antes de que la sentencia se ensamble

Ejemplo:

texto db "esto es un texto" ; reserva memoria y la inicializa

longitud equ \$ - texto ; longitud de texto antes de ensamblarse

; \$ apunta a la posición siguiente a texto

Operador: high

Formato: high valor ó high expresión

Función: devuelve el byte superior de un valor numérico de 16 bits o expresión de dirección

Ejemplo:

constante equ 0abcdh

mov ah,high constante ; equivalea mov ah,0abh



Operador: low

Formato: low valor ó low expresion

Función: devuelve el byte inferior de un valor numérico de 16 bits o expresión de dirección

Ejemplo:

constante equ 0abcdh

mov ah,low constante ; equivale a mov ah,0cdh



CAPITULO V

Diseño de Software: Instrucciones de control, Codificación de las instrucciones.

5.1 Instrucciones de control

Direcciones corta, cercana y lejana

Una operación de salto alcanza una dirección corta de un desplazamiento de un byte, limitado a una distancia de -128 a 127 bytes.

Una operación de salto alcanza una dirección cercana por medio de un desplazamiento de una palabra, limitado a una distancia de -32768 a 32767 bytes dentro del mismo segmento.

Una dirección lejana puede estar en otro segmento y es alcanzada por medio de una dirección de segmento y un desplazamiento; CALL es la instrucción normal para este propósito.

	CORTA	CERCANA	LEJANA
INSTRUCCIONES	MISMO SEGMENTO -128 A 127	MISMO SEGMENTO -32768 A 32767	OTRO SEGMENTO
JMP	SI	SI	SI
Jnnn	SI	SI:80386 y POSTERIORES	NO
LOOP	SI	NO	NO
CALL	N/A	SI	SI

Etiquetas de instrucciones

Las instrucciones JMP, Jnnn (salto condicional) y LOOP requieren un operando que se refiere a la etiqueta de una instrucción.



Ejemplo:

JMP A90

...

A90:

MOV AH, 00

...

La instrucción loop

La instrucción JMP provoca un ciclo infinito. Pero es mas probable que una rutina realice un ciclo de un numero especifico de veces o hasta que se alcance una condición particular.

La instrucción LOOP, que sirve para este propósito, 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.

...

...

MOV CX,10 ; INICIALIZA EL CONTADOR

A20:

INC AX

ADD BX,AX

SHL DX,1 ; MULTIPLICA POR DOS A DX

LOOP A20

...

...



Registros de banderas (flags)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				O	D	I	T	S	Z		A		P		C

SF (bandera de signo).

Se establece de acuerdo con el signo (el bit de orden más alto) después de una operación aritmética. Positivo pone la bandera en 0 y negativo lo pone en 1.

JG y JL prueban esta bandera.

TF (bandera de trampa).

Cuando esta en 1, hace que el procesador ejecute en modo de un solo paso, esto es, una instrucción a la vez bajo el control del usuario. Ya estableció esta bandera cuando ingreso el comando **t** en debug.

IF (bandera de interrupción).

No permite interrupción cuando esta en 0 y permite interrupción cuando esta en 1.

DF (bandera de dirección).

Utilizado en operaciones de cadena para determinar la dirección de transferencia de datos. Cuando la bandera esta en 0, la operación incrementa los registros DI y SI, haciendo que la transferencia sea de izquierda a derecha. Usando la bandera en 1, la operación decrementa DI y SI haciendo que la transferencia de datos sea de derecha a izquierda.

OF (bandera de desbordamiento).

Indica un acarreo interno y uno externo en el bit de signo de alto orden después de una operación aritmética con signo



5.2 Codificación de las instrucciones

La instrucción cmp

Es utilizada para comparar dos campos de datos.

[ETIQUETA:] CMP (REG/MEM),(REG/MEM/INM)

...

CMP BX , 0

JZ B50

...

B50:

...

La instrucción de salto condicional

[ETIQUETA:] Jnnn Direccion_corta

SALTOS CON BASE EN DATOS SIN SIGNO

SIMBOLO	DESCRIPCION	BANDERA EXAMINADA
JE / JZ	SALTA SI ES IGUAL O SALTA SI ES CERO	ZF
JNE / JNZ	SALTA SI NO ES IGUAL O SALTA SI NO ES CERO	ZF
JA / JNBE	BIFURCA SI ES MAYOR O SALTA SI NO ES MENOR O IGUAL	CF , ZF
JAE / JNB	SALTA SI ES MAYOR O IGUAL O SALTA SI NO ES MENOR	CF
JB / JNAE	SALTA SI ES MENOR O SALTA SI NO ES MAYOR O IGUAL	CF
JBE / JNA	SALTA SI ES MENOR O IGUAL O SALTA SI NO ES MAYOR	CF , AF



SALTOS CON BASE EN DATOS CON SIGNO

SIMBOLO	DESCRIPCION	BANDERA EXAMINADA
JE / JZ	SALTA SI ES IGUAL O SALTA SI ES CERO	ZF
JNE / JNZ	SALTA SI NO ES IGUAL O SALTA SI NO ES CERO	ZF
JG / JNLE	SALTA SI ES MAYOR O SALTA SI NO ES MENOR O IGUAL	ZF, SF, OF
JGE / JNL	SALTA SI ES MAYOR O IGUAL O SALTA SI NO ES MENOR	SF , OF
JL / JNGE	SALTA SI ES MENOR O SALTA SI NO ES MAYOR O IGUAL	SF , OF
JLE / JNG	SALTA SI ES MENOR O IGUAL O SALTA SI NO ES MAYOR	ZF ,SF, OF

PRUEBAS ARITMETICAS ESPECIALES

SIMBOLO	DESCRIPCION	BANDERA EXAMINADA
JS	SALTA SI EL SIGNO ES NEGATIVO	SF
JNS	SALTA SI EL SIGNO ES POSITIVO	SF
JC	SALTA SI HAY ACARREO	CF
JNC	SALTA SI NO HAY ACARREO	CF
JO	SALTA SI HAY DESBORDAMIENTO	OF
JNO	SALTA SI NO HAY DESBORDAMIENTO	OF



Llamada a procedimientos

Operaciones CALL y RET

La instrucción CALL transfiere el control a un procedimiento llamado, y la instrucción RET regresa del procedimiento llamado al procedimiento original que hizo la llamada. RET debe ser la última instrucción en un procedimiento llamado.

Los formatos son:

[ETIQUETA:] CALL PROCEDIMIENTO

[ETIQUETA:] RET [INMEDIATO]

5.3 El segmento de pila

El segmento de pila es un área de memoria que se reserva para hacer almacenamiento temporal de datos (16 bits) o direcciones de retorno para los subprogramas (subrutinas).

La dirección de este segmento se mantiene en el registro SS. El registro puntero de pila SP contiene el desplazamiento de la cima de la pila cuya dirección absoluta es por tanto SS:SP. El registro de base de pila BP se utiliza para acceder a los datos del segmento de pila así: [bp] , [bp+2], [bp+si], etc, representan ss:bp, ss:bp+2, ss:bp+si, respectivamente.

nombre segment stack

db n dup(?)

nombre ends

En donde:

nombre es el identificador del segmento de programa de tipo pila.

n es el numero de direcciones o tamaño del segmento de pila declarado (que estará comprendido entre ss:0000 y ss:n-1



Apilamiento: PUSH

PUSH dato

Se apila o pone el valor de dato sobre la cima de la pila, es decir SP disminuye su valor en dos unidades y se escribe el valor del dato en la dirección actual (con los octetos en orden invertido).

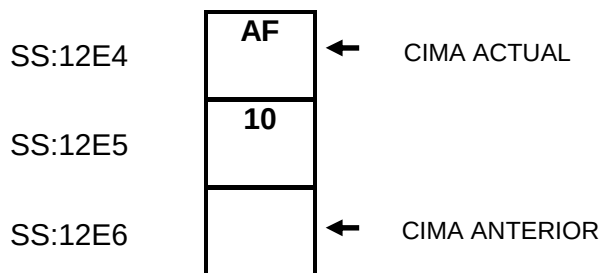
El dato puede ser un registro o un valor de memoria.

Ejemplo:

Si AX=10afh y SP=12e6h, entonces PUSH AX

Hace: SP = 12e4h (12e6-2) y pone el valor de AX en SS:SP.

Así en la pila se tiene:



Desapilamiento: POP

POP destino

Se desapila la palabra a la que apunta SP, esto es, se copia esta palabra en el destino (con los octetos en orden invertido) y SP aumenta su valor en dos unidades.

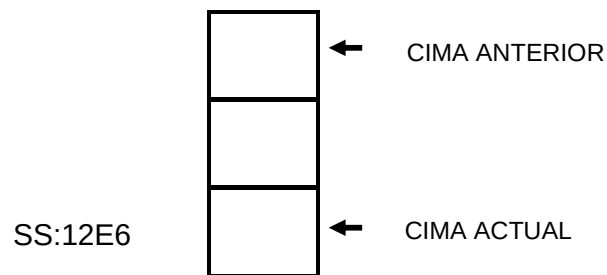


El dato puede ser un registro o un valor de memoria.

Ejemplo:

Si la pila se encuentra como en la figura anterior, entonces POP BX

Pone: SP en 12e6 y BX recibe el valor 10af.



Acceso a los datos del segmento de pila

Los datos del segmento de pila pueden ser accedidos (sin cambiar la cima de la pila) a través del registro base de pila BP:

`mov ax,[bp]` ; mueve a ax los dos octetos ubicados en SS:BP

`mov byte ptr [bp+4],12h` ; compara el octeto dado en ss:bp+4 con 12h



PROGRAMACION EN LENGUAJE ENSAMBLADOR



CAPITULO VI

Instrucciones del ensamblador: Instrucciones de uso más frecuente. Ejemplos de programación

6.1 INSTRUCCIONES DE USO MAS FRECUENTE

De carga de registros y direcciones

MOV (transferencia)

Sintaxis: MOV destino, origen.

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Transfiere datos de longitud byte o palabra del operando origen al operando destino. Pueden ser operando origen y operando destino cualquier registro o posición de memoria direccionada de las formas ya vistas, con la condición que origen y destino tengan la misma dimensión.

Existen ciertas limitaciones, como que los registros de segmento no admiten el direccionamiento inmediato: es incorrecto MOV DS,4000h; pero no lo es por ejemplo MOV DS,AX o MOV DS,VARIABLE.

No es posible, asimismo, utilizar CS como destino (es incorrecto hacer MOV CS,AX aunque pueda admitirlo algún ensamblador).

Ejemplos: mov ds,ax
 mov bx,es:[si]
 mov si,offset dato

En el 2do ejemplo, no se coloca en SI el valor de la variable dato sino su dirección de memoria o desplazamiento respecto al segmento de datos.



XCHG (intercambiar)

Sintaxis: XCHG destino, origen

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

- Intercambia el contenido de los operandos origen y destino.
- No pueden utilizarse registros de segmentos como operandos.

Ejemplo: xchg bl,ch

xchg mem_pal,bx

XLAT (traducción)

Sintaxis: XLAT tabla

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Se utiliza para traducir un byte del registro AL a un byte tomado de la tabla de traducción. Los datos se toman desde una dirección de la tabla correspondiente a BX + AL, donde BX es un puntero a el comienzo de la tabla y AL es un índice. Indicar tabla al lado de XLATt es una redundancia opcional.

Ejemplo: mov bx,offset tabla

mov al,4 xlat

LEA (carga dirección efectiva)



Sintaxis: LEA destino, origen

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Transfiere el desplazamiento del operando fuente al operando destino. Otras instrucciones pueden a continuación utilizar el registro como desplazamiento para acceder a los datos que constituyen el objetivo. El operando destino no puede ser un registro de segmento.

En general, esta instrucción es equivalente a MOV destino,OFFSET fuente.

LEA dx,datos[si]

En el ejemplo de arriba, el valor depositado en DX es el offset de la etiqueta datos mas el registro SI. Esa sola instrucción es equivalente a estas dos:

mov dx,offset datos

add dx,si

LDS (carga un puntero utilizando DS)

Sintaxis: LDS destino, origen

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Traslada un puntero de 32 bits (dirección completa de memoria compuesta por segmento y desplazamiento), al destino indicado y a DS. A partir de la dirección indicada por el operando origen, el procesador toma 4 bytes de la memoria: con los dos primeros forma una palabra que deposita en destino y, con los otros dos, otra en DS.



Ejemplo:

punt dd 12345678h

lds si,punt

Como resultado de esta instrucción en DS:SI se hace referencia a la posición de memoria 1234h:5678h; 'dd' sirve para definir una variable larga de 4 bytes (denominada punt en el ejemplo).

LES (carga un puntero utilizando ES)

Sintaxis: LES destino, origen

Esta instrucción es análoga a LDS, pero utilizando ES en lugar de DS.

LAHF (carga AH con los indicadores)

Sintaxis: LAHF

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Carga los bits 7, 6, 4, 2 y 0 del registro AH con el contenido de los indicadores SF, ZF, AF, PF Y CF respectivamente.

El contenido de los demás bits queda sin definir.

SAHF (copia AH en los indicadores)

Sintaxis: SAHF



Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - X X X X X

Transfiere el contenido de los bits 7, 6, 4, 2 y 0 a los indicadores SF, ZF, AF, PF y CF respectivamente.

Manipulación del registro de estado

CLC (baja el indicador de acarreo)

Sintaxis: CLC

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - - - - - 0

Borra el indicador de acarreo (CF) sin afectar a ninguno otro.

CLD (baja el indicador de dirección)

Sintaxis: CLD

Indicadores: OF DF IF TF SF ZF AF PF CF

- 0 - - - - - - -

Pone a 0 el indicador de dirección DF, por lo que los registros SI y/o DI se autoincrementan en las operaciones de cadenas, sin afectar al resto de los indicadores. Es necesario colocarlo antes de las instrucciones de manejo de cadenas si no se conoce con seguridad el valor de DF.

CLI (baja indicador de interrupción)

Sintaxis: CLI



Indicadores: OF DF IF TF SF ZF AF PF CF

- - 0 - - - - -

Borra el indicador de activación de interrupciones IF, lo que desactiva las interrupciones enmascarables.

CMC (complementa el indicador de acarreo)

Sintaxis: CMC

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - - - - - X

Complementa el indicador de acarreo CF invirtiendo su estado.

STC (pone a uno el indicador de acarreo)

Sintaxis: STC

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - - - - - 1

Pone a 1 el indicador de acarreo CF sin afectar a ningún otro indicador.

STD (pone a uno el indicador de dirección)

Sintaxis: STD

Indicadores: OF DF IF TF SF ZF AF PF CF

- 1 - - - - - - -

Pone a 1 el indicador de dirección DF, por lo que los registros SI y/o DI se autodecrementan en las operaciones de cadenas, sin afectar al resto de



los indicadores. Es necesario colocarlo antes de las instrucciones de manejo de cadenas si no se conoce con seguridad el estado de DF.

STI (pone a uno el indicador de interrupción)

Sintaxis: STI

Indicadores: OF DF IF TF SF ZF AF PF CF

- - 1 - - - - -

Pone a 1 la bandera de desactivación de interrupciones IF y activa las interrupciones enmascarables. Una interrupción pendiente no es reconocida, sin embargo, hasta después de ejecutar la instrucción que sigue a STI.

Manejo de la Pila

POP (extraer de la pila)

Sintaxis: POP destino

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Transfiere el elemento palabra que se encuentra en lo alto de la pila (apuntado por SP) al operando destino que a de ser tipo palabra, e incrementa en dos el registro SP.

Ejemplos: pop ax
pop pepe

PUSH (introduce en la pila)

Sintaxis: PUSH origen



Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Decrementa el puntero de pila (SP) en 2 y luego transfiere la palabra especificada en el operando origen a la cima de la pila. El registro CS aquí se puede especificar como origen.

Ejemplo: push cs

POPF (extrae los indicadores de la pila)

Sintaxis: POPF

Indicadores: OF DF IF TF SF ZF AF PF CF

X X X X X X X X X

Traslada al registro de los indicadores la palabra almacenada en la cima de la pila; a continuación el puntero de pila SP se incrementa en dos.

PUSHF (introduce los indicadores en la pila)

Sintaxis: PUSHF

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Decrementa en dos el puntero de pila y traslada a la cima de la pila el contenido de los indicadores.

Instrucciones de transferencia de control (incondicional)



CALL (llamada a subrutina)

Sintaxis: CALL destino

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Transfiere el control del programa a un procedimiento, salvando previamente en la pila la dirección de la instrucción siguiente, para poder volver a ella una vez ejecutado el procedimiento. El procedimiento puede estar en el mismo segmento (tipo NEAR) o en otro segmento (tipo FAR). A su vez la llamada puede ser directa a una etiqueta (especificando el tipo de llamada NEAR -por defecto- o FAR) o indirecta, indicando la dirección donde se encuentra el puntero. Según la llamada sea cercana o lejana, se almacena en la pila una dirección de retorno de 16 bits o dos palabras de 16 bits indicando en este caso tanto el offset (IP) como el segmento (CS) a donde volver.

Ejemplos: call proc1

dir dd 0f000e987h

call dword ptr dir

En el segundo ejemplo, la variable dir almacena la dirección donde saltar. De esta manera -conociendo su dirección puede llamarse también a un vector de interrupción guardando previamente los flags en la pila (PUSHF), porque la rutina de interrupción retornara con IRET en vez de con RETF.

JMP (salto)

Sintaxis: JMP direccion

JMP SHORT direccion

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -



Transfiere el control incondicionalmente a la dirección indicada en el operando. La bifurcación puede ser también directa o indirecta como anteriormente vimos, pero además puede ser corta (tipo SHORT) con un desplazamiento comprendido entre -128 y 127; o larga, con un desplazamiento de dos bytes con signo. Si se hace un JMP SHORT y no llega el salto (porque esta demasiado alejada esa etiqueta) el ensamblador dará error. Los buenos ensambladores (como TASM) cuando dan dos pasadas colocan allí donde es posible un salto corto, para economizar memoria, sin que el programador tenga que ocuparse de poner *short*. Si el salto de dos bytes, que permite desplazamientos de 64 Kb en la memoria sigue siendo insuficiente, se puede indicar con FAR que es largo (salto a otro segmento).

Ejemplos: `jmp etiqueta`

`jmp far ptr etiqueta`

RET / RETF (retorno de subrutina)

Sintaxis: `RET [valor]` o

`RETF [valor]`

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Retorna de un procedimiento extrayendo de la pila la dirección de la siguiente dirección. Se extrae el registro de segmento y el desplazamiento en un procedimiento de tipo FAR (dos palabras) y solo el desplazamiento en un procedimiento NEAR (una palabra). Si esta instrucción es colocada dentro de un bloque PROC-ENDP el ensamblador sabe el tipo de retorno que debe hacer, según el procedimiento sea NEAR o FAR.

En cualquier caso, se puede forzar que el retorno sea de tipo FAR con la instrucción RETF.



Valor, si es indicado permite sumar una cantidad *valor* en bytes a SP antes de retornar, lo que es frecuente en el código generado por los compiladores para retornar de una función con parámetros. También se puede retornar de una interrupción con RETF 2, para que devuelva el registro de estado sin restaurarlo de la pila.

Instrucciones de transferencia de control (condicional)

Las siguientes instrucciones son de transferencia condicional de control a la instrucción que se encuentra en la posición IP + desplazamiento (desplazamiento comprendido entre -128 y +127) si se cumple la condición. Algunas condiciones se pueden denotar de varias maneras. Todos los saltos son cortos.

JA/JNBE Salto si mayor, si no menor o igual, si CF=0 y ZF=0.

JAE/JNB Salto si mayor o igual, si no menor, si CF=0.

JB/JNAE/JC Salto si menor, si no superior ni igual, si acarreo, si CF=1.

JBE/JNA Salto si menor o igual, si no mayor, si CF=1.

JCXZ Salto si CX=0. JE/JZ Salto si igual, si cero, si ZF=1.

JG/JNLE Salto si mayor, si no menor ni igual, si ZF=0 y SF=0.

JGE/JNL Salto si mayor o igual, si no menor, si SF=0.

JL/JNGE Salto si menor, si no mayor ni igual, si SF<>OF.

JLE/JNG Salto si menor o igual, si no mayor, si ZF=0 y SF<>OF.

JNC Salto si no acarreo, si CF=0.

JNE/JNZ Salto si no igual, si no cero, si ZF=0.

JNO Salto si no desbordamiento, si OF=0.

JNP/JPO Salto si no paridad, si paridad impar, si PF=0.

JNS Salto si no signo, si positivo, si SF=0.

JO Salto si desbordamiento, si OF=1.

JP/JPE Salto si paridad, si paridad par, si PF=1.

JS Salto si signo, si SF=1.

Instrucciones de transferencia de control (gestión de bucle)



LOOP (bucle)

Sintaxis: LOOP desplazamiento

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Decrementa el registro contador CX; si CX es cero, ejecuta la siguiente instrucción en caso contrario transfiere el control a la dirección resultante de sumar a IP + desplazamiento. El desplazamiento debe estar comprendido entre -128 y +127.

Ejemplo:

```
mov cx,10
```

```
bucle: ..... .....
```

```
loop bucle
```

Con las mismas características que la instrucción anterior:

LOOPE/LOOPZ Bucle si igual, si cero. Z=1 y CX<>0

LOOPNE/LOOPNZ Bucle si no igual, si no cero. Z=0 y CX<>0

Instrucciones de transferencia de control (interrupciones)

INT (interrupción)

Sintaxis: INT n (0 ≤ n ≤ 255)

Indicadores: OF DF IF TF SF ZF AF PF CF

- - 0 0 - - - -

Inicializa un procedimiento de interrupción de un tipo indicado en la instrucción. En la pila se introduce al llamar a una interrupción la dirección de retorno formada por los registros CS e IP y el estado de los indicadores.



INT 3 es un caso especial de INT, al ensamblarla el ensamblador genera un solo byte en vez de los dos habituales; esta interrupción se utiliza para poner puntos de ruptura en los programas.

Ejemplo: int 21h

INTO (interrupción o desbordamiento)

Sintaxis: INTO

Indicadores: OF DF IF TF SF ZF AF PF CF

- - 0 0 - - - -

Genera una interrupción de tipo 4 (INT 4) si existe desbordamiento (OF=1), de lo contrario se continúa con la instrucción siguiente.

IRET (retorno de interrupción)

Sintaxis: IRET

Indicadores: OF DF IF TF SF ZF AF PF CF

x x x x x x x x x

Devuelve el control a la dirección de retorno salvada en la pila por una interrupción previa y restaura los indicadores que también se introdujeron en la pila. En total, se sacan las 3 palabras que fueron colocadas en la pila cuando se produjo la interrupción.

Instrucciones de entrada salida (e/s).



IN (entrada)

Sintaxis: IN acumulador, puerto.

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Transfiere datos desde el puerto indicado hasta el registro AL o AX, dependiendo de la longitud byte o palabra respectivamente. El puerto puede especificarse mediante una constante (0 a 255) o a través del valor contenido en DX (0 a 65535).

Ejemplo: in ax,0fh
 in al,dx

OUT (salida)

Sintaxis: OUT puerto, acumulador

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Transfiere un byte o palabra del registro AL o AX a un puerto de salida. El puerto puede especificarse con un valor fijo entre 0 y 255 o a través del valor contenido en el registro DX (de 0 a 65535).

Ejemplo: out 12h,ax
 out dx,al

Instrucciones aritméticas (suma).



AAA (ajuste ASCII para la suma)

Sintaxis: AAA

Indicadores: OF DF IF TF SF ZF AF PF CF

? - - - ? ? x ? x

Convierte el contenido del registro AL en un numero BCD no empaquetado. Si los cuatro bits menos significativos de AL son mayores que 9; si el indicador AF esta en 1, se suma 6 a AL, 1 a AH, AF se pone a 1, CF se iguala a AF y AL pone sus cuatro bits menos significativos a 0.

Ejemplo: add al,bl

aaa

En el ejemplo, tras la suma de dos números BCD no empaquetados colocados en AL y BL, el resultado (por medio de AAA) sigue siendo un numero BCD no empaquetado.

ADC (suma con acarreo)

Sintaxis: ADC destino, origen

Indicadores: OF DF IF TF SF ZF AF PF CF

x - - - x x x x x

Suma los operandos origen, destino y el valor del indicador de acarreo y el resultado lo almacena en el operando destino. Se utiliza normalmente para sumar números grandes, de mas de 16 bits, en varios pasos, considerando lo que nos llevamos (el acarreo) de la suma anterior.

Ejemplo: adc ax,bx



ADD (suma)

Sintaxis: ADD destino, origen

Indicadores: OF DF IF TF SF ZF AF PF CF

X - - - X X X X X

Suma los operandos origen y destino almacenando el resultado en el operando destino. Se activa el acarreo si se desborda el registro destino durante la suma.

Ejemplos: add ax,bx

add cl,dh

DAA (ajuste decimal para la suma)

Sintaxis: DAA

Indicadores: OF DF IF TF SF ZF AF PF CF

? - - - X X X X X

Convierte el contenido del registro AL en un par de valores BCD: si los cuatro bits menos significativos de AL son un número mayor que 9, el indicador AF se pone a 1 y se suma 6 a AL. De igual forma, si los cuatro bits menos significativos de AL tras la operación anterior son un número mayor que 9, el indicador CF se pone a 1 y se suma 60h a AL.

Ejemplo: add al,cl

daa

En el ejemplo anterior, si AL y CL contienen dos números BCD empaquetados, DAA hace que el resultado de la suma (en AL) siga siendo también un BCD empaquetado.



INC (incrementar)

Sintaxis: INC destino

Indicadores: OF DF IF TF SF ZF AF PF CF

X - - - X X X X -

Incrementa el operando destino. El operando destino puede ser byte o palabra.

Obsérvese que esta instrucción no modifica el bit de acarreo (CF) y no es posible detectar un desbordamiento por este procedimiento (utilice ZF).

Ejemplos: inc al

inc es:[di]

inc ss:[bp+4]

inc word ptr cs:[bx+di+7]

Instrucciones aritméticas (resta).

AAS (ajuste ASCII para la resta)

Sintaxis: AAS

Indicadores: OF DF IF TF SF ZF AF PF CF

? - - - ? ? x ? x

Convierte el resultado de la sustracción de dos operandos BCD no empaquetados para que siga siendo un numero BCD no empaquetado. Si el nibble inferior de AL tiene un valor mayor que 9, de AL se resta 6, se decrementa AH, AF se pone a 1 y CF se iguala a AF. El resultado se guarda en AL con los bits de 4 a 7 puestos a 0.

Ejemplo: sub al,bl



aas

En el ejemplo, tras la resta de dos números BCD no empaquetados colocados en AL y BL, el resultado (por medio de AAS) sigue siendo un número BCD no empaquetado.

DAS (ajuste decimal para la resta)

Sintaxis: DAS

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - X X X X X

Corrige el resultado en AL de la resta de dos números BCD empaquetados, convirtiéndolo también un valor BCD empaquetado. Si el nibble inferior tiene un valor mayor que 9 o AF es 1, a AL se le resta 6, AF se pone a 1. Si el nibble mas significativo es mayor que 9; CF es 1, entonces se resta 60h a AL y se activa después CF.

Ejemplo: sub al,bl

das

En el ejemplo anterior, si AL y BL contienen dos números BCD empaquetados, DAS hace que el resultado de la resta (en AL) siga siendo también BCD empaquetado.

DEC (decrementar)

Sintaxis: DEC destino

Indicadores: OF DF IF TF SF ZF AF PF CF

X - - - X X X X -



Resta una unidad del operando destino. El operando puede ser byte o palabra. Obsérvese que esta instrucción no modifica el bit de acarreo (CF) y no es posible detectar un desbordamiento por este procedimiento (utilice ZF).

Ejemplo: dec ax
dec mem_byte

NEG (negación)

Sintaxis: NEG destino

Indicadores: OF DF IF TF SF ZF AF PF CF
X - - - X X X X X

Calcula el valor negativo en complemento a dos del operando y devuelve el resultado en el mismo operando.

Ejemplo: neg al

SBB (resta con acarreo)

Sintaxis: SBB destino, origen

Indicadores: OF DF IF TF SF ZF AF PF CF
X - - - X X X X X

Resta el operando origen del operando destino y el resultado lo almacena en el operando destino. Si es 1 el indicador de acarreo además resta una unidad más. Los operandos pueden ser de tipo byte o palabra. Se utiliza normalmente para restar números grandes, de mas de 16 bits, en varios pasos, considerando lo que nos llevamos (el acarreo) de la resta anterior.



Ejemplo: sbb ax,ax
sbb ch,dh

SUB (resta)

Sintaxis: SUB destino, origen

Indicadores: OF DF IF TF SF ZF AF PF CF

X - - - X X X X X

Resta el operando destino al operando origen, colocando el resultado en el operando destino. Los operandos pueden tener o no signo, siendo necesario que sean del mismo tipo, byte o palabra.

Ejemplos: sub al,bl
sub dx,dx

Instrucciones aritméticas (multiplicación).

AAM (ajuste ASCII para la multiplicación)

Sintaxis: AAM

Indicadores: OF DF IF TF SF ZF AF PF CF

? - - - X X ? X ?

Corrige el resultado en AX del producto de dos números BCD no empaquetados, convirtiéndolo en un valor BCD también empaquetado. En AH sitúa cociente de AL/10 quedando en AL el resto de dicha operación

Ejemplo: mul bl
aam



En el ejemplo, tras el producto de dos números BCD no empaquetados colocados en AL y BL, el resultado (por medio de AAA) sigue siendo, en AX, un numero BCD no empaquetado.

IMUL (multiplicación entera con signo)

Sintaxis: IMUL origen

Indicadores: OF DF IF TF SF ZF AF PF CF

x - - - ? ? ? ? x

Multiplica un operando origen con signo de longitud byte o palabra por AL o AX respectivamente. Si *origen* es un byte el resultado se guarda en AH (byte mas significativo) y en AL (menos significativo), si *origen* es una palabra el resultado es devuelto en DX (parte alta) y AX (parte baja). Si las mitades menos significativas son distintas de cero, independientemente del signo, CF y OF son activados.

Ejemplo: imul bx

imul ch

MUL (multiplicación sin signo)

Sintaxis: MUL origen (*origen* no puede ser operando inmediato)

Indicadores: OF DF IF TF SF ZF AF PF CF

x - - - ? ? ? ? x

Multiplica el contenido sin signo del acumulador por el operando origen. Si el operando destino es un byte el acumulador es AL guardando el resultado en AH y AL, si el contenido de AH es distinto de 0 activa los indicadores CF y OF. Cuando el operando origen es de longitud palabra el acumulador es AX quedando el resultado sobre DX y AX, si el valor de DX es distinto de cero los indicadores CF y OF se activan.



Ejemplo: mul byte ptr ds:[di]

mul dx

mul cl

Instrucciones aritméticas (división).

AAD (ajuste ASCII para la división)

Sintaxis: AAD

Indicadores: OF DF IF TF SF ZF AF PF CF

? - - - x x ? x ?

Convierte dos números BCD no empaquetados contenidos en AH y AL en un dividendo de un byte que queda almacenado en AL. Tras la operación AH queda a cero. Esta instrucción es necesaria ANTES de la operación de dividir, al contrario que AAM.

Ejemplo: aad

div bl

En el ejemplo, tras convertir los dos números BCD no empaquetados (en AX) en un dividendo, la instrucción de dividir genera un resultado correcto.

DIV (división sin signo)

Sintaxis: DIV origen (*origen* no puede ser operando inmediato)

Indicadores: OF DF IF TF SF ZF AF PF CF

? - - - ? ? ? ? ?

Divide, sin considerar el signo, un numero contenido en el acumulador y su extensión AH, AL si el operando es de tipo byte o DX, AX si el



operando es palabra) 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. DX o AH deben ser cero antes de la operación. Cuando el cociente es mayor que el resultado mínimo que puede almacenar, cociente y resto quedan indefinidos produciéndose una interrupción. En caso de que las partes mas significativas del cociente tengan un valor distinto de cero se activan los indicadores CF y OF.

Ejemplo: div bl
div mem_pal

IDIV (división entera)

Sintaxis: IDIV origen (*origen* no puede ser operando inmediato)

Indicadores: OF DF IF TF SF ZF AF PF CF
 ? - - - ? ? ? ? ?

Divide, considerando el signo, un numero contenido en el acumulador y su extensión entre el operando fuente. El cociente se almacena en AL o AX según cual operando sea byte o palabra y de igual manera el resto en AH o DX. DX o AH deben ser cero antes de la operación. Cuando el cociente es positivo y superior al valor mínimo que puede almacenarse (7fh;7fffh), o cuando el cociente es negativo e inferior al valor mínimo que puede almacenarse (81h u 8001h) entonces cociente y resto quedan indefinidos, generándose una interrupción, lo que también sucede si el divisor es 0.

Ejemplo: idiv bl
idiv bx



Instrucciones aritméticas (conversiones).

CBW (conversiones de byte en palabra)

Sintaxis: CBW

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Copia el bit 7 del registro AL en todos los bits del registro AH, es decir, expande el signo de AL a AX como paso previo a una operación de 16 bits.

CWD (conversión de palabra a doble palabra)

Sintaxis: CWD

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Expande el signo del registro AX sobre el registro DX, copiando el bit más significativo de AH en todo DX.

Instrucciones de manipulación de cadenas

CMPS/CMPSB/CMPSW (compara cadenas)

Sintaxis: CMPS cadena_destino, cadena_origen

CMPSB (bytes)

CMPSW (palabras)

Indicadores: OF DF IF TF SF ZF AF PF CF

X - - - X X X X X



Compara dos cadenas restando al origen el destino. Ninguno de los operandos se alteran, pero los indicadores resultan afectados. La cadena origen se direcciona con registro SI sobre el segmento de datos DS y la cadena destino se direcciona con el registro DI sobre el segmento extra ES. Los registros DI y SI se autoincrementan o autodecrementan según el valor del indicador DF en una o dos unidades, dependiendo de si se trabaja con bytes o con palabras. *Cadena* origen y *cadena* destino son dos operandos redundantes que indican el tipo del dato (byte o palabra) a comparar, colocar CMPSB o CMPSW para indicar bytes/palabras. Si se indica un registro de segmento, se sustituirá en la cadena origen al DS ordinario.

Ejemplo:

lea si,origen

lea di,destino cmpsb

LODS/LODSB/LODSW (cargar cadena)

Sintaxis: LODS cadena_origen

 LODSB (bytes)

 LODSW (palabras)

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Copia en AL o AX una cadena de longitud byte o palabra direccionada sobre el segmento de datos (DS) con el registro SI. Tras la transferencia, SI se incrementa o decrementa según el indicador DF en una o dos unidades, según este manejando bytes o palabras. *Cadena_origen* es un operando redundante que indica el tipo del dato (byte o palabra) a cargar, colocar LODSB o LODSW para indicar bytes/palabras.



Ejemplo:

```
cld
lea si,origen
lods b
```

MOVS/MOVSb/MOVSsw (mover cadena)

Sintaxis: MOVS cadena_destino, cadena_origen
 MOVSb (bytes)
 MOVSsw (palabras)

Indicadores: OF DF IF TF SF ZF AF PF CF
 - - - - - - - - -

Transfiere un byte o una palabra de la cadena origen direccionada por DS:SI a la cadena destino direccionada por ES:DI, incrementando o decrementando a continuación los registros SI y DI según valor de DF en una o dos unidades, dependiendo de si se trabaja con bytes o con palabras. *Cadena origen* y *cadena destino* son dos operandos redundantes que indican el tipo del dato (byte o palabra) a comparar, colocar MOVSb o MOVSsw para indicar bytes/palabras. Si se indica un registro de segmento, se sustituirán la cadena origen al DS ordinario.

Ejemplo: lea si,origen
 lea di,destino
 movsw

SCAS/SCASb/SCASsw (explorar cadena)

Sintaxis: SCAS cadena_destino
 SCASb (bytes)
 SCASsw (palabras)



Indicadores: OF DF IF TF SF ZF AF PF CF
X - - - X X X X X

Resta de AX o AL una cadena destino direccionada por el registro DI sobre el segmento extra. Ninguno de los valores es alterado pero los indicadores se ven afectados. DI se incrementa o decrementa según valor de DF en una o dos unidades –según este trabajando con bytes o palabras- para apuntar al siguiente elemento de la cadena. *Cadena_destino* es un operando redundante que se indica el tipo del dato (byte o palabra), colocar SCASB o SCASW para indicar bytes/palabras.

Ejemplo: lea di,destino
mov al,50
scasb

STOS/STOSB/STOSW (almacena cadena)

Sintaxis: STOS *cadena_destino*
STOSB (bytes)
STOSW (palabras)

Indicadores: OF DF IF TF SF ZF AF PF CF
- - - - - - - -

Transfiere el operando origen almacenado en AX o AL, al destino direccionado por el registro DI sobre el segmento extra. Tras la operación DI se incrementa o decrementa según indicador DF para apuntar al siguiente elemento de la cadena. *Cadena_destino* es un operando redundante que se indica el tipo del dato (byte o palabra) a cargar.

Ejemplo: lea di,destino
mov ax,1991
stosw



REP/REPE/REPZ/REPNE/REPNZ (repetir)

Sintaxis: REP repetir operación de cadena
 REPE/REPZ repetir operación de cadena si igual/si cero
 REPNE/REPNZ repetir operación de cadena si no igual (si no 0)

Estas instrucciones se pueden colocar como prefijo de otra instrucción de manejo de cadenas, con objeto de que la misma se repita un número determinado de veces incondicionalmente o hasta que se verifique alguna condición. El número de veces se indica en CX. Por sentido como se deben utilizarse las siguientes combinaciones:

Prefijo	Función	Instrucciones

REP	Repetir CX veces	MOVS, STOS
REPE/REPZ	Repetir CX veces mientras ZF=1	CMPS, SCAS
REPNE/REPNZ	Repetir CX veces mientras ZF=0	CMPS, SCAS

Instrucciones de operaciones lógicas a nivel de bit.

AND (y)

Sintaxis: AND destino, origen

Indicadores: OF DF IF TF SF ZF AF PF CF
 0 - - - x x ? X 0

Realiza una operación Y entre el operando origen y destino quedando el resultado en el destino. Son dos operandos byte o palabra, pero ambos del mismo tipo.



Ejemplos: and ax,bx
and bl,byte ptr es:[si+10h]

NOT (no)

Sintaxis: NOT destino

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Realiza el complemento a uno del operando destino, invirtiendo cada uno de sus bits. Los indicadores no resultan afectados.

Ejemplo: not ax

OR (O)

Sintaxis: OR destino, origen

Indicadores: OF DF IF TF SF ZF AF PF CF

0 - - - x x ? X 0

Realiza una operación a nivel de bits entre los dos operandos, almacenándose después el resultado en el operando destino.

Ejemplo: OR AX,BX

TEST (comparación)

Sintaxis: TEST destino, origen

Indicadores: OF DF IF TF SF ZF AF PF CF

0 - - - x x ? X 0



Realiza una operación entre los dos operandos pero sin almacenar el resultado. Los indicadores son afectados con la operación

Ejemplo: test al,bh

XOR (O exclusivo)

Sintaxis: XOR destino, origen

Indicadores: OF DF IF TF SF ZF AF PF CF
 0 - - - x x ? x 0

Operación OR exclusivo a nivel de bits entre los operandos origen y destino almacenándose el resultado en este ultimo.

Ejemplo:

xor di,ax

Instrucciones de control del procesador

NOP (operación nula)

Sintaxis: NOP

Indicadores: OF DF IF TF SF ZF AF PF CF
 - - - - - - - - -

Realiza una operación nula, es decir, el microprocesador decodifica la instrucción y pasa a la siguiente. Realmente se trata de la instrucción XCHG AX,AX.

ESC (salida a un coprocesador)

Sintaxis: ESC ciclo_operacion,origen



Indicadores: OF DF IF TF SF ZF AF PF CF
- - - - -

Se utiliza en combinación procesadores externos, tales como los coprocesadores de coma flotante o de E/S, y abre al dispositivo externo el acceso a las direcciones y operandos requeridos. Al mnemónico ESC le siguen los ciclos de operación apropiados para el coprocesador así como la instrucción y la dirección del operando necesario.

Ejemplo:

esc 21,ax

HLT (parada hasta interrupción reset)

Sintaxis: HLT

Indicadores: OF DF IF TF SF ZF AF PF CF
- - - - -

El procesador se detiene hasta que se restaura el sistema o se recibe una interrupción. Como en los PC se producen normalmente 18,2 interrupciones de tipo 8 por segundo (del temporizador) algunos programadores utilizan HLT para hacer pausas y bucles de retardo. Sin embargo, el método no es preciso y puede fallar con ciertos controladores de memoria.

LOCK (bloquea los buses)

Sintaxis: LOCK

Indicadores: OF DF IF TF SF ZF AF PF CF
- - - - -



Es una instrucción que se utiliza en aplicaciones de recursos compartidos para asegurar que no accede simultáneamente a la memoria de un procesador. Cuando una instrucción va precedida por LOCK, el procesador bloquea inmediatamente el bus, introduciendo una señal por la patilla LOCK.

WAIT (espera)

Sintaxis: WAIT

Indicadores: OF DF IF TF SF ZF AF PF CF

- - - - -

Provoca la espera del procesador hasta que se detecta una señal en la patilla TEST. Ocurre, por ejemplo, cuando el coprocesador ha terminado una operación indica su finalización. Suele preceder a ESC para sincronizar las acciones del procesador y coprocesador.

Instrucciones de rotación y desplazamiento.

RCL (rotación a la izquierda con acarreo)

Sintaxis: RCL destino, contador

Indicadores: OF DF IF TF SF ZF AF PF CF

X - - - - - X

Rotar a la izquierda los bits del operando destino junto con el indicador de acarreo CF el numero de bits especificado en el segundo operando. Si el numero de bits a desplazar es 1, se puede especificar directamente, en caso contrario el valor debe cargarse en CL y especificar CL como segundo operando. No es conveniente que CL sea mayor de 7 en bytes; 15 en palabras.



Ejemplos:

rcl ax,1

rcl al,cl

rcl di,1

RCR (rotación a la derecha con acarreo)

Sintaxis: RCR destino, contador

Indicadores: OF DF IF TF SF ZF AF PF CF

X - - - - - - - X

Rotar a la derecha los bits del operando destino junto con el indicador de acarreo CF el numero de bits especificado en el segundo operando. Si el número de bits es 1 se puede especificar directamente; en caso contrario su valor debe cargarse en CL y especificar CL como segundo operando:

Ejemplos:

rcr bx,cl

rcr bx,1

ROL (rotación a la izquierda)

Sintaxis: ROL destino, contador

Indicadores: OF DF IF TF SF ZF AF PF CF

X - - - - - - - X

Rota a la izquierda los bits del operando destino el numero de bits especificado en el segundo operando, que puede ser 1; CL previamente cargado con el valor del numero de veces.



Ejemplos:

rol dx,cl

rol ah,1

ROR (rotación a la derecha)

Sintaxis: ROR destino, contador

Indicadores: OF DF IF TF SF ZF AF PF CF

x - - - - - x

Rota a la derecha los bits del operando destino el numero de bits especificado en el segundo operando. Si el número de bits es 1 se puede poner directamente, en caso contrario debe ponerse a través de CL.

Ejemplos:

ror cl,1

ror ax,cl

SAL/SHL (desplazamiento aritmético a la izquierda)

Sintaxis: SAL/SHL destino, contador

Indicadores: OF DF IF TF SF ZF AF PF CF

x - - - x x ? x x

Desplaza a la izquierda los bits del operando el numero de bits especificado en el segundo operando que debe ser CL si es mayor que 1 los bits desplazados.

SAR (desplazamiento aritmético a la derecha)

Sintaxis: SAR destino, contador



Indicadores: OF DF IF TF SF ZF AF PF CF

X - - - X X ? X X

Desplaza a la derecha los bits del operando destino el numero de bits especificado en el segundo operando. Los bits de la izquierda se rellenan con el bit de signo del primer operando. Si el número de bits a desplazar es 1 se puede especificar directamente, si es mayor se especifica a través de CL.

Ejemplos:

sar ax,cl

sar bp,1

SHR (desplazamiento a la derecha)

Sintaxis: SHR destino, contador

Indicadores: OF DF IF TF SF ZF AF PF CF

X - - - X X ? X X

Desplaza a la derecha los bits del operando destino el numero de los bits especificados en el segundo operando. Los bits de la izquierda se llena con cero. Si el número de bits a desplazar es 1 se puede especificar directamente en el caso en que no ocurra se pone el valor en CL:

Ejemplos:

shr ax,cl

shr cl,1



6.2 EJEMPLO DE PROGRAMACION

TITLE - CONTADOR DE CARACTERES CONTENIDO EN UNA CADENA

DATOS SEGMENT

TITCADENA DB "INGRESE CADENA A EVALUAR : \$"

TITCARACT DB "INGRESE CARACTER A BUSCAR: \$"

COLUMNAB DB " \$"

DATOS ENDS

;

CODIGO SEGMENT

ASSUME CS:CODIGO,DS:DATOS

INICIO:

MOV AX,DATOS

MOV DS,AX

;

MOV DX,OFFSET TITULO1

MOV AH,9h

INT 21H

;

MOV CL,0

OTRO:

MOV DL,CL

MOV AH,2h

INT 21h ;Imprimir caracter ascii

;

LEA DX,COLUMNAB

MOV AH,9h

INT 21h ; Imprimir caracter en blanco

;

ADD CL,1

CMP CL,255

JNE OTRO

FINAL:

MOV AH,4Ch

INT 21H

CODIGO ENDS

END INICIO



CAPITULO VII

Herramientas de programación: Bibliotecas objetos. Macros. Procedimientos. Programación.

7.1 MACROS

Las macros son útiles para los siguientes propósitos:

- Simplificar y reducir la cantidad de codificación repetida
- Reducir errores causados por la codificación repetitiva
- Linealizar un programa en lenguaje ensamblador para hacerlo mas legible

Una definición sencilla de una macro

Las macros deben ser definidas antes que cualquier definición de segmento:

Ejemplo:

```
INITZ  MACRO          ; DEFINE MACRO
      MOV  AX , FLDA
      ADD  AX , FLDB
      MOV  FLDC , AX
      ENDM
```

Uso de parámetros en macros

Para hacer una macro flexible, puede definir nombres en ella como argumentos mudos (ficticios).

Ejemplo:

```
PROMPT MACRO MENSAJE ; ARGUMENTO MUDO
      MOV  AH , 09H
      LEA  DX , MENSAJE
      INT 21H
      ENDM
```



Invocación: PROMPT MOSTRAR

El parámetro en la macroinstrucción corresponde al argumento mudo en la definición original de la macro.

Definición de macro: (argumento)

xyz macro mensaje

Macroinstrucción : (parámetro)

xyz mens2

Se puede definir una macro con cualquier número de argumentos mudos, separados por coma, hasta la columna 120 de una línea. El ensamblador sustituye los parámetros de la macroinstrucción por los argumentos mudos en la definición de la macro, entrada por entrada, de izquierda a derecha.

Si se quiere que aparezca un comentario dentro de una expansión, utilice la directiva de listado .LALL antes de solicitar la macroinstrucción.

.LALL

PROMPT COMENTARIO

Uso de una macro dentro de una definición de macro

Sea dos21 una macro sencilla que carga una función en el registro ah y emite la int 21h.

```
DOS21 MACRO DOSFUNC
    MOV AH , DOSFUNC
    INT 21H
ENDM
```



Usar esta función dos21 para aceptar entrada desde el teclado, codifique:

```
LEA DX , NOMBREPAR  
DOS21 0AH
```

Ahora suponga que tiene otra macro, llamada disp, que carga la función 02h de la int 21h en el registro ah para desplegar un carácter.

```
DISP  MACRO CHAR  
      MOV AH , 02H  
      MOV DL , CHAR  
      INT 21H  
      ENDM
```

Uso de una macro dentro de una definición de macro

Entonces para desplegar un signo de interrogación podemos cambiar disp para aprovechar la macro dos21 dentro de disp.

```
DISP  MACRO CHAR  
      MOV DL , CHAR  
      DOS21 02H  
      ENDM
```

La directiva local

Algunas macros necesitan que se definan elementos de datos y etiquetas de instrucciones dentro de la definición de macro. Si utiliza la macro más de una vez dentro del mismo programa, el ensamblador define los elementos de datos para cada aparición.

Los nombres duplicados harían que el ensamblador genere un mensaje de error. Para asegurar que cada nombre generado es único codifique la directiva local inmediatamente después de la instrucción macro.



Ejemplo:

```
DIVIDE MACRO DIVIDENDO,DIVISOR,COCIENTE
    LOCAL COMP
    LOCAL OUT
    MOV AX , DIVIDENDO
    MOV BX , DIVISOR
    SUB CX,CX
COMP:  CMP AX , BX
    ...
    ...
OUT:   MOV COCIENTE , CX
    ENDM
```

Incluir (include) desde una librería de macros

Definir una macro y usarla una sola vez en un programa no es muy productivo. El enfoque es catalogar las macros en una biblioteca en disco bajo un nombre descriptivo como macro.lib. Usted solo tiene que reunir todas las definiciones de sus macros en un archivo y almacenar el archivo en disco.

Como incluir:

```
IF1
    INCLUDE D:MACRO.LIB
ENDIF
```

La directiva purge

La ejecución de una instrucción include hace que el ensamblador incluya todas las definiciones de macros que están especificadas en la biblioteca. Para eliminar las que no se necesitan usamos la directiva purge.



```
IF1
    INCLUDE D:MACRO.LIB
ENDIF
    PURGE PROMPT,DIVIDE
...
```

Concatenación (&)

El carácter ampersan (&) indica al ensamblador que concatene texto o símbolos. La siguiente macro move proporciona la generación de la instrucción movsb, movsw o movsd.

```
MOVE MACRO TAG
    REP MOVS&TAG
ENDM
```

La directiva repetición

La directiva de repetición REPT, IRP e IRPC hace que el ensamblador repita un bloque de instrucciones terminadas por ENDM. Estas directivas no tienen que estar contenidas en una definición macro, pero si lo están, es necesario un ENDM para finalizar la repetición y un segundo ENDM para terminar la definición macro.

Masm 6.0 introdujo los términos repeat, for y forc para rept, irp e irpc , respectivamente.

Repetición: rept

La directiva rept provoca la repetición de un bloque de instrucciones hasta endm de acuerdo con el número de veces en la expresión de entrada.

```
REPT EXPRESION
```



Ejemplo:

BEEPSPKR MACRO

MOV AH , 02H

MOV DL , 07

REPT 5

INT 21H

ENDM

ENDM

Repetición indefinida: irp

Hace que se repita un bloque de instrucciones hasta ENDM.

El formato general es:

IRP ARG_MUDO , <ARGUMENTOS>

Ejemplo:

IRP N , <3,19,17,25,28>

DB N

GENERA: DB 3

DB 19

DB 17

DB 25

DB 28

Repetición indefinida con carácter: irpc

Hace que se repita un bloque de instrucciones hasta ENDM.

El formato general es:

IRPC ARG_MUDO , CADENA



Ejemplo:

IRPC N , 39758

DW N

GENERA: DW 3
 DW 9
 DW 7
 DW 5
 DW 8

Directivas condicionales

El lenguaje ensamblador permite usar varias directivas condicionales. Usamos if1 anteriormente para incluir una entrada de biblioteca solo durante la pasada 1 de un ensamblado.

Las directivas condicionales son muy útiles dentro de una definición de macro, pero no están limitadas a ese propósito. Cada directiva IF debe tener su correspondiente ENDIF para terminar una condición que se prueba. Un ELSE opcional puede proporcionar un área alterna.

```
IFxx (Condición)
...
ELSE (Opcional)
...
ENDIF (Fin del IF)
```

Directivas condicionales: tipos

IF expresión. Si la expresión que se evalúa es diferente de cero, el ensamblador ensambla las instrucciones dentro del bloque condicional.



IFE expresión. Si la expresión que se evalúa es cero, el ensamblador ensambla las instrucciones dentro del bloque condicional.

IF1 (sin expresión). Si el ensamblador esta procesando la pasada 1, actúa sobre las instrucciones en el bloque condicional.

IF2 (sin expresión). Si el ensamblador esta procesando la pasada 2, actúa sobre las instrucciones en el bloque condicional.

IFDEF símbolo. Si el símbolo esta definido en el programa o declarado como EXTRN, el ensamblador procesa las instrucciones en el bloque condicional.

IFNDEF símbolo. Si el símbolo no esta definido en el programa o declarado como EXTRN, el ensamblador procesa las instrucciones en el bloque condicional.

IFB <argumento>. Si el argumento esta en blanco, el ensamblador procesa las instrucciones en el bloque condicional. Se debe incluir los <>

IFNB <argumento>. Si el argumento no esta en blanco, el ensamblador procesa las instrucciones en el bloque condicional. Se debe incluir los <>.

IFIDN <arg-1>,<arg-2>. Si la cadena del argumento 1 es idéntica a la cadena del argumento 2, el ensamblador procesa las instrucciones en el bloque condicional.

IFDIF <arg-1>,<arg-2>. Si la cadena del argumento 1 es diferente a la cadena del argumento 2, el ensamblador procesa las instrucciones en el bloque condicional.

IF e IFE pueden usar operadores relacionales EQ, NE, LT, LE, GT y GE.



Ejemplo:

```
DOS21      MACRO DOSFUNC , DXADDRES
            MOV AH, DOSFUNC
            IFNB <DXADDRES>
                MOV DX , OFFSET DXADDRES
            ENDIF
            INT 21H
        ENDM
```

La directiva Exitm

Una definición de macro puede contener una directiva condicional que pruebe buscando una condición grave. Si la condición es verdadera, el ensamblador sale desde cualquier expansión posterior de macro. La directiva EXITM sirve para este propósito:

```
IFxx (CONDICION)
... (CONDICION NO VALIDA)
EXITM
...
```

Si el ensamblador encuentra EXITM en una expansión de una macroinstrucción, descontinúa la expansión de la macro y reasume el procesamiento después de ENDM.

7.2 PROCEDIMIENTOS Y SUBPROGRAMAS

Enlace de subprogramas

Hasta el momento hemos visto programas que han consistido en un solo modulo ensamblado y autónomo. Sin embargo, es posible desarrollar un programa que conste de un programa principal enlazado con uno o más subprogramas ensamblados por separado. Las razones para hacer esto podrían ser:

- Enlazar entre lenguajes



- Facilitar el desarrollo de proyectos grandes
- Traslapar partes de un programa durante la ejecución a causa del gran tamaño del programa.

Segmentos

El formato para una directiva segment es:

nom_seg segment [alinear] [combinar] [´clase`]

tipo align (alinear)

El operador alinear le indica al ensamblador que alinee el segmento nombrado al inicio de una frontera particular de almacenamiento.

byte (mas adecuado para procesadores 8088)

word (mas adecuado para procesadores 8086/80286)

dword (mas adecuado para procesador 80386)

para (frontera de párrafo. divisible entre 16. por defecto)

page (frontera de pagina. divisible entre 256)

Tipo combine (combinar)

El operador combinar le indica al ensamblador y al enlazador si combina segmentos o los mantiene separados.

NONE (El segmento esta separado de manera lógica de los otros segmentos. Por defecto).

PUBLIC (Combina los segmentos que están definidos como PUBLIC y tienen el mismo nombre de segmento y clase).

COMMON (Si segmentos comunes tienen el mismo nombre y clase, el enlazador les da la misma dirección base. Durante la ejecución el segundo segmento se traslapa en el primero)



Tipo class (clase)

Ya hemos usado los nombres de clase 'stack', 'data' y 'code'. Se puede asignar el mismo nombre de clase a segmentos relacionados de modo que el ensamblador y el enlazador los agrupen. Esto es, aparecerán como segmentos uno después del otro.

Las dos instrucciones siguientes SEGMENT no relacionadas generan resultados idénticos, es decir, un segmento de código independiente alineado en una frontera de párrafo:

```
CODESEG SEGMENT PARA NONE 'Code'
```

```
CODESEG SEGMENT 'Code'
```

Llamadas intrasegmento

Las instrucciones CALL usadas hasta este momento han sido llamadas intrasegmento; esto es, el procedimiento llamado está en el mismo segmento de código que el procedimiento que llama.

Una llamada CALL es cercana si el procedimiento llamado está definido o si es por omisión NEAR (esto es, dentro de 32k). La operación CALL empuja el registro IP a la pila y reemplaza el IP con el desplazamiento de la dirección destino.

Una llamada intrasegmento puede ser cercana, como se describió, o lejana si AL llamada es a un procedimiento definido como lejano dentro del mismo segmento.

```
CALL proc_cerc    ; Llamada cercana: guarda  
...              ; en la pila el IP. Enlaza a proa_cerc  
...              ;
```



```
proc_cerc PROC NEAR
    ...
    ...
    RET          ;Regreso cercano: saca IP y
proc_cerc ENDP   ;regresa
```

Llamadas intersegmento

Una llamada es clasificada como lejana si el procedimiento llamado esta definido como FAR o como EXTRN, con frecuencia en otro segmento.

La operación CALL primero guarda en la pila el contenido del registro CS e inserta una nueva dirección de segmento en CS, después guarda en la pila el IP e inserta un nuevo desplazamiento de dirección en el IP.

La diferencia entre un CALL cercano y uno lejano es básicamente que un CALL cercano solo reemplaza el desplazamiento IP, mientras que un CALL lejano reemplaza tanto la dirección del segmento CS como del desplazamiento IP.

```
CALL proc_lej    ; Llamada lejana: guarda
...              ; en la pila el CS y el IP. Enlaza a
...              ; proc_lej
proc_lej PROC FAR
    ...
    ...
    RET          ; Regreso lejano: saca IP , CS
proc_lej ENDP    ; y regresa
```

Atributos extrn y public

```
EXTRN SUBPROG:FAR
MAINPROG PROC FAR
    ...
    CALL SUBPROG
    ...
MAINPROG ENDP
```



```
;-----  
PUBLIC SUBPROG  
SUBPROG PROC FAR  
...  
...  
RET  
SUBPROG ENDP
```

Uso de extrn

La directiva extrn indica al ensamblador que el elemento llamado un dato – procedimiento o etiqueta- esta definido en otro ensamblado.

```
EXTRN NOMBRE:TIPO [, ...]
```

La entrada tipo puede ser abs (constante), byte, dword, far, near, word.

byte, word y dword identifican datos a los que hace referencia un modulo, pero otro modulo lo define.

NEAR, FAR identifican a un procedimiento o etiqueta de instrucción a los que hace referencia un modulo pero otro lo define.

Uso de public

La directiva PUBLIC indica al ensamblador y al enlazador que la dirección de un símbolo especificado definido en el ensamblado actual estará disponible para otros módulos.

```
public simbolo [, ...]
```

```
page 60, 132  
title p23main1 - llama al subprograma  
extrn p23sub1:far  
;-----  
stacksg segment para stack 'stack'
```



```
                dw    64 dup(?)
stacksg ends
;-----
datasg segment para 'data'
qty      dw    0140h
price    dw    2500h
datasg  ends
;-----
codesg segment para 'code'
begin    proc far
        assume cs:codesg,ds:datasg,ss:stacksg
        mov  ax , datasg
        mov  ds , ax
        mov  ax , price
        mov  bx , qty
        call p23sub1
        mov  ax , 4c00h
        int  21h
begin    endp
codesg  ends
        end begin

page 60, 132
title p23sub1 - subprograma llamado
;-----
codesg segment para 'code'
p23sub1 proc far
        assume cs:codesg
        public p23sub1
        mul  bx           ;bx = precio  ax = cantidad
        ret              ;dx:ax = producto
p23sub1 endp
codesg  ends
        end p23sub1
```



Datos comunes en subprogramas

Un requerimiento común en programación es procesar en un modulo datos que están definidos en otro modulo.

```
page 60, 132
title p23main4 - llama al subprograma
    extrn p23sub4:far
    public qty,price
;-----
stacksg segment para stack 'stack'
    dw    64 dup(?)
stacksg ends
;-----
datasg  segment para public 'data'
qty     dw    0140h
price   dw    2500h
datasg  ends
;-----
codesg  segment para public 'code'
begin   proc far
    assume cs:codesg,ds:datasg,ss:stacksg
    mov  ax , datasg
    mov  ds , ax
    call p23sub4
    mov  ax , 4c00h
    int  21h
begin   endp
codesg  ends
        end begin
```



```
page 60, 132
title p23sub4 - subprograma llamado
    extrn qty:word,price:word
;-----
codesg segment para public 'code'
p23sub4 proc far
    assume cs:codesg
    public p23sub4
    mov ax , price
    mov bx , qty
    mul bx          ;bx = precio  ax = cantidad
    ret             ;dx:ax = producto
p23sub4 endp
codesg ends
    end p23sub4
```

Definición de datos en ambos programas

Los subprogramas con frecuencia tienen que definir sus propios datos, así como hacer referencia a datos en el programa que los llama.

```
page 60, 132
title p23main5 - llama al subprograma
    extrn p23sub5:far
    public qty
;-----
stacksg segment para stack 'stack'
    dw 64 dup(?)
stacksg ends
;-----
datasg segment para 'data'
    qty dw 0140h
datasg ends
;-----
codesg segment para public 'code'
```



```
begin    proc far
        assume cs:codesg,ds:datasg,ss:stacksg
        mov  ax , datasg
        mov  ds , ax
        call p23sub5
        mov  ax , 4c00h
        int  21h
begin    endp
codesg  ends
        end begin
```

page 60, 132

title p23sub5 - subprograma llamado

extrn qty:word

;-----

datasg segment para 'data'

price dw 2500h

datasg ends

;-----

codesg segment para public 'code'

p23sub5 proc far

assume cs:codesg

public p23sub5

mov bx , qty

push ds

assume ds:datasg

mov ax , datasg

mov ds , ax

mov ax , price

mul bx ;bx = precio ax = cantidad

pop ds

ret ;dx:ax = producto

p23sub5 endp

codesg ends



end p23sub5

Paso de parámetros

Otra forma de hacer que se conozcan los datos por los subprogramados llamados es por medio del paso de parámetros, en el un programa pasa datos físicamente mediante la pila. En este caso asegúrese de que cada PUSH hace referencia a una palabra, ya sea en memoria o en un registro.

Pila de la estructura del programa

La pila de la estructura del programa es la parte de la pila que el programa que llama utiliza para pasar parámetros y que el subprograma llamado utiliza para accederlos.

EJEMPLO :

```
page 60, 132
title p23main6 - llama al subprograma
extrn p23sub6:far
;-----
stacksg segment para stack 'stack'
        dw    64 dup(?)
stacksg ends
;-----
datasg  segment para 'data'
qty     dw    0140h
price   dw    2500h
datasg  ends
;-----
codesg  segment para public 'code'
begin   proc far
        assume cs:codesg,ds:datasg,ss:stacksg
        mov  ax , datasg
        mov  ds , ax
```



```
push price
push qty
call p23sub6
mov ax, 4c00h
int 21h
begin endp
codesg ends
end begin
```

page 60, 132

title p23sub6 - subprograma llamado

;-----

codesg segment para public 'code'

p23sub6 proc far

assume cs:codesg

public p23sub6

push bp

mov bp, sp

mov ax, [bp+8]

mov bx, [bp+6]

mul bx ;bx = precio ax = cantidad

pop bp

ret 4 ;dx:ax = producto

p23sub6 endp

codesg ends

1. Un PUSH carga price en la pila
2. Un PUSH carga qty en la pila
3. CALL guardo en la pila de la estructura el contenido de CS. Como el subprograma es PUBLIC, el enlazador combina los dos segmentos de código y direcciones CS es la misma para ambos.



4. También CALL guarda en la pila de la estructura el contenido del registro IP.

El enlazador (link)

Es el programa que básicamente realiza dos tareas:

- combina varios módulos objetos, realizando las conexiones necesarias entre ellos.
- convierte los módulos objetos en un modulo ejecutable

entrada: modulo.obj[modulo2.obj,...]

librería.lib[libreria2.lib,...]

salida: modulo.exe

modulo.map

comando: link objetos(s), ejecutable, mapa, librería(s)

Los objetos y la librerías se especifican separadas mediante el signo “+”

El gestor de librerías (lib)

Una librería de módulos objetos es una colección de objetos reunidos en un solo fichero. si se indica al enlazador el nombre de una o varias librerías, automáticamente inserta en el modulo ejecutable los módulos objeto que necesita. Con ello se evita tener que especificar todos los módulos necesarios en el comando link para poder generar el modulo ejecutable.

El gestor de librerías (lib) : operaciones

OPERACION	COMANDO
CREAR UNA LIBRERÍA DE OBJETOS	LIB LIBRERÍA;
AÑADIR UN MODULO OBJETO	LIB LIBRERÍA + MODULO;
ELIMINAR UN MODULO OBJETO	LIB LIBRERÍA – MODULO;
SUSTITUIR UN MODULO OBJETO	LIB LIBRERÍA –MODULO + MODULO



EXTRAER UN MODULO	LIB LIBRERÍA *MODULO
ELIMINAR UN MODULO Y EXTRAERLO	LIB LIBRERÍA -*MODULO
LISTAR EL DIRECTORIO EN PANTALLA	LIB LIBRERÍA;
LISTAR EL DIRECTORIO EN FICHERO	LIB LIBRERÍA,FICHERO;
LISTAR EL DIRECTORIO EN IMPRESORA	LIB LIBRERÍA,LPT1;

Lib

lib /out:consola.lib pantalla.obj ent_sal.obj

Genera la librería consola.lib a partir que contiene pantalla y ent_sal

lib /list consola.lib

Lista los módulos contenidos en consola.lib

7.3 INTERFAZ CON LENGUAJES DE ALTO NIVEL

La utilización del ensamblador con lenguajes de alto nivel esta justificada:

- Cuando se desea realizar cierta función no soportada por el lenguaje de alto nivel e incluirla en el repertorio del lenguaje
- Cuando se desea mejorar la velocidad de ejecución de una función implementada en un lenguaje de alto nivel.

En la comunicación lenguaje de alto nivel-ensamblador intervienen los elementos:

- El modulo llamador
- El modulo llamado



- El interfaz de comunicación entre ambos

El interfaz

Es lo que hace posible que un programa escrito en un lenguaje de alto nivel se pueda comunicar con un modulo ensamblador.

Existen dos partes o aspectos del interfaz:

- el interfaz de control
- el interfaz de datos

El interfaz de control

Corresponde a la llamada a la subrutina y al retorno de esta.

a) La llamada

Respecto a la llamada, casi todos los lenguajes de alto nivel disponen de una sentencia CALL, en la que pueden especificarse opcionalmente una serie de parámetros.

La llamada puede ser: NEAR (dentro del segmento de código) o FAR (a otro segmento distinto). En el primer caso, solo se transmite el desplazamiento, y en el segundo, la dirección completa (segmento y desplazamiento)

b) El retorno

Se realiza vía instrucción RET

Puesto que normalmente los parámetros se pasan a través de la pila, hay que especificar un argumento en RET que hace que se restaure el puntero de la pila antes de devolver el control. Si se especifica un valor erróneo, el programa se queda “colgado”. En algunos lenguajes esta restauración del puntero de pila la realiza el propio modulo llamador.



El interfaz de datos

Corresponde a los datos que se manejan y conocen a ambos lados del interfaz. Los datos residen en memoria, no se utilizan registros. Existen muchas maneras de pasar información de un modulo a otro, pero la forma mas segura y flexible es mediante la colocación de los argumentos sobre la pila.

Otros tipos de interfaz de datos son:

- a) Mediante COMMON, que es un área de memoria accesible para ambos módulos (C, PASCAL, ETC)
- b) Mediante variables declaradas PUBLIC, que permite al modulo ensamblador acceder al contenido por el nombre, mecanismo idéntico al existente entre módulos ensamblador.

Los parámetros

El paso de los parámetros sobre la pila puede hacerse de dos maneras:

- directa: se pasa por valor
- indirecta: se pasa la dirección

El tipo de método utilizado depende del lenguaje. Algunos lenguajes permiten pasar valores, casi todos permiten direcciones y otros permiten ambos sistemas.

Este último caso nos permite elegir uno u otro método. ¿Cuál deberíamos utilizar?

- En el caso de que la subrutina cambie el parámetro, estamos obligados a utilizar la dirección (método indirecto).
- Si el parámetro no va a ser modificado, podría pasarse el valor (método directo)



Los registros

En general puede decirse que:

- CS apunta al segmento de código del modulo llamador
- SS y SP corresponden a la pila del modulo llamador
- DS y/o ES apuntan al segmento de datos del modulo llamador

Se deben preservar los registros utilizados salvándolos sobre la pila y recuperarlos antes de devolver el control (con RET) al modulo llamador.

La pila

La subrutina llamada puede usar la pila del modulo llamador, pero no existe manera de saber el espacio disponible. Si lo que se necesita es razonable (menos de 32 palabras), normalmente debería ser suficiente. Si se necesita mas espacio, la subrutina llamada debe reservar su propio espacio en memoria.

Interfaz con c

La comunicación entre un modulo en lenguaje c y un modulo ensamblador tiene las siguientes características:

- los argumentos se pasan por valor o por referencia. el lenguaje c intenta, a menos que se le diga que haga otra cosa, poner el valor sobre la pila, en lugar de su dirección. Para forzar al C a que se pase una dirección, se le pone a la variable el prefijo "&".

Por ejemplo:

subr(&i , j)

El primer parámetro contiene la dirección de la variable "i" y el segundo el valor de la variable "j"

- Las direcciones relativas (desplazamiento) están referidas al registro DS



- Como todas las funciones c retornan un valor entero en el registro AX, el modulo ensamblador debe hacer lo mismo. si la función se calculo correctamente, AX debe ser cero.
- Las variables tipo cadena (string) usadas como argumento pasan la dirección siempre.

Por ejemplo:

subr(s)

- El procedimiento debe declararse PUBLIC
- El procedimiento debe ser de tipo near
- En el modulo ensamblador la instrucción de retorno de ser ret n, siendo **n** igual al numero de bytes que ocupan los parámetros sobre la pila.

Los tipos de datos soportados por el lenguaje C son:

- Enteros
Pueden ser de 1 byte sin signo, de 2 bytes o de 4 bytes.
- Cadenas
Una cadena almacena como asciiz, es decir, una serie de características seguido de un byte con 00h, que indica el final de la cadena.
- Punto flotante
Utilizan los formatos estándar del coprocesador 8087

Lenguaje c y ensamblador

El siguiente programa en lenguaje C llama a las subrutinas CLS y CURSOR escritas en lenguaje ensamblador:



```
/* programa prg.c*/  
main()  
{ cls();  
  cursor(10,20);  
  printf("fin de programa");  
}
```

La primera subrutina CLS no tiene parámetros. La segunda pasa dos parámetros por valor, esto es, en la pila se encontraran los valores pasados 10 y 20. En C se apilan los parámetros empezando por el último. Así, en este caso primero se apila 20 y luego 10.

También se pueden pasar parámetros por referencia, es decir los valores pasados son direcciones en donde se localizan los datos.

El programa en lenguaje ensamblador, al que llamaremos VIDEO.ASM es el siguiente:

```
_text segment byte public `code`      ; nombre _text y clase code  
    assume cs:_text                   ; requeridos por el compilador  
    public cls, cursor                ; simbolos publicos  
cls  proc near                        ; subrutina cercana _cls  
    push bp  
    mov ah , 6  
    mov al , 0  
    mov cx , 0  
    mov dx , 184fh  
    mov bh , 7  
    int 10h  
    pop bp  
    ret  
cls  endp  
cursor proc near                      ; subrutina cercana _cursor  
    push bp                           ; salva bp para acceder a la pila
```



```
        mov bp , sp
        mov dh , [bp+4]          ; numero de fila = parametro 1
        mov dl , [bp+6]          ; numero de columna = parametro 2
        mov bh , 0
        mov ah , 2
        int 10h
        pop bp                    ; desapila valor inicial de bp
        ret 4                     ; retorna y desapila dos parametros
cursor endp
_text ends
end
```

Como obtener prg.exe

- Se compila prg.c y se obtiene prg.obj
- Se compila video.asm y se obtiene video.obj
- Se produce prg.exe con link respondiendo con cs + prg + video
<enter> a la pregunta de modulos de objetos y con prg <enter>
para el modulo ejecutable.

Algunos puntos de interés

Para versiones de C que son sensibles al uso de mayúsculas y minúsculas, el nombre de los módulos en lenguaje ensamblador debe ser escrito exactamente igual que en la referencia del programa en c.

La mayoría de las versiones de C pasan los parámetros a la pila en una secuencia inversa que la de otros lenguajes.

adds(m,n)

La instrucción guarda en la pila **n** y después **m** en ese orden, y luego llama a ADDS.

Al regresar del modulo llamado, el modulo en c suma 4 al SP para desechar los parámetros usados. El procedimiento común en el modulo de lenguaje



ensamblador llamado para acceder los dos parámetros pasados es como sigue:

```
PUSH SP
MOV BP , SP
MOV DH , [BP+4]
MOV DL , [BP+6]
...
...
POP BP
RET
```

Algunas versiones de c necesitan que un modulo en ensamblador que cambia los registros DI y SI los guarde en la pila al entrar y los saque al salir del subprograma en ensamblador.

El modulo en ensamblador debe regresar los valores, si se necesitan, como una palabra en el AX o dos palabras en el par dx:ax.

En algunas versiones de c, un programa en ensamblador que pone en uno la bandera DF debe ponerse en cero (CLD) antes de regresar.

Enlace de “c” a ensamblador

```
#include <stdio.h>
int main(void)
{ int temp_row , temp_col;
  printf("ingrese fila cursor: ");
  scanf("%d",&temp_row);
  printf("ingrese columna cursor: ");
  scanf("%d",&temp_col);
  set_curs(temp_row,temp_col);
  printf("nueva localizacion cursor:");
}
```



```
_data segment word `data`
    row equ [bp+4]          ; parametros
    col equ [bp + 6]        ; argumentos
_data ends

_text segment byte public `code`
dgroup group _data
    assume cs:_text , ds:dgroup , ss:dgroup
    public _set_curs

_set_curs proc near
    push bp                ; registro bp del que llama
    mov bp , sp            ; apunta a los parametros
    mov ah , 02h           ; peticion para colocar el cursor
    mov bx , 0             ; pagina de video
    mov dh , row           ; renglon de bp+4
    mov dl , col           ; renglon de bp+6
    int 10h                ; llama al bios
    pop bp                 ; restaura bp
    ret                   ; regresa a donde fue llamado
_set_curs endp
_text ends
end
```



CAPITULO VIII

Interrupción: Introducción. Servicios. Interrupciones BIOS y DOS.

8.1 INTRODUCCION.

Una interrupción es un suceso que hace que el microprocesador pare la ejecución del programa en curso para atender otra actividad requerida.

Existen 2 tipos de interrupciones:

Externas: Provocadas por los periféricos del ordenador

Internas: Generadas por el propio microprocesador cuando ocurren ciertas condiciones especiales.

Los tipos de interrupciones internas son cinco:

0 – Error en división

1 – Paso a paso (modo depuración de programas)

2 – No es enmascarable (no se puede desactivar por programa).

3 – Sirve para indicar puntos donde debe pararse el programa (modo depuración de programas)

4 – Desbordamiento (overflow). Se dispara por la instrucción INTO (interrupción si overflow)

Existen en total 256 interrupciones (códigos 0 a 255). Mediante este código se apunta a una tabla de vectores de interrupción. Cada vector de interrupción contiene la dirección de la rutina de tratamiento de la interrupción (rutina de servicio de la interrupción).

Algunas de las interrupciones se reservan para el BIOS (Basic Input Output System) y otras para el sistema operativo (DOS).



8.2 Interrupciones BIOS.

Corresponden a las interrupciones 00h a 1Fh. Se dividen en cinco grupos:

Grupo	Números	Tipo
1	00h – 07h	Interrupciones internas del microprocesador 8088/8086
2	08h – 0Fh	Interrupciones del controlador de interrupciones 8259
3	10h – 1Ah	Puntos de entrada del BIOS
4	1Bh – 1Ch	Rutinas del usuario
5	1Dh – 1Fh	Parámetros del BIOS

Las interrupciones internas del microprocesador 8088/8086 se activan cuando ocurre una condición excepcional (por ejemplo, una división por cero) durante el proceso de las instrucciones.

El 8259 es el controlador de interrupciones. Acepta como entradas las señales de petición de interrupciones de diferentes dispositivos del sistema. Tras recibir la petición, este controlador pasa la señal junto con un código identificador al procesador principal (8088/8086).

Las rutinas del usuario son llamadas por otras rutinas de interrupción.

Los parámetros del BIOS no corresponden realmente con rutinas de servicio, sino que son direcciones que apuntan a tablas de datos para las interrupciones 10h y 13h.

Interrupción 00h: División por cero

Es una interrupción interna que se activa automáticamente cada vez que el cociente, en una instrucción de dividir (DIV o IDIV), es demasiado grande para poder ser almacenado en un registro (AL o AX). Esta interrupción hace que se emita el mensaje “Divide Overflow” y se devuelve el control al DOS.

Interrupción 01h: Ejecución paso a paso

Esta interrupción permite ejecutar un programa instrucción por instrucción. se utiliza para depurar programas con el programa DEBUG del DOS. El DOS



inicializa el vector de interrupción con una dirección que contiene una instrucción IRET, es decir, que no hace nada.

Interrupción 02h: No enmascarable

Las interrupciones pueden ser activadas (permitidas) o desactivadas (inhibidas):

Para activarlas (Bandera IF=1): instrucción STI

Para desactivarlas (bandera IF=0): instrucción CLI

Todas las interrupciones, excepto esta, pueden ser inhibidas. Genera el mensaje "Parity Check 1" o "Parity Check 2", inhibe las interrupciones (con CLI) y para el microprocesador (con HLT).

Interrupción 03h: BreakPoint

Esta interrupción permite que se ejecute un programa hasta que se encuentre un indicador de parar la ejecución. Se utiliza en el programa DEBUG (comando G).

Igual que en la interrupción 1, el DOS inicializa el vector correspondiente con una dirección que contiene la instrucción IRET.

Interrupción 04h: Overflow

Esta interrupción se activa mediante la instrucción INTO (interrupción si overflow).

Igual que en las interrupciones 1 y 3, el DOS inicializa el vector correspondiente con una dirección que contiene las instrucciones IRET.

Interrupción 05h: Imprimir pantalla (hardcopy)

Esta interrupción salva la posición actual del cursor, imprime la información alfanumérica de la pantalla (a la vez que va desplazando el cursor) y restaura el cursor.

Esta rutina se ejecuta con las interrupciones permitidas.



Esta interrupción hace lo mismo que la tecla PrtSc.

Si se pulsa la tecla PrtSc mientras se esta imprimiendo, se ignora.

La dirección 50h:0 contiene el estado:

Valor	Significado
0	No se ha invocado la función de imprimir pantalla u operación sin error
1	Impresión de la pantalla en curso
FFh	Error al imprimir

Interrupción 08h: Temporizador (8353 timer)

El temporizador del sistema emite una interrupción cada 1,193,182 / 65536 nanosegs., es decir, 18.2 veces por segundo aprox. Cada interrupción se llama un "tic", que produce una actualización del contador de tiempos. Este contador se encuentra en una doble palabra en el área de datos del BIOS:

40h:6Ch = 46Ch = palabra inferior

40h:6Eh = 46Eh = palabra superior

La interrupción 08h (es decir cada tic) invoca una interrupción 1Ch. La interrupción 1Ch apunta a una instrucción IRET. Cambiando el vector de interrupción, se puede apuntar a una rutina del usuario.

Interrupción 09h: Teclado

El BIOS activa esta interrupción cada vez que se pulsa una tecla. Debe considerarse a todos los efectos como una interrupción del sistema.

Interrupción 10h: Entrada/Salida pantalla

Según el valor de AH, existen 16 diferentes funciones. Estas funciones son: de establecimiento/interrogación del modo de funcionamiento de la pantalla, de transferencia de caracteres y graficas.



Interrupción 1Bh: Break de teclado

Se invoca cuando se pulsa Ctrl-Break. Inicialmente apunta a IRET, pero el DOS actualiza el vector para que apunte a la misma rutina que la interrupción 23h del DOS.

8.3 Interrupciones DOS.

Corresponden a las interrupciones 20h a 3Fh. Son las siguientes:

Nro	Descripción
20h	Terminar programa
21h	Petición de función
22h	Dirección de terminación. Especifica la dirección a donde se bifurcara cuando se termine el programa. Esta dirección se copia en PSP
23h	Dirección de salida por Ctrl-Break
24h	Manejador de error crítico
25h	Lectura de disco absoluta
26h	Grabación en disco absoluta
27h	Terminar, pero quedar residente
28h	No se usa (reservada)
...	
3Fh	No se usa (reservada)

Las funciones DOS

Corresponden a la interrupción 21h. Cada función se distingue por un valor de AH. Las rutinas que se ejecutan sirven para lectura de teclado, salida por pantalla e impresora, gestión de ficheros, gestión de la memoria, etc.).



Estas funciones se agrupan en:

Valores de AH (hex)	Tipo de función
0 a 12	Entrada / Salida de carácter
13 a 24	Gestión de ficheros
25 a 26	Funciones no asociadas a dispositivos
27 a 29	Gestión de ficheros
2A a 2E	Funciones no asociadas a dispositivos
2F a 38	Otras funciones
39 a 3B	Gestión de directorios
3C a 46	Gestión de ficheros

Las funciones de la 2Fh en adelante corresponden a las nuevas funciones añadidas a partir de DOS 2.0



CAPITULO IX

Operadores y Pseudo operadores: Ejemplos.

9.1 OPERADORES

Un operador es un modificador que se usa en el campo de operandos de una sentencia ensamblador. Se puede utilizar varios operadores y comunicarse entre si en una misma sentencia.

Hay 5 tipos de operadores:

- Aritméticos
- Lógicos
- Relacionales
- De retorno de valores
- De atributos

9.2 PSEUDO-OPERADORES

Las pseudo-operaciones son una parte importante de la programación en lenguaje ensamblador. Una pseudo-op es un directivo al ensamblador para que realice una operación. Entonces una pseudo-op puede ser considerada como una orden del ensamblador u operando más que como una instrucción de ensamblamiento que será traducida a código maquina. Las pseudo-op no producen código maquina equivalente.



Relacion de pseudo-operadores

&	DBIT	ENDP	IRPC
=	DD	ENDS	LABEL
!	DP	EQU	.LALL
%	DQ	EVEN	.SALL
%OUT	DT	EXITM	.XALL
::	DW	EXTRN	.LFCOND
ASSUME	ELSE	GROUP	.LIST
COMMENT	END	INCLUDE	.XLIST
.CREF	ENDIF	IF,IFE,IF2,...	LOCAL
DB	ENDM	IRP	MACRO
.MSFLOAT	NAME	ORG	PAGE
PROC	PUBLIC	PURGE	.SFCOND
STRUC	SUBTTL	.TFCOND	TITLE



ADMINISTRACION DE ARCHIVOS, PERIFERICOS Y MEMORIA



CAPITULO X

Administración: Introducción I/O de disco. Administración secuencial de archivos en disco.

10.1 INTRODUCCION I/O DE DISCO.

En algún momento, un programador serio tiene que estar familiarizado con los detalles técnicos de la organización en disco, en particular para el desarrollo de programas de utilería que examinan el contenido de los discos flexibles y duros.

Características de los discos

Para procesar registros en disco, es útil estar familiarizado con los términos y características de su organización. Un disco flexible tiene dos lados, mientras que un disco duro contiene varios discos de dos lados.

Pistas y sectores

Cada lado de un disco flexible o de un disco duro contiene varias pistas concéntricas, numeradas iniciando con 00, la pista más externa. Cada pista esta formateada en sectores de 512 bytes en donde se almacena la información.

Cilindros

El cilindro es el conjunto vertical de todas las pistas con el mismo número en cada superficie de un disco flexible o de un disco duro. por tanto el cilindro 0 es el conjunto de pistas numeradas con 0, el cilindro 1 es el conjunto de pistas con numero 1, y así sucesivamente.

Una referencia a las caras (cabezas) de un disco, pistas y sectores es por medio de un número. Los números de lado y pista empiezan con 0, pero los sectores pueden ser numerados de una de dos formas:



Dirección pista cilindro: los números de sector en cada pista inician en 1, así que el primer sector en el disco tiene la dirección cilindro 0, pista 0, sector 1.

Número de sector relativo: los sectores pueden ser numerados de manera relativa al inicio del disco, de modo que el primer sector en el disco que esta sobre el cilindro 0, pista 0 tiene la dirección de sector relativo 0.

Controlador de disco

El controlador de un disco esta localizado entre el procesador y la unidad de disco y maneja toda la comunicación entre ellos. El controlador acepta información del procesador y la convierte a una forma que pueda usar el dispositivo.

Por ejemplo:

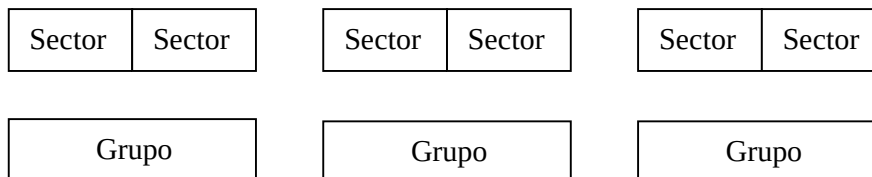
El procesador puede enviar una petición de datos de un cilindro, cabeza de disco y sector específico. El papel del controlador es proporcionar los comandos apropiados para mover el brazo de acceso al cilindro necesario, seleccionar la cabeza de lectura/escritura y aceptar la información del sector cuando esta llegue a la cabeza de lectura/escritura.

Grupos

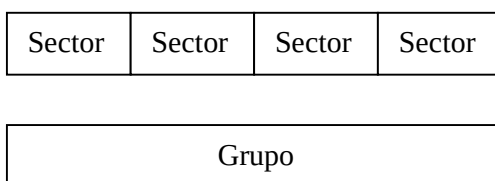
Un grupo es un conjunto de sectores que el dos trata como una unidad de espacio de almacenamiento. El tamaño de grupo es siempre una potencia de 2, como 1, 2, 4 u 8 sectores. Por lo común un disco duro tiene cuatro sectores por grupo. En un dispositivo de disco que utiliza un sector por grupo, sector y grupo son los mismos. Un archivo empieza en una frontera de grupo y necesita un mínimo de un grupo aunque solo ocupe uno de los cuatro sectores. Un grupo se puede traslapar de una pista a otra.



Un disco con dos sectores por grupo se vería así:



Un disco con cuatro sectores por grupo se vería así:



Un archivo de 100 bytes almacenado en un disco con cuatro sectores por grupo utiliza $4 \times 512 = 2,048$ bytes de almacenamiento, aunque solo un sector contendría información.

El DOS almacena los grupos para archivo en orden ascendente, aunque un archivo puede estar fragmentado de manera que resida, por ejemplo, en los grupos 8, 9, 10, 14, 17 y 18.

En los discos duros, las capacidades varían considerablemente por dispositivo y por partición. Las operaciones útiles para la determinación del número de cilindros, sectores por pista o cabezas de lectura/escritura incluyen las funciones 1fh y 440dh con código secundario 60h ambas de la int 21h.

10.2 AREA DE SISTEMA Y DE DATOS

Para dar cuenta de la información almacenada en disco, el DOS reserva ciertos sectores para sus propios objetivos. La organización de disco flexibles y de discos duros varía de acuerdo con su capacidad. Un disco duro y algunos



discos flexibles están formateados para autoarranque, esto es, son capaces de procesar el inicio cuando se enciende o reinicia la computadora.

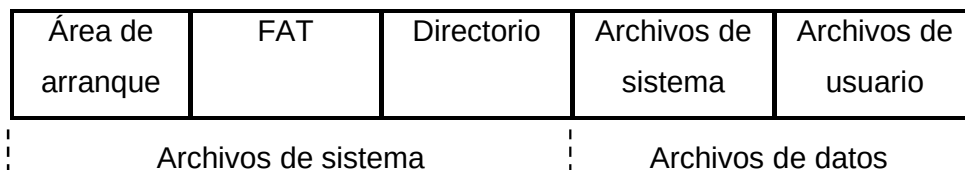
Área de sistema:

El área de sistema es la primer área del disco, en la(s) pista(s) mas externa(s) iniciando con el lado 0, pista 0, sector 1. La información que el dos almacena y mantiene en su área de sistema es utilizada para determinar, por ejemplo, la ubicación de cada archivo que será accesado.

Los 3 componentes del área de sistema son:

- Registro de arranque
- Tabla de asignación de archivos (FAT)
- Directorio

El área de sistema y el de datos están acomodados así:



Sectores asignados para el área de sistemas

La lista siguiente da la organización de varios tipos de dispositivos y muestra los números de los sectores de inicio y final para el registro de arranque, la FAT y el directorio (sectores en términos de numero de sector relativo, en donde el sector relativo 0 es cilindro 0, pista 0, sector 1, el primer sector del dispositivo).

En los discos duros, las ubicaciones del registro de arranque y de la FAT por lo común son las mismas que un disco flexible, el tamaño de la FAT y la ubicación del directorio varían por dispositivo.



Dispositivo	Arranque	FAT	Directorio	Sector/Grupo
5.25" 360kb	0	1-4	5-11	2
5.25" 1.2mb	0	1-14	15-28	1
3.5" 720kb	0	1-6	7-13	2
3.5" 1.44mb	0	1-18	19-32	1

Área de datos.

El área de datos en un disco o disco flexible de arranque empieza con los archivos del sistema DOS llamados IOSYS.SYS y MSDOS.COM (para MS-DOS) o IBMBIO.COM e IBMDOS.COM (para ibm pc dos).

Cuando utiliza FORMAT /S para formatear un disco, el DOS copia sus archivos de sistema a los primeros sectores del área de datos.

Los archivos del usuario siguen de manera inmediata a los archivos de sistema o, si no existen archivos de sistema, empiezan al inicio del área de datos.

Un disco flexible de dos lados formateado con nueve sectores por pista contiene la información siguiente:

Lado 0, pista 0, sector 1	Registro de arranque
Lado 0, pista 0, sector 2-3	Tabla de asignación de archivos (FAT)
Lado 0, pista 0, sector 4-7	Directorio
Lado 1, pista 0, sector 1-3	Directorio
Lado 1, pista 0, sector 4-sig	Área de datos

Los registros para los archivos de datos empiezan en el lado 1, pista 0, sector 3 al 9. El sistema almacena los registros siguientes en el lado 0, pista 1, después en el lado 1, pista 1, después lado 0, pista 2, y así sucesivamente.



Registro de arranque.

El registro de arranque contiene las instrucciones que cargan los archivos del sistema IOSYS.SYS, MSDOS.COM y COMMAND.COM desde el disco a la memoria. Todos los discos formateados tienen este registro aun cuando no estén almacenados en ellos los archivos del sistema.

El registro de arranque contiene la información siguiente, en orden de dirección de desplazamiento.

00H	SALTO CERCANO O LEJANO DE LA RUTINA DE ARRANQUE EN EL DESPLAZAMIENTO 1EH o 3EH EN EL REGISTRO ARRANQUE
03H	NOMBRE DEL FABRICANTE Y NUMERO DE LA VERSION DEL DOS CUANDO FUE CREADO EL ARRANQUE
0BH	BYTES POR SECTOR, POR LO COMUN 200H (512)
0DH	SECTORES POR GRUPO (1, 2, 4 U 8)
0EH	SECTORES RESERVADOS
10H	NUMERO DE COPIAS DE LA FAT (1 o 2)
11H	NUMERO DE ENTRADAS EN EL DIRECTORIO RAIZ
13H	SI EL VOLUMEN ES MENOR QUE 32 MB
15H	BYTE DE DESCRIPCION DE ,
16H	NUMERO DE SECTORES POR LA FAT
18H	NUMERO DE SECTORES POR PISTA
1AH	NUMERO DE CABEZAS DE LECTURA/ESCRITURA
1CH	NUMERO DE SECTORES OCULTOS
1EH	CARGADOR DE LA RUTINA DE ARRANQUE
20H	SI EL VOLUMEN ES MENOR QUE 32H, NUMERO TOTAL DE SECTORES
24H	NUMERO DE UNIDAD FISICA (PARA DISCOS FLEXIBLES = 0, PARA DISCO DURO UNIDAD C =80H
25H	RESERVADO PARA EL DOS
26H	SECTOR AMPLIADO DE ARRANQUE
27H	IDENTIFICACION DEL SISTEMA ,
2BH	ETIQUETA DEL VOLUMEN



36H	RESERVADO PARA EL DOS
3EH-1FFH	A PARTIR DE DOS 4.0, EL CARGADOR DE ARRANQUE EMPIEZA AQUÍ.

El dos 4.0 amplía el registro de arranque con campos adicionales desde 20h hasta 1ffh, por tanto, el registro original de arranque es de 20 (32) bytes, mientras que la versión ampliada es de 200h

Directorio

Todos los archivos en un disco empiezan en una frontera de grupo que es el primer sector del grupo. Para cada archivo, el DOS crea una entrada de directorio de 32 (20h) bytes que describe el nombre del archivo, la fecha en que creado, su tamaño y la ubicación de su grupo inicial.

Las entradas del directorio tienen el formato siguiente:

Byte	Propósito
00h – 07h	Nombre del archivo, como es definido en el programa que crea el archivo. El primer byte del nombre del archivo también puede indicar el estado del archivo. 00h el archivo nunca ha sido utilizado 05h actualmente el primer carácter del nombre del archivo es e5h 2eh la entrada es para un subdirectorio e5h el archivo ha sido borrado
08h – 0ah	Extensión del nombre de archivo 0bh atributo del archivo, define el tipo de archivo 00h archivo normal 01h archivo que solo puede ser leído (solo lectura) 02h archivo oculto, en una búsqueda de directorio no se muestra



08h – 0ah	04h archivo del sistema dos, no desplegado por una búsqueda de directorio 08h etiqueta de volumen 10h subdirectorio 20h archivo resguardado, que indica si el archivo fue reescrito en su ultima actualización
0ch – 15h	reservado para el dos
16h – 17h	hora del día que el archivo fue creado o actualizado por ultima vez almacenada en 16 bits en formato binario (hhhhhhmmmmmmsssss)
18h – 19h	Fecha de creación o ultima actualización, almacenada en 16 bits (yyyyyyymmmmmddddd). el año puede ser 0-119 (0=1980), mes puede ser 1-12 y el día puede ser 01-31
1ah – 1fh	Grupo inicial del archivo. El número es relativo a los dos sectores del directorio.
1ch – 1fh	Tamaño del archivo en bytes. Cuando crea un archivo, el dos calcula y almacena su tamaño en este campo.

Tabla de asignación de archivos

El objetivo de la FAT es asignar espacio en disco para archivos. La FAT contiene entrada para cada grupo en el disco. Cuando crea un archivo o revisa un archivo existente. El DOS revisa las entradas asociadas a la FAT de acuerdo con la ubicación del archivo en el disco. La FAT empieza en el sector 2, inmediatamente después del registro de arranque. En un disco en donde un grupo consta de 4 sectores, el mismo número de entradas de la FAT puede hacer referencia a cuatro veces la información que los discos en donde un grupo consiste en un sector.



En consecuencia, el uso de grupo con sectores múltiples reduce el número de entradas en la FAT y permite al DOS direccionar un espacio de almacenamiento mayor en disco.



CAPITULO XI

Programación: Manejo de ficheros. Ejemplos.

11.1 MANEJO DE ARCHIVOS

Un archivo es una colección de datos (bloques de octetos) que con un nombre se escriben o almacenan en algún dispositivo magnético. Estos datos pueden ser después leídos o recuperados del archivo.

El sistema operativo dos provee varias formas para trabajar con archivos usando el lenguaje ensamblador. Una de ellas consiste en asignarle a un archivo un número de 16 bits al cual denominaremos descriptor del archivo (FILE HANDLE).

Por medio del descriptor se realizan las operaciones básicas de lectura, escritura y movimiento del archivo.

Funciones

Las funciones del dos para manejar archivos son:

Creación	: 3ch
Apertura	: 3dh
Cierre	: 3eh
Lectura	: 3fh
Escritura	: 40h
Borrar archivo	: 41h
Movimiento del puntero de archivo	: 42h

Todas se ejecutan con: Int 21h

Estas instrucciones utilizan el indicador de acarreo CF para determinar si se han ejecutado correctamente (sin acarreo)



Para desactivar o borrar el acarreo se puede utilizar, por ejemplo:

clc ; (clear carry) borra acarreo
sub al,al ; hace al=0 y borra acarreo

Se pueden emplear las instrucciones de salto corto según el acarreo:

jc destino ; (jump if carry) saltar si hay acarreo
jnc destino ; (jump if not carry) saltar si no hay acarreo

NOTA: el valor del descriptor de un archivo que se crea o abre es asignado (elegido) por el sistema operativo DOS.

Creación de archivos: 3ch

Se debe tener:

- En AH el código 3ch
- En DS:DX la dirección del comienzo del nombre del archivo a crearse (este nombre debe terminar con 0)
- En CX el valor 0 (modo normal de creación)

Y se ejecuta con Int 21h.

Resultado:

- Si hay acarreo no se crea el archivo y AX resulta con el código de error (Ej. disco lleno)
- Si no hay acarreo, se crea el archivo y AX resulta con el descriptor asignado

Apertura de archivos: 3dh

Se debe tener:

- En AH el código 3dh
- En DS:DX la dirección de inicio del nombre del archivo a abrirse (el nombre debe terminar con 0)



- En AL el código de acceso: 0 = solo lectura
1 = solo escritura
2 = lectura/escritura

Y se ejecuta con Int 21h.

Resultado:

- Si hay acarreo no se puede abrir y AX contiene el código de error.
- Si no hay acarreo, se abre el archivo y AX resulta con el descriptor asignado.

Cierre de archivos: 3eh

Se debe tener:

- En AH el código 3eh
- En BX el descriptor del archivo a cerrar

Y se ejecuta con int 21h.

Resultado:

- Si hay acarreo no se puede cerrar el archivo y AX contiene el código de error
- Si no hay acarreo, se cierra el archivo

Lectura de archivos: 3fh

Se debe tener:

- En AH el código 3fh
- En BX el descriptor del archivo a leer
- En CX el numero de octetos a ser leídos
- En DS:DX la dirección del área de recepción de datos (buffer)



y se ejecuta con Int 21h.

Resultado:

- Si hay acarreo, error de lectura con código de error en AX
- Si no hay acarreo, AX contiene el numero de octetos leídos, siendo este cero si se esta en el final del archivo

Escritura de archivos: 40h

Se debe tener:

- En AH el código 40h
- En BX el descriptor del archivo a leer
- En CX el numero de octetos a ser escritos
- En DS:DX la dirección del área de datos a copiar (buffer)

Y se ejecuta con int 21h.

Resultado:

- Si hay acarreo, error de escritura con código de error en AX
- Si no hay acarreo, la operación de escritura ha sido correcta

Borrar archivos: 41h

Se debe tener:

- En AH el código 41h
- En DS:DX la dirección del nombre de archivo (termina con 0)

Y se ejecuta con int 21h.

Resultado:

- Si hay acarreo, error de borrado con código de error en AX
- Si no hay acarreo, la operación de borrado ha sido correcta



Movimiento del puntero de un archivo: 42h

Dentro de un archivo es posible desplazarse a una dirección específica y luego ejecutar operaciones de lectura o de escritura a partir de la dirección siguiente.

Utilizando esta función, el archivo puede ser tratado como uno de acceso aleatorio o directo. Por ejemplo, si se trabaja con un archivo formado por registros de 150 octetos de longitud, para acceder al cuarto registro será necesario localizar el comienzo del cuarto bloque de 150 octetos, este es dado por el número 3×150 , pues los octetos del archivo se numeran empezando por 0.

NOTA:

- Las clases de operaciones a realizar en un archivo dependen del modo de acceso que se haya escogido en el momento de crearlo o de abrirlo.
- Igual que en el direccionamiento de memoria, las direcciones en un archivo emplean un par de palabras de 16 bits $i : j$.

Se debe tener:

- En AH el código 42h
- En BX el descriptor del archivo a leer
- En el par CX:DX el desplazamiento del apuntador, esto es, el número de direcciones que ha de moverse con respecto a un punto de referencia.
- en al el código del punto de referencia:
 - 0 = para el comienzo del archivo
 - 1 = para la posición actual
 - 2 = para el final del archivo

y se ejecuta con int 21h.



Resultado:

- Si hay acarreo, operación incorrecta
- Si no hay acarreo, el par DX:AX contiene la dirección actual del apuntador que debe ser igual a:
= Dirección del puntero de referencia + valor de CX:DX

11.2 EJEMPLO

PAGE 40, 132

TITLE ALEATORI - PROCESA EL ARCHIVO AZAR EN FORMA ALEATORIO

COMMENT *

ABRE EL ARCHIVO EN MODO LECTURA/ESCRITURA, SI EXISTE, O CREARLO
INICIANDO TODOS SUS REGISTROS CON CARACTERES DE ".".

SOLICITA EL INGRESO DE UN NUMERO DE REGISTRO ENTRE 1 Y 50

SI EL REGISTRO EXISTE, SE MUESTRA SU CONTENIDO Y SE ESPERA EL
INGRESO DE LA NUEVA INFORMACION. PUEDE PULSAR ENTER PARA NO
MODIFICAR EL REGISTRO

*

;-----

PROGRAMA SEGMENT

ASSUME CS:PROGRAMA, DS:PROGRAMA

ORG 100H

COMIENZO:

MOV AX , CS

MOV DS , AX

CALL BORRAR

CALL ABRIR

JNC ABIERTO

CALL CREAR

JC FIN

ABIERTO:

CALL PROCESAR

CALL CERRAR

FIN:

MOV AH , 4CH

INT 21H

;-----

;

BORRAR PROC NEAR



```
MOV AH , 6H                ; SUBROUTINA PARA BORRAR PANTALLA
MOV AL , 0                  ; MOVER ARRIBA (SCROLL UP)
MOV CX , 0                  ; TODA LA PANTALLA
MOV DX , 184FH              ; DESDE FILA = CH = 0 , COLUMNA = CL = 0
MOV BH , 07H               ; HASTA FILA = DH = 24 , COLUMNA = DL = 79
INT 10H                    ; VIDEO NORMAL = FONDO NEGRO Y
                           ; RELIEVE = BLANCO

MOV AH , 2H                ; Y UBICAR CURSOR EN
MOV DX , 0                  ; FILA = DH = 0 , COLUMNA = DL = 0
MOV BH , 0                  ; DE LA PAGINA ACTUAL
INT 10H
RET

BORRAR ENDP

;
ABRIR PROC NEAR
MOV DX , OFFSET NOMBRE     ; SUBROUTINA PARA ABRIR ARCHIVO
MOV AH , 3DH               ; (SI EXISTE) EN EL MODO AL = 2
MOV AL , 2H                ; PARA LEER O ESCRIBIR
INT 21H                    ; VARIABLE ARCH CONTIENE DESCRIPTOR
JC FIN_ABRIR
MOV ARCH , AX

FIN_ABRIR:
RET

ABRIR ENDP

;
CREAR PROC NEAR
MOV DX , OFFSET NOMBRE    ; SUBROUTINA PARA CREAR ARCHIVO
MOV AH , 3CH               ; E INICIAR CON BLANCOS 50
MOV CX , 0                 ; REGISTROS CON 30 OCTETOS C/U
INT 21H
JC FIN_CREAR
MOV ARCH , AX
MOV CX , LONGITUD_REG
MOV DX , OFFSET CADENA     ; (CADENA HA SIDO INICIADA
ADD DX , 2H                ; CON BLANCOS, VER AREA DE DATOS)
MOV BX , ARCH              ; ESCRIBIR 50 VECES SECUENCIALMENTE)
MOV CODIGO , MINIMO        ; CONTENIDO DE CADENA EN ARCHIVO SIGUIENTE:
CMP CODIGO , MAXIMO
JA FIN_CREAR
MOV AH , 40H
```



```
                INT 21h
                INC CODIGO
                JMP SIGUIENTE

FIN_CREAR:
                RET

CREAR  ENDP
;
CERRAR  PROC NEAR
                MOV BX , ARCH                ; RUTINA PARA CERRAR ARCHIVO
                MOV AH , 3EH
                INT 21H
                RET

CERRAR  ENDP
;
PROCESAR PROC NEAR
    CALL LEER_COD                ; SUBROUTINA DE PROCESAMIENTO
                CMP CODIGO , INVALIDO      ; DE REGISTROS. INVOCA A
                JNE CONTINUAR              ; VARIAS SUBROUTINAS: LEER_COD
                JMP FIN_PROCESAR           ; MOSTRAR_REG Y ACTUALIZAR

CONTINUAR:
                CALL MOSTRAR_REG            ; PROCESAR SE EJECUTA MIENTRAS
                CALL ACTUALIZAR             ; CODIGO SEA VALIDO
                JMP PROCESAR

FIN_PROCESAR:
                RET

PROCESAR ENDP
;
LEER_COD PROC NEAR
                MOV DX , OFFSET MENSAJE1    ; SUBROUTINA DE LECTURA DE CADENA
                MOV AH , 9                  ; DE CODIGO Y CONVERSION A BINARIO
                INT 21H
                MOV CODIGO , 0
                MOV DX , OFFSET CADENA_COD
                MOV SI , DX
                MOV AH , 10
                INT 21H
                MOV DX , OFFSET CLINEA
                MOV AH , 9
                INT 21H
                MOV CH , 0
```



```
MOV CL , BYTE PTR [SI+1]
MOV BX , 0

SIG_CAR:
    CMP BX , CX
    JAE PROBAR
    XCHG AX , CODIGO
    PUSH CX
    MOV CX , 10
    MUL CL
    POP CX
    XCHG AX , CODIGO
    MOV AL , BYTE PTR [SI+BX+2]
    SUB AL , "0"
    SUB AH , AH
    ADD CODIGO , AX
    INC BX
    JMP SIG_CAR

PROBAR:
    CMP CODIGO , MINIMO
    JB VALOR_INVALIDO
    CMP CODIGO , MAXIMO

VALOR_INVALIDO:
    MOV CODIGO , INVALIDO

FIN_LEER_COD:
    RET

LEER_COD ENDP
;

MOSTRAR_REG PROC NEAR
    CALL MOVER_APUNT                ; SUBROUTINA PARA MOSTRAR
    MOV DX , OFFSET CADENA          ; REGISTRO DE CODIGO LEIDO
    ADD DX , 2                      ; LLAMA A SUBROUTINA
    MOV CX , LONGITUD_REG           ; MOVER APUNT PARA UBICAR
    MOV AH , 3FH                   ; EL APUNTADOR DEL ARCHIVO
    INT 21H                        ; AL COMIENZO DEL REGISTRO
    MOV DX , OFFSET CADENA          ; DEL CODIGO CORRESPONDIENTE
    ADD DX , 2
    MOV AH , 9
    INT 21H
    RET
MOSTRAR_REG ENDP
```



```
;
ACTUALIZAR PROC NEAR
    MOV DX , OFFSET MENSAJE2
    MOV AH , 9
    INT 21H
    MOV DX , OFFSET CADENA
    MOV SI , DX
    MOV AH , 10
    INT 21H
    MOV DX , OFFSET CLINEA
    MOV AH , 9
    INT 21H
    MOV BL , BYTE PTR [SI+1]
    MOV BH , 0
    CMP BX , 0
    JE FIN_ACTUAL
```

```
BLANCO:
    CMP BX , LONGITUD_REG
    JA  ESCRIBIR
    MOV BYTE PTR [SI+BX+2], " "
    INC BX
    JMP BLANCO
```

```
ESCRIBIR:
    CALL MOVER_APUNT
    MOV DX , OFFSET CADENA
    ADD DX , 2
    MOV CX , LONGITUD_REG
    MOV AH , 40H
    INT 21H
```

```
FIN_ACTUAL:
    RET
```

```
ACTUALIZAR ENDP
```

```
;
MOVER_APUNT PROC NEAR
    MOV BX , ARCH
    MOV AX , CODIGO
    DEC AX
    MOV CX , LONGITUD_REG
    MUL CL
    MOV CX , 0
```



```
XCHG AX , DX
MOV AH , 42H
MOV AL , 0
INT 21H
RET
MOVER_APUNT ENDP
;
MINIMO EQU 1 ; DEFINICION DE CONSTANTES
MAXIMO EQU 50 ; DE PALABRAS (16 BITS)
LONGITUD_REG EQU 30
INVALIDO EQU 0 ; INDICADOR DE CODIGO
MENSAJE1 DB "REGISTRO ? $"
MENSAJE2 DB "INGRESE INFORMACION : $"
NOMBRE DB "AZAR",0
ARCH DW ? ; = DESCRIPTOR DE ARCHIVO
CADENA DB 31,?,31 DUP(".") ; AREA DE TRANSFERENCIA INICIADA CON "."
CLINEA DB 10,13,"$"
CADENA_COD DB 4,?,4 DUP(?) ; = CADENA PARA LECTURA DEL
CODIGO DW ? ; CODIGO EN CARACTERES ASCII
;
PROGRAMA ENDS ; FIN DEL SEGMENTO PROGRAMA
(CODIGO Y DATOS)
END COMIENZO
```



CAPITULO XII

Bios: Rutinas de Servicios.

12.1 PROCESAMIENTO DE LA PANTALLA

La instrucción Int, para la mayoría de los propósitos maneja entrada y salida. Los dos tipos de interrupciones tratados aquí son las funciones de Int 10h del BIOS y las funciones de Int 21h del DOS para mostrar salidas en pantalla y aceptar entradas desde el teclado.

FUNCIONES DE INT 10H (BIOS)	FUNCIONES DE INT 21H (DOS)
02H Fija del cursor	02H Despliega en pantalla
06H Recorre la pantalla	09H Despliega en pantalla
	40H Despliega en pantalla

La pantalla

La pantalla es una malla de posiciones direccionables, en las cuales se puede colocar el cursor.

UBICACIÓN EN PANTALLA	FORMATO DECIMAL	
	RENGLON	COLUMNA
Esquina superior izquierda	00	00
Esquina superior derecha	00	79
Centro de pantalla	12	39/40
Esquina inferior izquierda	24	00
Esquina inferior derecha	24	79

El sistema proporciona espacio en la memoria para un área de despliegue de video o buffer. El área de despliegue monocromático inicia en la localidad de BIOS b0000[0]h y permite utilizar 4k bytes de memoria: 2k para caracteres y



2k para atributos para cada carácter, como video inverso, intermitencia, intensidad y subrayado.

Colocación del cursor

(El modo grafico no permite el uso del cursor)

- se debe cargar el numero de pagina (o pantalla), por lo común 0 en el registro BH.
- en el registro DX el renglón y columna requerida.

Ej.: mov bh , 00
 mov dh , 05 ; renglón 5
 mov dl , 12 ; columna 12
 mov ah , 02h ; función colocación
 int 10h

Limpiar la pantalla

Para limpiar toda la pantalla especifique en el renglon:columna (00:00h) y en el renglón columna final (18:4fh)

Cargar los registros:

AH = función 06h
AL = 00h para la pantalla completa
BH = numero del atributo
CX = renglon:columna inicial
DX = renglon:columna final

El atributo 71h establece la pantalla en fondo blanco (7) con primer plano azul (1)



Procesamiento avanzado de la pantalla

FUNCIONES DE INT 10H (BIOS)	FUNCIONES DE INT 10H (BIOS)
00H Establece el modo de video	0BH Establece la paleta de colores
01H Establece el modo de cursor	0CH Escribe el pixel punto
02H Establece la posición del cursor	0DH Lee el pixel punto
03H Lee la posición del cursor	0EH Escribe en teletipo
04H Lee la posición de la pluma óptica	0FH Obtiene el modo actual de video
05H Selecciona la pagina activa	11H Genera carácter
06H Recorre la pantalla hacia arriba	12H Selecciona rutina alterna de pantalla
07H Recorre la pantalla hacia abajo	13H Despliega cadena de caracteres
08H Lee el atributo o carácter en la posición del cursor	1BH Regresa la información de funcionalidad o de estado
09H Despliega el atributo o carácter en la posición del cursor	1CH Guarda o restaura el estado de video
0AH Despliega el carácter en la posición del cursor	

Adaptadores de video

MDA	Adaptador de pantalla monocromática
HGC	Tarjeta de gráficos Hércules
CGA	Adaptador de gráficos en colores
EGA	Adaptador de gráficos mejorado
MCGA	Adaptador de gráficos en multicolores (PS/2 modelos 25 y 30)
VGA	Matriz de gráficos de video

El adaptador de video consta de tres partes:

- El controlador de video. genera las señales de rastreo del monitor para el modo seleccionado, texto o grafico.



- El video de BIOS, actúa como un interfaz con el adaptador de video, contiene rutinas como para establecer el cursor y desplegar caracteres.
- El área de despliegue. contiene la información que el monitor va a mostrar. las interrupciones que manejan el despliegue en pantalla de forma directa transfieren a esta área sus datos. Las localidades del adaptador de video dependen de los modos de video que se estén usando.

Direcciones de inicio para adaptadores de video

A000:[0]	Utilizada para descripción de fuentes cuando esta en modo de texto y para gráficos de alta resolución para EGA, MCGA, VGA
B000:[0]	Modo de texto monocromático para MDA, EGA, VGA
B100:[0]	Para HCG
B800:[0]	Modos de texto para CGA, MCGA, EGA y VGA y modos gráficos para CGA, EGA, MCGA y VGA.

Especificaciones del modo de video

La función 00h de Int 10h de BIOS, puede designar el modo para el programa que se esta ejecutando actualmente o puede conmutar entre texto y grafico.

Para designar un nuevo modo:

```
mov ah , 00h ; petición para designar el modo
mov al , 03h ; texto o estándar a color 80 x 25
int 10h      ; llama al BIOS
```

Para monitores desconocidos, puede utilizar Int 10h y función 0Fh, la cual regresa en al el modo de video actual.



12.2 MODO TEXTO

El modo de texto se utiliza para el despliegue normal en la pantalla de caracteres ASCII.

MODO	TAMAÑO	TIPO	ADAPTADOR	RESOLUCION	COLORES
00	25 FILAS , MONO , 40 COLUMNAS	MONO	CGA EGA MCGA VGA	320 X 200 320 X 350 320 X 400 360 X 400	
01	25 FILAS , COLOR , 40 COLUMNAS	COLOR	CGA EGA MCGA VGA	320 X 200 320 X 350 320 X 400 360 X 400	16 16 DE 64 16 DE 262,144 16 DE 262,144
02	25 FILAS , MONO , 80 COLUMNAS	MONO	CGA EGA MCGA VGA	640 X 200 640 X 350 640 X 400 720 X 400	
03	25 FILAS , COLOR , 80 COLUMNAS	COLOR	CGA EGA MCGA VGA	640 X 200 640 X 350 640 X 400 720 X 400	16 16 DE 64 16 DE 262,144 16 DE 262,144
07	25 FILAS , MONO , 80 COLUMNAS	MONO	MDA EGA VGA	720 X 350 720 X 350 720 X 400	

Modo de texto: Byte de atributo

Un byte de atributo en modo de texto determina las características de cada carácter mostrado. Cuando un programa establece un atributo, permanece activado; esto es, todos los caracteres subsecuentes desplegados tienen el mismo atributo hasta que otra operación lo cambie.

➤ Formato:

INTERMITENCIA	FONDO PANTALLA			INTENSIDAD	CARÁCTER DESPLEGADO		
BL	R	G	B	I	R	G	B
7	6	5	4	3	2	1	0

Los bits RGB: 000 es negro y 111 es blanco



Modo de texto: despliegue monocromático

Para un monitor monocromático, el bit 0 establece el atributo de subrayado

➤ Combinaciones:

FONDO	FRENTE	CARACTERISTICAS	FONDO				PRIMER PLANO				HEX
			BL	R	G	B	I	R	G	B	
NEGRO	NEGRO	NO DESPLIEGA	0	0	0	0	0	0	0	0	00H
NEGRO	BLANCO	NORMAL	0	0	0	0	0	1	1	1	07H
NEGRO	BLANCO	INTERMITENTE	1	0	0	0	0	1	1	1	87H
NEGRO	BLANCO	INTENSO	0	0	0	0	1	1	1	1	0FH
BLANCO	NEGRO	VIDEO INVERSO	0	1	1	1	0	0	0	0	70H
BLANCO	NEGRO	INVERSO, INTERMITENTE	1	1	1	1	0	0	0	0	F0H
		SUBRAYADO	0	0	0	0	0	0	1		01H

Modo de texto: despliegue a color

➤ Combinaciones:

COLOR	I	R	G	B	COLOR	I	R	G	B
NEGRO	0	0	0	0	GRIS	1	0	0	0
AZUL	0	0	0	1	AZUL CLARO	1	0	0	1
VERDE	0	0	1	0	VERDE CLARO	1	0	1	0
CIAN	0	0	1	1	CIAN CLARO	1	0	1	1
ROJO	0	1	0	0	ROJO CLARO	1	1	0	0
MAGENTA	0	1	0	1	MAGENTA CLARO	1	1	0	1
CAFÉ	0	1	1	0	AMARILLO	1	1	1	0
BLANCO	0	1	1	1	BLANCO BRILLANTE	1	1	1	1



Páginas de pantalla

Los modos de texto le permiten almacenar datos en memoria de video en páginas. Los números de página son desde 0 hasta 3 para el modo normal de 80 columnas (y 0 hasta 7 para las de 40 cols). En modo de 80 cols, las direcciones de inicio son:

- página 0 a partir de b800:[0]
- página 1 a partir de b900:[0]
- página 2 a partir de ba00:[0]
- página 3 a partir de bb00:[0]

Solo se puede desplegar una página a la vez.

Cada carácter que se muestra en la pantalla necesitan dos bytes de memoria: un byte para el carácter y un segundo byte para su atributo. De esta forma una página completa de caracteres, para 80 columnas y 25 líneas, se necesita $80 \times 25 \times 2 = 4000$ bytes.

La cantidad de memoria realmente asignada a cada página es 4k o 4096 bytes, así que después de cada página la siguen 96 bytes no utilizados.

Interrupción 10h del BIOS para el modo texto

La interrupción conserva el contenido de los registros BX, CD, DX, DI, SI y BP, pero no el AX.

Modo de video: 00h

Se requiere:

Código 00h en ah

Código de modo en AL; 03 para color y 07 para monocromático

Tamaño del cursor: 01h

Se requiere:

CH (Bits 4-0) = Parte superior del cursor

CL (Bits 4-0) = Parte inferior del cursor



Para ajustar el tamaño del cursor entre la parte superior y la inferior:
0:14 (ch:cl)

Establece posición del cursor: 02h

Se requiere:

AH = 02h
BH = Numero de pagina (0,1,2,3)
DH = Renglón
DL = Columna

Lee posición del cursor: 03h

Se requiere:

AH = 03h
BH = 0h ; MODO NORMAL

Devuelve: AX y BX sin cambios

CH = Línea inicial cursor
CL = Línea final cursor
DH = Renglón
DL = Columna

Seleccionar pagina activa: 05h

Se requiere:

AH = 05h
AL = Numero de pagina (0,1, 2, 3)

Recorrer hacia arriba la pantalla: 06h

Se requiere:

AH = 06h
AL = Numero de líneas o cero para toda la pantalla
BH = ATRIBUTO
CX = Renglon:Columna Inicial
DX = Renglon:Columna Final



Recorrer hacia abajo la pantalla: 07h

Se requiere:

AH = 07h

AL = Numero de líneas o cero para toda la pantalla

BH = ATRIBUTO

CX = Renglon:Columna Inicial

DX = Renglon:Columna Final

Leer atributo o caracter: 08h

Se requiere:

AH = 08h

BH = NUMERO DE PAGINA

Devuelve:

AL = CARÁCTER

AH = ATRIBUTO

Para un carácter no ascii, devuelve 00h

Desplegar atributo o caracter: 09h

Se requiere:

AH = 09h

AL = Un carácter ascii

BH = Numero de pagina

BL = Atributo

CX = Numero de veces que la operación despliega de manera repetida el carácter que esta en AL.

Desplegar un carácter: 0ah

Se requiere:

AH = 0Ah

AL = Un carácter ascii

BH = Numero de pagina

CX = Numero de veces que la operación despliega de manera repetida el carácter que esta en AL.



Esta función utiliza el atributo actual

Escribir en teletipo: 0eh

Se requiere:

- AH = 0Eh
- AL = Un carácter ascii
- BH = Numero de pagina
- BL = Color del primer plano (modo grafico).

Obtener modo de video: 0fh

Se requiere:

- AH = 0Fh

Devuelve:

- AL = Modo actual de video
- AH = Caracteres por línea
- BH = Numero de pagina actual

Despliega una cadena de caracteres: 13h

Se requiere:

- AH = 13h
- AL = Subfuncion
- BH = Numero de pagina
- BL = Atributo
- BP = Dirección de la cadena
- CX = Longitud de la cadena
- DX = Posición relativa inicio pantalla

Subfuncion:

- 00 Despliega el atributo y la cadena; no avanza el cursor
- 01 Despliega el atributo y la cadena; avanza el cursor
- 02 Despliega el carácter y después el atributo; no avanza cursor



03 Despliega el carácter y después el atributo; avanza el cursor

12.3 MODO GRAFICO

MODO	TIPO	ADAPTADOR	RESOLUCION	COLORES
04	COLOR	CGA,EGA,MCGA,VGA	320 X 200	4
05	MONO	CGA,EGA,MCGA,VGA	320 X 200	
06	MONO	CGA,EGA,MCGA,VGA	640 X 200	
0D	COLOR	EGA,VGA	320 X 200	16
0E	COLOR	EGA,VGA	640 X 200	16
0F	MONO	EGA,VGA	640 X 350	
10	COLOR	EGA,VGA	640 X 350	16
11	COLOR	MCGA, VGA	640 X 400	2 DE 262,144
12	COLOR	VGA	640 X 400	2 DE 262,14416
13	COLOR	MCGA,VGA	320 X 200	256 DE 262,144

Modo grafico: píxeles

El modo grafico utiliza píxeles para generar patrones en color. Cada byte representa 4 píxeles.

➤ Formato:

BYTE	C1	C0	C1	C0	C1	C0	C1	C0
PIXEL	0		1		2		3	

NRO	C1	C0	PALETA 0	PALETA 1
0	0	0	FONDO	FONDO
1	0	1	VERDE	CIAN
2	1	0	ROJO	MAGENTA
3	1	1	CAFE	BLANCO



Interrupción 10h del BIOS para el modo grafico

Lee posición pluma óptica: 04h

Se requiere:

AH = 04h

Devuelve:

AH = 0 si el estado de funcionando y 1 si no esta

DX = renglón en DH y columna en DL

CH/BX = posición de píxel, con línea (horizontal) de la malla
en el BH y columna o punto en el BX

Establece una paleta de colores: 0bh

Se requiere:

AH = 0Bh

BH = 00 selecciona el color de fondo, en donde BL contiene el
numero del color en los bits 0-3 (cualquiera 16
colores)

BH = 01 selecciona la paleta para gráficos, en donde BL
contiene la paleta (0 ó 1)

Escribe un píxel punto: 0ch

Se requiere:

AH = 0Ch

AL = Color del píxel

BH = Numero de pagina

CX = Columna

DX = Renglón

Lee un píxel punto: 0dh

Se requiere:

AH = 0Dh

BH = NUMERO DE PAGINA

CX = COLUMNA

DX = RENGLON



Devuelve:

AL = Color del píxel

Determinación del tipo de adaptador de video

Determinar si esta instalado VGA: 1ah

Se requiere:

AH = 1ah

AL = 0

Devuelve:

AL = 1ah si es VGA



CAPITULO XIII

Programas Residentes en memoria.

13.1 EL DOS

Administración de la memoria del dos

Administrar la memoria implica describir el proceso de arranque, la inicialización del dos, el prefijo segmento programa, el entorno (ambiente), control de memoria, el cargador de programas y los programas residentes.

Programas principales del dos

Los cuatro programas principales del dos son: el registro de arranque, IO.SYS, MSDOS.SYS y COMMAND.COM

El registro de arranque, esta en la pista 0, sector 1 de cualquier disco que usted formatee con FORMAT /S. cuando inicializa la computadora. El sistema carga de manera automática el registro de arranque del disco y lo envía a la memoria. El registro de arranque a su vez carga IO.SYS del disco a la memoria.

IO.SYS

Es una interfaz de bajo nivel con las rutinas del BIOS en ROM. En la iniciación determina el estado de los dispositivos y el equipo asociado con la computadora y establece direcciones de la tabla de interrupciones hasta la 20h.

También maneja la entrada/salida entre la memoria y los dispositivos externos, como el monitor o los discos. Después carga el MSDOS.SYS.

MSDOS.SYS

Es una interfaz de alto nivel para programas que establece las direcciones de la tabla para las interrupciones de la 20h a la 3fh.



Administra el directorio y los archivos en disco, bloqueo y desbloqueo de registros en disco, las funciones de la Int 21h y otros servicios. Después carga el COMMAND.COM

COMMAND.COM

Maneja los diferentes comandos, como DIR y CHKDSK, y ejecuta todas las peticiones de programas .COM, .EXE y .BAT. Es responsable de cargar los programas ejecutables de disco a memoria.

Mapa de memoria después de cargar los programas de sistema del dos

DIRECCION DE INICIO	CONTENIDO
F0000H	AREA ROM DEL SISTEMA
E0000H	ROM BIOS
D0000H	ROM BIOS
C0000H	ROM BIOS
B0000H	BUFFERS DE VIDEO
A0000H	BUFFERS DE VIDEO
xxxx0H	PORCION TRANSITORIA DE COMMAND.COM, EN EL TOPE DE LA RAM
	PROGRAMAS DE USUARIO
	PROGRAMAS RESIDENTES (SI HAY ALGUNO)
xxxx0H	PORCION RESIDENTE DE COMMAND.COM
00500H	AREA DE COMUNICACIÓN DEL DOS
00400H	AREA DE DATOS DEL BIOS
00000H	TABLA DE DIRECCIONES DE INTERRUPCION
NOTA:	La memoria convencional va de 0000H a A0000H (640 kb) El área de la memoria superior va de A0000h a FFFF0h (1 Mb) El área de la memoria alta es de 64 Kb y va de FFFF0h a FFFFFh La memoria extendida esta arriba de HMA

Área de memoria alta

Para determinar la presencia del dos en el HMA.

MOV AX , 3306H ; Petición de la versión del dos
INT 21H



La operación regresa:

BL = Numero principal de la versión (7)

BH = Numero secundario de la versión (1)

DL = Numero de la revisión en los tres bits inferiores (2-0)

DH = Banderas de la versión del dos, en donde el bit cuatro (4)
= 1 significa en el HMA

Verificación del espacio disponible en el HMA

La Int 2fh del dos (de multiplexión), entre otros servicios, también proporciona una verificación del espacio disponible en el HMA.

```
MOV AX , 4A01H ; Petición de espacio en HMA  
INT 2FH
```

La operación regresa:

BX = numero de bytes libres disponibles en el HMA (cero si el
dos no esta cargado en el área alta)

ES:DI = dirección del primer byte libre en el HMA (ffff:ffff si el
dos no esta cargado en el área alta)

Command.com

El sistema carga las tres partes del command.com en la memoria, ya sea de manera permanente durante una sesión o bien cuando se requiera de manera temporal.

Parte 1:

La parte residente de COMMAND.COM carga de forma inmediata MSDOS.SYS en donde reside durante el procesamiento. La parte residente maneja errores para e/s de disco y las interrupciones siguientes:



INT 22H Dirección de terminación

INT 23H Manejador de CTRL + BREAK

INT 24H detección de error en READ/WRITE en disco o incorrecta
imagen de la memoria de la FAT

INT 27H Termina pero permanece residente

Parte 2:

La parte de inicialización de command.com sigue inmediatamente a la parte residente y contiene la configuración para archivos autoexec. Cuando el sistema inicia, la parte de inicialización toma el control y determina la dirección del segmento en la cual el sistema cargara programas para su ejecución.

Ninguna de las rutinas de inicialización se requiere durante la sesión. En consecuencia, su primera petición para cargar un programa desde disco hace que el dos traslape la parte de inicialización de la parte del COMMAND.COM siempre que resida en memoria.

Parte 3:

La parte transitoria de command.com es cargada en un área alta de memoria "transitoria" implica que si es necesario, el dos puede traslapar esta área con otros programas requeridos.

La parte transitoria muestra una petición común en pantalla y acepta y ejecuta las peticiones. Contiene un cargador de reubicación que carga archivos .EXE y .COM desde disco hacia la memoria para su ejecución.

Cuando se pide la ejecución de un programa, la parte transitoria construye un segmento de programa en la mas baja localidad de memoria disponible.



Crea el PSP en 00h, carga el programa ejecutable pedido en el desplazamiento 100h, establece la dirección de salida y pasa el control al programa que cargo.

Prefijo segmento de programa (PSP)

El dos carga programas .COM y .EXE para su ejecución y crea un PSP en el desplazamiento 00h y el programa mismo en el desplazamiento 100h del segmento. El PSP contiene los campos siguientes de acuerdo con la posición relativa.

00-01h	una instrucción de la int 21h (cd20h) para facilitar el regreso al dos
02-03h	Dirección del segmento del último párrafo de la memoria asignada al programa, como xxxx0. por ejemplo 640k se indica como 00a0h, para significar a0000[0]
04-09h	reservado para el dos
0a-0dh	Dirección de terminación (dirección del segmento para int 22h)
0e-11h	Dirección de salida de ctrl + break (dirección de segmento para int 23h)
12-15h	Dirección de salida de error critico (dirección de segmento para int 24h)
16-17h	reservado para el dos
18-2bh	tabla de manejadores de archivos por omisión
2c-2dh	Dirección de segmento del entorno del programa
2e-31h	reservado por el dos
32-33h	longitud de la tabla de manejadores de archivos
34-37h	apuntador lejano a la tabla de manejadores
38-4fh	reservado por el dos
50-51h	llama a la función del dos (int 21h y retf)
52-5bh	reservado por el dos
5c-6bh	Área de parámetro 1. formateado como un fcb (# 1) estándar no abierto
6c-7fh	Área de parámetro 2. formateado como un fcb (# 2) estándar no abierto, traslapado, si el fcb es 5ch esta abierto
80-ffh	buffer por omisión para un dta



13.2 PROGRAMAS RESIDENTES

Una de las rutinas de servicio más interesantes del dos es la que corresponde a la interrupción 27h. Como la interrupción 20h, sirve para parar la ejecución del programa en curso y devolver el control al dos. Pero además permite que una porción del programa permanezca en memoria.

La información que se queda en memoria se convierte, de esta forma, en una extensión del dos. Esta información puede ser código ejecutable y/o datos. Por ejemplo, rutinas de manejo de interrupciones, rutinas de intercepción del teclado, datos de comunicación entre programas, etc. la información queda residente en memoria mientras el dos esta funcionando.

Antes de que se ejecute la Int 27h, el registro DX debe contener el desplazamiento respecto al segmento de la posición siguiente a la última que se quiere dejar residente. El tamaño máximo de la información que se puede dejar residente es de 64kb. Un programa que utiliza esta técnica tiene normalmente dos partes:

- La parte de instalación de la información a dejar residente
- La información que se va a dejar residente

Esquema de instalación 1

```
;
; Esquema de instalación de una rutina cuya dirección se sitúa en el área de
; Comunicaciones entre aplicaciones (dirección 0:4f0h)
;-----
;
;
; Segmento para acceso al área de comunicaciones entre aplicaciones
;-----
;
;
SEG CERO SEGMENT AT 0H
    ORG    4F0H    ; Área de comunicaciones entre aplicaciones
DES_RUT    DW    ?    ; Dirección rutina: desplazamiento
SEG_RUT    DW    ?    ; Dirección rutina: segmento
SEG CERO ENDS
;
;-----
;
;
; Parámetros:
PARAMS      STRUC
    DW    ?    ; BP salvado por la subrutina
```



```
RETORNO DD ? ; Dirección de retorno
PN DD ? ; Dirección de parámetro N
;
P1 DD ? ; Dirección de parámetro 1
PARAMS ENDS
;
;-----
;
CODIGOSEGMENT
ASSUME CS:CODIGO
ORG 100H ; Por ser tipo COM
EMPEZAR:
JMP INSTALAR; Bifurca a la rutina de instalacion
;
;-----
;
; Definición de variables
;-----
;
...
;
;-----
;
; Rutina a instalar
;-----
;
RUTINA PROC NEAR ; Comienzo procedimiento
PUSH BP ; Salvar BP
MOV BP , SP ; BP = SP (Apuntar al área de parámetros)
;
;-----
;
; Salvar registros afectados
;
PUSH ...
;
; Instrucciones de la rutina
;
...
; Restaurar registros afectados
;
POP ...
;
;-----
;
MOV SP , BP ; SP = BP
POP BP ; Restaurar BP
RET NRET
RUTINA ENDP
;
;-----
;
; Instalación
;-----
;
; Almacena la dirección de la rutina en el área de comunicaciones entre aplicaciones
; deja residente la rutina
;
INSTALAR PROC
;
```



```
; Dirección segmento SEGCERO con el registro ES
;
;
ASSUME ES:SEGCERO
MOV AX , 0          ; AX = 0
MOV ES , AX        ; ES = 0
;
; Poner dirección de la subrutina en el área de comunicaciones entre aplicaciones
;
MOV DES_RUT , OFFSET RUTINA      ; Desplazamiento
MOV SEG_RUT , CS                 ; segmento
;
; Terminar rutina de instalación, pero dejar residente la rutina
;
MOV DX , OFFSET INSTALAR        ; DX = Última dirección +1 respecto
                                ; al segmento de la rutina a instalar
INT 27H                        ; Terminar pero dejar residente
;
INSTALAR ENDP
;
;-----
;
;
CODIGOENDS
        END EMPEZAR
```

Esquema de instalación 2

```
;
;
; Esquema de instalación de una rutina de servicio de una interrupción
; (de número NRO_INT )
;-----
;
;
; Segmento correspondiente a los vectores de interrupción
;-----
;
;
INTERRS  SEGMENT AT 0H
        ORG    NRO_INT * 4
DIREC   LABEL DWORD      ; Dirección rutina de servicio
INTERRS  ENDS
;
;-----
;
; Segmento Principal:
;-----
;
;
CODIGOSEGMENT
        ASSUME CS:CODIGO,DS:CODIGO
        ORG 100H      ; Por ser tipo COM
EMPEZAR:
        JMP INSTALAR; Bifurca a la rutina de instalación
;
;-----
;
;
; Definición de variables
;-----
;
;
        ...
;
;-----
;
; Rutina de servicio
```



```
;-----  
;  
;  
RUTINA PROC  
    CLI                      ; Inhibir interrupciones  
;  
;  
;-----  
;  
;  
; Salvar registros afectados  
;  
    PUSH ...  
;  
; Instrucciones de la rutina  
;  
    ...  
; Restaurar registros afectados  
;  
    POP ...  
    STI                      ; Permitir Interrupciones  
    IRET                    ; Una interrupción debe terminar con IRET  
RUTINA ENDP  
;  
;  
;-----  
;  
; Instalación  
;-----  
;  
; Almacena la dirección de la rutina en el vector de interrupción y  
; deja residente la rutina  
;  
INSTALAR PROC  
;  
; Direccionar segmento de vectores de interrupciones con ES  
;  
    ASSUME ES:INTERRS  
    MOV AX , 0              ; AX = 0  
    MOV ES , AX            ; ES = 0  
;  
; Poner en el vector de interrupción la dirección de la rutina  
;  
    CLI                      ; Inhibir interrupciones  
    MOV DIREC , OFFSET RUTINA ; Desplazamiento  
    MOV DIREC + 2 , CS       ; segmento  
;  
; Terminar rutina de instalación, pero dejar residente la rutina  
;  
    MOV DX , OFFSET INSTALAR ; DX = Ultima dirección +1 respecto  
                                ; al segmento de la rutina  
                                ; interrupción  
    STI                      ; Permitir interrupciones  
    INT 27H                  ; Terminar pero dejar residente  
;  
INSTALAR ENDP  
;  
;-----  
;  
CODIGO ENDS  
    END EMPEZAR
```



Función

La función 31h de la int 21h, también sirve para terminar y dejar residente un programa.

Pero además:

- Permite reportar un código de retorno en al
- Supera la limitación de las 64kb de la int 27h. en DX se especifica la cantidad de memoria (en párrafos) que debe permanecer. El resto se libera.

PAGE 40, 132

TITLE RELOJ

COMMENT *

INSTALACION DE RUTINA RESIDENTE PARA ESCRIBIR LA HORA
EN LA ESQUINA SUPERIOR DERECHA DE LA PANTALLA
EL TEMPORIZADOR (TIMER) EMITE UNA INTERRUPCION TIPO 1CH
CON LA FRECUENCIA $f = 1193180 / 65536 = 18.206481$ VECES/SEG
POR MINUTO: $f \times 60 = 1092$
POR HORA: $1092 \times 60 = 65543 = 32760 \times 2$

*

;-----

; Simbolos

PANTALLA EQU 0B800H ; Segmento memoria de pantalla

;

;Segmento correspondiente a los vectores de interrupcion del DOS

INTERRS SEGMENT AT 0H

ORG 1CH * 4

DIREC LABEL DWORD ; Direccion rutina reloj

INTERRS ENDS

;

;-----

;

; Segmento principal

;-----

;

CODIGO SEGMENT



```
ASSUME CS:CODIGO, DS:CODIGO
    ORG 100H                      ; Origen por ser tipo COM
EMPEZAR:
    JMP INSTALAR
;
; Variables
;
HORA DB 8 DUP (?)                ; Hora en formato hh:mm:ss
;
;-----
;
; Rutina reloj
RELOJ PROC
    CLI                          ; Inhibir interrupciones
;
; Salvar registros
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI
    PUSH DI
    PUSH DS
    PUSH ES
;
; Obtener hora de las palabras 40h:6ch (inferior) y 40h:6eh (superior)
;
    MOV AX, 40H                  ; Segmento
    MOV ES, AX
    MOV SI, 6CH                  ; DESPLAZAMIENTO
    MOV AX, ES:[SI]              ; PARTE INFERIOR CONTADOR
    MOV DX, ES:[SI+2]            ; PARTE SUPERIOR CONTADOR
    MOV BX, 32771
    DIV BX                       ; DX = RESTO, AX = HORAS * 2
    SHR AX, 1                    ; AX = HORAS (DIVIDIR ENTRE 2)
    MOV CH, AL                   ; CH = HORAS
;
    MOV AX, DX
    MOV DX, 0
    MOV BX, 1092
```



```
DIV BX                ; DX = RESTO , AX = MINUTOS
MOV CL , AL           ; CL = MINUTOS
;
MOV AX , DX
MOV DX , 0
MOV BX , 18
DIV BX
MOV DH , AL           ; DH = SEGUNDOS
;
; PONER LA HORA EN FORMATO DECIMAL
;
MOV BH , 10           ; BN = 10 (CONSTANTE)
MOV AH , 0
MOV AL , CH           ; AX = HORAS
DIV BH               ; AX HORAS EN DECIMAL
ADD AX , 3030H
MOV WORD PTR HORA , AX
MOV HORA + 2 , ":"
;
MOV AH , 0
MOV AL , CL           ; AX = MINUTOS
DIV BH               ; AX MINUTOS EN DECIMAL
ADD AX , 3030H
MOV WORD PTR HORA + 3 , AX
MOV HORA + 5 , ":"
;
MOV AH , 0
MOV AL , DH           ; DH = SEGUNDOS
DIV BH               ; AX SEGUNDOS EN DECIMAL
ADD AX , 3030H
MOV WORD PTR HORA + 6 , AX
MOV HORA + 8 , ":"
;
MOV AH , 0
MOV AL , DL           ; DL = CENTESIMAS
DIV BH               ; AX CENTESIMAS EN DECIMAL
ADD AX , 3030H
MOV WORD PTR HORA + 9 , AX
;
; Direccionar zona memoria de pantalla con es:[di]
```



```
;
MOV AX , PANTALLA
MOV ES , AX
MOV DI , 2 * (80 - LENGTH HORA)
;
; MOVER LA HORA
;
MOV CX , LENGTH HORA
MOV SI , 0 ; DESPLAZAMIENTO INICIAL
MOVER:
MOV AL , HORA[SI] ; CARACTER CAMPO HORA
MOV ES:[DI] , AL ; MOVERLO A MEMORIA PANTALLA
INC DI
MOV BYTE PTR ES:[DI] , 07H ; MOVER ATRIBUTO NORMAL
INC SI
INC DI
LOOP MOVER
;
; RESTAURAR REGISTROS
;
FIN:
POP ES
POP DS
POP DI
POP SI
POP DX
POP CX
POP BX
POP AX
STI ; PERMITIR INTERRUPCIONES
IRET ; UNA INTERRUPCION DEBE TERMINAR CON IRET
RELOJ ENDP
;
;-----
;
;
; Instalación
;-----
;
;
INSTALAR PROC
;
;
```



```
;DIRECCIONAR SEGMENTO DE VECTORES DE INTERRUPCIONES CON ES
;
    ASSUME ES:INTERRS
    MOV AX , 0
    MOV ES , AX
;
; PONER EN EL VECTOR 1CH = 28 LA DIRECCION DE LA RUTINA RELOJ
;
    CLI
    MOV WORD PTR [DIREC] , OFFSET RELOJ ; DESPLAZAMIENTO
    MOV WORD PTR [DIREC + 2] , CS ; SEGMENTO
;
; TERMINAR RUTINA DE INSTALACION, PERO DEJAR RESIDENTE LA RUTINA
;
    MOV DX , OFFSET INSTALAR
    STI
    INT 27H
INSTALAR ENDP
;
;-----
;
;
CODIGO ENDS
END EMPEZAR
```

REFERENCIAS BIBLIOGRAFICAS

1. Francisco Charte Ojeda. "Programación ensamblador para DOS, Linux y Windows". Edit. Anaya Multimedia. Madrid. 2003.
2. J.Terry Godfrey. "Lenguaje ensamblador para microcomputadoras" Edit. Prentice Hall. 1991
3. Miguel Ángel Rodríguez-Rosello. "PROGRAMACIÓN ENSAMBLADOR EN ENTORNO MS-DOS 8088/8086.
4. Willian H. Murriay / Chris H. Papas. "PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR 80386/80286". España. Ed. McGraw-Hill. Inc USA 1987.
5. Jaime Martinez Charte. "ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORAS". Ed. Addison Wesley longman de mexico. SA. Primera edicion. 2003.
6. Meter Abel. "LENGUAJE ENSAMBLADOR Y PROGRAMACION PARA PC IBM Y COMPATIBLES". Ed. Pearson Educación. Tercera edicion. 1996.
7. Ronald J. Tocci. "SISTEMAS DIGITALES". Ed. Prentice-Hall Hispanoamericana. 1996.
8. Maynard Kong. "LENGUAJE ENSAMBLADOR, MACRO ENSAMBLADOR". Universidad Catolica del peru. Fondo editorial. 1999.