PROGRAMACION EN BAJO NIVEL

Por : J. LUIS C. JAUREGUI T.

LICENCIADO EN INFORMATICA

Texto para la materia INF 475 de las Carreras de Licenciatura en Informática e Ingeniería Electrónica de la Facultad de Ciencias y Tecnología de la Universidad Mayor de San Simón

Cochabamba – Bolivia

Prologo

El presente texto es el resultado de un largo tiempo de haber dictado la materia INF 475 – Taller de programación en bajo nivel.
Su propósito es suministrar a los estudiantes todo el material que se expone en clases y los ejercicios similares a los que se pedirán en los exámenes.
¿ Como aprobar la materia INF 475 ?
Se recomienda en primer lugar disponer y estudiar todo el material, consistente en este texto y un diskette con programas fuente y ejecutables.
En segundo lugar, realizar prácticas en computador sobre los ejercicios propuestos.
En tercer lugar, asistir a clases.
Existe aun mucho por hacer para tener un curso completamente actualizado y espero hacerlo en próximas revisiones. Mientras, espero que este texto sea de provecho y les ayude a algo mas que aprobar la materia.

Lic. J. Luis C.Jauregui T.
Cochabamba, marzo de 2002

INDICE

CAPITULO I: INTRODUCCION

I.1.- DE LOS CIRCUITOS LOGICOS A LOS PROCESADORES

De los circuitos lógicos surge la posibilidad de convertir señales de control en señales de acción; por ejemplo, construir el control remoto de una puerta con un mando a distancia que la abra o cierre es un ejercicio de circuitos lógicos. Las señales de control, sus fuentes y codificación pueden ser las siguientes:
Señal 1.- un mando de control remoto (M) que emite dos señales:

0 = cerrar la puerta
1 = abrir la puerta

Señal 2.- un sensor (A) en la puerta que indica:

0 = puerta cerrada
1 = puerta no cerrada

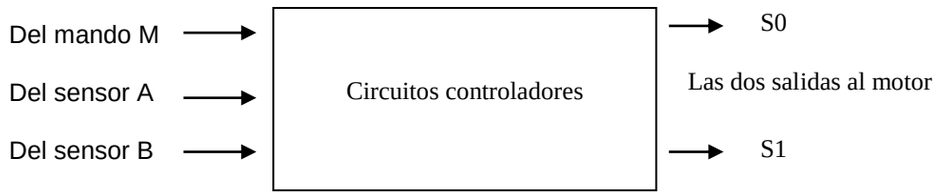
Señal 3.- un sensor (B) en el lugar donde la puerta queda abierta e indica:

0 = puerta abierta
1 = puerta no abierta

La puerta puede operada por un motor que reciba los siguientes comandos (acciones):

01 = mover la puerta para abrirla
10 = mover la puerta para cerrarla
00 = detenerse

A nivel electrónico, las señales pueden asociarse con tensiones eléctricas; un 1 corresponderá a la tensión superior a alguna constante y 0 debajo o la misma constante.
Los sensores y el mando podrán llegar al control de la puerta con un solo hilo o conexión mientras al motor deberán entrar dos conexiones, una para cada dígito de los dos que se usan para representar sus comandos.
Esquemáticamente, el dispositivo se vera así:



Para diseñar los circuitos se construye una tabla con todas las posibilidades de entradas y sus respectivas salidas:

Entradas M	A	B	Salidas S0	S1	Explicación
0	0	0	0	0	Se ordena cerrar la puerta, y la puerta esta cerrada, por tanto no debe hacerse nada
0	0	1	0	0	
0	1	0	1	0	Se ordena cerrar la puerta, y la puerta no esta cerrada, por tanto debe moverse hasta cerrarse
0	1	1	1	0	
1	0	0	0	0	Se ordena abrir la puerta, y la puerta esta abierta, por tanto no debe hacerse nada
1	0	1	0	0	
1	1	0	0	1	Se ordena abrir la puerta, y la puerta no esta abierta, por tanto debe moverse hasta abrirse
1	1	1	0	1	

Por inspección, se deduce que S0 es 1 solo cuando M es 0 y A es 1,sin importar el valor de B que puede ser 0 o 1, por lo que:

S0 = (no M) y A

de la misma manera, S1 es 1 solo cuando M es 1 y B es 1, por lo que:

S1 = M y B

Algo mas que debe considerarse es que la señal del mando puede desaparecer cuando el mando se aleje, por lo que es necesario tener un dispositivo de que almacene su señal.
De este modo, con tres compuertas lógicas (un NO y dos Y) y con un elemento de memoria que retenga las señal del mando de control remoto, puede tenerse un muy eficiente y a la vez simple dispositivo de control de la puerta.

Este tipo de dispositivos tiene propósito especifico; solo resuelven un problema bien definido. Pero, como es posible construir dispositivos de propósito general ?, procesadores capaces de resolver una gran cantidad de problemas y no solo uno especifico ?
La clave de los procesadores es su flexibilidad, que se logra con la capacidad de recibir programas, tal cual los conocemos en lenguajes de alto nivel.
Mientras que el circuito controlador de puertas enlaza rígidamente las señales con los circuitos, un procesador recibe las señales, y las trata según un algoritmo. Por ejemplo, el programa para obtener la señal S0 puede ser:

Paso 0: leer y almacenar la señal M en un elemento de memoria
Paso 1: obtener la negación del elemento de memoria
Paso 2: leer y almacenar la señal A en otro elemento de memoria
Paso 3: procesar ambos elementos de memoria por un circuito Y
Paso 4: entregar el resultado al motor
Paso 5: volver al Paso 0

Se puede ahora imaginar que un procesador debe tener lo siguiente:

- una memoria capaz de contener un programa
- un elemento secuenciador capaz de seguir los pasos de programa
- dispositivos de entrada capaces de leer datos
- elementos auxiliares de memoria
- circuitos procesadores como los Y, O y NO
- dispositivos de salida capaces de entregar los resultados del proceso

Además, no solo es posible procesar señales tal como las hemos mencionado; es posible también procesar números. Para recordarlo, los números se pueden representar en binario como una serie de dígitos binarios (denominados 'bits')
con valor de posición. Recordando la codificación de los 8 primeros números:

Decimal	Binario
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

El cambio de base se logra mediante una serie de divisiones sucesivas entre 2.
Con los residuos se forma la representación binaria.
Por ejemplo, para la representación de 6:
6 dividido entre 2 es 3, residuo 0
3 dividido entre 2 es 1, residuo 1
1 dividido entre 2 es 0, residuo 1
Por tanto, la representación de 6 es 110.
Se puede verificar que:
6 = 4 + 2 + 0 = 1 * 2^2 + 1 * 2^1 + 0 * 2^0
Para identificar los bits se acostumbran las siguientes a 101:

1 0 1
bit '2' bit '1' bit '0'
bit 'mas alto' bit 'mas bajo'
bit 'mas significativo' bit 'menos significativo'

Un 'byte' es una serie de 8 bits. Un 'nible' es una serie de 4 bits. Un byte tiene dos nibles; el mas significativo (o mas alto) y el menos significativo (o mas bajo), siguiendo el convenio de los bits.
También, se acostumbra usar hexadecimal para acortar la notación del binario.
Por ejemplo, 127 tiene la representación 0111111 en binario, que se simplifica a 7F en hexadecimal. La técnica de conversión de binario a hexadecimal consiste en convertir el nible mas alto a hexadecimal y luego el mas bajo. En este ejemplo:

nible mas alto nible mas bajo
1111

	nible mas alto	nible mas bajo
en decimal:	7	15
en hexadecimal:	7	F

Cada bit puede tratarse como una señal lógica. El clásico ejemplo es el circuito 'sumador'; de dos bits. La tabla de la suma es:

A	B	A+B Byte Alto	Byte Bajo	Explicación
0	0	0	0	Representación de 0
0	1	0	1	Representación de 1
1	0	0	1	
1	1	1	0	Representación de 2

Las expresiones lógicas para cada uno de los bits son :

bit alto : A y B

bit bajo : A o B y no (A y B)

Se vera mas adelante que esta no es la única forma de representar números, y que la notación binaria de números puede emplearse para otros fines, por ejemplo para representar caracteres. La codificación mas usada es la ASCII (American Standard Code for Interchange Information) que asigna caracteres a números. Algunos casos concretos son:

Números	Caracteres
0-47	Varios, incluido espacio en blanco
48	0 (números)
49	1
50-63	2-9 y otros caracteres
64	A letras mayúsculas
65	B
66-96	C- Z y otros símbolos
97	a letras minúsculas
98	b
99-255	c- z y otros símbolos

Retomando la construcción de un procesador, sus partes, llamadas 'dispositivos', son:

- 'Unidad Aritmética - Lógica', dotada con los usuales Y,O y NO y también circuitos mas complejos, como sumadores que pueden operar sobre solo un par de bits u otros mas complejos que pueden operar sobre bytes.

- 'Registros', memorias que pueden contener resultados intermedios para entregar a los dispositivos procesadores o recibir de ellos.
  Normalmente tienen 8, 16 o 32 bits.

- 'Memoria', capaz de almacenar diversos algoritmos para diferentes procesos; debe estar compuesta de 'celdas' donde cada una puede ser manipulada individualmente para entregar o recibir su contenido.

- 'Buses' de comunicación entre los dispositivos. Son tres:

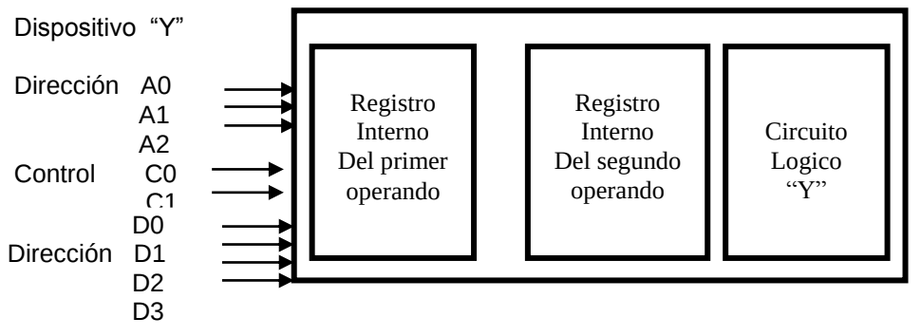
  - 'Bus de dirección': cada dispositivo recibe un número, denominado 'dirección' y en este bus se indica que dispositivo se quiere utilizar.
  - 'Bus de Control' : las señales que indican que tarea debe realizar el dispositivo seleccionado.
  - 'Bus de Datos' : en el fluirán los datos para dispositivos y sus resultados

Los buses son 'hilos' que se conectan a los dispositivos y son tratados por circuitos lógicos. Por ejemplo, un dispositivo con un circuito 'Y' en la Unidad Aritmética - Lógica podría tener las siguientes especificaciones:

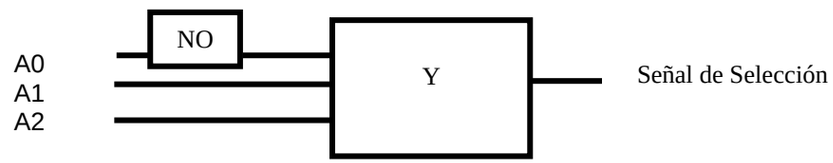
Dirección : número 6 (110) en tres bits ; A2,A1 y A0

Control: en 2 bits ; C1 y C0
01 - leer del bus al registro interno del primer operando
10 - leer del bus al registro interno del segundo operando
11 - escribir al bus el resultado 'Y lógico' de ambos operandos

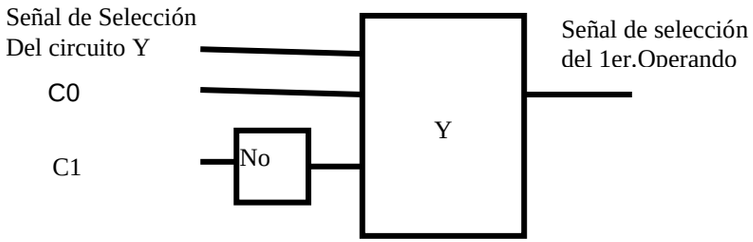
Datos: en 4 bits D3, D2, D1 y D0



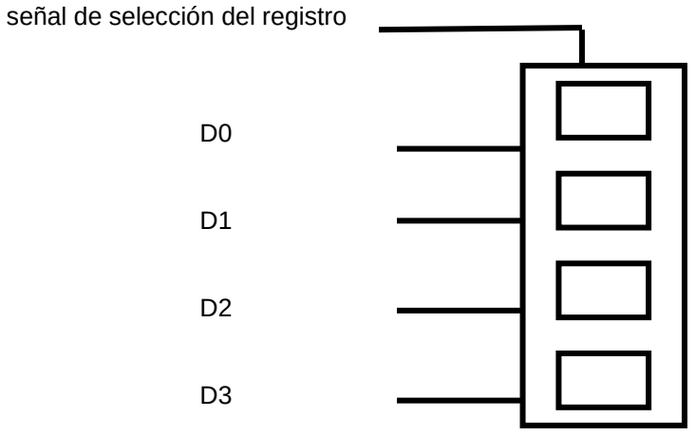
debe actuar de tal manera cuando se presente la dirección 6 (en binario 110) en el bus de direcciones, exista una acción. Una forma simple de lograrlo es colocar un circuito 'A2 y A1 y no A0'. La salida de este circuito será 1 solo cuando A2=1, A1=1 y A0=0



El registro interno del 1er. operando debe almacenar información solo cuando las líneas de control tienen el valor 01. Su circuito selector será:



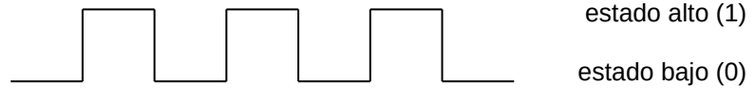
Los registros internos como dispositivos de almacenamiento estarán compuestos por cuatro elementos capaces de retener una señal solo
cuando se los seleccione, como en el siguiente esquema:



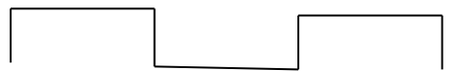
La cantidad de bits de cada bus se denomina 'ancho de bus'. El ancho del bus de datos es muy importante por que indica cuantos bits se pueden enviar y procesar simultáneamente en los dispositivos.
Si un procesador tiene ancho de bus de datos de 8 bits, para procesar 16 bits necesitara dos operaciones. En cambio, si tiene ancho de 16 bits, lo podrá hacer en una sola operación.
El ancho del bus de direcciones es también muy importante por que indica cuantas localidades de memoria están disponibles para acceder a ellas. Un bus de 16 bits solo accede a algo mas de 64000 posiciones de memoria, mientras que con 32 bits se accede a mas de 4 mil millones de posiciones.

- 'Unidad de control', que toma programas de la Memoria y comanda a la Unidad Aritmética - Lógica y los Registros para ejecutarlos.

El procesamiento secuencial de un programa se basa en una señal oscilante de estados "altos" y "bajos" llamada "frecuencia de reloj" ('clock') que provee alternativamente niveles 0 y 1.

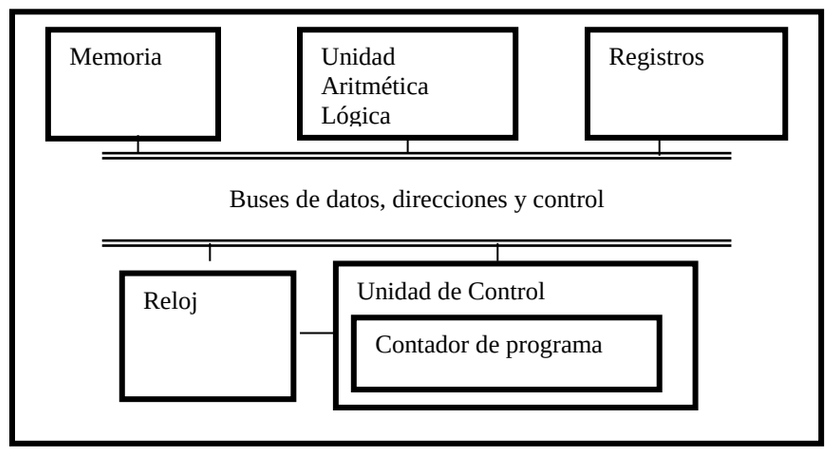


El tiempo en que se termina un estado bajo y luego otro alto, se denomina "ciclo de reloj". Un ciclo se llama también "hertz", en honor del físico alemán Otto Hertz.
Esta señal esta conectada a un sumador que incrementa en 1 un registro. Los valores cambiantes en este registro aunque repetidos en el tiempo ocasionan que se alternen "ciclos de búsqueda", donde el procesador toma su siguiente instrucción de la memoria y "ciclos de ejecución", donde se encarga de cumplirla.
Un registro especial 'Program Counter' (PC) se mantiene la dirección de la próxima instrucción.
En realidad los ciclos pueden tomar varios ciclos de reloj; a continuación por ejemplo, cada uno toma dos ciclos de reloj.



Búsqueda ejecución búsqueda
Algunas instrucciones complejas pueden tomar varios ciclos de búsqueda y ejecución cuando por ejemplo necesitan tomar varios elementos de la memoria para completarse, deben buscarlos usando ciclos de ejecución.
Otras, pueden requerir mas tiempo para ejecutarse tomado varios ciclos de ejecución.

La frecuencia de reloj es el dato que generalmente se indica como característica del procesador; 300 Mega hertz (Mhz) por ejemplo significa que 300 millones de veces por segundo se alternan estados altos y bajos. El mecanismo encargado de la secuencia y que también realiza algunas otras tareas como verificar que exista un dispositivo externo solicitando una interrupción y la atienda se denomina 'master control'. Esta implementado en electrónica pura y tiene forma de un ciclo.
De manera esquemática, el procesador con todas sus partes tiene la siguiente conformación, llamada también 'micro-arquitectura':



Por ejemplo, se podría construir un procesador con el anterior dispositivo 'Y' y tres registros Ra, Rb y Rc con las siguientes especificaciones:

Direcciones: 3 para Ra
5 para Rb
6 para Rc

Control : 1 escribir del bus al registro
0 leer del registro al bus (validas para ambos registros Ra y Rb)

Datos : 4 bits

Los programas en memoria son los contenidos que deben depositarse en los buses para ejecutar un algoritmo. Siguiendo el ejemplo anterior, para obtener 'Ra y Rb' en Ra, se escribiría el siguiente algoritmo:

	Bus de direcciones			Bus de control	
	A2	A1	A0	C1	C 0
L0.- Leer de Ra	0	1	1	0	0
L1.- Escribir primer operando en 'Y'	1	1	0	0	1
L2.- Leer de Rb	1	0	1	0	0
L3.- Escribir segundo operando en 'Y'	1	1	0	1	0
L4.- Leer de resultado de 'Y'	1	1	0	1	1
L5.- Escribir en Ra	0	1	1	0	1

En hexadecimal, el programa seria:

0C
19
14
1A
1B
0D

Al lenguaje de programación que resulta de este manejo de señales en los buses se denomina 'micro-código' y la memoria donde reside se llama 'micro-memoria'.

I.2.- ARQUITECTURA DE COMPUTADOR Y LENGUAJE DE MAQUINA

Los diseñadores de procesadores avanzaron un paso mas; observaron que muchas operaciones eran tan frecuentes (como la anterior, el uso del circuito Y por ejemplo), que era conveniente crear un nivel superior donde las operaciones frecuentes se codificaran y estuvieran disponibles con la invocación del código. Por ejemplo, suponiendo que se tienen los circuitos Y y O y los tres registros Ra, Rb y Rc, se podría disponer la siguiente codificación de operaciones binarias:

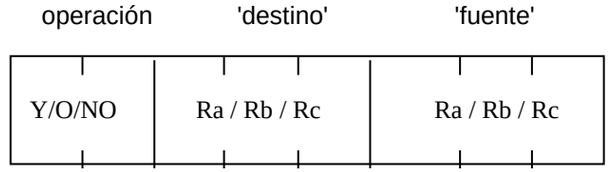
Operaciones: Y = 01
O = 10

Operandos : Ra = 001 De los dos operandos que participan en la operación
Rb = 010 , se llamara 'destino' al operando que se altera
Rc = 100 con la operación y 'fuente' al otro operando.

La operación

'destino' = 'destino' operación 'fuente'

se codificará:



Por ejemplo: Ra = Ra Y Rb se codificaría 01001010 en binario y 4A en hexadecimal

A todas las convenciones que se establezcan para este nivel se denominan 'lenguaje de maquina'.

Para su implementación se requiere:

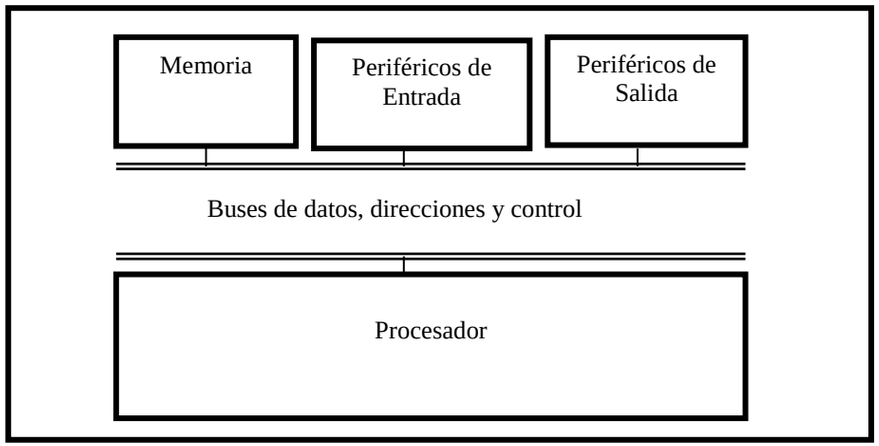
- una memoria que almacene códigos de instrucciones y resida fuera del procesador. En contraposición, la memoria dentro del procesador se denomina 'micro-memoria'. Esta memoria también podría usarse para almacenar datos y no solo código de programación. Se usa la simbología '[nnnn]' para indicar el contenido de la dirección de memoria nnnn.

- buses que conecten esta memoria con el procesador. También serán necesarias señales de datos, dirección y control.

- un contador de programa que recorra la memoria. En contraposición, el contador dentro del procesador se denomina 'micro - contador de programa'. Pasa a ser uno mas de los registros del procesador.

- dentro de la unidad de control del procesador será necesario incluir una 'Unidad de decodificación' que analice cada instrucción en lenguaje de maquina y asigne al micro-contador de programa la posición dentro de la micro-memoria en que empieza el programa en micro-código correspondiente a la instrucción en lenguaje de maquina.

La arquitectura general de un computador es:



En contraposición, la arquitectura dentro del procesador se denomina 'micro-arquitectura' y su operación es muy similar;

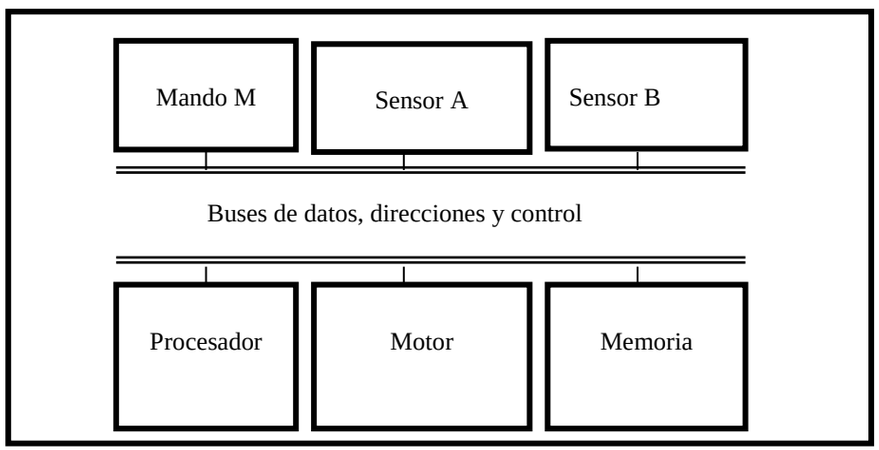
- cada periférico y posición de memoria tiene una dirección y también acepta comandos de control para enviar o recibir datos por el bus cuando es seleccionada.

- la Unidad de Control del procesador tiene un contador de programa que indica la dirección de memoria con la próxima instrucción a ejecutar.

- la próxima instrucción a ejecutar es llevada a la Unidad de Decodificación que arranca un micro programa para ejecutarla.

- las micro instrucciones usaran los buses de la micro - arquitectura o llegaran a usar también los buses de la arquitectura para manejar la memoria y los periféricos.

Para construir un dispositivo capaz de realizar el algoritmo P0 a P6, se necesitaría la siguiente arquitectura:



Históricamente, las 'mini-computadoras' de los años 70 y anteriores tenían toda una tarjeta destinada al procesador. Las 'micro computadoras' de fines de esa década hasta ahora, tienen un 'micro - procesador', donde toda una tarjeta de procesador se ha convertido en un solo circuito VLSI (Very Large Scale of Integration). A fines de los años 80 se denomino a toda esta arquitectura CISC (Complex Instruction Set Code) y se propuso la arquitectura RISC (Reduced
Instruction Set Code) que elimina el proceso de decodificacion, la micro-memoria y requiere que la memoria contenga los micro-programas. Se considera que en 1995 esta arquitectura fracaso frente a los procesadores INTEL. El propio diseñador de uno de los primeros procesadores RISC (SPARC para SUN) ha pasado a diseñar una nueva arquitectura basada ya también en instrucciones muy largas (64 o 128 bits) y la aplica en procesadores que emulan los procesadores INTEL mediante un sistema patentado como 'code-morphing'. Busca aplicarlo a computadoras portátiles.

Ejercicios (Grupo1)

1.- Que condiciones deben asegurarse al encendido de un procesador respecto a su programa control maestro y su micro-memoria ?

2.- Que problemas surgen y como se resolverían en un ambiente de proceso paralelo con mas de un procesador operando simultáneamente en un solo computador ?

3.- Como se podría lograr que un procesador atienda interrupciones enviadas como señales desde un dispositivo ?

4.- Diseñar un circuito que active cuatro bujías en el orden 1-4-2-3 a partir de un reloj y un circuito contador cíclico de 0 a 3.

5.- Que elementos se deberán añadir a un procesador si se quiere aumentar una memoria 'cache' de alta velocidad pero reducido tamaño para mantener copias de la memoria común y asi evitara acceder a ella ?

6.- Que elementos se deberán añadir a un procesador si se quiere aumentar una lista con las próximas instrucciones a ejecutar de tal manera que se puedan ir decodificando mientras otra instrucción se ejecuta ?

7.- Esbozar un micro-programas de multiplicación de bytes por corrimientos para una maquina con 4 registros de 2 words divisibles en bytes, un circuito de corrimientos y un sumador.

8.- Que elementos se deberán añadir a un procesador si se quiere simular una cantidad inmensa de memoria usando un disco ?

I.3.- SISTEMAS OPERATIVOS

De la misma manera en que el ensamblador surgio como una simplificacion y uniformación de la programación basada en micro-programas reemplazando micro-programas por instrucciones de ensamblador, la agrupación de programas en
ensamblador muy usuales como 'servicios', también simplifica y uniforma la programación en ensamblador y en general. Por ejemplo, leer caracteres del teclado es un servicio que deberá existir como rutina en todo programa que pida caracteres y es deseable que se implemente una sola vez.
Al conjunto de estos 'servicios' se denomina 'Sistema Operativo'.
Los servicios mas usuales de un sistema operativo son:

- Al encendido del equipo, activar debidamente los periféricos y la memoria.

- Cargar, ejecutar y terminar programas invocados por el usuario.

- Operaciones usuales con periféricos de entrada y salida, como leer un carácter de teclado o imprimir un carácter por impresora.

Un sistema operativo es también un administrador de los recursos de un computador, a saber:

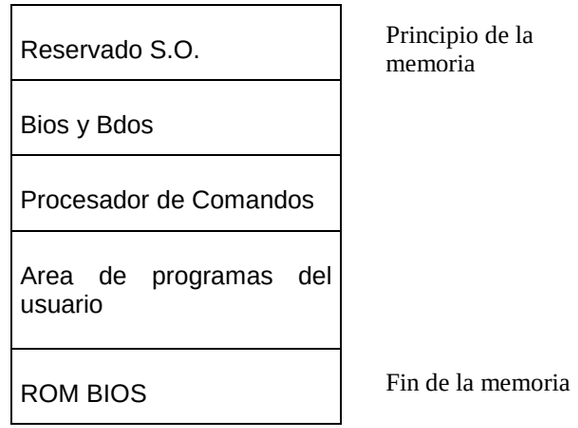
- Memoria

- Periféricos de entrada y salida

- Información de periféricos de almacenamiento

- Procesadores

A partir de ahora nos concentraremos en el ambiente usual de los procesadores Intel y los sistemas operativos DOS y Windows. Por construcción, al encenderse un procesador, su contador de programa apunta siempre a la dirección FFFF0h.
Los computadores deben tener en esa dirección el llamado ROM BIOS, con el programa iniciador SETUP.
El programa iniciador se basa en tablas que indican la dotación básica del computador; cuanta memoria tiene, que periféricos están conectados y algunas características que no pueden ser extraídas de los propios periféricos.
Luego de activar los periféricos, informando si han existido errores, el programa busca en los discos, en cierto orden definido en el ROM BIOS, una localización bien definida. Allí espera encontrar una 'firma' de Sistema Operativo que indique cual es el contenido del disco. En los discos duros, el elemento típico es una 'tabla de partición' que indica como esta dividido el disco. Usualmente existe una única partición (división), cuando todo el disco corresponde a C:. Pero alternativamente puede existir mas de una partición; por ejemplo cuando existen F: y G: en un solo disco. También pueden existir dos sistemas operativos en el mismo disco; por ejemplo UNIX y DOS. En la tabla de partición, se indica que es lo que contiene cada partición y que parte del disco ocupa.
Si ninguna partición tiene sistema operativo, busca en otro disco o diskette.
En los diskettes no existe tabla de partición, sino solamente un 'registro de arranque' (boot record), que también existe en los discos.
Si existe mas de un sistema operativo, el ROM BIOS pregunta cual se quiere utilizar y en función de la respuesta accede a una partición.
En la partición existe el 'registro de arranque' que informa de las características del sistema operativo y también un programa el arranque del sistema operativo. El ROM BIOS copia a memoria el programa de arranque y pasa a
ejecutarlo.
Este programa busca y copia a memoria el programa IO.SYS, que contiene los servicios básicos de entrada y salida y además, el programa SYSINIT, que al ser ejecutado busca el archivo CONFIG.SYS para configurarse, carga el
programa MSDOS.SYS y por ultimo COMMAND.COM que ejecuta AUTOEXEC.BAT si existe.
El proceso de arranque culmina con una organización de la memoria con el siguiente aspecto:



Un área 'reservada' es como cualquier otra pero no debe ser usada por ningún otro programa ya que su información es imprescindible para el sistema operativo, por ejemplo, en que modo (texto/gráfico) se esta usando la pantalla.

BIOS (Basic Input Output System) esta en IO.SYS y tiene los servicios primitivos para las operaciones de entrada y salida; por ejemplo, como leer un sector de un disco. Se asocia con el 'nivel físico'; eso es, 'tal cual es'. Es dependiente de la electrónica, lo que se denomina 'dependiente del hardware'.

BDOS (Basic Disk Operating System) esta en MSDOS.SYS y tiene servicios a 'nivel lógico'; eso es, 'tal cual se ve', por ejemplo abrir un archivo de disco. Lo que realmente existe en el disco son pistas y sectores y ciertos manejos hacen
que existan archivos y directorios.

Existen medios bien definidos para acceder a los servicios de BIOS y BDOS de modo que esta disponibles mediante instrucciones de bajo nivel.

El procesador de comandos COMMAND o WIN es la interfaz con el usuario. Básicamente consiste en un medio para indicar los programas que se quieren ejecutar.
Algunos programas muy usuales como DIR (ver el contenido de un directorio) son 'internos', ya que residen dentro del propio COMMAND y los demás, son externos, que deben ser buscados en un disco y llevados a memoria para su ejecución.
Algunos programa provistos con el propio sistema operativo son denominados 'utilitarios' y generalmente están relacionados con aspectos algo internos del sistema operativo, como SCANDISK que explora un disco corrigiendo los errores debidos a un corte intempestivo de energía.

Los programas para el procesador de comandos tienen formatos bien definidos; existen básicamente dos que se diferencian con la extensión del nombre:

.COM : imagen de memoria en disco; debe empezar en un lugar preciso (100H) y no puede tener tamaño mayor a 65536 bytes.

.EXE : formato 'relocalizable' sin ninguna de la restricciones del formato .COM pero empieza con una 'tabla de relocalización' que indica ciertos puntos del programa que deben ser ajustados para la ejecución. Existe una variante algo diferente para WINDOWS que ejecutada en DOS se cancela inmediatamente.

Históricamente, DOS apareció en 1982 con la primera PC con el procesador 8088 y en esa época existía una versión propia de IBM, IBM-DOS y otra para equipos compatibles denominada Microsoft-DOS. El antecesor de DOS fue CP/M (Control Program for Microcomputers) desarrollado por Digital Corp. para los procesadores 8080, la generación inmediatamente anterior a la serie 80x86. La relación es tan directa que gran parte de un manual de CP/M es directamente aplicable a DOS. Esto se hizo con la intención que los programas para CP/M pudieran fácilmente convertirse en programas para DOS.

I.4.- ENSAMBLADORES Y DEPURADORES

Un paso mas a la simplificacion y uniformizacion de la programación fue la aparición de 'ensambladores'. Usualmente, con un nuevo sistema operativo aparecen juntamente un 'editor' y un 'ensamblador' basados en el.
Si bien la programación en lenguaje de maquina se puede lograr con algún mecanismo de acceso a la memoria para depositar allí contenidos que representen instrucciones de lenguaje de maquina, es deseable tener mejores medios para
hacerla mas cómoda y eficientemente.
La mejor forma es disponer de un 'editor de texto' donde se pueda escribir con múltiples facilidades de corrección un programa en forma muy entendible y luego llevarlo a lenguaje de maquina. Para el ejemplo de la instrucción Y se podría
convenir que la expresión

AND Ra , Rb

se convierta en la instrucción que hace Ra = Ra Y Rb, que se vio es

4A hexadecimal.

A estos convenios como 'AND' se los denomina 'mnemotecnicos' (mnemonics).

Un 'editor de texto' es un programa que lee caracteres, los despliega según el código ASCII y los almacena tal cual en disco. Esta es la forma mas sencilla de almacenar información, contrapuesta con otras que guardan efectos especiales de
impresión y despliegue. Lo único inusual en un archivo de texto son los separadores de líneas horizontales, que pueden ser uno o los dos caracteres 0D y 0A en hexadecimal, 13 y 10 en decimal.
El 'ensamblador' es el programa que toma el texto y lo convierte en un programa ejecutable en lenguaje de maquina según algún formato del sistema operativo.
Las técnicas que se emplean para construir ensambladores están basadas en la teoría de autómatas, maquinas teóricas con los mínimos recursos necesarios para realizar cómputos.
Por realizar la construcción de un programa es que los ensambladores recibieron su nombre y el lenguaje que reciben se denomina 'lenguaje ensamblador' o 'ensamblador'.
Para crear un lenguaje ensamblador, y en general cualquier lenguaje se deben definir 4 elementos:

- los símbolos del lenguaje; usualmente son:

  - Palabras y símbolos reservados propios del lenguaje, como 'AND', 'Ra', la coma , el espacio en blanco, 'Rb' y los delimitadores de línea en el ejemplo anterior.

  - Símbolos del usuario, como variables y nombre de rutinas.

  - Constantes, como 3, 'No' que resultan imprescindibles.

Se denominan 'léxico' del lenguaje.

- Las formas que deben seguir las expresiones en el lenguaje; por ejemplo para el anterior se puede convenir:

AND <nombre de registro>,<nombre de registro o constante>

Se denominan 'sintaxis' del lenguaje.

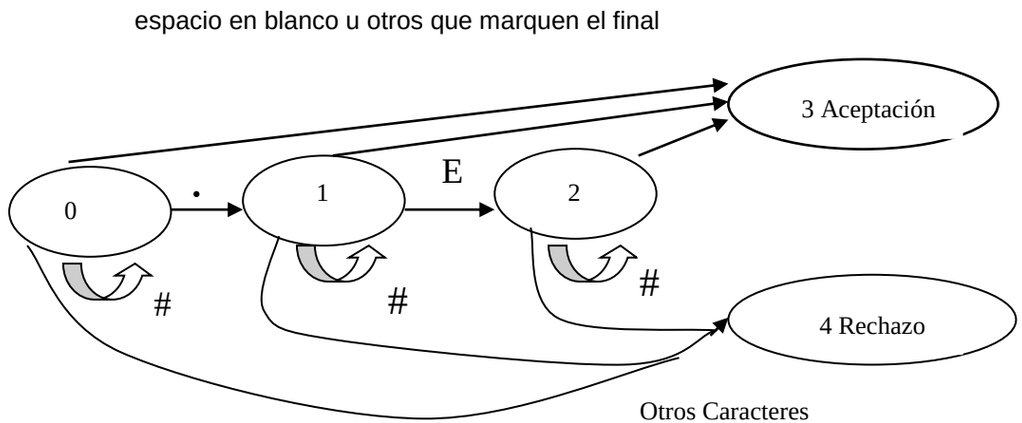
- El sentido de las expresiones del lenguaje, por ejemplo

AND Ra,5.3

sigue la sintaxis correcta pero la constante debe ser un número entero Se denomina 'semántica' del lenguaje.

- La traducción, que se denomina 'generación de código'. En el ejemplo anterior, como formar el código de maquina según los campos de bits, que se vieron antes.

El léxico de los lenguajes se representa muy bien con los denominados 'autómatas finitos' que se asemejan a grafos orientados donde los nodos son 'estados' del proceso de identificación de un símbolo del lenguaje y los arcos corresponden a los caracteres encontrados. Por ejemplo, para identificar un número con la notación ####.###E### se podría construir el siguiente:



Los símbolos se representan también por las llamadas 'gramáticas regulares' El gráfico se interpreta así:

Al aparecer un número, se ingresa al Nodo 0
Mientras sigan existiendo números '#', se mantiene en el Nodo 0
El punto '.' lleva al Nodo 1
Los caracteres terminadores como el espacio en blanco terminan el reconocimiento positivamente. Otros, negativamente.

En el Nodo 1
Mientras sigan existiendo números '#', se mantiene en el Nodo 2
La letra E 'E' lleva al Nodo 2
Los caracteres terminadores como el espacio en blanco terminan el reconocimiento positivamente. Otros, negativamente.

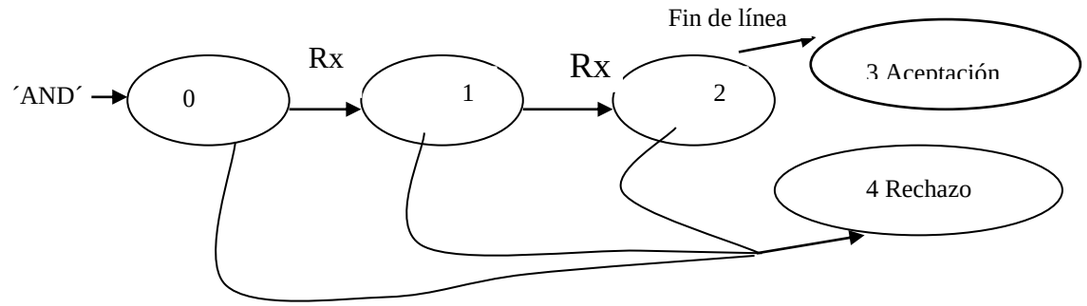
En el Nodo 2
Mientras sigan existiendo números '#', se mantiene en el Nodo 2
Los caracteres terminadores como el espacio en blanco terminan el reconocimiento positivamente. Otros, negativamente.

Su implementación mas sencilla se puede lograr con una estructura CASE y algunos CASE anidados en lenguajes de alto nivel:
Por ejemplo, para el anterior

.. se asume que se ha leído un carácter en C que resulta ser un número

```
N=0
X=C
MIENTRAS n<>3 y n<>4
  LEER un carácter en C
  CASOS de N
   0: CASOS de C
     '0'..'9' : N=0, X=X+C
       '.'   : N=1, X=X+C
       ' '   : N=3
     otros : N=4
     FIN-CASOS

   1: CASOS de C
     '0'..'9' : N=1, X=X+C
       'E'   : N=2, X=X+C
       ' '   : N=3
     otros    : N=4
     FIN-CASOS

   2: CASOS de C
     '0'..'9': N=2, X=X+C
       ' '   : N=3
     otros : N=4
     FIN-CASOS

  FIN-MIENTRAS
```

La sintaxis de un ensamblador es relativamente sencilla y también puede implementarse con autómatas finitos, por ejemplo para la instrucción AND se puede tener el siguiente



Se interpreta así:

Se llama al analizador lexicográfico para que entregue un símb Otros Símbolos

Al aparecer el símbolo 'AND', se ingresa al Nodo 0
Allí se llama al analizador lexicográfico para leer otro símbolo y si es un registro Rx, se pasa al Nodo 1. Otro símbolo lleva al rechazo.

En el Nodo 1
Se llama al analizador lexicográfico para leer otro símbolo y si es un registro Rx, se pasa al Nodo 2. Otro símbolo lleva al rechazo.

En el Nodo 2
Se llama al analizador lexicográfico para leer otro símbolo y si es fin de línea, se termina. Otro símbolo lleva al rechazo.

Su implementación es simplemente lineal:

```
Leer símbolo.
Si es 'AND'
  Leer símbolo
  Si es un registro Rx
    Leer símbolo
    Si es un registro Rx
      Leer símbolo
      Si es fin de línea
        Generar código
      Si no
        Error
      Fin Si
    Si no
      Error
    Fin Si
  Si no
    Error
  Fin Si
Fin Si
```

Un 'depurador' es un programa capaz de mostrar la ejecución de un programa en ensamblador tanto en instrucciones como en datos. Muestran los registros del procesador, la próxima instrucción a ejecutar, las posiciones de memoria y
otras.

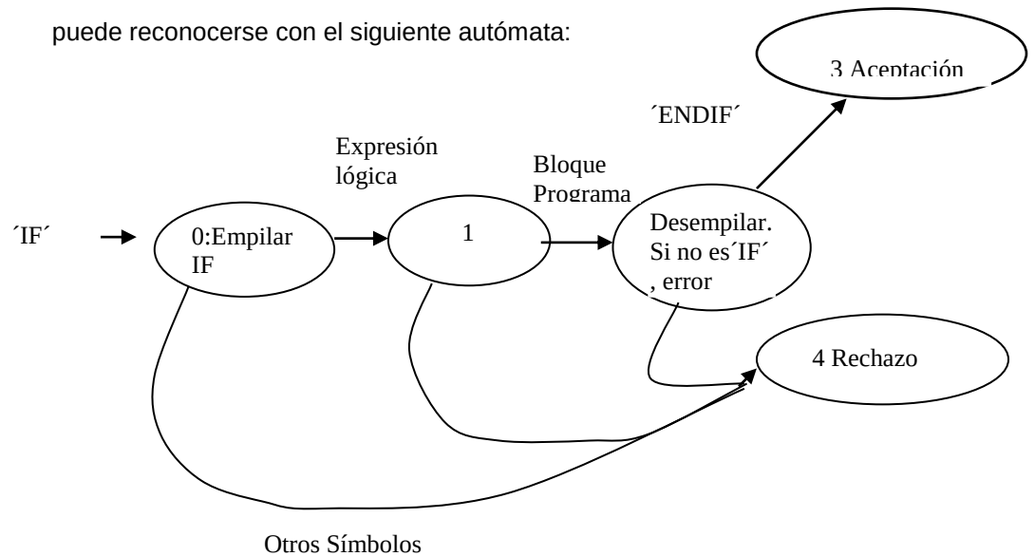
I.5.- LENGUAJES DE PROGRAMACION Y ESTRUCTURAS DE DATOS

Otro paso mas a la simplificacion y uniformicen de la programación es la definición y construcción de 'lenguajes de alto nivel'.
En contraposición al ensamblador, no están limitados a la traducción unívoca de una expresión a una instrucción en código de maquina, si no mas bien, sus expresiones pueden implicar grandes programas en código de maquina que
inclusive accedan a servicios del sistema operativo. Por ejemplo, una instrucción podría ser 'leer una línea de un archivo de texto'; ello implica un programa que use los servicios del sistema operativo para leer caracteres hasta
encontrar la marca de fin de línea.

En la sintaxis se incluyen estructuras de programación mas poderosas como IF y WHILE que pueden inclusive anidarse; esto es, usar un IF dentro de otro.
Estas estructuras se reconocen con los 'autómatas de pila', similares a los autómatas finitos pero con una pila a la que se llevan los símbolos encontrados
Por ejemplo, el reconocimiento de una sentencia IF que tenga la forma :

IF expresión bloque-programa ENDIF

puede reconocerse con el siguiente autómata:



Dentro del bloque-programa puede existir otra vez 'IF'

En la generación de código existen algunas alternativas:

- la generación de código de maquina directamente, lo que se llama 'ejecución compilada'.
Una variante muy atractiva es el uso de un 'entorno de ejecución', un programa que queda en memoria (se llama ‘residente’), con un conjunto de servicios. El código generado supone que este entorno esta disponible y solo realiza llamadas a el. En la 'ejecución compilada' convencional cada uno de los servicios que se requieren se copia dentro del programa. Ello les da autonomía ya que no dependen del entorno de ejecución y se denominan también 'stand - alone'.
- la generación de un 'código intermedio', que necesita de un programa ejecutor llamado 'interprete'. La mejor característica de esta modalidad es la flexibilidad que permite usar los recursos del interprete mientras el programa se ejecuta; por ejemplo se puede permitir que el usuario introduzca una expresión matemática y se evalúe. En un interprete esto es tan simple como disponer de la expresión mientras el programa se traduce; simplemente se entrega al interprete la expresión matemática.
En la modalidad compilada normalmente el programa debe contener todos los medios para aceptar cualquier expresión. En el grado máximo, cada programa acaba conteniendo una copia casi completa del compilador.

Se puede ver que la evolución ha tenido como constante la introducción de símbolos, como una instrucción de alto nivel (símbolo) que se reemplaza por un programa en bajo nivel. Otros símbolos muy importantes son las estructuras de datos, donde un nombre se asocia con un dato con ciertas características.
Por ejemplo, de un nombre asociado a una cadena de caracteres, puede extraerse una parte que sigue siendo cadena de caracteres. No así de un número, del que en caso extraerse una parte, puede no obtenerse un número similar, al perderse el signo, por ejemplo. Los tipos de datos mas conocidos son:

- Números
- Cadenas de caracteres
- Fechas
- Horas

Cada uno de ellos tiene un modo especifico de almacenamiento en memoria.

I.5.1.- Representación de números

La forma mas natural para representar números es naturalmente el binario; asignar a cada bit el sentido de un dígito binario con valor de posición de tal manera que un byte con sus ocho bits corresponda a un único número.
Por combinatoria elemental, los 8 bits, cada uno independiente de los demás, con dos posibles estados (0 y 1) dan una cantidad de combinaciones posibles de 2 elevado a la octava potencia, que es 256; desde 00000000 a 11111111.
En decimal, este es el rango 0 a 255.
Como la representación binaria es larga, una gran cantidad de ceros y unos, se trata generalmente de usar hexadecimal (hexa para abreviar); la representación de números en base 16. Cada dígito hexa esta en el rango de cero a quince (0 a F) y agrupa 4 dígitos binarios (un nibble) según la siguiente tabla:

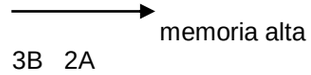
Binario	Hexa
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

La conversión de un byte a hexa sigue la sencilla regla de convertir independientemente los 2 nibbles que componen el byte. Por ejemplo, para convertir 01101110 a hexa:

1.- Identificar los nibbles superior e inferior: 0110 y 1110
2.- convertirlos a hexa: 6 y E
3.- formar el número convertido: 6E

Para representar rangos mayores de números, se deben usa varios bytes creando estructuras multibyte; por ejemplo, con dos bytes, se representan números hexa en el rango 0000 a FFFF (0 a 65535 en decimal). A veces se usara la letra "h" (mayúscula o minúscula) para como sufijo para denotar que se trata de números hexa.

La forma en que se almacenan en memoria es "invertida"; los bytes mas altos se almacenan en la memoria alta; por ejemplo un número hexa como 2A3B se almacenaría como:



Si bien esta forma es aparentemente extraña, es la mas natural para el proceso de números; nuestra notación, al escribir de derecha a izquierda, no lleva a contradecirnos al realizar

procesos, por ejemplo sumas, que deben realizarse de derecha a izquierda. Bajo el esquema "invertido" , los números se escriben de derecha a izquierda y también se suman en el mismo sentido.

La aritmética que rige con solo un byte es muy similar a la aritmética de números naturales, pero en realidad es la aritmética de las "clases residuales modulo 256" (la estructura básica de grupo cíclico con 256 elementos). Esto se hace evidente cuando se considera la finitud de la representación; solo 8 bits y se trata de desbordarla, por ejemplo; cuando
se suma FF y 01, el resultado 100 deja 00 en un byte y 1 como "acarreo" de la suma para otro byte, que no interesa mientras se ha definido el uso de un solo byte.

Por consideraciones algebraicas sobre las clases residuales, es posible definir simétricos aditivos (números negativos); en el caso anterior, FF tiene las mismas características de -1; sumado con 1 da 0.
Prácticamente, considerando que 256 tiene el mismo papel que 0, el número -x, dentro de la aritmética de clases, es el mismo que 256-x. Por ejemplo, -1 es 256-1 que es 255 o FF en hexa.
Un algoritmo de obtención de números negativos dice:

1.- cambiar 0 por 1 y viceversa
2.- sumar 1

La formula "256 - x" es también la demostración de ese algoritmo; puede escribirse como "(255-x) + 1"; considerando que 255 es 11111111 (solo dígitos 1 binario) restar cualquier número de el da por resultado cambiar 0 por 1 y 1 por 0.

x	1-x
0	1
1	0

Intercambio de 1 y 0 al restarse de 1

Una interpretación mas profunda de esta forma de representar números negativos puede obtenerse de la "libertad de elección de representantes" para la aritmética de clases. Los números negativos no son mas que otra
elección respecto a los positivos.

El rango que puede representarse en un byte llega a ser -128 a +127 (decimal). En una word, llega a ser -32768 a +32767. Es importante destacar que las reglas aritméticas para la manipulación de números con signo y sin signo son las mismas, y el signo es entonces solo interpretación del número. En particular, el bit mas alto en 1 indica la presencia del signo negativo.

Los números binarios se pueden utilizar en representación de cantidades enteras, pero son preferidos en el manejo de índices y punteros en arreglos o memoria dinámica.

También se pueden usar para manejar números fraccionarios estableciendo el convenio que el número entero que normalmente representa un número binario es un número con cierta cantidad de decimales multiplicado por una potencia de diez de tal manera que resulta en un entero. Por ejemplo, si la potencia de diez es 100, el número almacenado como 200 en binario, representara a 2.00
Dados los números 4.00 y 2.00, con sus respectivas representaciones 400 y 200 las operaciones aritméticas suma, resta, multiplicación y división se realizaran así:

Operación	Calculo		Ajuste	Resultado
4.00 + 2.00	400 + 200 =	600	Ninguno	6.00
4.00 - 2.00	400 – 200 =	200	Ninguno	2.00
4.00 * 2.00	400 * 200 =	80000	Dividir entre 100: 80000/100	8.00
4.00 / 2.00	400 / 200 =	2	Multiplicar por 100: 2*100	2.00

Alternativamente al binario, existe la representación BCD (binario codificado decimal); bajo este esquema, cada dígito decimal se representa en un nibble. Los números de 0 a 9 tienen idéntica representación que en binario, pero a partir de 10, se requieren dos nibbles; un nibble para el dígito 1 y otro para el 0. En hexa, 10 se representa como A.
Prácticamente, se trata los dígitos binarios como dígitos hexa.
Cuando se usan los dos nibbles de un byte, se denomina BCD empaquetado, y cuando solo se usa el nibble inferior, se denomina BCD desempaquetado.
Por ejemplo, el número 1234;

empaquetado : 34 12

desempaquetado: 04 03 02 01

(contenidos de la memoria en hexa)

El BCD se utiliza en la representación de cantidades de "punto fijo" como por ejemplo el dinero de las aplicaciones comerciales; siempre se maneja con dos dígitos decimales (entiéndase fraccionarios) ya que no se manejan
fracciones de centavos. Bajo este esquema, a una cantidad de dígitos BCD se les asigna una parte fraccionaria.
El BCD permite un tratamiento muy similar al de las operaciones manuales y mantiene la precisión acorde a la numeración decimal.

Otra representación es la de "punto flotante" utilizada mayormente en cálculos científicos y de ingeniería.
En este esquema, se considera la representación binaria de números fraccionarios, también como sumas de potencias negativas de 2. Así, 0.5 decimal es 0.1 binario.
La posición del punto decimal puede ser manipulada para conseguir los números m y c, llamados "mantisa" y "característica" respectivamente tales que

$$n = s * m * 2^{c} \quad y \quad 1 <= m < 2$$

si n es diferente de 0
s es 1 o -1

Se usara n = s * m E c c es entero con signo

Mientras n sea diferente de 0, su representación binaria tendrá un bit mas significativo de 1 y será de la forma 1xyzuv.rstq; que es igual a 1.xyzuvrstq por una potencia adecuada. A esta forma se llama "normalizada" de la mantisa y se almacena solo la parte fraccionaria. Se dice que se ha aplicado la "técnica del bit escondido".

Para el llamado "real corto", se crea una estructura multibyte con las siguientes partes:

```
          bits
31  30    23  22              0
s   cccccccc  mmmmmmmmmmmmmmmmmmmmmmm
```

s	= signo del número
cccccccc	= característica
mmmmmmmmmmmmmmmmmmmmmmm	= mantisa

El signo negativo se representa con 1 y el positivo con 0.
La característica es un número en el rango -126 a +127 y se representa como un byte sin signo al que se le ha sumado 127. -127 y +128 se usan para representar 0, números no normalizables en el rango e infinitos positivo, negativo,
indeterminado y no-números.
De la mantisa solo se guarda la parte fraccionaria, asumiéndose el 1 entero.
Los números muy pequeños como 2 E -130 no pueden ser normalizados y se representan como 0.0001 E -126. La característica en 0 indica esta situación.
Cuando la mantisa es 0 y la característica es 0, se esta representando el número 0. Existen +0 y -0 según el signo.

Este formato es usado por los coprocesadores matemáticos que consideran números de 4 bytes ("real corto"), 8 bytes ("real largo") y 10 bytes ("real temporal").
Lo mejor de esta representación es la posibilidad de manejar tanto números muy grandes como muy pequeños por el ajuste de la característica.

Ejemplo:
Representar 200.625 como real corto.

1.- Calcular las representaciones binarias de las parte entera y fraccionaria

200 = 11001000b y 0.625 = 0.101b

2.- Formar el número

200.625 = 11001000.101b

3.- Obtener la forma normalizada

200.625 = 11001000.101b = 1.1001000101b E 111b

4.- Obtener la característica de la representación sumando 127 (1111111)

111 + 1111111 = 10000110

5.- Obtener la mantisa en 23 bits añadiendo ceros a la derecha

1001000101 0000000000000

6.- Formar la representación

0 10000110 1001000101 0000000000000

7.- Formar bytes de 8 bits y expresar en hexa

```
01000011 01001000 10100000 00000000
 4  3    4  8    A   0    0   0
```

8.- Invertir el orden

00 A0 48 43

Ejercicios (Grupo 2)

1.- Indicar los bytes para almacenar el número decimal 31100 en binario, BCD empaquetado y desempaquetado.

2.- Dado el número hexadecimal 010305, que números decimales representa si es interpretado como binario, luego como BCD empaquetado y finalmente como desempaquetado.

3.- Como número de tres bytes FA1C00, que números decimales representa si es interpretado como número sin signo y luego con signo ?

4.- Dado el número n, cuantos bytes se necesitan para almacenarlos en binario, BCD empaquetado y desempaquetado ?

5.- Dados n bytes, que rango de números binarios sin signo, con signo, BCD desempaquetado y empaquetado se pueden representar con ellos ?

6.- Dada la representación de un número sin signo en n bytes, como puede obtenerse su representación en n+1 bytes ?

7.- Dada la representación de un número con signo en n bytes, como puede obtenerse su representación en n+1 bytes ?

8.- Como se almacenaría el número 1 como real corto?
00 18 80 44, que número representa ?
Resp. 00 00 80 3F y 1024.75

I.5.2.- Representación de cadenas de caracteres.

A fines del siglo pasado y principios del presente se creo la llamada codificación ASCII (iniciales de American Standard Code for Interchange Information). Es una relación biunivoca entre números (en el rango 0 a 255) y símbolos (letras, números, caracteres gráficos, letras griegas, símbolos matemáticos y otros).
Con el tiempo se modifico y existe un standard no único ya que los fabricantes de computadoras e impresoras tratan de darle a cada país sus caracteres propios; por ejemplo, a los países nórdicos se les dota de símbolos propios de su alfabeto y que no existen en otra lengua escrita.

La representación de cadenas de caracteres es la aplicación de esta tabla;
por ejemplo "JUAN" se almacenaría como:

4A	55	41	4E

(contenidos de la memoria en hexa)

Nótese que la representación ASCII de los números 0 a 9 es 30 a 39 (en hexa); cuando en el BCD desempaquetado se fuerza a que el nibble superior tenga el valor 3, la representación se denomina "número ASCII", por la coincidencia con las representación de los números en el ASCII. Por ejemplo 123 se almacenara como:

31	32	33

(contenidos de la memoria en hexa)

La forma en que la información es manejada por usuarios esta naturalmente en ASCII. Debe imaginarse entonces que para almacenar en binario, BCD empaquetado o desempaquetado, debe aplicarse un programa de conversión desde el ASCII hasta la estructura; por ejemplo, 123 se almacena en un solo byte como 7B.

Estas son cadenas de longitud fija, que siempre ocupan una cantidad conocida de caracteres. Se usan en registros de archivos almacenados en disco y se rellenan con espacios en blanco (20h) cuando los caracteres usados son menos
que los disponibles; por ejemplo si el espacio reservado para un nombre fuera de 6 caracteres, JUAN se almacenaría así:

4A	55	41	4E	20	20

Otras cadenas pueden ser de longitud variable, como las denominadas ASCIIZ, cadenas almacenadas usando ASCII terminadas con 0.
Por ejemplo JUAN en ASCIIZ se almacenaría como:

4A	55	41	4E	00

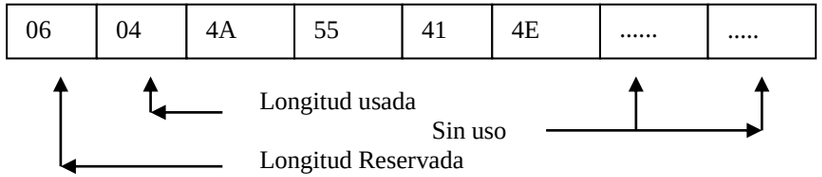
También puede incluirse el primer lugar la cantidad de caracteres:

04	4A	55	41	4E

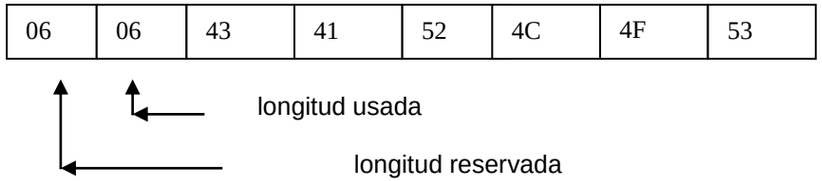
Este tipo de cadenas también se denominan delimitadas y son adecuadas para almacenar mensajes o constantes de caracteres.

Otras son las 'cadenas estructuradas' que permiten longitud variable dentro de un espacio fijo. Se añaden dos bytes mas, uno con la longitud fija usada y otro con la longitud variable usada. Por ejemplo si se reservan 6 posiciones,

JUAN se almacenaría como:

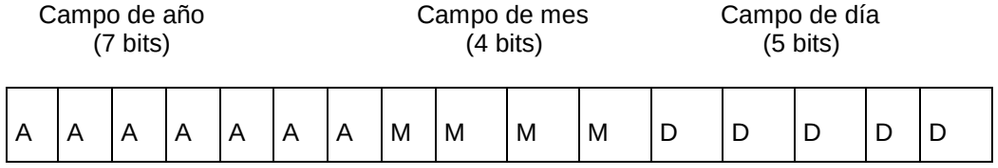


Esta estructura también podría contener CARLOS;

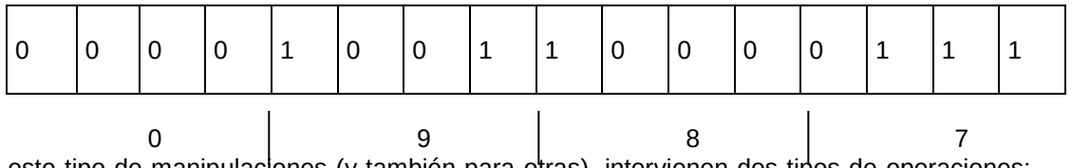


I.5.3.- Representaciones de fechas y horas

Para aprovechar adecuadamente la memoria, otras estructuras compuestas de partes definen "campos" en grupos de bytes. Por ejemplo, para representar fechas, compuestas de día, mes y año se usa el siguiente convenio sobre
dos bytes.



Los 7 bits altos se destinan al año; permiten representar un número entre 0 y 127; en DOS y WINDOWS se asocian a los años 1980 y 2107. Los 5 bits bajos se destinan al día; representan números entre 0 y 31. Los restantes 4 bits representan el mes; entre 0 y 15; suficiente para los doce meses. Por ejemplo, el 7/12/1984 se representa como:



En este tipo de manipulaciones (y también para otras), intervienen dos tipos de operaciones; lógicas y corrimientos.

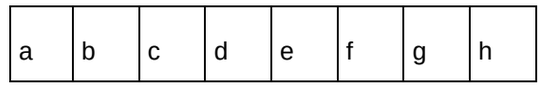
Las operaciones lógicas típicas AND, OR y NOT están bien definidos sobre bits, sobre bytes se realizan "bit a bit" de acuerdo a la posición relativa de cada bit. Por ejemplo 2A OR 33 se calcula de la siguiente manera:

```
2A = 0 0 1 0 1 0 1 0
33 = 0 0 1 1 0 0 1 1

     0 0 1 1 1 0 1 1 = 3B
```

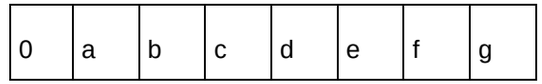
Dado X, 0 o 1, se tiene:

```
X OR 0 = X      ; X queda invariante con la operación
X OR 1 = 1      ; el resultado siempre es 1

X AND 0 = 0     ; el resultado siempre es 0
X AND 1 = X     ; X queda invariante
```

Los corrimientos, como su nombre lo indica, son el movimientos bits hacia la derecha o la izquierda. Por ejemplo, dado un byte



donde cada letra es un bit 0 o 1, un corrimiento a la derecha en un bit dará por resultado:



El bit mas bajo se pierde y el bit mas alto siempre es 0.

Un corrimiento a la izquierda en un bit dará por resultado:

El bit mas alto se pierde y el bit mas bajo siempre es 0.

Un corrimiento de n bits (a derecha o izquierda) será la reiteración de n veces el corrimiento de un bit (a derecha o izquierda).

Para el ejemplo anterior de la representación de fechas, dados año, mes y día en words, se podría aplicar el siguiente algoritmo para construir la word que representa toda la fecha:

1.- mes = corrimiento de mes a la izquierda 5 bits

2.- fecha = día OR mes
3.- año = corrimiento de año a la izquierda 9 bits
4.- fecha = fecha OR año

Para el caso anterior del 7/12/1964;

día = 0000000000000111
mes = 0000000000001100
año = 0000000001000000

1.- mes = 0000000110000000 ; el mes va a su posición correcta

2.- día = 0000000000000111 ; día y mes se sobreponen sin afectarse; cada uno, tiene ceros en
fecha = 0000000110000111 la parte que corresponde al otro.

3.- año = 1000000000000000

4.- fecha = 1000000110000111 = 8187 (en hexa)

Recíprocamente, dada una fecha en una word, se obtienen día, mes y año de la siguiente manera:

1.- día = fecha AND 0000000000011111

2.- mes = fecha AND 0000000111100000
mes = corrimiento de mes 5 bits a la derecha

3.- año = fecha AND 1111111000000000
año = corrimiento de año 9 bits a la derecha

En el caso del día, puede verse que la elección de 0 y 1 es tal que los 5 bits bajos, que son quienes representan el día quedan invariantes y los demás, quedan en 0, dejando una word que efectivamente puede usarse como día.
Para el caso del año, puede hacerse una simplificacion; es posible acceder a la parte alta de la fecha y correrla a la izquierda un solo bit. Se obtiene el año, pero en un solo byte.

La representación de horas se da como un ejercicio.

I.6.- EJEMPLO DE UN COMPILADOR

El programa LASM254.EXE es el compilador de un lenguaje muy simple que ilustra las técnicas descritas para la construcción de compiladores.

Considera las siguientes estructuras de datos:

- S_BYTE (ENTERO1) 1 byte c/signo
- S_WORD (ENTERO2) 1 word c/signo
- US_BYTE (BINARIO1) 1 byte s/signo
- US_WORD (BINARIO2) 1 word s/signo
- STRING (CADENA) cadena de longitud variable hasta 255 caracteres

Acepta los siguientes operadores (en orden de precedencia):

1.- Paréntesis

2.- Operadores lógicos
AND Y
OR O
NOT NO !

3.- Operadores relacionales
>= => ò MAYOR_IGUAL
<= =< ó MENOR_IGUAL
= IGUAL
< MENOR
> MAYOR
<> DIFERENTE

4.- Aritméticos y de cadenas (1)
+ : suma
- : resta
& : concatenación de cadenas

5.- Aritméticos (2)
* : multiplicación
/ : división
mod : modulo

El formato de un programa debe ser:

```
declaración de variables
CODE            ; palabra reservada de inicio de código
                ; pueden usarse tanto mayúsculas como
                ; minúsculas.
                ; Punto y coma (;) se puede usar para introducir
                ; comentarios

NO_ASUMIR_ENTORNO       ; palabra reservada opcional

desarrollo del programa principal
```

ETIQUETA1: ; declaración y desarrollo de rutinas

ETIQUETA2:

etc..

Sus estructuras de programación con sus palabras reservadas equivalentes son:

Lectura/escritura de variables en pantalla:
    LEER INPUT READ
    OUTPUT WRITE ESCRIBIR

Condicional simple:
    IF SI
    ELSE SI_NO
    ENDIF FIN_SI

Multicondicional:
    DOCASE EN_LOS_CASOS
    CASE CASO
    OTHERWISE OTROS_CASOS
    ENDCASE FIN_CASOS

Ejecución de rutinas:
    DO EJECUTAR

Repetición condicional:
    WHILE MIENTRAS
    WEND FIN_MIENTRAS

Repetición de al menos una vez:
    REPEAT REPETIR
    TO UNTIL HASTA

Repetición n veces:
    FOR PARA
    NEXT FIN_PARA

Terminación del programa (no obligatoria al final ni del programa ni las rutinas)
    END FIN

Se puede usar cualquiera de las palabras de cada línea indistintamente; por ejemplo IF a=b tiene el mismo efecto que SI a=b
Los archivos asociados al compilador son:

LASM254 .PAS : Código fuente Pascal
LEXICO1 .PAS : Unidad de análisis lexicográfico

LASM254 .EXE : Compilador
LEX .ERR : Tabla de errores del compilador
LEX .TAB : Tabla de símbolos del compilador
LASM254T.LIB : Librería de rutinas básicas
LA254CAL.MAC : Macros para rutinas básicas (modalidad no residente)
LA254I50.MAC : Macros para rutinas básicas (modalidad residente)
LASM254R.ASM : Código fuente del entorno residente
LASM254R.COM: Entorno residente
FREE50 .ASM : Liberación del entorno residente
FREE50 .COM : Liberación del entorno residente

TEST .254 : Programa de prueba
TEST_ING.254 : Programa de prueba con palabras reservadas en ingles

NE .COM : Norton Editor: editor ASCII con la posibilidad de comparar visualmente dos archivos

Los programas fuentes para este compilador deben tener la extensión .254 que es la sigla de la materia para la que se desarrollo.
El funcionamiento del compilador se basa en la generación de un archivo .ASM que luego es ensamblado por TASM y enlazado por TLINK (que son invocados por el propio compilador). El programa de prueba TEST.254 se puede ensamblar con el siguiente comando: LASM254 TEST
Y se genera TEST.COM que se puede ejecutar directamente con el comando: TEST
Con los comandos LASM254 TEST TEST.EXE y LASM254 TEST TEST.ASM se generan respectivamente TEST.EXE y TEST.ASM, el primero ejecutable y el segundo programa fuente.
Se ha desarrollado también un entorno de ejecución (LASM254R.COM) que contiene todas las rutinas básicas de entrada, salida y concatenación de variables. Se instala como un programa residente en la interrupción 50H y debe cargarse antes de ejecutar un programa que lo requiera. Solo debe cargarse una vez.
Se puede liberar con FREE50.COM.
La palabra reservada opcional NO_ASUMIR_ENTORNO hace que los programas incluyan dentro de si las rutinas básicas, con lo que resultan mas grandes, pero a la vez no requieren del entorno de ejecución.

El programa de ejemplo TEST.254 es el siguiente:

```
a cadena (20)
b cadena (20)
c cadena (20)
d cadena (20)
x cadena (20) 'PROGRAMA DE PRUEBA'
```

```
n binario2 0

código
no_asumir_entorno
escribir x
escribir 'Quien eres tu ?'
leer a
ESCRIBIR 'Que haces tu ?'
LEER B
EScribir 'Quien eres ?'
leer c
EScribir 'Que haces?'
leer d

escribir 'Termine de aprender'

n=0
repetir
 n=n+1
 escribir 'Dime quien eres y te diré que haces'
 leer x
 si x=a
       escribir x ', tu haces 'b
 si_no
  si x=c
       escribir x ', tu haces 'd
  si_no
       escribir x,', no te conozco'
  fin_si
 fin_si
 Escribir 'Seguimos S/N ?'
 read x
until x='n' o x='N'
escribir 'Se contestaron ' n ' preguntas'
```

Realiza la tarea de almacenar los nombres y actividades de dos personas y luego espera que se le de un nombre de persona e indica que actividad realiza.

Se crea el ejecutable con LASM254 TEST TEST.COM

Ejercicios (Grupo 3)

1.- Como se almacenaría 'JUAn' como ASCIIZ ?

2.- La cadena 'SOL', interpretada como número binario, que número hexadecimal representa ? e interpretada como BCD empaquetado ? y como desempaquetado ?

3.- Como se representaría 'JUAN' dentro de una cadena estructurada de 10 bytes disponibles ?

4.- Que relación existe entre las representaciones de mayúsculas y minúsculas en ASCII ?

5.- Como pueden convertirse las representaciones ASCII '0' a '9' en los números 0 a 9 ?

6.- Como pueden convertirse las representaciones ASCII 'A' a 'F' en los números hexa A a F ?

7.- Algunos sistemas de transmisión de datos solo son capaces de transmitir 7 bits aunque reciban 8 fijando el bit mas alto en 0. Como afectaría esta situación a un mensaje en ASCII transmitido en este sistema ? En particular, que ocurriría con las letras acentuadas ?

8.- La hora del día se almacena en dos bytes según la formula
horas * 2048 + minutos * 32 + segundos / 2

Como se almacenaría la hora 4:05:49 ? Que hora representa ABCD ? Dar algoritmos de conversión.

9.- Idear un esquema para almacenar y manipular "conjuntos"

10.- Idear un esquema para almacenar gráficos en un rectángulo de 80 x 48 elementos en blanco y negro. Cada elemento gráfico recibe el nombre de "pixel" (picture element) y se puede representar en un bit; cero para blanco y 1 para negro.

11.- Un esquema de almacenamiento condensado de textos acepta solo letras (26) y los símbolos (espacio en blanco) . , + - y finalmente /. Solo usa 5 bits por letra. Cuanto ahorra este esquema en el almacenamiento en un texto de puras letras ?

12.- Si se sabe que un texto contiene en gran parte letras, pero también cualquier símbolo, como puede aplicarse la técnica anterior para almacenarlo condensado.

13.- Usando LASM254 compilar el programa TEST con la generación de código tal que asuma el entorno de ejecución, ejecutarlo debidamente y comparar con el tamaño del programa que no requiera el entorno.

14.- Generar el programa TEST.ASM y analizarlo.

15.- Generar el programa TEST.EXE y compararlo con TEST.COM

16.- Escribir un programa de calculo de factorial para LASM254 y probarlo

17.- Escribir un programa de calculo de coeficientes binomiales usando una rutina que calcule factoriales.

II.- PROGRAMACION BASICA

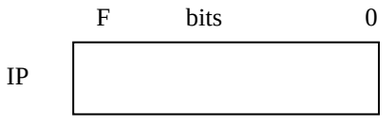
Los microprocesadores son dispositivos microscópicos (su tamaño real es del orden de micras y residen en el centro de los dispositivos que se conocen omochips) con capacidad de"proceso"; esto es, convertir cierta "materia prima" en "producto elaborado", como por ejemplo a partir de dos números, obtener su suma.

Siguen la forma de los algoritmos; esto es, secuencias de instrucciones que se ejecutan en cierto orden.
Para usar un procesador, se debe, diseñar un algoritmo, codificarlo bajo las reglas de su lenguaje y almacenarlo

A continuación se estudiaran los registros y las características de la serie 80x86 de INTEL.

II.1.- Registros

II.1.1.- Puntero de instrucción IP



Contiene la dirección de la siguiente instrucción a ejecutar.

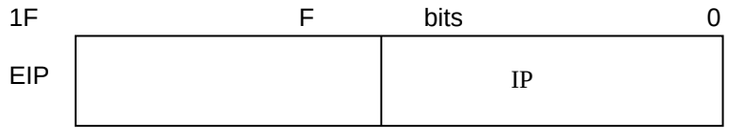
El termino PC (Program Counter o Contador de Programa) se uso hasta el 8080 pero desde el 8086 se ha empezado a aprovechar cierta forma de paralelismo al crear dos unidades independientes; la BIU (Bus Interface Unit) encargada de buscar instrucciones en el bus y la EU (Execution Unit) encargada de ejecutarlas, que trabajan independientemente. Así, mientras la BIU esta buscando una instrucción, la EU esta ejecutando la ya encontrada y al terminar, ya dispone de otra.

Un grupo de instrucciones muy importante de los procesadores en general y que hace que este esquema paralelo falle, son las instrucciones que alteran la secuencia de ejecución. En lenguajes de alto nivel esto corresponde a las formas GOTO o la ejecución de sub-programas. Se las llama "bifurcaciones" o "saltos".

En estos casos, la cola almacenada es inútil y debe ser desechada para buscar las instrucciones que efectivamente corresponden.

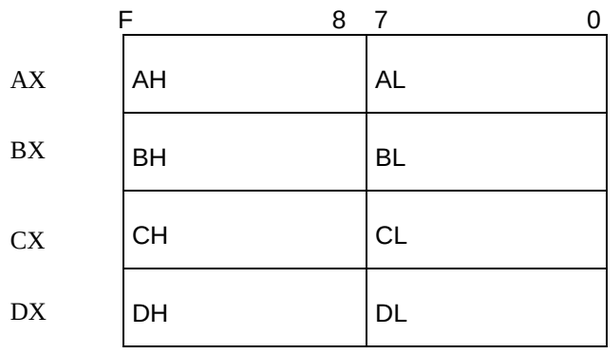
Ya que la BIU usa una cola circular donde almacena instrucciones, el PC ha sido reemplazado por el IP (Instruction Pointer), puntero a la siguiente instrucción en la cola. Ello sin embargo es transparente; el usuario no debe hacer ninguna consideración sobre la pila de instrucciones.

Su tamaño de 16 bits (de 0 a F) permite manejar números (que para su propósito son direcciones de memoria) en el rango de 0 a 64K - 1. En el acápite dedicado a los segmentos se explicara como se usa para manejar cantidades mayores de memoria. Desde el 80386 se amplia a 32 bits con en nombre EIP, pero se los puede usar efectivamente solo en el 'modo protegido', que se mencionara en otro capitulo.



También se ha incluido un dispositivo 'predictor de saltos' con el que se mejora el aprovechamiento de la cola.

II.1.2.- Registros de propósito general

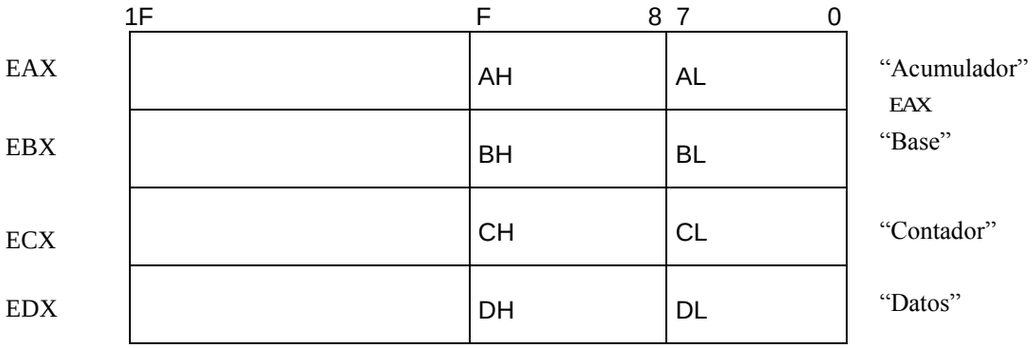


Como su nombre lo indica, pueden usarse para todo (o casi todo) tipo de tarea. Como se muestra, son de 16 bits, y pueden partirse en dos registros de 8 bits, un alto (High) y otro bajo (Low). El tamaño de 16 bits es la "longitud de palabra de datos" y da una idea del nivel de paralelismo del procesador; indica que 16 bits, una word, pueden fluir entre la memoria y el procesador simultáneamente. En un procesador con palabra de datos de 8 bits, para trasladar 16 bits, se requiere el doble del tiempo ya que se necesitan dos operaciones de acceso a la memoria.

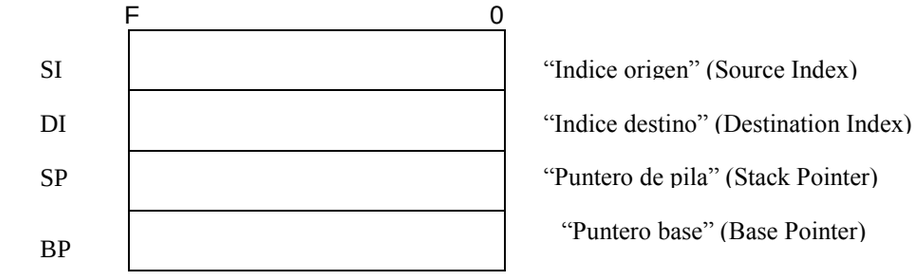
El acumulador AX es un tanto "preferido" para operaciones aritméticas y aunque se pueden hacer sumas y restas en muchos registros, en el se hacen algo mas eficientemente. Las operaciones de multiplicación y división lo requieren imprescindiblemente.

El registro base BX tiene la peculiaridad de poder apuntar a la base de una estructura. Una estructura es generalmente una forma multibyte, caracterizada por estar n un lugar y tener cierta longitud. Como se ha indicado, [nnnn] es la forma de indicar el contenido de la posición de memoria nnnn. Con BX se hace referencia al contenido del propio registro, pero es posible indicar [BX], con lo que se hace referencia la posición de memoria cuya dirección esta en el registro. Esta característica no es común a los demás registros; no es posible indicar por ejemplo [AX].
El contador CX es preferido en ciertas operaciones repetitivas que lo utilizan implícitamente; por ejemplo REP repite una instrucción cuantas veces se indica en CX y no es posible que considere otro registro.
También, es posible realizar sumas, restas y otras operaciones en CX.

El registro de datos DX, aparte de todas las operaciones usuales, en algunas se combina con AX para formar una doble word que es usada con propósitos específicos. DX es la parte alta y AX la parte baja.
Desde el 386, se amplían a 32 bits y se pueden usar tanto en modo real como en modo protegido, que se verán mas adelante



II.1.3.- Punteros e Indices



Pueden ser usados en muchas operaciones, como sumas, restas, carga de valores, como los registros de propósito general, pero tienen las siguientes peculiaridades.

SI y DI tienen la misma característica que BX y puede utilizarse para hacer operaciones en memoria.
Los índices se utilizan especialmente en las llamadas "instrucciones de cadenas". En realidad, cualquier estructura multibyte que ocupe posiciones contiguas puede ser procesada por este tipo de instrucciones. Existe movimiento, carga, recuperación, búsqueda y comparación de estas estructuras. Se explicaran con detalle mas adelante.

SP tiene la muy especial función de manejar pilas. Como se sabe, una pila es una estructura donde "el ultimo en entrar es el primero en salir" y se asocia a los "autómatas de pila", capaces de reconocer "lenguajes libres de contexto".
Existen instrucciones que empalan y desempilan registros y posiciones de memoria. La utilidad de las pilas se manifiesta en tres aspectos:

- Llamadas a rutinas; para implementar la capacidad de sub - programación, cada vez que se invoca un sub-programa, se empila la dirección siguiente a la instrucción de invocación, de tal manera que la instrucción de retorno del sub-programa puede simplemente tomar el ultimo elemento empilado. Con este sencillo esquema es posible anidar sub-programas dentro de sub-programas inclusive recursivamente sin que exista riesgo de confusión si se maneja la pila con un mínimo cuidado.

- Salvaguarda de registros; considerando que solo se disponen de ocho registros, en algunas ocasiones los sub-programas pueden usar los mismos registros que los programas que los invocan. De manera inmediata, resulta que el programa invocador pierde los valores que estaba manipulando, al ser usados por el sub-programa. Para que esto no ocurra, el sub-programa debe llevar los valores de los registros que usa en común con el programa invocador, a la pila. Al terminar, debe restituirlos a sus valores originales tomándolos de la pila, eso si, procediendo en el orden inverso al que los inserto. Por ejemplo si se hizo:

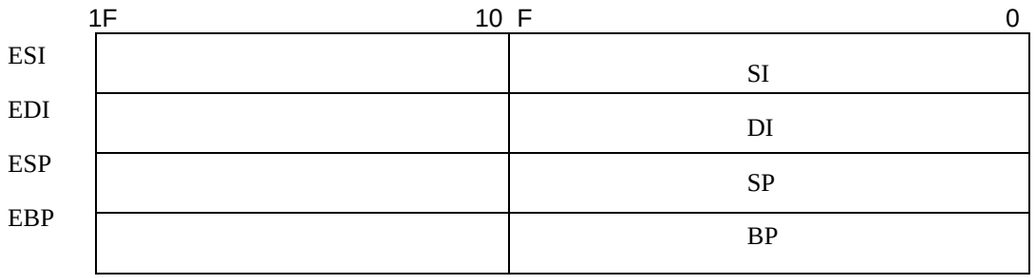
```
Empilar AX
Empilar BX
Empilar CX
```

```
Debe recuperarse:   Desempilar CX
                    Desempilar BX
                    Desempilar AX
```

- "Registros de activación"; el mecanismo para que los lenguajes de alto nivel manejen sub-programas con "variables locales" y capacidad de llamadas recursivas, consiste en la asignación de las variables locales en la pila, en tiempo de ejecución. De esta manera, cada llamada tiene sus propias variables diferenciadas de otras llamadas, aunque sea al mismo sub-programa. Las áreas (franjas, frames) dentro de la pila reservadas así, con sus estructuras complementarias, se llaman "Registros de activación".

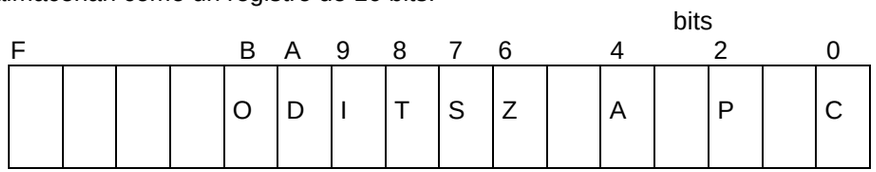
Solo se empalan registros de 16 bits y SP siempre apunta al ultimo elemento empilado.

BP tiene capacidad de trabajar eficientemente dentro de la pila, con la característica de BX, que es direccionar estructuras.
Desde el 386, se amplían a 32 bits y se pueden usar tanto en modo real como en modo protegido, que se verán mas adelante



II.1.4.- Banderas o "Flags"

Se almacenan como un registro de 16 bits.



Casi cada vez que se realizan operaciones, el resultado es someramente evaluado y reflejado en los bits de este registro. Conjuntamente, existen instrucciones que basan su conducta en los bits de este registro. Típicamente;

1.- Se usa una instrucción; las banderas se actualizan según el resultado generado. Por ejemplo, la suma de dos bytes.
2.- Se usa una instrucción basada en el resultado. Por ejemplo, bifurcar

C ("Carry", "Acarreo") en representaciones de números multibyte, para realizar una suma, se deben ir sumando uno a uno los bytes (o words) que la componen. Cuando el resultado de una suma excede el byte, debe llevarse un acarreo para el siguiente byte mas alto. Por ejemplo, al sumar 0101 y 02ff:

	byte bajo	byte alto	
	FF	02	
	01	01	
acarreo →	1 00	04	Resultado: 04 00

que se suma
a 01 y 02

Luego de la ejecución de algunas instrucciones, como por ejemplo sumas, C refleja este resultado intermedio con 0 (NC o "No carry") o 1 (CY o "Carry Yes").

P ("Parity","Paridad") indica la paridad de bits 1 en el resultado; si el resultado de una instrucción, independientemente de su propio valor, tiene un número par de bits 1, P en queda con el valor 1 (PE o "Parity Even"). En otro caso queda en 0 (PO o "Parity Odd").

A ("Auxiliary","Acarreo auxiliar") indica el acarreo entre el nibble superior y el nibble inferior. Es útil en operaciones BCD empaquetado y existe instrucciones que lo utilizan de manera transparente al programa.
Es 1 (AC o "Auxiliary Carry") cuando este acarreo existe y 0 (NA o "No Auxiliary") cuando no.

Z ("Zero") con 1 (ZR o "Zero") indica que el resultado es 0 y con 1 (NZ o "No Zero") indica que el resultado no es 0.

S ("Sign","Signo") con 1 (NG o "Negative") indica que el resultado es un número negativo y con 0 (PL o "Plus") indica que es positivo.

T ("Trap","Trampa") pone el procesador en "simple paso"; este es un modo en que el procesador ejecuta las instrucciones una a una. Requiere de un entorno especial para lograrlo.

I ("Interrupt","Interrupción"). Algunos dispositivos basan su comunicación con el procesador en señales de interrupción; por ejemplo, el teclado comunica al procesador que ha recibido la pulsación de una tecla mediante una señal (que en realidad pasa por un dispositivo "controlador de interrupciones"). Este bit habilita con 1 (EI o "Enable interrupt") y inhabilita con 0 (DI o "Disable interrupt") este tipo de comunicación.
Puede ser fijado por el usuario.

D ("Direction","Dirección"). Las instrucciones de cadena pueden procesar hacia adelante; de principio a fin o en reversa, al revés. Con 0 (UP), se procesa hacia adelante y con 1 (DN o "Down"), en reversa.

O ("Overflow","Sobrepaso"). La operaciones aritméticas de números con signo pueden dar resultados que exceden el rango de las representaciones; por ejemplo, al sumar 1 a 127 (7F), se tiene el resultado 128 (80), que es en realidad usado para -128. Con 1 (OV u "Overflow"), este bit informa que existe esta situación y con 0 (NV o "No Overflow"), que no es así.

Desde el 386, el registro de banderas tiene 32 bits; se aumentan 4 banderas y su significado se analizara al ver el modo protegido.

II.1.5.- Segmentos

Considerando que los tamaños de las rutinas de programas y las áreas de datos que ocupan no son excesivamente grandes, entre otras razones, se ha creído conveniente "segmentar" la memoria. Por este medio, para acceder a la memoria se especifican dos parámetros; un "segmento" y un "desplazamiento" ("offset") para calcular la dirección efectiva según la siguiente relación:

dirección efectiva = segmento * 10h + desplazamiento

Tanto segmento como desplazamiento se dan en 16 bits y solo se consideran 20 bits de la dirección efectiva. Se dice entonces que la "palabra de dirección" es de 20 bits. Con esta cantidad se pueden dar 2^20 (1 Mega, aprox. 1,000,000) direcciones diferentes. Se usa la notación

ssss : oooo

para denotar la dirección de segmento "ssss" y desplazamiento "oooo".
Por ejemplo 2223:200 (asumiéndolos como números decimales) es la dirección efectiva:

2223*10H + 200 = 22230 + 200 = 22430

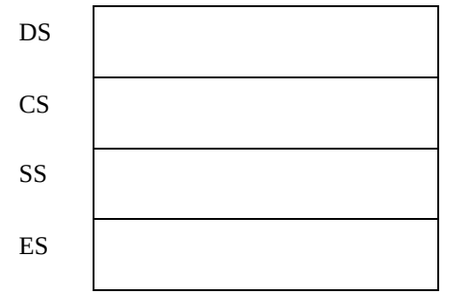
Esta modalidad se denomina 'modo real' y también 'segmentación de 16 bits'.
Nótese que fijando el segmento en un valor constante y con todas las variaciones posibles del desplazamiento, se tiene un área contigua de memoria de direccionable por 16 bits (los del desplazamiento) y por ende, tamaño de 64 K. A esta dimensión se llama también "segmento completo".

Nótese también la versatilidad de los segmentos; son un esquema muy flexible para identificación de la memoria ya que en todas las direcciones de la forma hhhh0 (en hexa) puede empezar un segmento, ocupar solo lo que necesita, que bien puede ser mucho menos que 64K y en la próxima dirección de la misma forma, puede empezar otro segmento.

También, con el mismo desplazamiento se pueden usar diferentes lugares de memoria; por ejemplo, 2334:0100 es otro que 5000:0100, aunque en ambos casos se indica el desplazamiento 100.

Para el procesador, los "registros de segmento" son los encargados de contener el valor del segmento. Son los siguientes:

F 0



Segmento de datos (Data segment)
Segmento de código (Code Segment)
Segmento de Pila (Stack Segment)
Segmento extra (Extra Segment)

Toda dirección debe tener su parte de segmento en alguno de estos registro; las instrucciones que consideran ambos valores, segmento y desplazamiento explícitamente son muy pocas.

DS se asocia a los datos; las direcciones explícitas como [nnnn] o las formas [BX] se tratan como desplazamientos para combinarse con el valor de DS y calcular una dirección efectiva.

CS se asocia al IP. Cuando se da una dirección de programa, se trata como desplazamiento y se combina con CS.

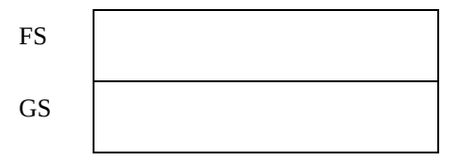
SS se asocia a la pila. Tanto SP y BP se combinan con el.

ES se usa en operaciones de cadenas.

Las combinaciones con registros de segmento no son fijas; se pueden cambiar a voluntad indicando el segmento que se quiere usar de la forma que antes se indico. Por ejemplo:

[1233] es la dirección DS:1233
DS:[1233] es la misma
ES:[1233] es otra si ES tiene otro valor que DS

En realidad, las combinaciones anteriores solo se tratan de "combinaciones por omisión".
Desde el 386 existen también dos nuevos registros de segmento con el fin de descongestionar las tareas de ES. No tienen un acrónimo elegido como los demás; simplemente se llaman F y G.



Segmento F
Segmento G

II.2.- INSTRUCCIONES, MODOS DE DIRECCIONAMIENTO Y UNIDADES DE MEDIDA

Como se dijo, las instrucciones se indican con "mnemotecnicos", códigos de abreviaciones de sus propósitos; por ejemplo MOV es el mnemotécnico de la instrucción que 'mueve' (copia contenidos). Siempre están ubicadas en una dirección y constan de cuando mas tres partes:

iii dddd,ffff

iii es el mnemotécnico de la instrucción. Siempre existe

dddd es el "operando destino", que es alterado por la instrucción. Puede no existir (al ser el operando implícito) o no afectarse

ffff es el "operando fuente" que participa de la instrucción, Generalmente no es afectado; puede no existir.

Ejemplos:

MOV AX,BX ; mueve el contenido del registro BX a AX. AX es destino y BX es fuente.
INC CL ; incrementa CL en 1. AL es destino y no existe fuente.
CLC ; fija la bandera C en 0. No existen destino ni fuente explícitos.
TEST AX,DX; ni AX ni DX son afectados. Se realiza una operación AND con ambos y se afectan las banderas según el resultado, que no queda disponible en ningún registro.

Los operandos solo pueden ser algunos de ciertos "modos de direccionamiento" posibles;

- "registro" : simplemente indicar un registro
- "inmediato": un valor constante colocado en la posición inmediata siguiente a la instrucción
- "directo" : una dirección entre corchetes [ ]
- "indirecto": la dirección apuntada por un registro

Los modos indirectos posibles son:

[SI]	[DI]	[BP]	[BX]
[SI+c]	[DI+c]	[BP+c]	[BX+c]
[BX+SI]	[BX+DI]	[BP+SI]	[BP+DI]
[BX+SI+c]	[BX+DI+c]	[BP+SI+c]	[BP+DI+c]

c es un desplazamiento de hasta 16 bits. Los modos que incluyen una base (BX o BP) se llaman "basados", los que incluyen un índice (SI o DI), "indexados" y los que incluyen una dirección, "directos" y se derivan todas las combinaciones; así por ejemplo [SI+BX+c] es el modo "indexado basado directo". También pueden usarse otras formas ;[c][SI][BX] , [SI+c][BX] son equivalentes a [SI+BX+c].

Los modos que incluyen a BP, toman a SS como segmento por omisión y los demás, a DS.

Dentro de cada instrucción se reservan campos específicos para identificar el modo de direccionamiento; estos son generalmente mod (de 2 bits) , r/m (de 3 bits) y w (1 bit).

w 1 operando de 16 o 32 bits
0 operando de 8 bits

mod Direccionamiento directo e indirecto
00 | Sin desplazamiento
01 | Con un byte adicional de desplazamiento
10 | Con dos bytes adicionales de desplazamiento
11 | Modo registro

r/m registro o forma de calcular la posición de memoria usada en el direccionamiento.

	Modo Registro 8 bits (w=0)	Modo Registro 16 o 32 bits (w=1)	Modo Indirecto	Segmento
000	AL	AX EAX	[BX+SI]	DS
001	CL	CX ECX	[BX+DI]	DS
010	DL	DX EDX	[BP+SI]	SS
011	BL	BX EBX	[BP+DI]	SS
100	AH	SP ESP	[SI]	DS
101	CH	BP EBP	[DI]	DS

110	DH	SI ESI	[BP]	SS
111	BH	DI EDI	[BX]	DS

Para que se usen 32 bits en lugar de 16 es necesario el 'prefijo de modificación de operando' 66H antes de la instrucción.
El modo inmediato es un caso especial de la codificación de cada instrucción y es admitido solo en algunas instrucciones.
El modo directo es codificado con mod = 00 y r/m=110 con lo que el direccionamiento [BP] debe ser reemplazado por [BP+0]
La siguiente tabla resume lo indicado:

	M o d			
r / m	00	01	10	11
000	[BX+SI]	[BX+SI+d8]	[BX+SI+16]	AL AX EAX
001	[BX+DI]	[BX+DI+d8]	[BX+DI+16]	CL CX ECX
010	[BP+SI]	[BP+SI+d8]	[BP+SI+16]	DL DX EDX
011	[BP+DI]	[BP+DI+d8]	[BP+DI+16]	BL BX EBX
100	[SI]	[SI+d8]	[SI+16]	AH SP ESP
101	[DI]	[DI+d8]	[DI+16]	CH BP EBP
110	[d8]	[BP+d8]	[BP+16]	DH SI ESI
111	[BX]	[BX+d8]	[BX+16]	BH DI EDI

d8 es un desplazamiento adicional al final de la instrucción que se convierte a 16 bits por "extensión de signo" que resulta en usar el byte adicional como un número signado. Por ejemplo si el desplazamiento debe ser 0007, basta con dejar 07 con extensión de signo. Pero, si es 1234h o 0080h, no es posible usar solo un byte. Si fuera FFFF, se podría usar extensión de signo con FF.
Usando BX,SI o DI se asume el segmento DS: pero si se usa BP, se asume por omisión SS:

Ejemplo:

La instrucción INC incrementa en uno su operando; su codificación en dos bytes es:

byte 1 byte 2
1111111 w mod 000 r/m

Modo inmediato: no existe por que no tiene mucho sentido.

Modo registro:

INC AX w=1 por que AX es de 16 bits
mod=11 por que es un registro
r/m=000 por ser AX
se codifica : 11111111 11000000 = ff c0

INC EAX es similar a INC EAX, pero aumentando el prefijo 66h:
66 ff c0

INC BL w=0 por que BL es de 8 bits
mod=11 por que es un registro
r/m=011 por ser BL
se codifica en: 11111110 11000011 = fe c3

Modo directo:
INC [1239] w=0 si se trata de incrementar exactamente esa posición de un byte se usa INC BYTE PTR [1239]
w=1 si se trata de incrementar la palabra de dos bytes que empieza en esa posición. Se usa INC WORD PTR [1239]
mod=00
r/m=110
INC BYTE PTR [1239] es 11111110 00000110 = fe 06 39 12
INC WORD PTR [1239] es 11111111 00000110 = ff 06 39 12

Para incrementar una doble word, lo que no puede verse con DEBUG y si es posible en TD, se debe seguir la codificación de w=1 y preceder con 66H: INC DWORD PTR [1239] es 66 ff 06 39 12 PTR es abreviación de PoinTeR (puntero)

Modo indirecto:
INC WORD PTR [BX] w=1, mod=00, r/m=111
11111111 00000111 = ff 07

INC BYTR PTR [BP+SI] w=0, mod=00, r/m=010
11111110 00000010 = fe 02

INC WORD PTR [BX+DI+000A] w=1, mod=10, r/m=001
11111111 10000001 0A 00 = ff 81 0A 00

Esta ultima instrucción puede ser codificada mediante la "extensión de signo"; esto es, ya que todos los bits de la parte alta del desplazamiento son iguales al ultimo bit de la parte baja, basta indicar solo la parte baja y usar mod =01

11111111 01000001 0A = ff 41 0A

El objetivo de este modo es reducir el espacio ocupado por el programa codificado.
Para permitir la operación con otros segmentos existe la instrucción de sobre-escritura de segmento que hace que deje de usarse el segmento por omisión propio de la instrucción para usarse el segmento indicado MOV AX, [ES:90] usa el segmento extra para mover el contenido de la posición 90 a AX.
Como instrucción, debe anteceder como prefijo a la instrucción que lo requiere con el formato 001_seg_110 para ES, CS, DS y CS. FS y GS tiene otro formato.

Instrucción Codificación
ES: 26H (seg=00)
CS: 2EH (seg=01) Ej. MOV AX,[123] asume por omisión DS
SS: 36H (seg=10) Para usar el segmento extra: ES: MOV AX,[123]

DS: 3EH (seg=11)
FS: 64H
GS: 65H

Los modos de direccionamiento se corresponden con ciertas estructuras y formas de asignación de memoria de alto nivel. Por ejemplo, las variables globales se definen con una simple dirección y pueden ser:

- "átomos"; se acceden por su sola dirección
- "arreglos"; requieren un índice
- "registros"; también una dirección
- "arreglos de registros"; un índice y una dirección

La asignación de espacio puede ser:

- "estática"; una dirección fija accesible por todo elemento del programa, Se llama también “global”
- "local"; en la pila como registro de activación
- "dinámica"; en un lugar definido solo en tiempo de ejecución

Tipos de Variables	Atomo	Arreglo	Registro	Arreglo de Registros
Estáticas	[constante]	[xI]	[constante]	[xI+constante]
Locales	[BP+constante]	[BP+xI]	[BP+constante]	[BP+xI+constante]
Dinámicas	[BX+constante]	[BX+xI]	[BX+constante]	[BX+xI+constante]

La abreviación xI representa SI o DI; para variables estáticas puede ser BX.

Se pueden usar también SI y DI en átomos y registros de variables dinámicas.

Desde el 386, es posible usar cualquiera de los registros extendidos excepto ESP para direccionamiento indirecto. Además, se pueden incluir sumas entre cualquiera de ellos, incluyendo dos sumandos iguales. También se puede incluir un factor de escala entre 2,4 y 8 multiplicando alguno de ellos. Algunos modos de direccionamiento validos son:

[EAX+EBX]
[EAX+EAX + 1232]
[ECX+4*ESI]
[2*EBP+EDI]
[EBX+8*EBP + 23222222]

En el modo real, la parte alta de los registros extendidos debe ser 0. En otro caso, el equipo se "colgara".
Para codificar instrucciones que usen estos modos de direccionamiento, se debe incluir el 'prefijo de tamaño de dirección' 67h antes de la instrucción, con lo que mod y r/m tienen una tabla diferente.
Para los modos que usan simplemente un registro de 32 bits, mod debe ser 00 y r/m se interpreta como si correspondiera a un registro. EBP (101) también indica que no se usa ningún registro y se trata de direccionamiento directo en 32 bits (solo en modo protegido). ESP(100) indica que la dirección se forma con una suma de registros donde uno tiene un factor de escala. Se debe incluir un byte luego de la instrucción con la siguiente interpretación :

f1 f 0	i 2 i 1 i 0	r 2 r 1 r 0

f1 y f 0 (f) son la potencia de 2 que es el factor de escala. 0 indica que no hay factor de escala, 1 que es 2, 2 que es 4 y 3 que es 8.

i2, i1 e i0 (i) son el registro que se multiplica por el factor. 100, que corresponde a ESP indica que mas bien no existe registro índice

r2, r1 e r0 (r) son el otro registro al que se suma el producto.

Cuando mod=01, se suma un desplazamiento en 8 bits, que extiende su signo a 32 bits. EBP ya no es tratado de manera diferente.
Cuando mod=10, se suma un desplazamiento en 32 bits.
Cuando mod =11, (registro) el prefijo 67h no tiene sentido.

Los prefijos 66h y 67h no son necesarios para operaciones de 32 bits cuando en modo protegido se selecciona por defecto ese tamaño con el D-bit de los descriptores de segmentos.

De manera similar que a los bits, a las direcciones superiores de memoria se las denomina "altas"; y a las direcciones inferiores, se las denomina "bajas". Se dice que la memoria "crece" hacia las direcciones superiores y decrece hacia las inferiores.

Ejemplo: Recordando que INC se codifica con: 1111111 w mod 000 r/m

INC BYTE PTR [ABCD1239]

w=0 se trata de incrementar un byte
mod=00
r/m=101 (EBP)
67h 11111110 00000101 39 12 CD AB=67 fe 05 39 12 CD AB

INC WORD PTR [EBX] w=1, mod=00, r/m=011
67h 11111111 00000011 = 67 ff 03

INC BYTE PTR [EAX+EBX] w=0, mod=00, r/m=100, f=0 ,i=0 r=3
67h 11111110 00000100 00000011 = 67 fe 04 03

INC WORD PTR [EAX+4*EDI+4] w=1, mod=01, r/m=100 f=10 i=111 r=0
67 11111111 01000100 10111000 04 = 67 ff 44 B8 04

UNIDADES DE MEDIDA DE LA MEMORIA

Mientras el bit es la "unidad de información" por que no es posible
fragmentar la información "0" o "1" entregada por un bit, la "unidad de direccionamiento" es el byte ya que la operaciones con la memoria no pueden direccionar por si solas a un bit en particular de un byte. Mediante otras manipulaciones, si es posible.

Sobre la unidad del byte, se definen:

"word"	dos bytes
"dword"	dos words (doble word)
"qword"	cuatro words (cuádruple word)
"tbyte"	diez bytes
"fword"	seis words

Estas formas se denominan "multibyte" ya que implican mas de un byte.
Cuando se almacenan en memoria, ocupan (generalmente) lugares consecutivos y los bytes de las direcciones mas altas se denominan "bytes mas altos" o también "bytes mas significativos". Recíprocamente, al byte de la dirección mas baja se denomina "bytes mas bajo" o también "bytes menos significativos".

Otras unidades son:

"párrafo"	16 bytes	
"pagina"	256 bytes	
"K"	1024 bytes	(aproximadamente mil)
"M" (mega)	1024 K	(aproximadamente un millón)
"G" (giga)	1024 M	(aproximadamente mil millones)
"T" (tera)	1024 G	(aproximadamente un billón)

Nótese que 1024 es la potencia 10 de 2.

Por si solo, un byte no tiene un significado propio y depende de una "interpretación". En realidad las diferentes estructuras de la información no son si no interpretaciones diferentes de bytes.

Ejercicios (Grupo 4)

1.- Si AX tiene el número decimal 1823, cuales son los valores hexadecimales de AH y AL ?

2.- Un programa tiene el siguiente aspecto:

100h invoca la rutina de 200h
...
120 termina programa

200h invoca la rutina de 300h
...
210 termina rutina

300h
...
340 termina rutina

considerando que IP = 100h, SP=7000, que valores se conocerán con certeza en el área 6ff0 a 7000 y que valor tendrá SP cuando IP=300 ? Considerar que una instrucción de invocación de rutina ocupa 3 bytes.

3.- Un programa usa los registros AL,BX y SI. Llama a una rutina que usa AH, BL y DI. Que registros debe resguardarse para coordinar correctamente el programa con la rutina ?

4.- Después de calcular 56h+33h se actualizan las banderas S,Z,A,P y C. Cuales deben ser sus valores ?

5.- El registro r tiene los siguientes campos:

a byte
b word
c array [0..5] of longint

Si se localiza en el área apuntada por BX, como se puede acceder a cada uno de sus campos ? En especial , como se puede acceder al n-ésimo byte del entero largo del campo "c" ?

6.- Dado x un arreglo [0..4] de dobles words, como puede accederse a la n-ésima word primero usando direccionamiento standard y luego direccionamiento de 32 bits ? .Considerar que x esta desde la posición 200h.

7.- Un programa se carga en 434:100 y tiene una longitud de 234h bytes. A continuación se quiere cargar otro programa dejando entre ambos un mínimo de 100h bytes y que también se haga en un offset 100h. Cual es la dirección mas baja donde puede hacerse ?

8.- La instrucción MOV tiene dos operandos; fuente y destino, efectuando la copia del operando fuente al destino. Una de sus formas codificación es:

100010 d w mod reg r/m

Uno de sus operandos siempre es un registro codificado en el campo reg de manera similar a r/m y el campo d indica si se mueve al registro (1) o del registro (0).

Como se escribiría y codificaría la instrucción que:

a) mueva AX sobre BX
b) mueva la posición de memoria 0982 sobre DL
c) mueva SP sobre la posición das por la suma de BX y SI
d) mueva BP sobre la posición apuntada en SI
e) sobre la posición 9a mueva el contenido de SI
f) sobre BX+SI+9DDE mueva el contenido de CL

9.- Porque se necesita un solo campo w en la anterior instrucción ?
Por que no es posible mover de memoria a memoria ?

10.- MOV tiene también un modo inmediato con la codificación

1100011w mod 000 r/m data1 data2

Explicar por que es necesario indicar WORD PTR o BYTE PTR en una instrucción MOV con modo inmediato y direccionamiento directo.

11.- Cuantos bits tiene una word ?, una dword ? una qword ? y una fword ?

12.- Dado el valor 01011111, cuanto vale el nibble superior ? y el inferior ?
y el bit mas significativo ? y el menos significativo ?

13.- Cuantos párrafos tiene un mega?

14.- A la memoria que puede ser modificada se la denomina RAM (random access memory). Existe también la ROM (read only memory) que aunque también es de acceso aleatorio, no puede ser modificada. Como puede verificarse que una dirección de memoria es ROM ?

15.- Cuantas paginas tiene un giga ?

16.- En que se diferencian 234 y [234] ?

17.- Si una estructura de información de 9 bytes es almacenada crecientemente con la memoria a partir de 200, que posiciones ocupa ?

II.3 Turbo Debug

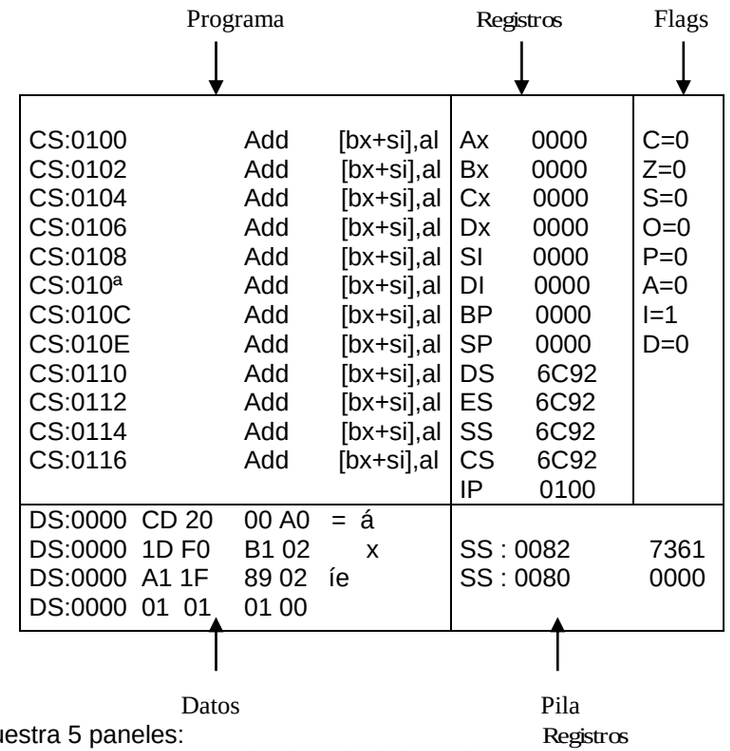
TURBO DEBUG es el programa de depuración provisto por Borland para sus productos Turbo C, Turbo Assembler y otros. DEBUG es el depurador de Microsoft que ha tenido pocos cambios desde DOS 2.
Para acceder a TD basta con escribir TD "prompt" del DOS del directorio que contiene TD y presionar Enter;

```
C:\TC\BIN\>TD
```

Para trabajar con un archivo, se puede indicar como parámetro;

```
C:\TC\BIN\>TD KEYB.COM
```

Por ser un comando externo, es necesario estar en el mismo directorio que el programa TD.EXE o indicar donde esta previamente al nombre o disponer de un PATH adecuado.
Puede abrir múltiples ventanas . Presenta en principio la siguiente ventana CPU:



Muestra 5 paneles:
Programa : se pueden introducir inmediatamente instrucciones en ensamblador
Datos : se pueden introducir inmediatamente números y cadenas de caracteres
Pila : se pueden introducir inmediatamente números y cadenas de caracteres
Registros : se pueden alterar valores inmediatamente
Flags : se pueden alterar valores inmediatamente

Se tienen las siguientes facilidades en teclas de función y otras
F1-Help : mostrar una ayuda sensitiva al contexto
F2-Bkpt : fijar/cancelar punto de parada (break point) en la dirección del cursor
F3-Mod : abre una ventana de modulo
F4-Here : ejecuta hasta la instrucción del cursor
F5-Zoom : agranda/achica la ventana
F6-Next : pasa a la siguiente ventana
F7-Trace : traza una instrucción
F8-Step : traza una instrucción, pero si es una rutina, la ejecuta
F9-Run : corre el programa
F10-Menu : accede al menú principal
TAB : avanza entre paneles de una ventana
Shift TAB : retrocede entre paneles de una ventana
ALT F1 : Tópico de ayuda previamente consultado
ALT F2-Bkpt At: fijar punto de parada (break point)
ALT F3-Close : cierra la ventana actual
ALT F4-Back : retrocede al anterior paso de ejecución (opuesto a F7). Para que funcione adecuadamente se debe fijar previamente la opciónFull en la ventana Execution History
ALT F5-User : muestra la pantalla del usuario
ALT F6-Undo : reabre las ventanas cerradas con ALT F3
ALT F7-Instr : traza una instrucción de maquina. Es similar a F7 pero en algunas instrucciones como INT, F7 ejecuta toda la interrupción mientras que ALT F7 entra a la interrupción
ALT F8-Rtn : corre el programa hasta una instrucción RET
ALT F9-To : corre el programa hasta una dirección
ALT F10-Local : accede al menú local. Muchas opciones locales pueden ser accedidas con la tecla Control y una letra.
Shift F1 : Indice de ayuda
Shift F3 : Copia a la lista Clipboard
Shift F4 : Copia de la lista Clipboard
Control F2 : Recarga el programa actual
Control F4 : Inspecciona memoria
Control F5 : Mueve con los cursores la ventana actual a otra posición en la pantalla. Con Shift y los cursores se puede cambiar el tamaño de la ventana.
Control F7 : Evalúa y modifica expresiones.

Se puede acceder al Menú principal con F10 o ALT. Sus opciones y sub-opciones son:

File: Open - cargar un archivo de programa, normalmente con extensiones EXE o COM
Change dir - cambiar de directorio
Get Info - mostrar información del programa en depuración, la memoria y sistema operativo
Resident - termina y queda residente. Un programa que ejecute INT 3 trae TD nuevamente.
Quit (ALT X) - salir

Edit: Copy (Shift F3) - copia el elemento del cursor a 'Clipboard', una ventana que contiene una lista de direcciones y contenidos.
Paste (Shift F4) - recupera de 'Clipboard' al cursor
Copy to log - copia a 'Log', una ventana de lista de elementos y puntos de parada.
Dump pane to log - copia todo el panel a 'Log', una ventana de registro mas amplio de elementos y eventos.

View: abre las siguientes ventanas que muestran:

Breakpoints: puntos de parada
Stack: la pila, aplicable en la depuración de lenguajes de alto nivel
Log: lista con puntos de parada, contenidos y partes de paneles
Watches: expresiones que muestran valores y contenidos de memoria
Variables: etiquetas y variables locales y globales de módulos

Module (F3): Módulos. Es aplicable a programas compuestos de uno o mas módulos compilados con las opciones /zi o /zd. En ellas se establece relación entre el programa fuente y el programa ejecutable, con lo que puede verse el modulo fuente.

File: Archivo de disco

Cpu: muestra los 5 paneles Programa, Datos, Pila, Registros y Flags
Los menús locales de cada uno de ellos son:
Programa:
Go to (^G) : muestra desde una dirección dada
Origin (^O) : muestra desde el actual CS:IP
Follow (^F) : Muestra el código siguiente para instrucciones de bifurcación
Caller (^C) : Muestra el programa que llamo al actual basándose en la pila
Previous (^P) : vuelve al código que previamente mostraba
Search (^S) : busca un contenido de memoria
View source (^S) : muestra el modulo fuente del código del cursor. Son necesarias mismas condiciones que para abrir un modulo fuente.
Mixed (^M) : para un modulo fuente, muestra las líneas fuente de tres maneras:
No : solo muestra códigos desensamblados
Yes: muestra códigos desensamblados y líneas fuente
Both: reemplaza código desensamblado por líneas fuente Marca con un símbolo las líneas fuente.
New cs:ip (^N) : hace la presente dirección el actual CS:IP
Assemble (^A) : permite la introducción de programa. Es posible introducir instrucciones inmediatamente pero ^A muestra las instrucciones previamente usadas y es sible re-usarlos.
I/O (^I) : permite la lectura o escritura de puertos de 8 o 16 bits.

Registros:
Increment (^I) : incrementa en 1 el actual registro
Decrement (^D) : decrementa en 1 el actual registro
Zero (^Z) : pone en 0 el actual registro
Change (^C) : cambia el valor del actual registro. Es posible cambiar valores inmediatamente pero ^C muestra los valores previamente usados y es posible re-usarlos.
Registers (^R) : muestra registros en 32 bits o deja de mostrarlos

Flags:
Toggle (^T) : cambiar valores binarios. Es también posible hacerlo con la barra espaciadora.
Pila:
Go to (^G) : muestra desde una dirección dada
Origin (^O) : muestra desde el actual SS:SP
Follow (^F) : Muestra el código siguiente para instrucciones de bifurcación
Previous (^P) : vuelve al código que previamente mostraba
Change (^C) : cambia el valor actual. Es posible cambiar valores inmediatamente pero ^C muestra los valores previamente usados y es posible re-usarlos.
Datos:
Go to (^G) : muestra desde una dirección dada
Search (^S) : busca un contenido de memoria
Next (^N) : busca el siguiente contenido
Change (^C) : cambia el valor actual. Es posible cambiar valores inmediatamente pero ^C muestra los valores previamente usados y es posible re-usarlos.
Follow (^F) : pasa mostrar desde la posición actual.
Puede afectar el panel de programa e interpretar la posición actual como direcciones lejana o cercana.
Puede afectar el panel de datos e interpretar la posición actual como offset, segmento o ambos.
Previous (^P) : vuelve al lugar que previamente mostraba
Display as (^D) : muestra como binarios: byte, word, long (4 bytes), comp (8 bytes); reales float (4 bytes) Real (6 bytes), Double (8 bytes) y Extended (10 bytes).
Block (^B) : permite manejar bloques que se indican por posición y tamaño:
Clear : pone en 0
Move : mueve a otra posición
Set : fija en un valor

Read : lee un archivo
Write : escribe un archivo

Dump: idéntica al panel de datos de la ventana CPU

Registers: identifica al panel de registros de la ventana CPU

Numeric Processor: muestra el coprocesador matemático:

Empty ST (0)	im = 1	Ie = 0
Empty ST (0)	dm = 1	de = 0
Empty ST (0)	zm = 1	ze = 0
Empty ST (0)	om = 1	oe = 0
Empty ST (0)	um = 1	ue = 0
Empty ST (0)	pm = 1	pe = 0
Empty ST (0)	iem = 1	ir = 0
Empty ST (0)	pc = 3	cc = 1
	rc = 0	st = 0
	ic = 0	

Execution History: secuencia de instrucciones ejecutadas

Windows messages: propio de windows

Clipboard: lista de contenidos guardados con ALT F3

Another: Permite abrir otra ventana Module,File o Dump

Run : Casi todas sus opciones son teclas de función.
Run (F9)
Go to cursor (F4)
Trace into (F7)
Step over (F8)
Execute to (ALT F9)
Until return (ALT F8)
Animate : ejecuta el programa instrucción a instrucción automáticamente.
Pide un tiempo en décimas de segundo entre cada instrucción.
Back trace (ALT F4)
Instruction trace (ALT F7)
Arguments : permite fijar los argumentos tal como se pasarían a un programa desde el procesador de comandos.
Program Reset (Ctrl F2)

Breakpoints: Toggle (F2) fija un punto de parada en la dirección actual
At (ALT F2): fija un punto de parada en una dirección. Es posible especificar las condiciones en que detendrá un programa en ejecución y las acciones que generara
Change Memory global: requiere un área de memoria mediante inicio y tamaño. Cuando es alterada durante la ejecución de un programa, se detiene.
Expresión true global: requiere una expresión que se evalúa en cada instante. Cuando es cierta durante la ejecución de un programa, se detiene.
Hardware breakpoint: requiere de una tarjeta especial de depuración por hardware
Delete all: borra todos los breakpoints

Data: Inspect: permite ver el contenido en una dirección
Evalúa/Modify (Control F4): permite introducir una expresión que puede ser evaluada y modificada en función de un parámetro
Add watch (Control F7): añade una expresión a la ventana Watch. La abre si no lo esta.
Function return: permite ver el resultado de una función. En aplicable para C o Pascal de Borland.

Options:
Language: Fija el lenguaje en que se interpretaran las expresiones.
La opción Source hace que el propio TD vea en que lenguaje C, Pascal o Asembler están escritas.
Macros: permite grabar y eliminar macros, secuencias de teclas en una sola
Display options: algunas opciones de despliegue como líneas por pantalla e intercambio (swap) de pantallas del usuario y de TD.
Path for source: fija el directorio donde están los programas fuente.
Save: guarda las opciones fijadas (Option), las ventanas abiertas (Layout) y las macros creadas (Macro) en el archivo de configuración u otro de tal manera que al volver a ingresar a TD, ya están establecidas.
Restore: Restituye un archivo de configuracion previamente guardado con Save.

Window: Casi todas sus opciones son teclas de función.
Zoom (F5)
Next (F6)
Next pane (TAB)
Size/Move (Ctrl F5)
Iconize/Restore : Reduce la ventana actual a un mínimo tamaño o si esta reducida la restituye a su tamaño original.
Close (ALT F3)
Undo close (ALT F6)
User screen (ALT F5)
A continuación esta la lista de ventanas abiertas que se pueden elegir.

Help: Casi todas sus opciones son teclas de función.
Index (Shift F1)
Previous topic (ALT F1)
Help on help : Informa como usar Help.

TD puede usarse para explorar un computador cuando se dispone de alguna información técnica. Algo muy interesante es la pantalla.
Existe un área de la memoria donde se almacena toda la pantalla reservando dos bytes por cada carácter; el primero contiene el número ASCII del carácter y el segundo contiene un "atributo" que define la presentación del carácter.
En modo monocromático los valores del atributo pueden ser:

07= normal
87= centelleante
0F= intenso
8F= centelleante e intenso
70= vídeo inverso
00= invisible negro
77= invisible blanco

En monitores VGA, esta área se sitúa desde B800:0 y en monitores Hercúleas desde B000:0.

Con esta información, es posible realizar algunas manipulaciones.

- Ver el área de pantalla: en el panel de datos de la ventana CPU con ^G fijar
  0B800:0

- Probar los atributos en la primera fila. Poner los siguientes valores:
  41 07 41 87 41 0f 41 8f 41 70 41 00 41 77

- Cargar, ver el código y ejecutar un programa;
  Acceder al menú principal con F10 y luego Enter sobre File o directamente usar ALT F.
  Elegir EJEMPLO.EXE o escribir su nombre
  Ejecutarlo con F9
  Ver con ALT F5 su efecto

- Copiar el contenido de la memoria a un archivo de disco.
  ^B W
  rbx (para fijar parte alta de la longitud del archivo)
  rcx (para fijar parte baja de la longitud del archivo)
  w

Ejercicios (Grupo 5)

1.- Calcular x de tal manera que el lugar x:100 sea el mismo que DS:0

2.- Verificar que el segmento f000 es ROM. Intentar escribir en el viendo que no es posible modificarlo.

3.- Averiguar la codificación de la instrucción CLC. Introducirla y luego ver el código que la genera.

4.- Averiguar a que instrucciones corresponden las codificaciones siguientes:

ff 07
fe 02
ff 81 0A 00

5.- Fijar las banderas en 0.

6.- Introducir el siguiente programa en 100H y trazarlo con F7 viendo los cambios en los registros, las banderas y la posición 200 de memoria:

```
mov ax,1
mov bx,[200]
add ax,bx
mov [200],ax
```

7.- Introducir el siguiente programa en 100H y trazarlo con F7. Que efecto tiene en el registro AL ?

```
mov ah,1
int 21
```

8.- Aumentar int 3 luego de int 21 al programa anterior y ejecutarlo con F9

9.- Almacenar el número decimal 127 en binario, BCD empaquetado y luego desempaquetado a partir del desplazamiento 200 sin interesar el segmento.

10.- Cargar DX,AX con el número hexadecimal 23fab31.

II.4.- INSTRUCCIONES BASICAS

A continuación se describirá un conjunto básico de instrucciones con los siguientes convenios:
dest. = operando destino
fuente = operando fuente
r = registro de propósito general, puntero o índice.
rn = registro de propósito general, puntero o índice de n bytes.
m = modo de direccionamiento a memoria. Casi siempre se combina con DS para datos o CS para programa. Se puede indicar explícitamente. Por ejemplo ES:[200] usa CS como segmento.
s = registro de segmento
c = inmediato
dir. = dirección; generalmente un desplazamiento de 16 bits.

II.4.1.- MOV dest. , fuente.

Copia el contenido del operando fuente al operando destino. Las formas posibles son:
MOV r,r registro a registro
MOV r,c valor inmediato a registro
MOV r,m memoria a registro; asume DS si no se indica
MOV m,r registro a memoria; asume DS si no se indica
MOV m,c valor inmediato a memoria; en DEBUG se debe indicar BYTE PTR o WORD PTR en m para indicar que se trata de un Byte o una Word.
MOV s,r16 registro a segmento
MOV s,m memoria a segmento
MOV r16,s segmento a registro
MOV m,s segmento a memoria

Nótese que:
- ambos operandos deben ser del mismo tamaño en bits. Es posible indicar MOV AL,BL pero no MOV AL,BX (AL es de 8 bits y BX de 16). MOV AL,[BX] si es posible ya que [BX] se refiere a un byte por ser AL un byte.
- no se puede mover memoria a memoria
- no se puede mover un valor inmediato a un segmento
- en las formas r,m y m,r se usa un operando en memoria del mismo tamaño ue r. Así MOV AL,[67] mueve el byte de [67] a AL. MOV AX,[566] mueve a word [566] ([567] a AH y [566] a AL).
- no se afectan banderas

Los usos de MOV son muchísimos. Algunos son:
- Inicializar un registro con un valor: MOV AL,5
- Hacer que un puntero apunte a una dirección de memoria: MOV SI,200
- Cargar un registro con el contenido de una dirección de memoria:
MOV AL,[200]
- Copiar el contenido de un registro a otro: MOV AX,BX
- Cargar un registro con el contenido de una dirección de memoria apuntada por un registro : MOV AL,[SI]
- Llevar un registro a una dirección de memoria: MOV [BX],DX
-Cargar un registro de segmento con un valor. MOV ES,400 no es Posible; se debe pasar por un registro de 16 bits: MOV AX,400
MOV ES,AX
- Cargar una valor inmediato a memoria: MOV [SI],78 no es precisa ya que no es claro si se trata de inicializar el byte apuntado por SI con 78 o la word apuntada por SI. Se debe usar: MOV BYTE PTR [SI],78 para indicar que es el byte y MOV WORD PTR [SI],78 para indicar que es la word
- Mover de un lugar de memoria a otro. MOV [SI],[DI] no es posible, además que no se indica si son bytes o words quienes se quieren mover. En todo caso se debe usar un registro intermedio:

para bytes: MOV AL,[DI] para words: MOV AX,[DI]
MOV [SI],AL MOV [SI],AX

- Como se indico, los modos de direccionamiento asumen DS por omisión. Es posible cambiar de segmento indicando el segmento; MOV ES:[SI],AL mueve al segmento ES, desplazamiento SI.

II.4.2.- XCHG dest. , fuente.

Intercambia dest. y fuente.

Ambos operandos son afectados. Sus formas posibles son:
XCHG r,r
XCHG r,m
XCHG m,r
Nótese que:
- ambos operandos deben ser del mismo tamaño en bits
- no se puede intercambiar memoria a memoria
- no se pueden involucrar segmentos
- no se afectan banderas

Puede ser usado en procesos de ordenamiento que involucran estos intercambios; por ejemplo para intercambiar AL y AH, basta:
XCHG AL,AH
En cambio con MOV, debe hacerse: MOV BL,AL
MOV AL,AH
MOV AH,BL

Para intercambiar elementos de memoria entre si, puede hacerse:
MOV AL,[SI]
XCHG [DI],AL
MOV [SI],AL

II.4.3.- Instrucciones de pila.

Siempre manejan operandos de 16 bits

PUSH op. empila un registro o memoria o una constante de 16 o 32 bits.
PUSHF empila el registro de banderas
POP op. desempila el tope de la pila a un registro o memoria, siempre de 16 bits
POPF desempila el tope de la pila a las banderas

Nótese que el SP siempre apunta al ultimo registro empilado. Se usa para salvar registros. Por ejemplo, PUSH AX inserta en la pila el registro AX. Puede usarse también para recuperar las banderas en un registro; PUSHF
POP DX

II.4.4.- Sumas y Restas

Suman y restan el operando fuente al operando destino

ADD dest. , fuente: dest. = dest. + fuente
ADC dest. , fuente: dest. = dest. + fuente + C (C es la bandera C)
SUB dest. , fuente: dest. = dest. - fuente
SBB dest. , fuente: dest. = dest. - fuente - C (C es la bandera C)
Sus pares de operandos posibles son:

```
r,r
r,m
r,c
m,c
m,r
```

Las reglas son:
- ambos deben ser del mismo tamaño en bits
- en la forma m,c se debe indicar el tamaño del operando memoria con BYTE PTR o WORD PTR o DWORD ptr para 8 o 16 o 32 bits.
- no existe la forma m,m ni pueden usarse registros de segmento
- afectan las banderas C,Z,S,Z,P,Z,O según el resultado

Para sumar 4 y 3 se puede usar:

```
MOV AL,3          MOV AL,3          MOV BYTE PTR [200],3
ADD AL,4          MOV BL,4          ADD BYTE PTR [200],4
(resultado en AL) ADD AL,BL         (resultado en [200])
                  (resultado en AL)
```

Las sumas en AL y AX son privilegiadas en tanto tienen codificaciones especiales que son mas cortas y eficientes que las demás.

La suma con acarreo se usa para números multibyte, por ejemplo sumar la word de [200] y [201] a [300] y [301] podría hacerse así:

```
MOV AL,[200]
ADD [300],AL
MOV AL,[201]
ADC [301],AL
```

En ese caso en particular, es preferible usar una operación de 16 bits:

```
MOV AX,[200]
ADD [300],AX
```

En un número multibyte de tres bytes, se puede hacer;

Con operaciones de byte:

```
MOV AL,[200]
ADD [300],AL
MOV AL,[201]
ADC [301],AL
MOV AL,[202]
ADC [302],AL
```

Combinando bytes y words:

```
MOV AX,[200]
ADD [300],AX
MOV AL,[202]
ADC [302],AL
```

La instrucción de resta SUB es similar a ADD; por ejemplo para restar 3 de 4; MOV AL,4

```
SUB AL,3
```

SBB tiene un uso similar a ADC; por ejemplo restar de la word de 300 la word de 200: MOV AL,[200]

```
SUB [300],AL
MOV AL,[201]
SBB [301],AL
```

II.4.5.- NEG dest.

Calcula 0 - dest. en dest. y actualiza banderas.

II.4.6.- Instrucciones de control del Acarreo

Fijan el valor del acarreo

CLC Fija la bandera C en 0
STC Fija la bandera C en 1
CMC Fija la bandera C en su complemento; si esta en 1, la fija en 0 y si esta en 0, la fija en 1.

Un uso de esta instrucción es el manejo uniforme de operaciones multibyte; por ejemplo para sumar dos números de tres bytes en [200] y [300]

```
CLC
MOV AL,[200]
ADC [300],AL
MOV AL,[201]
ADC [301],AL
MOV AL,[202]
ADC [302],AL
```

De esta manera, todas las operaciones son ADC.

II.4.7.- Incrementos y decrementos.

Suman y restan 1

INC dest.: dest. = dest. + 1
DEC dest.: dest. = dest. - 1

El operando puede ser r o m. Afectan las banderas Z,S,C,P,A,O según el resultado. La bandera C queda invariante; esta es una consideración para operaciones multibyte. Esta es la única diferencia importante con ADD dest.,1 y SUB dest.,1 que también suman y restan 1 al operando destino.
INC y DEC están creadas para la actualización de punteros en operaciones de representaciones multibyte. Por ejemplo, para sumar dos números de tres bytes:

```
CLC
MOV SI,200
MOV DI,300
MOV AL,[SI]
ADC [DI],AL
INC SI
INC DI
MOV AL,[SI]
ADC [DI],AL
INC SI
INC DI
MOV AL,[SI]
ADC [DI],AL
```

Este programa muestra repeticiones que pueden ser programadas en instrucciones de ciclos, idealmente:

```
CLC
MOV SI,200
MOV DI,300
repetir 3 veces
   MOV AL,[SI]
   ADC [DI],AL
   INC SI
   INC DI
fin repetir
```

II.4.8.- Comparación.

Compara dos operandos dejando el resultado en las banderas

CMP dest. , fuente: sigue exactamente las reglas de SUB, pero el operando destino no es afectado. Solo se afectan las banderas y en base a ellas, el siguiente grupo de instrucciones hace ciertas "tomas de decisiones". Por ejemplo, CMP AL,BL calcula la resta de BL a AL y deja tanto AL como BL sin cambios. Si son iguales, la bandera Z queda en 1 indicando que el resultado es 0; si no, queda en 0 indicando que el resultado es diferente de 0 por que son desiguales.

II.4.9.- Bifurcaciones.

Fija un nuevo valor para IP según una condición.

JMP dir. Bifurca a una dirección incondicionalmente. Por ejemplo, el ejercicio anterior podría hacerse así:

```
0100   CLC
0101   MOV SI,200
0104   MOV DI,300
0107   MOV AL,[SI]
0109   ADC [DI],AL
010B   INC SI
010C   INC DI
010D   JMP 0107
```

aunque no dispone de un mecanismo de control. Se usara también el termino "salto" para denotar una bifurcación.

JZ dir. Si Z=1, el programa pasa a la dirección indicada <dir> o continua a la siguiente instrucción. Tiene el nombre equivalente JE. Por ejemplo, un problema de programación típico como:

```
si AL<>0 entonces
  BL=1
si-no
  BL=0
fin si
```

se esquematizaría así:

```
 comparar AL con 0
 si es 0 , saltar a ES_CERO
 BL=1
 saltar a FIN
ES_CERO: BL=0
FIN:
```

ES_CERO y FIN son "etiquetas" que marcan puntos de programa. Se siguen de dos puntos (:) para denotar esa condición. Se implementaría así:

```
0100   CMP AL,0
0102   JZ 0108          si AL es igual a 0, AL-0 es también 0 y Z=1
```

```
0104   MOV BL,1          <bloque si>

0106   JMP 010A          es necesario saltar al fin del programa
                         en otro caso, continuaría al bloque si-no

0108   MOV BL,0        <bloque si-no>

010A   INT 3
```

En la programación en TD, es laborioso ver las direcciones finales que ocuparan ciertos puntos del programa, y puede empezarse programando así:

```
0100   CMP AL,0
0102   JZ 0102         se ponen las direcciones de bifurcación
0104   MOV BL,1          en ellas mismas
0106   JMP 0106
0108   MOV BL,0
010A   INT 3
```

Una vez que se identifican en 10A y en 108 como las etiquetas, se re-escriben con los comandos A 102 y A 106.

JNZ dir. Si Z=0, el programa pasa a la dirección indicada <dir> o continua a la siguiente instrucción. Para el problema de la suma de números de tres bytes, se aplica la siguiente técnica:

```
0100   CLC
0101   MOV SI,200
0104   MOV DI,300
0107   MOV CL,3        CL se fija en 3
0109   MOV AL,[SI]
010B   ADC [DI],AL
010D   INC SI
010E   INC DI
010F   DEC CL          se decrementa CL
0110   JNZ 109         si CL llego a 0, el programa debe terminar
                           si no, repetir el ciclo.
0113   INT 3
```

Nótese cuan importante es que INC y DEC no afecten el acarreo; si lo hicieran, no podría transmitirse el acarreo de la suma de 10B ADC [DI],AL para el siguiente ciclo.
Las instrucciones de bifurcación pueden usarse en cualquier parte y dependen solo de las banderas, pero si se usan inmediatamente a una instrucción CMP dest.,fuente, se puede considerar:

	Relación de dest. a fuente	Bifurcación	Traducción	Nombres equivalentes
Para números en general:	=	JZ	(jump zero)	JE (equal)
	<>	JNZ	(jump no zero)	JNE
Para números sin signo:	>	JA	(jump above)	JNBE
	>=	JAE	(jump above equal)	JNB
	<	JB	(jump below)	JNAE
	<=	JBE	(jump below equal)	JNA
Para números con signo:	>	JG	(jump greater)	JNLE
	>=	JGE	(jump greater equal)	JNL
	<	JL	(jump less)	JNGE
	<=	JLE	(jump less equal)	JNG

Otras son:

Instrucción /equivalentes	bifurca si:
JC / JBE	C=1
JNC / JNB	C=0
JP / JPE	P=1
JNP / JPO	P=0
JO	O=1
JNO	O=0
JS	S=1
JNS	S=0

Es importante hacer notar que todas las bifurcaciones condicionales solo pueden ser a direcciones que estén 128 bytes abajo o 127 bytes arriba del IP luego de la instrucción. Para efectuar saltos mas largos se debe usar una bifurcación incondicional. Por ejemplo si desde 100 se quiere bifurcar a 800 si Z=1:

```
0100 JNZ 0105    se salta el salto con la condición contraria
0102 JMP 0800    salto incondicional
0105 .. siguiente instrucción
```

LOOP dir. Es otra instrucción de bifurcación muy importante y útil; decrementa CX, y si es diferente de 0, bifurca y si no, continua a la siguiente instrucción. No afecta las banderas pero solo puede saltar a direcciones que estén 128 bytes abajo o 127 bytes arriba del IP luego de la instrucción.

```
0100   CLC
0101   MOV SI,200
0104   MOV DI,300
0107   MOV CX,3        CX se fija en 3. Necesariamente se usa CX
010A   MOV AL,[SI]
010C   ADC [DI],AL
```

```
010E    INC SI
010F    INC DI
0110    LOOP 10A        se decrementa CX y si CX llego a 0, el programa debe terminar si no, repetir el ciclo.
0112    INT 3
```

JCXZ dir. es una bifurcación que opera cuando CX es 0. No depende de banderas ni las afecta y esta también restringida a -128 y +127 bytes desde IP.

LOOPE dir. Como LOOP, decrementa CX y continua si CX<>0 y además si Z=1
/LOOPZ Se espera que una instrucción actualice previamente las banderas.

LOOPNE dir. Como LOOP, decrementa CX y continua si CX<>0 y además si Z=0
/LOOPNZ Se espera que una instrucción actualice previamente las banderas.

II.4.10.- Multiplicaciones y Divisiones.

Multiplican y dividen el acumulador

MUL op. : Multiplica el acumulador por el operando fuente.
Sus reglas son:
El operando puede ser r o m. Si es de 8 bits, calcula AX=AL * op. Si es de 16 bits, calcula DX,AX=AX * r/m. Si es de 32 bits, calcula EDX,EAX=EAX * r/m.
Trata los operandos como números sin signo. Las banderas C y O indican con 1 si la parte alta (AH para operaciones de bytes y DX para operaciones de word) es diferente de 0. Las demás quedan indefinidas.
Nótese que no se aceptan valores inmediatos como operandos. Un ejemplo típico es el calculo del factorial de CX en AX:

```
0200 MOV AX,1
0203 JCXZ 0209
0205 MUL CX
0207 LOOP 0203
0209
```

IMUL op. : Es similar a MUL pero trata los operandos como números con signo.

DIV op. : Divide el acumulador por el operando fuente. Sus reglas son:

El operando puede ser r o m. Si es de 8 bits, calcula AX / op. dejando el cociente en AL y el residuo en AH. Si es de 16 bits, calcula DX,AX / op. dejando el cociente en AX y el residuo en DX.
Cuando el cociente no es de la dimensión esperada, la instrucción responde ejecutando la "interrupción 0". Trata los operandos como números sin signo. Las banderas quedan indefinidas.

IDIV op. : Es similar a DIV pero trata los operandos como números con signo.

II.4.11.- Rutinas. Llamada y retorno.

CALL dir. empila la siguiente dirección y bifurca a la rutina de la dirección.

RET desempila una dirección y bifurca a ella.
Por ejemplo, para obtener el producto de los factoriales de 2 y 3
usando el programa de calculo de factorial de CX en AX;

```
0100 MOV CX,2
     CALL 200
     MOV BX,AX
     MOV CX,3
     CALL 200
     MUL BX
```

Se debe incluir:

```
0209 RET
```

Ejercicios Usando TD (Grupo 6)

1.- Cargar los siguientes operandos destinos con lo siguiente:
- AL con BL
- AX con BX
- AX con el contenido de las posiciones 100 y 101

Cuidar que la parte mas alta de memoria se debe cargar en AH y la mas baja en AL. Usar instrucciones de byte y de word.
- AX con el contenido de las posiciones 100 y 102
- AL con el byte apuntado por BX
- AX con la word apuntado por BX
- AH con 0
- AX con el byte apuntado por BX
- SI con la base de la estructura situada en 200
- CH con el byte apuntado por la word apuntada por SI
- los 32 bits DX,AX (DX es la parte alta) con el número 140000 (expresado en decimal)

2.- Cargar en AL el byte apuntado por segmento y desplazamiento en 0:0. Indicación: cargar 0 en el segmento DS, acceder al desplazamiento 0 y cargar el desplazamiento y segmento

(parte superior en la memoria mas alta) en SI y ES y luego acceder con ellos usando la sobre-escritura de segmento.

3.- Intercambiar el contenido de AL con AH usando BL como intermediario.

4.- Un vector de bytes V [1..10] se carga en la posición DS:43. Si AL contiene la variable I, acceder a V[I].

5.- El registro r se carga en DS:50 con la siguiente estructura:

```
record
  x:byte
  y:word
  z:longint
end
```

Como se accedería a r.x, r.y, r.z para cargarlos en AL, AX y DS,AX respectivamente ?

6.- Como se respondería a la pregunta en el ejercicio 5 si el inicio de r estuviera apuntado por DS:SI ?

7.- Si un arreglo de registros del tipo de r (ver ejercicio 5) estuviera localizado en DS:200, como se accedería a r.x[i],
r.y[i], r.z[i] para cargarlos en AL, AX y DS,AX respectivamente ?

8.- Intercambiar AL con SI suponiendo que SI tiene un valor menor a FF.

9.- Intercambiar DS y ES.

10.- Sumar y restar 3 y 4 en AL y AH respectivamente

11.- Sumar y restar los bytes de 180 y 181 con los de 200 y 201 considerándolos como bytes individuales dejando el resultado en 300-301.

12.- Sumar y restar las palabras [180]-[181] y [200]-[201] usando solo sumas y restas de bytes dejando el resultado en [300]-[301].

13.- Similar al anterior, pero con instrucciones de words.

14.- Similar al anterior, pero dejando el resultado en [180]-[181]

15.- Sumar y restar los números de tres bytes [180]-[182] y [200]-[202] usando sumas de bytes y words. Dejar el resultado en [180]-[182] y luego en [300]-[301].

16.- Sumar el byte sin signo en [200] y una word sin signo en [204] dejando el resultado en AX.

17.- Restar el byte sin signo en [200] de una word con signo en [204] dejando el resultado en AX.

18.- Sumar un byte con signo en [200] con una word con signo en [204] dejando el resultado en [204].

19.- Multiplicar los bytes de [200] y [300] y dejar el resultado en [400].

20.- Multiplicar los bytes de [180] y [200] por el contenido de AL dejando los resultados en ellos mismos.

21.- Cargar los registros de propósito general con valores diferentes, salvarlos en la pila y luego recuperarlos.

22.- Investigar operativamente si SP se decrementa antes o después de efectuada la instrucción PUSH. Indicación: Usar PUSH SP.

23.- Usando instrucciones PUSH y POP, intercambiar AX, BX, CX y DX entre si.

24.- Insertar en la pila el área DS:[10] a DS:[13] y luego recuperarla en DS:[100] a DS:[103].

25.- Insertar en la pila el área DS:[10] a DS:[12] y luego recuperarla en DS:[200] a DS:[202]. Notar que la instrucción POP restituye words y el tamaño del área (3 bytes) no contiene word enteras.

26.- Usando instrucciones XCHG, intercambiar AX, BX, CX y DX entre si.

27.- Intercambiar los bytes DS:[10] y DS:[110] usando XCHG.

29.- Intercambiar las áreas DS:[180] a [183] con DS:[200] a [203].

30.- Usar la instrucción LEA para cargar en SI el desplazamiento apuntado por BX+CX.

31.- Obtener en AX todas las banderas.

32.- Fijar todas las banderas en 0.

33.- Buscar una coma en la cadena ASCIIZ a partir de [200]. Dejar DX en 1 si es así, en otro caso, en 0.

34.- En [200] se tiene una cadena donde el primer byte indica su longitud y a continuación esta la cadena. Determinar el AL el byte que sumado con todos los bytes de la cadena da el resultado 0.

35.- Dada una cadena en la posición 200 con la siguiente estructura:

l x x x x x x x

l es la longitud de la cadena y a continuación están los caracteres.
x x x son los caracteres mismos

Por ejemplo: 4 'juan'

a) Cambiar minúsculas por mayúsculas.
b) Verificar si la letra "A" esta dentro de la cadena dejar ZF en uno si es así
c) Cambiar las comas por espacios en blanco
d) Eliminar los espacios en blanco iniciales

36.- Dado un arreglo de números de 10 words a partir de [300], buscar secuencialmente 1232h. Dejar CF en 1 si es así y en 0 en otro caso.

37.- Sumar los siguientes 4 números multibyte de 3 bytes colocados a partir de [200] en [210] : 45aaff , 231233 , 331233 , 1111ff.

38.- Llenar el área a partir de [300] con los coeficientes binomiales para la quinta potencia.

39.- Empilar el valor 200 y llamar a una rutina que a su vez pueda usar ese valor como desplazamiento para dejar allí su cuadrado como word.
Indicación: Usar BP y considerar que la llamada a la rutina ha empilado una dirección.

40.- Dejar en [300] como ASCIIZ los divisores enteros del byte [200].

II.4.12.- Introducción al ensamblador

La programación con el depurador no puede si no tener un alcance limitado por la imposibilidad de disponer de efectos de edición adecuados como inserciones y eliminaciones de líneas de programa. Su utilidad radica en la visualización de los programas con sus características de registros, banderas, posiciones de memoria, entre otras, con lo que se puede adquirir confianza en la programación.
La mejor forma de desarrollar programas complejos es mediante el uso de los llamados "ensambladores"; estos son programas que traducen un texto (programa fuente) a un programa ejecutable (programa objeto).
De esta manera, se pueden aprovechar las características de los editores de texto ASCII. Se denomina así el formato mas simple donde no se introducen códigos propios de los editores que mejoran el aspecto de los textos como
negrillas, subrayados y otros.

"Lenguaje de maquina": se denomina conjunto de las instrucciones de un procesador.

El programa ensamblador acepta un lenguaje que incluye todo el lenguaje de maquina ("instrucciones") con ligeras y necesarias variaciones y añade una serie de "directivas", sentencias que tienen sentido en el ámbito del ensamblador y que pierden su identidad en el programa ejecutable.

Al lenguaje total, compuesto de instrucciones y directivas, se llaman "lenguaje ensamblador" ("assembly language") o simplemente "ensamblador" ("assembler").

Usualmente el archivo que contiene un programa ensamblador tiene la extensión ASM; por ejemplo PRUEBA.ASM. Las líneas fuente que contiene deben tener el siguiente formato:

nombre acción expresión ;comentario

El "nombre" es una definición del usuario que crea un símbolo que puede ser utilizado luego en el programa, pueden representar código, datos y constantes; solo debe existir si la "acción" lo requiere y bien puede no existir.

La "acción" es una directiva o una instrucción. Si no existe, la línea debe ser solo un comentario.

La "expresión" son el o los "parámetros" de la acción; por ejemplo operandos de instrucciones.

El "comentario" es opcional y debe ir precedido de punto y coma.

Ejemplos:

P0: MOV AX,BX

Crea el nombre P0; seguido de : es una etiqueta representa código en el punto donde esta la instrucción MOV AX,BX. Antes o después de esta instrucción se puede bifurcar a ella como por ejemplo con JMP P0.

STC

Solo se da la instrucción STC; no existe nombre ni expresión ni comentario.

A DB 100

Crea el nombre A; es una variable y representa un dato. Se reserva espacio de un byte con la directiva DB (Define Byte), se le da el valor inicial 100 (decimal) y el nombre A puede ser usado como operando memoria de instrucciones; por ejemplo se puede mover a AL con MOV AL,A o incrementar con INC A.

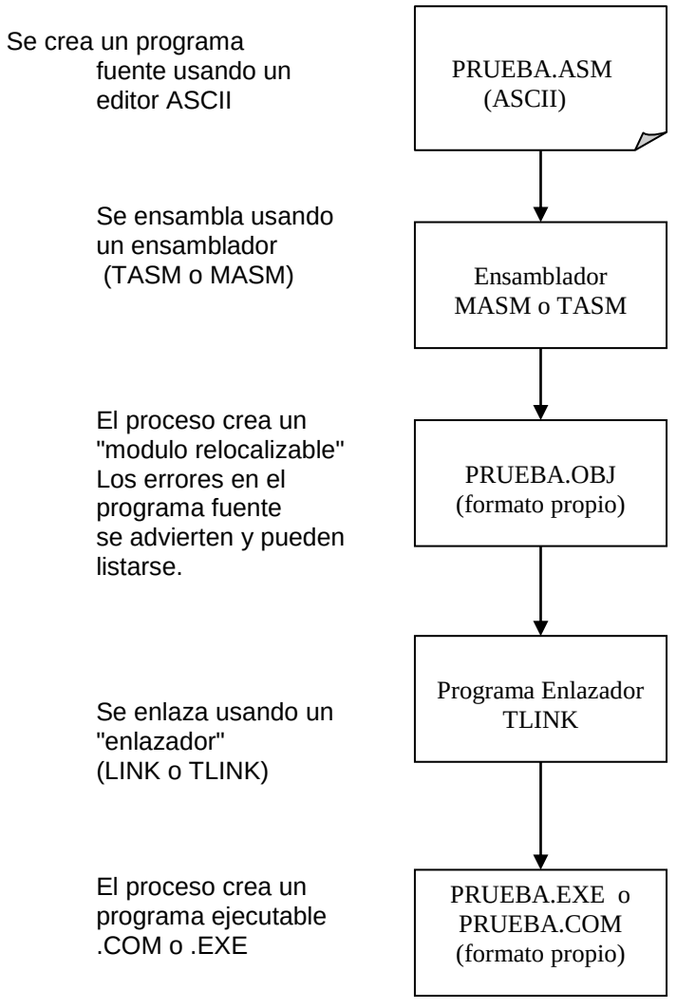
T_PANTALLA EQU 80*25

Crea el nombre T_PANTALLA; es una constante con el valor 2000 (resultado de 80*25). Se puede usar por ejemplo como operando inmediato; MOV CX,T_PANTALLA.

;**********

Denota con asteriscos (*) una línea divisoria que en realidad es ignorada por tratarse de un comentario al empezar con ;.

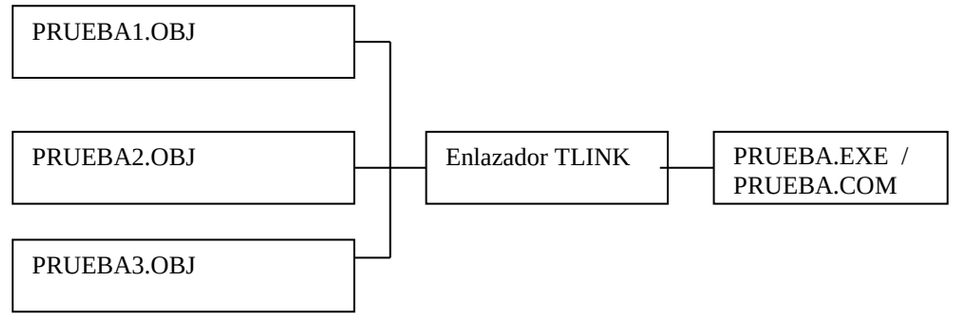
El proceso de creación de programas es el siguiente:



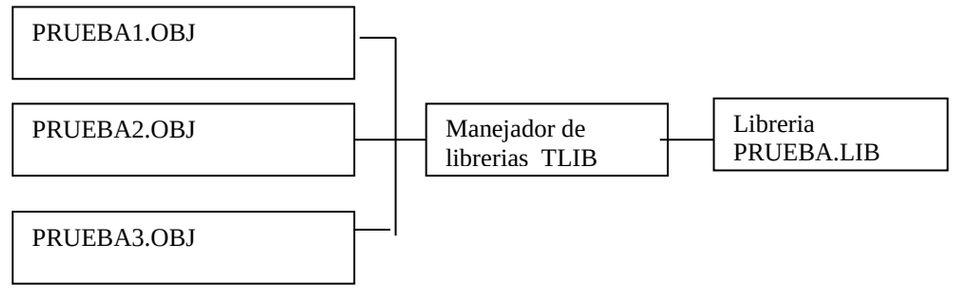
El programa ejecutable puede depurarse con TD y también ejecutarse desde el procesador de comandos.

La introducción del "enlazador" ha sido creada para efectivamente conectar varios programas. Existen las siguientes posibilidades:

- Usar varios módulos relocalizables para crear un programa ejecutable

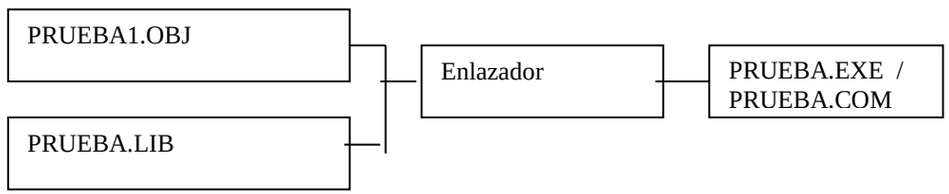


- Crear librerías de varios módulos relocalizables



La "librería" simplemente es un archivo que contiene a los módulos relocalizables. Es una facilidad orientada a no tener que manejar varios programas relocalizables individualmente, si no mas bien referirse a una librería con todos ellos.

- Enlazar un modulo relocalizable (o varios) con una librería para crear un programa ejecutable



Complementariamente existen:

- Listado del programa, generado por el ensamblador mostrando el código generado, símbolos, errores y referencias cruzadas (.LST).

- Mapa de memoria, generado por el enlazador, con direcciones del programa ejecutable (.MAP).

La mayor facilidad que se introduce es la definición de símbolos. Siempre deben empezar con una letra, componerse de letras, números o los símbolos ?,@,_ y $, pueden tener cualquier longitud aunque solo se consideran los primeros 31 caracteres. Los símbolos pueden ser:

- variables. Se definen con la siguiente sintaxis:

```
símbolo dx [ ? / ## / 'ccc' / nn DUP ( ## / 'ccc'  / ?) ]
```

dx puede ser db (byte), dw (word) , dd (double words), df (six word), dq (quadruple word), dt (ten byte)

Se puede indicar a continuación ningún, un o varios valores iniciales separados por comas. En blanco o ? indica ningún valor. Esto quiere decir que el valor no interesa y puede ser cualquiera.

Se puede inicializar con números o cadenas. Los números deben empezar en un dígito decimal y pueden ser binarios (terminados en B), hexadecimales (terminados en H) y decimales (no necesitan terminación, aunque puede usarse D). Las cadenas deben estar entre comillas simples. La misma comilla simple se representa por dos comillas. nn DUP (ccc) repite nn veces el contenido ccc entre paréntesis.

Una variable puede ser llamada por su nombre simplemente:

```
x db 0,0,0

mov x,al
```

En este ejemplo, por declararse un byte, las operaciones que la involucren son operaciones de tamaño byte, por ejemplo

```
mov x,0
```

no necesita ser

```
mov byte ptr x,0
```

Pero también se puede usar ptr para cambiar el tamaño de la variable, así

```
mov word ptr x,0
```

lleva 0 a la posición de x y su inmediata siguiente.

También se puede sumar o restar una constante para referirse a las posiciones contiguas.

```
mov x+1,bl
```

Se puede obtener la posición de la variable con el operador OFFSET

```
mov bx,OFFSET x
```

hace que bx apunte a la variable x. También se puede sumar o restar constantes a un OFFSET y obtener la diferencia entre dos OFFSETS.

Normalmente, las variables pueden declararse al final del programa constantes

Se pueden realizar operaciones aritméticas de suma, resta, multiplicación y división entre números determinando constantes que se pueden usar como operandos en modo inmediato, por ejemplo: mov cx,80*25.

Se definen con la siguiente sintaxis:

```
símbolo [=/EQU] constante numérica o cadena
```

se pueden usar en el programa asumiendo que cada vez que aparece el símbolo de la izquierda es reemplazado por la constante numérica o cadena de la derecha. Por ejemplo:

```
pantalla equ 0b800h

mov ax,pantalla        se convierte en      mov ax, 0b800h
```

- etiquetas

Se declaran con la siguiente sintaxis:

```
símbolo :
```

Se usan para bifurcaciones como en el siguiente ejemplo:

```
ciclo:   mul cx
         loop ciclo
```

Se pueden introducir comentarios en cualquier línea de programa colocando el carácter punto y coma (;) previamente, por ejemplo:

```
mov ax,0        ; borrar ax
```

Se pueden usar tanto mayúsculas como minúsculas.

Otras facilidades de programación que ofrece el ensamblador son:

- Inserción de prefijos de segmento
- Ensamblado condicional
- Definición y expansión de macros
- Incorporación de archivos

Un formato para un programa .COM con las llamadas 'directivas simplificadas' puede ser:

```
.model tiny
.code
.386
.387
org 100h
inicio:
.
. líneas de programa
.
int 20h              ; final del programa
.
ccc db
    dw
. variables
.

end inicio
```

Debe ser un texto ASCII y tener extensión ASM.
Esta disponible como ESQCOM.ASM (ESQueleto de programa .COM)
El encabezado y el final tienen la siguiente explicación:

```
.model tiny      = modelo de memoria diminuto; ideal para formato .COM
.code            = segmento de código. Util para datos en formato .COM
.386             = usa instrucciones del 386. Si no se usan, mejor no incluir
                   por que los tipos de datos de 32 bits deben ser mejor indicados.
.387             = usa instrucciones del 387. Si no se usan, no incomoda
org 100h         = origen del programa en 100H. Necesaria para formato .COM
inicio:          = etiqueta que marca el principio de programa
end inicio       = fin del programa. Empezar en la etiqueta 'inicio'
```

Por ejemplo, un calculo del factorial de 5 en un archivo llamado FACTOR.ASM puede ser:

```
.model tiny
.code
org 100h
inicio: mov cx, número
        mov ax,1
        jcxz fin
ciclo:  mul cx
        loop ciclo
fin:    mov factorial,ax
        int 20h
número dw  5
factorial dw ?
end inicio
```

Para simplificar el ensamblado puede usarse el siguiente programa C.BAT:

```
rem             Crea programa .COM para depuración o ejecución
@echo off
tasm %1 /z
if errorlevel 1 goto fin
tlink %1 /x/t
if errorlevel 1 goto fin
del %1.obj
cls
echo Ensamblado exitoso
echo Enter para seguir con Turbo Debug o parar con ^C
pause
td %1.com
exit
:fin
echo Se presentaron errores
```

El programa se ensamblaría con C FACTOR.
Si existen errores, se listan en pantalla.
Si no, se puede pasar con Enter a ver el programa en Turbo Debug o terminar con ^C. Puede aparecer un mensaje de confirmación que debe ser contestado afirmativamente.

Turbo Debug tiene la facilidad de mostrar y trazar el programa fuente original.
Para hacerlo es necesario generar un programa .EXE Se puede utilizar el siguiente formato en los archivos .ASM.

```
.MODEL SMALL
.DATA
.
ccc db
    dw
. variables
-STACK nn (nn es el tamaño reservado para la pila)

.CODE
inicio: mov ax,_data    ; se requiere fijar DS
        mov ds,ax       ; al segmento de datos
.
. líneas de programa
.
mov ah,4ch
int 21h              ; final del programa
END inicio
```

Esta disponible como ESQEXE.ASM (ESQueleto programa EXE)
El programa anterior, con las directivas simplificadas es:

```
.MODEL SMALL
.DATA
NÚMERO DW  5
FACTORIAL DW ?
.STACK 10
.CODE
INICIO: MOV AX,_DATA          ; se requiere fijar el DS
        MOV DS,AX             ; al segmento de datos
                              ; No se requiere en programas .COM
        MOV CX,NÚMERO
        MOV AX,1
        JCXZ FIN
CICLO: MUL CX
        LOOP CICLO
FIN:    MOV FACTORIAL,AX
        MOV AH,4Ch      ; se requiere esta forma
        INT 21H         ; para terminar correctamente
END INICIO
```

El programa ensamblador ED.BAT es:

```
rem                 Crea programa .EXE para depuración
@echo off
tasm %1 /z/zi
if errorlevel 1 goto fin
tlink %1 /x/l/v;
if errorlevel 1 goto fin
del %1.obj
td %1.exe
exit
:fin
echo Se presentaron errores
```

Crea un programa .EXE de tamaño mayor al usual con todos los elementos para ser depurado con conexión directa al modulo fuente y la posibilidad de ver las variables tal cual van evolucionando. El programa ensamblador E.BAT que genere el código necesario sin estas características es:

```
rem           Crea programa .EXE para ejecución
@echo off
tasm %1 /z
if errorlevel 1 goto fin
tlink %1 /x
if errorlevel 1 goto fin
del %1.obj
cls
echo Ensamblado exitoso
exit
:fin
echo Se presentaron errores
```

Ejercicios resueltos

1.- Promediar 5 bytes. Dejar el resultado en AL

```
.MODEL TINY
.CODE
ORG 100H
INICIO: MOV CX,5        ;       CX es el contador
        MOV SI,OFFSET a ;       SI apuntara al área a con los bytes a promediar
        MOV AL,0        ;       El resultado quedara en AL
P0:     ADD AL,[SI]
        INC SI          ;       Avance al siguiente byte
        LOOP P0
        MOV AH,0        ;       Debe ponerse en 0 porque la división
                        ;       considera AH como parte alta del dividendo
        MOV DL,5        ;       Se debe usar un registro por que la división
        DIV DL          ;       no acepta un operando inmediato.
        INT 20H
A DB 3,7,9,1Ah,0BCh
END INICIO
```

La suma no excede el tamaño de un byte; por ejemplo si los números fueran AA,FF,CC,DD,22 habría que sumarlos en una word.

```
INICIO: MOV CX,5
        MOV SI,200      ;        Alternativamente:
        MOV AX,0
                        ;                MOV BH,0
P0:     ADD AL,[SI]     ;        P0:     MOV BL,[SI]
        ADC AH,0        ;                ADD AX,BX

        INC SI
        LOOP P0
        MOV DL,5
        DIV DL
```

No debe hacerse ADD AX,[SI] ya que se tomarían dos bytes por vez.

2.- Promediar 5 words. Dejar el resultado en BX

```
.MODEL TINY
.CODE
ORG 100H
INICIO: MOV CX,5
        MOV SI,OFFSET A
        MOV AX,0
P0:     ADD AX,[SI]
        INC SI      ;   Como son words, hay que incrementar dos veces SI
        INC SI
        LOOP P0
        MOV DX,0    ;   Es necesario poner DX en 0 porque la división
        MOV CX,5    ;   lo considera parte alta del dividendo.
        DIV CX
        MOV BX,AX
        INT 20H
A DW 322,721,911,13Ah,10BCh
END INICIO
```

3.- En X se encuentra una lista de words. No se conoce la cantidad exacta de words pero se sabe que el ultima es anterior a la marca FFFF. Promediarlas y dejar el resultado en DX.

```
.MODEL TINY
.CODE
ORG 100H
INICIO: MOV SI,OFFSET X
        MOV AX,0
        MOV CX,0
P0:     CMP [SI],0FFFFh
        JZ FIN
        ADD AX,[SI]
        INC CX
        ADD SI,2          ; Otra forma de incrementar SI en 2.
        JMP P0
FIN:    MOV DX,0
        JCXZ NO_HAY
        DIV CX
NO_HAY: INT 20h
X DW 12,111,223,223,11H,123,22,0FFFFH
END INICIO
```

II.4.13.- Corrimiento y rotación de bits.

Todas las instrucciones siguientes manipulan bits a través del acarreo.

SHL dest. ,fuente SHR dest., fuente SAL dest., fuente ROR dest., fuente ROL dest., fuente RCR dest., fuente RCL dest., fuente	dest. puede ser un registro o una posición de memoria de tamaño byte o word. fuente puede ser solo 1 o CL. Con CL se mueve una cantidad variable de bits. Desde el 386 puede también ser cualquier constante entera.

Dado un registro de ocho bits y el acarreo:

```
7 6 5 4 3 2 1 0   Acarreo
a b c d e f g h   C
```

las instrucciones y sus efectos son:

```
SHL   <-    b c d e f g h 0   a      Shift Left
SHR   ->    0 a b c d e f g   h      Shift Right
SAR   ->    a a b c d e f g   h      Shift Aritmetic Right
ROL   <-    b c d e f g h a   a      Rotate Left
ROR   ->    h a b c d e f g   h      Rotate Right
RCL   <-    b c d e f g h C   a      Rotate Left with carry
RCR   ->    C a b c d e f g   h      Rotate Right with carry
```

El corrimiento a la derecha es equivalente a una división por 2 y el corrimiento a la izquierda, a una multiplicación. SAR ha sido creado para mantener el signo, con lo que sirve como una división de números con signo. Las banderas S,Z,P,C se actualizan, O solo se actualiza en operaciones unitarias y A queda indefinida.
En las rotaciones, A,S,Z,P no se alteran, C se actualiza y O solo se actualiza en operaciones unitarias.

II.4.14.- Operaciones lógicas.

Realizan operaciones lógicas bit a bit entre operandos o sobre un operando destino.

AND dest., fuente OR dest., fuente XOR dest., fuente NOT dest. TEST dest., fuente	Ambos operandos deben ser del mismo tamaño, solo uno puede ser memoria, no se pueden implicar registros de segmento y se acepta como fuente un operando inmediato. TEST realiza un AND pero solo actualiza banderas y no operandos.

Con TEST, AND, OR y XOR las banderas C y O siempre terminan en 0, S,Z y P se actualizan y A queda indefinida.
NOT no actualiza ninguna bandera.
Muchas veces se usa XOR reg,reg para dejar reg en 0.

II.4.15.- Interrupciones.

INT n ocasiona una llamada a la "interrupción" n , donde n es un número en el rango 0 a FFH. Las interrupciones están direccionadas desde el área 0:0 y cada una ocupa 4 bytes (2 para segmento en la parte alta y 2 para el desplazamiento en la parte baja). INT ejecuta una llamada lejana a ese punto empilando segmento y desplazamiento de la siguiente dirección a la invocación. También empila la banderas y luego fija T=0 e I=0.

IRET Es el retorno de una interrupción. Restituye las banderas empiladas con INT y ejecuta un retorno lejano a la siguiente instrucción a INT.

El propio DOS presta sus servicios a través de interrupciones, en particular la INT 21H presta lo siguiente:

Valor para el registro AH	Servicio
0	Termina el programa
1	Lee un carácter de teclado al registro AL y lo despliegaen pantalla como ASCII
2	Despliega el carácter de DL como ASCII. Usa AL internamente
7	Lee un carácter de teclado al registro AL .No lo despliega en pantalla.
9	Despliega la cadena apuntada por DS:DX terminada en $
0A	Lee la cadena estructurada apuntada por DS:DX. 0Dh queda al final de la cadena leída, por lo que hay que reservar un byte adicional.

Control C (^C) termina los programas si es ejecutado en medio de algunos servicios de INT 21H. La función 7 es una excepción y no es afectada por ^C.
Ejemplo: leer un carácter y terminar:

```
100     MOV AH,1
        INT 21h
        MOV AH,0
        INT 21h
```

INT 20h también puede ser usada para terminar un programa, aunque solo esta restringida al formato .COM

Ejercicios resueltos

1.- Leer una cadena de longitud fija

```
mov ah,1
mov cx,10
mov si,offset cadena
p0:int 21h
mov [si],al
inc si
loop p0
int 20h
cadena  db 10 dup ( ? )
```

2.- Desplegar una cadena Asciiz

```
mov ah,2
mov si,offset cadena
p0: mov dl,[si]
   or dl,dl
   jz fin
   int 21h
   inc si
   jmp p0
fin: int 20h
cadena  db 'roberto va a casa',13,10,0
```

3.- Desplegar el binario de AL

```
inicio:
mov ah,2
mov al,11001010b
mov cx ,8
p0: mov dl,'0' shr 1
   rol al,1
   rcl dl,1
   push ax     ; el servicio 2 destruye el valor de AL
   int 21h
   pop ax
   loop p0
int 20h
```

4.- Convertir 4 bytes a un hexa en AX

```
mov si,offset área
call cnv            ; llama a rutina de conversión de un byte
```

```
mov ah,bh
call cnv          ; llama a rutina de conversión de un byte
mov al,bh
int 3

cnv: call cnv1        ; llama a rutina de conversión de un nible
     mov cl,4
     shl bl,cl
     mov bh,bl
     call cnv1        ; llama a rutina de conversión de un nible
     or bh,bl
     ret

cnv1: mov bl,[si]
      inc si
      cmp bl,'9'
      ja p0
      and bl,0fh
      ret
p0:   cmp bl,'a'
      jb short p1
      sub bl,'a'-10
      ret
p1:   sub bl,'A'-10
      ret
área db '1a3b'
```

5.- Leer un número de 0 a 99 en decimal y llevarlo a binario en AL

```
inicio: call lee_num0
    mov cl, área
    xor ch,ch
    mov al,ch
    jcxz fin
    mov bl,10
    mov si,offset area+1
p0: mul bl
    mov bh,[si]
    and bh,0fh
    add al,bh
    inc si
    loop p0
fin: int 3

lee_num0:  mov si,offset area+1
    mov area,0
lee_num: mov ah,7
    int 21h
    cmp al,8
    jnz enter?
back: cmp area,0        ; caso de presionarse backspace
    jz lee_num
    mov ah,2
    mov dl,al
    int 21h
    mov dl,' '
    int 21h
    mov dl,8
    int 21h
    dec área
    jmp lee_num
enter?: cmp al,13       ; caso de presionarse enter
    jnz número?
    ret
número?:                ; caso de presionarse números
    cmp al,'0'
    jb lee_num
    cmp al,'9'
    ja lee_num
    cmp area,2
    jnz nuevo
    ret
nuevo: mov [si],al
    inc si
    inc área
    mov dl,al
    mov ah,2
    int 21h
    cmp area,2
    jnz lee_num
    ret
área db 0,0,0
```

6.- Desplegar AL como byte con signo

```
inicio: mov bl,10
    mov si,offset área
    or al,al
    jns positivo
    neg al
    mov byte ptr [si],'-'
    inc si
```

```
    mov dl,4
    jmp p0
positivo: mov dl,3
p0: mov cx,3
    add si,2
p1: xor ah,ah
    div bl
    or ah,30h
    mov [si],ah
    dec si
    loop p1
    mov cl,dl
    xor ch,ch
    mov si,offset área
    mov ah,2
p2: mov dl,[si]
    int 21h
    inc si
    loop p2
fin: int 3

área db 0,0,0,0
```

Ejercicios Usando TASM (Grupo 7)

Cadenas

1.- Leer cadenas estructuradas y leer cadenas de longitud fija.

2.- Concatenar cadenas
   a) longitud fija con longitud fija
   b) longitud fija con longitud fija con supresión de espacios en blanco
   c) estructurada con ASCIIZ
   d) estructurada con estructurada

3.- Obtener sub-cadenas; parte izquierda, derecha y central de:
   a) cadena estructurada
   b) cadena ASCIIZ
   c) cadena de longitud fija

4.- Asignar cadenas:
   a) estructurada a estructurada
   b) longitud fija a estructurada
   c) estructurada a longitud fija
   d) ASCIIZ a estructurada
   e) ASCIIZ a longitud fija
   f) estructurada a ASCIIZ

Números

5.- Leer una cadena exigiendo que sus caracteres solo sean números y un signo opcional.

6.- Dada una cadena de puros números, convertirla a binario:
   a) en un byte sin signo
   b) en una word sin signo
   c) en un byte con signo
   d) en una word con signo
   Considerar las posibilidades de que se exceda la capacidad del tipo de dato
   y no puede completarse la conversión.

7.- Para los tipos de datos del problema anterior, escribir los programas que los desplieguen en pantalla.

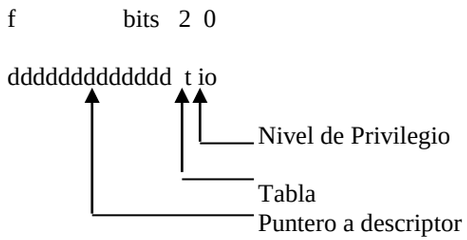
III.- PROGRAMACION AVANZADA

III.1.- MODO PROTEGIDO Y OTRAS CARACTERÍSTICAS DESDE EL 386

La segmentación que presentada en el capitulo anterior se denomina "modo real" o segmentación de 16 bits y apareció con los procesadores 8086 y 8088. Desde el procesador 286 aparece el "modo protegido" con capacidad de acceso a áreas de

memoria mas allá de un Megabyte. En este modo, los registros de segmento son "selectores" de "descriptores" de memoria. El esquema de acceso en modo protegido es el siguiente:

1.- Un selector (registro de segmento) se carga con un valor y su contenido se interpreta como:



Nivel de privilegio: un valor entre 0 y 3 con el que se pretende acceder al descriptor. 0 es el mas alto y 3 el mas bajo.

Tabla: Descriptores locales (1) o globales (0)
Puntero: dirección del descriptor

2.- Se accede al descriptor

Los descriptores globales están en una tabla llamada "tabla de descriptores globales" cuya base es apuntada por un registro llamado GDTR ("global descriptor tasa registre").
Existe un tratamiento similar para los descriptores locales.
La tabla se elige según el byte t; dentro de ella se usa el puntero completando los tres bits bajos con 0 con lo que queda de 16 bits.

Allí debe existir una estructura de descriptor de ocho bytes como sigue:

Base (b24-b31)				G	D	0	A	Limite (l 16-l 19)
P	IO	S	Tipo y acceso	Base (b16 – b 23)				
Base (b0 a b15)								
Limite (l 0 a l 15)								

En primer lugar, se verifica que el nivel de privilegio puesto en el selector sea al menos el del byte 5 (derechos de acceso) del descriptor. Si no es así, se genera un tratamiento denominado "excepción".

En el 286 no se usan los dos bytes mas altos aunque deben existir.
La base es la dirección de inicio del segmento en 32 bits (b0 a b31) o 24 bits en el 286. A este tipo de dirección se denomina "lineal" ya que esta sobre toda la memoria direccionable; 4 Gb desde el 386 o 16 Mb en el 286.
Sobre esta base se tiene acceso a una longitud máxima dada en los bits de longitud, 16 bits para el 286 (l0 a l15) y 20 bits desde el 386 (l0 a l19), donde además, se puede usar el bit de granularidad para dar una longitud de
segmento de 4 Gbytes. Toda la información del descriptor es copiada al procesador a una llamada "estructura invisible" de registros de tal manera que solo se accede a las tablas de descriptores solo cuando se cambia el valor del selector.

Los campos de bits del descriptor son los siguientes:

P = Presente: 1 si existe en la memoria, 0 si no.

IO = Nivel de privilegio mínimo para acceder al descriptor.

S = 1 para descriptores de segmentos de datos, código y pila.
0 para descriptores de segmentos del sistema, llamados también "puertas"

Tipo = Para segmentos de datos, código y pila sigue una codificación de tres bits que indica si el segmento crece hacia arriba (datos y código) o hacia abajo (pila), si puede leerse y/o escribirse en el, y si se "conforma" o no; esto es, si puede accederse con un nivel de privilegio superior o solo con el mismo. Para segmentos del sistema, sigue una codificación de cuatro bits que designa elementos de un ambiente de operación multiusuario y multitarea, por ejemplo:

- segmentos de estados de tareas, que son áreas con ambientes completos de procesador (todos los registros) correspondientes a una tarea.
- puertas, direcciones de tratamiento de excepciones y acceso a servicios del sistema.

Acceso = 0 si no se accedió antes, 1 en otro caso. No existe en puertas.

G = Granularidad; si el limite se especifica en bytes o en unidades de 4K.

D = "Default"; tamaño de los registros por omisión; 0 para registros de 16 y 1 para registros de 32 bits.

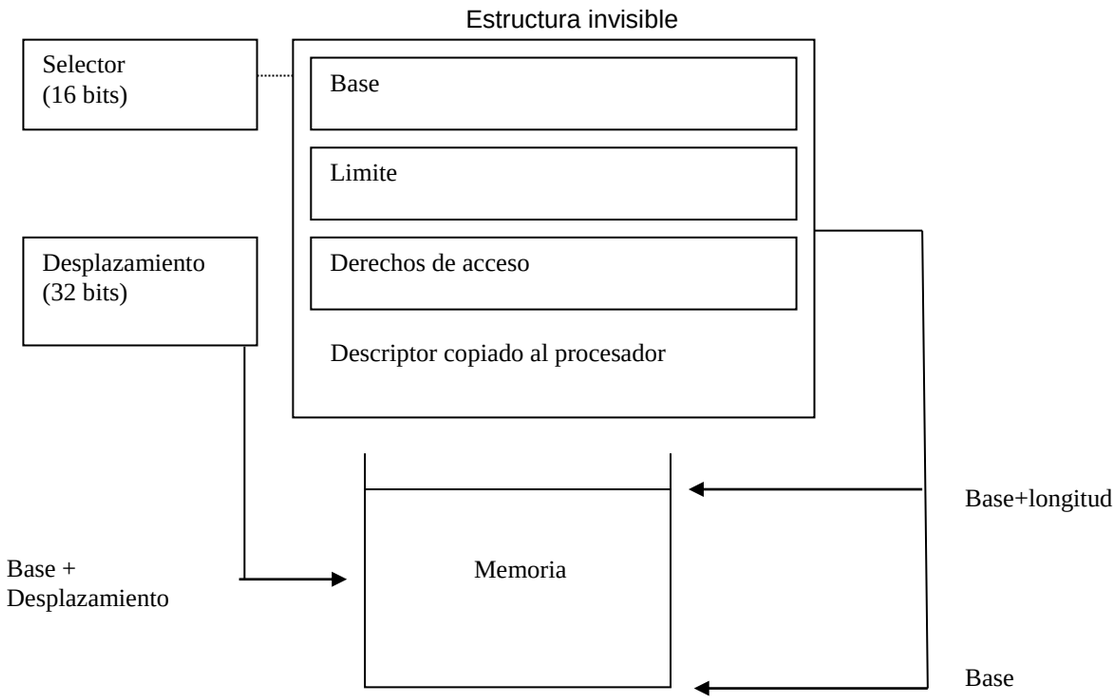
A = "Avail" Disponible para cualquier uso de programadores de sistemas operativos.

3.- Se especifica un desplazamiento.

Desde la base a la que hace referencia el selector, en el sentido de crecimiento del segmento según el descriptor, se obtiene una dirección lineal para acceder a memoria. Es efectiva si:

- El nivel de acceso del procesador satisface el nivel de acceso del segmento
- El desplazamiento no excede el limite del segmento.

En otro caso se genera un tratamiento de excepción.



Al activarse cualquier procesador INTEL ingresa al modo real. Se debe escribir un programa para entrar al modo protegido. El 286 necesita ser reiniciado para volver al modo real, mientras que desde el 386 es posible retornar al modo real por programación.

En el modo protegido, las instrucciones a ejecutarse se acceden según EIP y CS como selector.

Las banderas también se extienden al registro extendido EFLAGS de 32 bits:

Bits

12	11	10		0E	0D	0B	0A	09	08	07	06		04	02		00
	L	V	R		N	IO	O	D	I	T	S	Z		A	P		C

Las nuevas banderas son mas aplicables en ambientes multitarea y/o multiprogramación. Tienen los siguientes significados:

IO: ("Input/Output Privileges") Nivel de privilegio de entrada/salida en el rango de 0 a 3.
N : ("Nested Task") Indica si la tarea actual esta anidada dentro de otra.
R : ("Resume Flag") Indica si la ejecución de instrucciones debe ser o no reasumida luego de interrupciones de depuración.
V : ("Virtual Mode") Indica que dentro del modo protegido, se pase a simular el modo real "virtualmente".
L : ("Align") Aparece desde el 486. Indica si el procesador ha accedido una word en una dirección impar o una doble word en una posición que no es múltiplo de 4. Los programas para 486 se ejecutan mejor cuando se buscan words en posiciones pares y dobles words en posiciones múltiplos de 4.

N e IO ya existían en el 286.

Además, desde el 386 existen tres conjuntos nuevos de registros:

- De control: CR0,CR1,CR2 y CR3, todos de 32 bits.
  CR0 es también llamado MSW en el 286 (Machine Status Word) y contiene los siguientes bits:

PG	00000000	ET	TS	EM	MP	PE

PG (Paging) en 1 activa la 'paginación' y permite la traducción de la tabla de paginas en direcciones físicas. La tabla de paginas tiene formato propio y esta apuntada por CR3

PE (Protection Enabled) en 1 activa el modo protegido. Se debe cumplir con preparar la tablas de descriptores.

ET (Extension Type) en 1 indica que esta instalado un coprocesador 387 o superior

TS (Task Switched) en 1 indica que se ha efectuado un cambio de tareas

EM (Emulation) en 1 indica que el coprocesador esta emulado por software y cada instrucción ESC ocasiona una interrupción 7.

MP (Monitor Processor) en 1 indica que el coprocesador matemático esta presente.

CR1 es invisible y esta reserva por Intel para proyectos futuros

CR2 contiene la ultima dirección de falla de pagina

CR3 en los 20 bits mas altos contiene la base de la tabla de paginas

- Direcciones del sistema:

GDTR de 48 bits: los 32 bits altos contienen la dirección lineal de inicio de la tabla de descriptores globales y los 16 bajos contiene la cantidad de bytes usados por la tabla.

IDTR similar a GDTR, para los descriptores de interrupciones

LDTR de 16 bits, contiene el selector de la tabla local de descriptores que se ubica en la GDTR

TR de 16 bits, contiene el selector de la TSS (Task Status Segment), un segmento con formato propio e información sobre la tarea actual.

- De depuración: DR0 a DR7 todos de 32 bits.

DR0 a DR3 contienen direcciones lineales de parada que son comparada con las direcciones del EIP cada vez que se ejecuta una instrucción

DR4 y DR5 están reservados por Intel

DR6 y DR7 se componen de 30 campos de bits que deciden exactamente cual será el tratamiento de las direcciones lineales de parada.

- De verificación: TR6 y TR7 de 32 bits ambos. Contienen 10 campos de bits para probar a TLB (Translate Lateral Buffer) que es una tabla con las 32 entradas mas usadas de la tabla de paginas.

El siguiente programa activa el 'modo real plano', que permite desde modo real acceder mas allá de un Mb. La técnica consiste en activar el modo protegido y fijar en un segmento de 32 bits toda la memoria para luego volver al modo real.
De este modo, la estructura invisible ya esta fijada.
Prueba que localidades muy lejanas de la memoria están disponibles leyendo un carácter, llevándolo allí y luego desplegándolo en pantalla.

```
.model tiny
.code
; se necesitan instrucciones del modo protegido
.386p
org 100h
inicio: mov eax,cr0     ; verificar que no este en modo protegido
        test ax,1
        jz procede
        jmp en_modo_p
procede: xor eax,eax
        mov ax,ds
        shl eax,4
        xor ebx,ebx
        mov bx,offset gdt
        add eax,ebx             ; prepara mem48 para contener GDTR el
        mov dword ptr mem48[2],eax ; puntero de la tabla de tareas globales GDT
        lgdt fword ptr mem48       ; carga GDTR

        push ds
        cli
        mov eax,cr0     ; fija modo protegido
        or eax,1
        mov cr0,eax
        jmp p0          ; elimina la cola de instrucciones
p0:
        mov bx,8        ; fija los demás segmentos a la entrada 1 de la GDT
        mov gs,bx
        mov fs,bx
        mov es,bx
        mov ds,bx
        and al,0feh     ; cancela modo protegido
        mov cr0,eax
        jmp p1
p1:     sti

        pop ds

        mov edi,1000000h  ; SI apunta a un lugar mas allá de 1 Mb.
        xor ah,ah       ; lee un carácter
        int 16h
        mov [edi],al    ; lo lleva al lugar lejano
        mov dl,[edi]    ; lo vuelve a traer
        mov ah,2        ; lo despliega
        int 21h
        jmp fin
en_modo_p:
```

```
        mov dx, offset er
        mov ah,9
fin:    mov ah,4ch
        int 21h

mem48 dw 16     ; GDTR con limite de 16 bytes (2 entradas; 0 y 1).
      dd        ; La base la añadirá el programa
gdt db 8 dup (0); 2 entradas de descriptores. La primera (0) es nula.
; entrada 1
    db   0ffh   ; l0 a l15
    db   0ffh
    db   0      ; b0 a b23
    db   0
    db   0
    db   10010010b          ;92h    1       = segmento presente
                            ;       00      = iopl máximo
                            ;       1       = descriptor de datos
                            ;       0010    = tipo y acceso
    db   11001111b          ;0cfh   1       = granularidad en unidades de 4k
                            ;       1       = default en registros de 16 bits
                            ;       0       = obligatorio
                            ;       0       = disponible
                            ;       1111    = l16 a l19
    db   0h                 ;       b24 a b31
comment .

Resumen de la entrada 1 : base      :       ff000000
                          Limite    :       f0000000 ( f0000 * 1000 )
.
er db  'Maquina en modo protegido. No puede acceder al modo real plano$'

end inicio
```

III.2.- ENSAMBLADOR

III.2.1.- Algunos conceptos funcionales del ensamblador

Un principio básico en la operación de los ensambladores es el 'segmento'; un área de 64K contiguos (que puede ser 4 Gb en los 386)en la que pueden residir datos, código y pilas. Un 'segmento' tiene los siguientes 'atributos':
- nombre, como las etiquetas o las variables
- tamaño, 16 o 32 bits.
- alineamiento, el tipo de posiciones en que empieza; bytes, words o parágrafos.

- combinación, como se relaciona con otros segmentos.
- clase, un nombre que participa en la definición del orden en que se coloca.

Las directivas simplificadas .CODE y .DATA definen también segmentos de acuerdo a los modelos de memoria de los lenguajes de alto nivel. En los modelos TINY (diminuto) y SMALL (pequeño) que se han propuesto se tienen los siguientes
atributos y características.

Se asume que ciertos registros de segmento (CS, DS, ES y SS) apuntan a los segmentos.

En el modelo TINY:
.CODE
Nombre: _TEXT
Tamaño: 16 bits

Alineamiento: WORD; si se va a colocar en una posición impar, deja un byte y se ubica en una posición par; el principio de una word bien alineada para lectura o escritura del procesador.

Combinación: PUBLIC; si aparece otro segmento con el mismo nombre _TEXT, ambos forman un solo segmento. Se vera como es posible conectar (enlazar) varios programas y todos los segmentos de .CODE deberán entrar en 64K.

Clase: 'CODE'

Registros que se asumen apuntando a el: CS, DS, ES y SS. Efectivamente apuntan y no es necesario ajustarlos.

En el modelo SMALL:
.CODE da un segmento similar al del modelo TINY, con la diferencia que solo se asume CS apuntando a el.

.DATA da un segmento similar a .CODE con la diferencia del nombre que es _DATA; otro nombre de segmento, que no se combina con _TEXT (de .CODE). De esta manera el programa puede tener hasta 64 K en _TEXT y otros 64K en _DATA
totalizando 128K. Se asume que DS apunta a el, pero DS apunta al inicio del programa y por eso debe ajustarse para que apunte a el.

.STACK
Nombre: STACK
Tamaño: 16 bits
Alineamiento: PARA; ubica la próxima posición de la forma xxxx0h

Combinación: PUBLIC; si aparece otro segmento con el mismo nombre _TEXT, ambos forman un solo segmento. Se vera como es posible conectar (enlazar) varios programas y todos los segmentos de .CODE deberán entrar en 64K.

Clase: 'BSS' Registros que se asumen apuntando a el: SS. Efectivamente apunta y no es necesario ajustarlo.

Cada variable y etiqueta tiene los siguientes atributos:
- El segmento al que pertenece
- Un desplazamiento (offset) dentro del segmento
- Un tipo: WORD, DWORD, BYTE, FWORD, QWORD, TENBYTE, estructura y registro para las variables y FAR y NEAR para las etiquetas.

Cuando el ensamblador encuentra una referencia a una variable que no se ha definido no sabe si debe generar código para un byte o una word. Considérese el siguiente código:

```
0100  ??              B DB 0CCH
0101  C6 06 0111r 01 90    MOV A,1
0107  C6 06 0100r 01       MOV B,1
0111  ??              A DB 0CCH
```

B esta definida cuando se ensambla MOV B,1 a diferencia que MOV A,1 que esta antes de la definición de A. El ensamblador reserva espacio para la transferencia de una word y como luego no resulta necesario, lo rellena con 90h que es XCHG AX,AX. Se muestra como NOP (No OPeration). Este defecto puede subsanarse escribiendo MOV BYTE PTR A,1. Se esta noticiando al ensamblador que debe generar código para transferencia de una word.

Para transferencia de control a etiquetas que están en el mismo segmento de la instrucción de transferencia, basta con cambiar el valor de IP. Esto es una bifurcación cercana. Si esta en otro segmento, lo que se genera en otros modelos de memoria como MEDIUM, es una bifurcación lejana que cambia los valores tanto de CS como IP. Esto trae mayores complicaciones; el segmento no es relativo como el offset ; es una dirección absoluta en memoria y no puede decidirse en el instante del ensamblado en que lugar se ejecutara todas las veces el programa. Por esto, aparece el formato .EXE que contiene una tabla de
las direcciones que deben reajustarse dependiendo del lugar donde el programa es cargado. Por ejemplo, si una instrucción en el principio de un segmento salta a la primera instrucción de próximo segmento y se carga en 2000:0 se codificara
como JMP 3000:0. En cambio si se carga en 2500:0, se codificara 3500:0.

Otra facilidad del ensamblador es la capacidad de generar códigos de sobre-escritura de segmento. Si se tiene MOV A,1 y A es una variable tipo BYTE con offset 23 que esta en un segmento que se asume apuntado por ES, se ensambla MOV BYTE PTR [ES:23],1.

Para acceder a posiciones de memoria conocidas es necesario construir las direcciones con los atributos de variables; por ejemplo para acceder a 0:0 es necesario lo siguiente:

```
      MOV AX,0
      MOV ES,AX
MOV WORD PTR ES:0,0     ; se da tipo, segmento y desplazamiento.
```

III.2.2.- Enlace de programas y librerías

La directiva PUBLIC nombre hace que el nombre este disponible para ser usado como externo.
La directiva EXTRN nombre:tipo anuncia que nombre es del tipo dado, se localiza fuera del modulo y se encontrara al enlazarse. El ejemplo del factorial se puede construir con los dos siguientes programas independientes:

Programa principal PRO.ASM

```
.MODEL TINY
.CODE
ORG 100H
EXTRN FACTOR:NEAR            ; FACTOR es una rutina externa cercana
INICIO: MOV CX,NÚMERO        ; (en el mismo segmento)
      CALL FACTOR
      MOV FACTORIAL,AX
      INT 20H
NÚMERO DW  5
FACTORIAL DW ?
END INICIO
```

Sub-programa SUB.ASM

```
.MODEL TINY
.CODE              ; no se usa ORG 100H
PUBLIC FACTOR      ; la etiqueta FACTOR esta disponible para acceso externo
FACTOR: MOV AX,1
     JCXZ FIN
CICLO:  MUL CX
     LOOP CICLO
FIN:   RET
END                ; no se coloca una etiquera
```

Se ensamblan con los siguientes comandos:

```
TASM PRO
TASM SUB
```

Se enlazan con:

```
TLINK PRO SUB /t/x
```

Se crea el ejecutable PRO.COM

La librerías se crean con el programa TLIB que tiene las siguientes opciones:

```
TLIB librería [comandos] [,listado]
```

librería: es un nombre de archivo que generalmente tiene la extensión .LIB
listado : tiene la extensión .LST
Los comandos pueden ser:
 +<modulo objeto> : incluye modulo objeto
 -<modulo objeto> : elimina modulo objeto
 *<modulo objeto> : extrae modulo objeto; recupera .OBJ
-+<modulo objeto> : reemplaza modulo objeto
-*<modulo objeto> : elimina y extrae modulo objeto

Se puede crear una librería:

```
TLIB LSUB +SUB
```

Se enlaza con la librería con el comando:

```
TLINK PRO,,,LSUB /t/x
```

Se crea un programa PRO.COM idéntico al creado con TLINK PRO SUB /t/x. La ventaja es que la librería LSUB.LIB puede contener varias rutinas y no es necesario nombrarlas una a una.

III.2.3.- Registros y estructuras

Se pueden definir estructuras con la directiva STRUC.

nombre_estruc STRUC

solamente definiciones de datos con Dx con o sin valores iniciales

ENDS

y usar el nombre de la estructura para declarar una variable;

nombre nombre_estruc<valores iniciales separados por comas>; por ejemplo:

```
persona STRUC
nombre   db '                '       ; 20 bytes
teléfono dd
edad     db 20
ENDS

JUAN persona                         ; reserva 25 bytes; edad llega con valor 20.
```



```
ROBERTO persona <'Roberto',23232,30>; reserva y da valores iniciales
```

Nótese que las variables declaradas con múltiples valores iniciales no pueden recibir un nuevo valor inicial. No era posible inicializar

```
nombre db 20 DUP(?)
```

ya que hubiera tenido 20 valores iniciales. No pasa lo mismo con 'edad'

Además, es posible referirse a los desplazamientos dentro del registro con la notación x.nombre_campo. Por ejemplo

```
JUAN.edad       ; Se refiere al campo edad dentro de JUAN
[BX.edad]       ; suma 24 al valor de BX
```

También se pueden definir registros con campos de bits con la directiva RECORD.

nombre_rec RECORD nom_campo:longitud=valor inicial,....

Ejemplo:

```
ident RECORD t_c:3,r_v:4 ; solo se implican 7 bits, con lo que
                         el registro tiene tamaño de 1 byte.
                         t_c ocupa los 3 bits altos y r_v los
                         4 siguientes.

z ident <3,5>            ; z es un byte con el valor 01101010.
```

Existen tres operadores de registros:
- El nombre del campo, que resulta en la cantidad de bits que hay que correr a la derecha el campo para llevarlo a la posición mas baja.
- WIDTH <campo> el ancho de un campo.
- MASK <campo> la mascara para aislar el campo.

Ejemplo: Sean

```
FECHA RECORD anio:7,mes:4,dia:5
f FECHA
d db
m db
a db
```

Extraer dia en d:

```
mov al, byte ptr f
and al,mask dia
mov d,al
```

Extraer mes en m:

```
    mov ax,f
    and ax,mask mes
    mov cl,mes
    shr ax,cl
    mov m,al
```

Extraer año en a:

```
    mov al,byte ptr f+1
    and al,high (mask anio)
    mov cl, anio-8
    shr al,cl
    mov a,al
```

Formar f a partir de d,m,a:

```
    mov al,a
    mov cl, width mes
    shl ax,cl
    or al,m
    mov cl, width dia
    shl ax,cl
    or al,d
    mov f,ax
```

Aprovechando que width anio = 7:

```
    mov ah,a
    mov al,m
    mov cl,8-width mes
    shl al,cl
    shl ax,8-width anio
    or al,d
    mov f,ax
```

Extraer d,m y a en una sola operación:

```
    mov ax,f
    mov d,al
    and d,mask dia
    mov a,ah
    shl a,8-width anio
    and ax,mask mes
    mov cl,mes
    shl ax,cl
    mov m,al
```

III.2.4.- Directivas de macros

Una macro es un código que puede ser invocado por un nombre. Se declara con la directiva MACRO. El siguiente ejemplo Desp_h despliega el hexadecimal de dl basado en desp_l, macro que despliega un nible.

Macro:	Desp_h se implementaría como:
desp_dl MACRO	desp_h:
LOCAL p1,p0	push dx
cmp dl,9	mov cl,4
ja p0	shr dl,cl
or dl,30h	desp_dl ←
jmp short p1	pop dx — desp_dl se copia
p0: add dl,'A'-10	and dl,0fh — en cada una de las
p1: mov ah,2	desp_dl — invocaciones.
int 21h	ret
ENDM	

Las copias de la macro se denominan "expansiones". Como una macro normalmente se invoca mas de una vez, las etiquetas dentro del código se repiten y ello ocasiona errores si no se usa LOCAL con la lista de las etiquetas.

Se pueden indicar argumentos. Por ejemplo:

```
desp_reg MACRO reg
        local p0,p1
        cmp reg,9
        ja p0
        or reg,30h
        jmp short p1
p0:     add reg,'A'-10
p1:     mov ah,2
        mov dl.reg
        int 21h
ENDM
```

Se puede invocar con :

```
        desp_reg dl
o       desp_reg al
o       desp_reg <byte ptr [si]>
```

Debe existir ENDM para terminar una macro. EXITM también puede concluir la expansión de la macro dentro de una directiva condicional.
Al declarar la macro, los parámetros se deben indicar separados por comas y pueden omitirse, con lo que se considera un parámetro nulo.
Al invocar la macro, no es necesario separar los argumentos por comas. Inclusive se puede indicar un parámetro que contiene espacios en blanco o comas encerrado entre paréntesis angulares como en el ejemplo:

desp_reg <byte ptr [si]>

Se puede crear un archivo con varias macros y después incorporarlo a un texto fuente cuando se lo necesite con la directiva INCLUDE <archivo> que copia el archivo dentro del programa.

III.2.5.- Directivas de ensamblado condicional

El formato general es:

```
IFxxx
  bloque-if
ELSE
  bloque-else
ENDIF
```

Si IFxxxx se evalúa como verdadera, se ensambla el bloque IF, en otro caso el bloque ELSE. ELSE es opcional.

IF exp	es verdadera cuando exp es diferente de 0.
IFE exp	es verdadera cuando exp es 0.
IFDEF etiqueta	es verdadera cuando etiqueta esta definida
IFNDEF etiqueta	es verdadera cuando etiqueta esta no definida

En las siguientes, se requieren los paréntesis angulares:

IFB <argumento de macro>	es verdadera cuando <argumento ...> es blanco
IFNB <argumento de macro>	es verdadera cuando <argumento ...> no es blanco
IFIDN <arg1>,<arg2>	es verdadera cuando <arg1> y <arg2> son iguales
IFDIF <arg1>,<arg2>	es verdadera cuando <arg1> y <arg2> no son iguales

Una utilidad del ensamblado condicional en macros es la posibilidad de generar parámetros por omisión. Por ejemplo, si no se indica un parámetro para la siguiente macro, no se altera el valor de AL.

```
m MACRO x
ifnb x
    mov al,x
endif
endm
```

En la siguiente, si se pasa el parámetro CX, no se realiza un asignación a CX

```
n MACRO x
ifdif <x>,<CX>
    mov cx,x
endif
endm
```


III.2.6.- Pase de parámetros por pila a rutinas

El uso de "registros de activación" puede lograrse de la siguiente manera:

Modulo llamador:

```
PUSH ..     Empila n bytes de parámetros o direcciones a parámetros
CALL ..     Ejecuta la llamada a la rutina
```

Rutina de servicio:

```
SUB SP,x    Desplaza el SP hacia abajo la cantidad requerida para un área de variables locales (SUB SP,x)
PUSH BP     Empila BP por si estuviera en uso
MOV BP,SP   Hace BP igual a SP (MOV BP,SP)

    .... realiza sus tareas ....

POP BP      Desempila BP (POP BP)
ADD SP,X    Restituye SP a su posición original
RET N       Termina y descarta los parámetros pasados.
```

Definiendo adecuadamente una estructura se puede acceder cómodamente a las variables locales y a los parámetros. La apariencia será:

```
R STRUC
    dw    ; BP empilado
v_l dw    ; variables locales
    dw    ; dirección de retorno
par_n dw  ; ultimo parámetro empilado
par_1 dd  ;primer parámetro empilado
ENDS
```

Se podrá acceder con [BP.v_l], por ejemplo

Ejercicios (Grupo 8)

1.- Analizar las necesidades en registros de activación con relación a modelos de memoria con varios segmentos de código y datos.

2.- Escribir una rutina que reciba las direcciones de A (word) y B (word) como parámetros y deje B=A*A

III.3.- CONJUNTO DE INSTRUCCIONES 80x86

Existen dos grupos de instrucciones:

- Del programador de aplicaciones
- Del programador de sistemas operativos

El segundo grupo trata de la activación del modo protegido y la programación de sistemas operativos y redes que operan en modo protegido y controlan tareas que se lanzan en modo virtual 86. Queda fuera del alcance de este texto.

Se han visto varias de las instrucciones de programación de aplicaciones y ahora se completaran. El fabricante, INTEL, establece 8 grupos de instrucciones del programador de aplicaciones:

1.- Transferencia de datos
2.- Control del procesador
3.- Operaciones Aritméticas
4.- Operaciones booleanas y manejo de bits
5.- Manejo De cadenas
6.- Transferencia de control
7.- Interrupciones
8.- Sincronización

III.3.1.- TRANSFERENCIA DE DATOS

MOVSX r , r / m

Mueve la fuente al destino con extensión de signo de manera similar a la que se usa en los modos de direccionamiento y es aplicable para números con signo. El destino debe ser un registro y la fuente puede ser un registro o memoria pero siempre de tamaño menor que el destino. Por ejemplo;

MOVSX AX,DL
Si DL tiene FF, que es -1, AX pasa a tener FFFF que es también -1

MOVZX r , r / m

Mueve la fuente al destino y llena con ceros la parte alta. Se aplica muy bien a números con signo. El destino debe ser un registro y la fuente puede ser un registro o memoria pero siempre de tamaño menor que el destino. Por ejemplo;

MOVZX EAX,WORD PTR[SI]

XLAT

Traduce la tabla apuntada por BX según el registro de AL.
Se puede entender como AL = "[BX+AL]". No tiene operandos Su utilidad es muy limitada y puede aplicarse a encriptaciones de datos.

LEA r16 / r32,m

Carga dirección efectiva. Calcula el desplazamiento de la dirección m y la carga a un registro de 16 o 32 bits.
A diferencia de MOV, que no hace cálculos, LEA puede tener en m cualquier un modo de direccionamiento, calcular la dirección a la que se refiere y llevarla al registro destino.
LEA AX,[200] tiene el mismo efecto que MOV AX,200, que incluso es una forma mas eficiente.
LEA SI,[SI+BX] calcula SI+BX y lleva el resultado a SI.
MOV SI,[SI+BX] calcula SI+BX, accede a memoria en la posición que y lleva ese contenido a SI.
LEA SI,[BX+AX] no es posible por que BX+AX no es un modo de direccionamiento valido. Podría ser [EBX+EAX].
Al igual que MOV, se usa para iniciar un puntero en una dirección, con la posibilidad de efectuar un calculo.
Por ejemplo, si BX apunta al primer elemento de un arreglo de bytes y SI tiene un número de elemento (desde 0), con
LEA DI,[BX+SI], el registro DI queda apuntando a ese elemento.

LDS r16/r32,m
LSS, LES
LFS, LGS

Cargan en un registro de 16 o 32 bits y un segmento desde un puntero en memoria. El registro debe estar en la parte baja y el segmento (selector) en la parte alta. Cuando se usa memoria dinámica y las posiciones asignadas se guardan en un 'puntero', se reservan 4 bytes con el formato exacto para ser recuperadas con esta dirección. Por ejemplo, si se obtiene una asignación dinámica en 3120:AF5D en un puntero localizado en DS:1400, el contenido en memoria es:

DS:1400 5D AF 20 31
Para cargar ES:DI con ese valor basta LES DI, DWORD PTR [1400]

LAHF,SAHF

Carga AH con las banderas 8080 y fija los flags desde AH.

PUSHA,POPA

Salva y restituye todos los registros de propósito general, punteros e índices de 16 bits.

PUSHAD,POPAD

Salva y restituye todos los registros de propósito general, punteros e índices de 32 bits.

PUSHFD,POPFD

Salva y restituye las banderas en 32 bits.

ENTER w,b

Reserva w (tamaño word inmediato) bytes en la pila y copia b(tamaño byte) niveles de punteros de registros de activación.
Empila BP y asigna BP al inicio del registro de activación.
Como el número de niveles es constante, no es posible fijar en un nivel variable de niveles y si no se requieren punteros a registros de activación, basta con ENTER w,0.
Se puede aplicar en el pase de parámetros a rutinas y se discutirá posteriormente.

LEAVE

Revierte la operación de ENTER. No tiene operandos

IN dest.,fuente

Lee del puerto fuente (inmediato de 8 bits o DX) al destino AL o AX. Los puertos son las direcciones de dispositivos externos al procesador. Nótese que para manejar directamente un dispositivo se deben conocer los puertos asociados a el con todos sus funciones relacionadas. Los puertos se clasifican en:

- Datos : dependiendo del dispositivo, entrada, salida o ambos
- Estado : informan del estado interno del dispositivo
- Control: permiten comandar el dispositivo

Para cada puerto debe también conocerse el significado de los bits que se lee o mandan a el.

Por ejemplo, si se sabe que un dispositivo tiene su puerto de estado de un byte con el número 45h e informa de un evento, como por ejemplo que se ha presionado una tecla en el teclado colocando en 1 el bit 2 y además que en el puerto 46h esta un dato del evento, como la tecla presionada en una word, se puede escribir un programa de espera y lectura del evento con las siguientes instrucciones:

```
CICLO: IN AL,45H
       TEST AL,100B
       JZ CICLO
       IN AX,46H
```

OUT dest,fuente

Escribe en el puerto destino (inmediato de 8 bits o DX) la fuente AL o AX. Siguiendo el ejemplo de la instrucción IN, el dispositivo podría necesitar de un comando de reiniciación luego de comunicar un evento con el bit 1 puesto en 0 en el puerto 47h. Habría que completar el programa anterior con:

```
       XOR AL,AL
       OUT 47H,AL
```

Si el puerto fuera 122h habría que usar DX con

```
       MOV DX,122H
       OUT DX,AL
```

BSWAP r16/r32

Para registros de 32 bits intercambia los bytes 0 y 3, 1 y 2 del registro de 32 bits. Para registros de 16 bits, se consideran los registros de 32 bits de los que los registros de 16 bits son parte y los dos bytes bajos reciben intercambiados a los dos bytes altos. Es propia del 486.

SETcond r8/m8

Verifican una condición sobre las banderas actuales y fijan en 1 el operando si es así y en 0 si no lo es.
Las condiciones, las mismas que las de bifurcaciones condicionales, son las siguientes:

SETL / SETNGE	SETNL / SETGE
SETLE / SETNG	SETNLE / SETG
SETB / SETNAE	SETNB / SETAE
SETBE / SETNA /SETC	SETNBE / SETA / SETNC
SETO	SETNO
SETS	SETNS
SETP / SETPE	SETNP / SETPO
SETZ / SETE	SETNZ / SETNE

NOP

Es la instrucción XCHG AX,AX que no realiza ninguna acción y solo ocupa un byte (90). Se usa para cubrir espacios que por cualquier razón están dentro del programa pero no deberían estarlo. NOP significa No OPeration

III.3.2.- CONTROL DEL PROCESADOR

CLD,STD

Fijan la bandera de dirección en 0 (avance) y 1 (retroceso) respectivamente.

CLI,STI

Fijan la bandera de interrupción en 0 y 1, activando y desactivando respectivamente el procesamiento de interrupciones externas.

III.3.3.- OPERACIONES ARITMETICAS

Las siguientes operaciones de extensión de signo han sido creadas para convertir tipos de datos numéricos binarios con signo a otros de mayor tamaño.
En especial se usan para divisiones de números de mismo tamaño ampliando el tamaño del dividendo a un tamaño mayor que es lo que necesitan la instrucción IDIV para dividir números con signo. Si se trata de números sin signo, basta
con llenar de ceros la parte alta.

CBW

Extiende el signo de AL a AX. Para dividir F0h entre 2, se puede hacer:

```
MOV AL,0f0h
CBW
MOV BL,2
IDIV BL
```

CWD

Extiende el signo de AX a DX.

CWDE

Extiende el signo de AX a EAX

CDQ

Extiende el signo de EAX a EDX

DAA/DAS

Ajustes decimales para adición/ substracción. Se deben aplicar luego de sumar/restar en AL dos números BCD empaquetados. Se actualizan la banderas excepto O que queda indefinida. Por ejemplo. para calcular 19 + 1 se puede hacer:

```
MOV AL,19H
ADD AL,1
DAA          ; AL queda con 20H.
```

AAA,AAS

Ajustes ASCII para adición/ substracción. Se deben aplicar luego de sumar en AL dos números ASCII o BCD desempaquetados. Acarreo y auxiliar se actualizan y las demás banderas quedan indefinidas.
AH se incrementa/decrementa con el acarreo. Los cuatro bits mas altos terminan en 0 y no deben necesariamente empezar así.

AAM

AL se convierte de binario puro a decimal ASCII de dos dígitos en AX. Puede seguir a una multiplicación pero no necesariamente. Las banderas S,Z,P se actualizan y O,A,C quedan indefinidas. Por ejemplo, para calcular 2 * 6 se puede programar:

```
MOV AL,2
MOV BL,6
MUL BL
AAM          ; AX queda con 0102; AH=01 y AL=02
```

AAD

AX, con dos decimales ASCII se convierte en binario en AL. Puede anteceder a una división pero no necesariamente. Las banderas S,Z,P se actualizan y O,A,C quedan indefinidas Por ejemplo, para calcular 18 / 6 se puede programar:

```
MOV AX,0108h
MOV BL,6
AAD
DIV BL       ; AL queda con 03
```

CMPXCHG r/m,r

Según el tamaño de los operandos, se elige el acumulador AL,AX o EAX. Si r / m = acumulador entonces r / m =r , si no acumulador = r / m. Es propia del 486.

XADD r / m,r

Los dos operandos deben ser del mismo tamaño. r/m queda con la suma r/m + r y r con el valor de r/m. Es propia del 486.

La instrucción IMUL tiene desde el 386 las siguientes nuevas posibilidades:

IMUL r,imm El registro de 16 o 32 bits el multiplicado por el operando inmediato imm. Ejemplo: IMUL AX,8
IMUL r,r/m El registro es multiplicado por r/m.
Ejemplos: IMUL AX,BX IMUL AX,[BX]

IMUL r,r/m,imm El registro recibe el producto de r / m por el operando inmediato imm.

Ejemplos: IMUL AX,BX,8 IMUL AX,[BX],8

Los registros deben ser de 16 o 32 bits.

La difusión del coprocesador hace obsoletos todos los temas aritméticos complejos que deben ser dejados al coprocesador.

Ejercicios

1.- Almacenar 123423 y 3232 en memoria en ASCII y calcular su suma y resta.
2.- Almacenar 123423 y 3232 en memoria en BCD empaquetado y calcular su suma y resta.
3.- Multiplicar 9 por 8 como números en ASCII y obtener en AX 0702h
4.- Dividir 63 entre 9 almacenando 63 en ASCII como 0306h
5.- Dividir 4fh entre 0b usando CBW.
6.- Que diferencias hay entre usar CBW y MOVSX AX,AL? y usar CWD y MOVZX EAX,AL?
7.- Multiplicar un número de varios bytes en binario por otro de un byte.

III.3.4.- MANEJO DE BITS

BSF r,r/m
BSR

Revisan el operando fuente r/m y fijan la bandera Z en 1 si es 0. En otro caso dejan en el destino r la posición del primer bit 1. BSF explora en avance desde los bits menos significativos hacia los mas significativos y BSR a la inversa. Ambos operandos deben ser de 16 o 32 bits. Ejemplos:

```
BSF AX,BX   ; si BX=0 entonces Z=1
            ; si BX=2 entonces Z=0 y AX=1
```

BT/BTS r/m,r/inm8
BTR/BTC

Manipulan directamente un bit del operando destino según una posición indicada en el operando fuente. Todas llevan a C el bit del operando destino. BT solo verifica el bit, BTS lo fija en 1, BTR lo fija en 0 y BTC lo complementa.
Se consideran 4 bits del operando fuente si los registros son de 16 bits y 5 si son de 32 bits. Ejemplos con valores iniciales AX=12h y BX=2:

```
BT AX,BX      ; valores finales AX=12h y  C=0
BTS AX,BX     ; valores finales AX=16h y  C=0
BTR AX,BX     ; valores finales AX=12h y  C=0
BTC AX,BX     ; valores finales AX=16h y  C=0
```


SHLD r / m, r , i / CL
SHRD

Corrimientos de doble precisión. Forman un solo destino entre los dos operandos r y r/m , que pueden ser de 16 o 32 bits y se corren a la izquierda o derecha las veces que indica el tercer operando i(INMEDIATO DE 8 BITS) o CL.
Se actualizan S,Z,P y C. Ejemplo con AX=ffffh y BX=1

```
SHLD AX,BX,1    ; AX=fffe BX=2
```

III.3.5.- MANEJO DE CADENAS

Consideran tres tipos de operandos; bytes, words y dwords que se indican explícitamente con los sufijos B,W y D respectivamente; por ejemplo MOVSB es la instrucción MOVS con operandos byte.
Consideran un operando fuente apuntado por DS:SI y un destino apuntado por ES:DI. Ambos deben ser del mismo tipo. Tienen dirección de avance y retroceso dependientes de la bandera D. Según la dirección y el tipo del
operando actualizan los punteros a las siguientes direcciones. Por ejemplo, si la dirección es avance y los operandos son word, SI y DI se incrementan en 2.
Si son byte, se incrementan en 1. Si bien no necesitan operandos explícitamente en las instrucciones, se pueden indicar para generar la instrucción correcta. Por ejemplo:

```
MOVS data1,data2
```

exige que ambos, data1 y data2, sean del mismo tipo y si son dwords, se ensambla MOVSD. Se debe asegurar que previamente DS:SI apunte a data1 y ES:DI apunte a data2. La instrucción no fija valores iniciales para DS:SI ni ES:DI pero si actualiza SI y DI.

MOVS

Mueve fuente a destino. Si se usa sobre-escritura de segmento, el segmento indicado se combina tanto con SI como con DI.

CMPS

Compara fuente con destino y actualiza banderas. Si se usa sobre-escritura de segmento, el segmento indicado se combina tanto con SI como con DI.

SCAS

Compara el acumulador (AL,AX o EAX según sea el tipo) con el destino y actualiza banderas. Ignora la sobre-escritura de segmento.

LODS

Carga el acumulador con DS:SI. Acepta sobre-escritura de segmento.

STOS

Lleva el acumulador a ES:DI. Ignora la sobre-escritura de segmento.

INS

Lee el puerto indicado en DX y lleva a ES:DI. Ignora la sobre-escritura de segmento.

OUTS

Escribe DS:SI al puerto indicado en DX. Acepta sobre-escritura de segmento.

REP
REPE/REPZ
REPNE/REPNZ
Son prefijos que pueden anteceder solo a instrucciones de cadena. Si CX es 0 al inicio, no se ejecuta la instrucción.
Si no, se ejecuta y si se decrementa CX. Si CX es diferente de 0, se repite

Con REPE/REPZ se repite solo si también se tiene Z=1
Con REPNE/REPNZ se repite solo si también se tiene Z=0

EJEMPLOS

1.- Copiar la pantalla de B800:0 al área inmediatamente contigua.

```
.model tiny
.code
org 100h
inicio: mov ax,0b800h          ; apuntar al área de la pantalla
        mov DS,ax              ; con DS y ES
        mov ES,ax
        xor SI,SI              ; apuntar al inicio con SI
        mov DI,80*25*2         ; apuntar al final con DI
        mov CX,DI              ; cargar en CX la cantidad a mover
        cld                    ; fijar dirección en avance
        rep movsb              ; mover
        int 20h
end inicio
```

Se puede optimizar con movimiento de words:

```
        mov CX,80*25
        cld
        rep movsw
```

Aun mas con movimiento de dwords:

```
        mov CX,40*25
        cld
        rep movsd
```

2.- Comparar dos cadenas estructuradas apuntadas por SI y DI

```
.model tiny
.code
.386
org 100h
inicio: cld
        mov si, offset a+1
        lodsb
        mov di, offset b+1
        scasb
        pushf
        jb short p0
        mov al,[di-1]
p0:     movzx cx,al
        repe cmpsb
        pop ax
        jnz short fin0
        sahf
fin0:   jae short ma_yori
        mov dx, offset menor
        jmp short dsp
ma_yori: jz short ig_ual
        mov dx, offset mayor
        jmp short dsp
ig_ual: mov dx, offset igual
dsp:    mov ah,9
        int 21h
        int 20h
menor db 'menor$'
mayor db 'mayor$'
igual db 'igual$'
a db 10,4,'bsdfg     '
b db 10,4,'asdfg     '
end inicio
```

Ejercicios (Grupo 9)

1.- Borrar el área de la pantalla.

2.- Borrar la letras "o" en el área de pantalla

3.- Dadas dos cadenas estructuradas, buscar la primera dentro de la segunda y dejar AL el valor 1 si es así.

4.- Dadas dos cadenas ASCIIZ, compararlas y dejar en las banderas el resultado.

5- Dadas dos cadenas ASCIIZ, buscar la primera dentro de la segunda y dejar AL el valor 1 si es así.

III.3.6.- TRANSFERENCIA DE CONTROL

JMP dir Existen tres modos;

Directos : se indica una dirección y el programa bifurca a ella.

Indirectos: se indica la dirección de un contenido en memoria y el programa bifurca a la dirección allí depositada. La dirección se asume en el segmento DS: pero puede sobrescribirse con un prefijo de segmento.

Registro : un registro contiene la dirección de bifurcación

Modos directos:
- Corta: con desplazamiento desde la posición del CS:IP en un byte con signo. El punto de bifurcación debe estar 128 bytes antes o 127 bytes después. Por ejemplo:

```
JMP SHORT 100
JMP SHORT X     ; X es una etiqueta
```

- Cercana: con desplazamiento desde la posición del CS:IP en una word con signo. El punto de bifurcación debe estar en el mismo segmento. Por ejemplo:

```
JMP P0              ;P0 esta definida como cercana
JMP NEAR PTR P0     ;P0 esta definida de alguna manera  (DB, LABEL, etc.)
JMP _TEXT:100       ;se construye una dirección en el segmento _TEXT. Se debe
                     indicar un nombre de segmento y no un registro de segmento.
```

- Lejana: con dirección en términos de segmento y desplazamiento. El punto de bifurcación puede estar en cualquier parte. Por ejemplo:

```
JMP P0              ;P0 esta ya esta definida en otro segmento como lejana
PROC o LABEL FAR.

JMP FAR PTR P0      ;P0 esta definida de alguna manera (DB, LABEL, etc.)

JMP BIOS:100   ;se construye una dirección en el segmento BIOS.
```

Para bifurcar a una dirección conocida es un segmento conocido es necesario declarar un segmento con AT nnnn.
Se podría hacer BIOS SEGMENT AT 0040h para que JMP BIOS:100 sea realmente JMP 40h:100h.
ENDS debe terminar el bloque SEGMENT.

Modos indirectos:
- Cercana: tiene como operando la dirección de una word que tiene el desplazamiento al que se quiere bifurcar en el mismo segmento. Por ejemplo:

```
JMP P0              ;P0 esta definida como word
JMP WORD PTR P0     ;P0 esta definida de alguna manera  (DB, LABEL, etc.)
JMP ds:100          ;se construye una dirección en el segmento DS.
```

-Lejana : tiene como operando la dirección de una dword que tiene el segmento y desplazamiento donde se quiere bifurcar. Por ejemplo:

```
JMP P0                      ;P0 esta definida como dword
 JMP DWORD PTR P0           ;P0 esta definida de alguna manera (DB, LABEL, etc.)
JMP DWORD PTR CS:100        ; se construye una dirección en el segmento CS.
```

Modo registro: cualquier registro de 16 o 32 bits se puede usar para hacer una bifurcación cercana.

CALL Es similar a JMP pero difiere en:
- no existe forma corta
- la dirección en la llamada corta no es relativa.

RET n

Existen RET y RETF. Si se usa dentro de un procedimiento lejano, RET se ensambla como FAR. Se puede indicar un operando inmediato adicional RET n con lo que se descartan n bytes de la pila. Es muy útil cuando se pasan parámetros por la pila.

ELOOP dir.

Es similar a LOOP pero opera sobre ECX.

ELOOPE dir. /ELOOPZ

Como ELOOP, decrementa ECX y continua si ECX<>0 y además si Z=1 Se espera que una instrucción actualice previamente las banderas.

ELOOPNE dir. / ELOOPNZ
Como ELOOP, decrementa ECX y continua si ECX<>0 y además si Z=0. Se espera que una instrucción actualice previamente las banderas.

JECXZ dir.

Si ECX es cero bifurca a una dirección corta.

Ejercicios (Grupo 10)

1.- Crear un menú con varias opciones usando la llamada indirecta. Crear una tabla con posiciones de punteros a los procedimientos que hacen las opciones del menú

III.3.7.- INTERRUPCIONES

INTO Ejecuta INT 4 si la bandera O (overflow) esta en 1.

HLT Detiene el procesador hasta que llegue una interrupción externa.

BOUND r16,m

Compara un registro de 16 bits con dos words en la fuente m.
Si el registro es menor a la primera o mayor a la segunda, se genera una INT 5.
En el entorno DOS/WINDOWS las interrupciones se dividen en cuatro grupos:

Interrupciones del procesador
Interrupciones del controlador de interrupciones
Interrupciones del BIOS
Interrupciones del BDOS

Cada una agrupa varios servicios y requiere como indicador el contenido en un registro (normalmente AH) o varios. El rango de interrupciones 40h a 80h esta disponible para el usuario.

- Interrupciones del procesador

0 = Overflow de división.

1 = Simple paso, cuando se activa la bandera T = 1

2 = Línea NMI (Non-maskable interrupt). Las excepciones en el coprocesador también derivan en INT 2

3 = Interrupción en una sola instrucción Int 3 que se codifica 0CCh.

4 = Asociada a INTO (interrupción por Overflow)

5 = Print Screen de DOS. Asociada también a la instrucción BOUND, desde el 286.

6 = Código de operación no valido, desde el 286

7 = Coprocesador no disponible y se intenta una instrucción ESC o WAIT , desde el 286.

- Interrupciones del controlador de interrupciones

8 = Temporizador. La señal del reloj interrumpe 18.02 veces por segundo el procesador para actualizar un contador de horas, minutos, segundos y fechas.
Desde el 286 , Falla doble, cuando dos interrupciones separadas ocurren durante la misma instrucción.

9 = Teclado. Cada vez que se presiona o libera una tecla, se invoca esta interrupción para leer las teclas y generar acciones especiales como las de Control+Alt+Del.
Desde el 286, también con un intento de acceder mas alla del segmento del coprocesador.

A = En modo protegido, cuando el campo limite del segmento de estado de tareas (TSS) se invalida al ser menor a 002BH.

B = En modo protegido, cuando el segmento no esta presente (bit P=0 en descriptor).

C = En modo protegido, Overflow en el segmento de pila o el segmento no esta presente.

D = En modo protegido, violación general de protección; limite de tabla de descriptores excedido, regla de privilegio violada, tipo de segmento invalido, intento de escritura en segmento protegido, lectura en segmento de solo ejecución, limite de segmento excedido

E = Diskette

F = Reservada

- Interrupciones del BIOS

10 = Video

11 = Chequeo de equipo
12 = Tamaño de memoria
13 = Diskette
14 = Comunicaciones
15 = Cassete y servicios misceláneos
16 = Teclado
17 = Impresora
18 = Llamada a BASIC
19 = Reencendido
1A = Hora
1B = Control C
1C = Eco de temporizador
1D = Parámetros de vídeo
1E = Parámetros de disekette
1F = Extension de caracteres gráficos

- Interrupciones del BDOS

20 = Terminación de programa
21 = Servicios del DOS
22 = Dirección de terminación
23 = Dirección de ^C
24 = Error critico
25 = Lectura absoluta de disco
26 = Escritura absoluta de disco
27 = Terminar pero permanecer residente
28 = Interrupción inactiva
2F = Interrupción múltiple.
33 = Ratón (mouse)

Las interrupciones reciben sus parámetros y entregan sus resultados mediante registros. AH se usa normalmente para elegir un servicio de una interrupción. Por ejemplo para la INT 10H, AH=0 es la fijación del modo de vídeo. Se proporciona el archivo BIOS.MAC con macros que llaman a interrupciones con servicios de teclado y pantalla. Los parámetros de estas macros corresponden a registros.

Para parámetros de entrada:
- si no se indican, no se altera el correspondiente registro. En ese caso, el registro ya debe tener el parámetro.
- si se indican, se altera el correspondiente registro.

Para parámetros de salida:
- si se indican, reciben el contenido del correspondiente registro.
- si no se indican, no se realiza ninguna acción.

En todo caso, los registros implicados son alterados.

Las macros desarrolladas son:

<u>Asociadas a INT 10 para pantalla</u>

SET_VIDEO m
Fija modo de vídeo.
m (AL) = modo de vídeo. Los pre-definidos son:
tex_bn_40 : texto blanco y negro de 40 columnas.
tex_16c_40 : texto a 16 colores de 40 columnas.
tex_bn_80 : texto blanco y negro de 80 columnas.
tex_16c_80 : texto a 16 colores de 80 columnas.
graf_2 : gráfico de 2 colores, 640 filas x 480 columnas
graf_16 : gráfico de 16 colores, 640 filas x 480 columnas
graf_256 : gráfico de 256 colores, 320 filas x 200 columnas
Registros usados: AX

SET_TAM_CUR inicio,fin
Fija tamaño de cursor.
inicio (CH) = línea de inicio. Debe estar entre 0 y 7
fin (CL) = línea de fin. Debe estar entre 0 y 7
Registros usados: AH, CX

SET_CUR_OVR
Fija tamaño de cursor en sobre-escritura (grande).
Registros usados: AH, CX

SET_CUR_INS
Fija tamaño de cursor en inserción (pequeño).
Registros usados: AH, CX

SET_CUR_BOR
Borra el cursor.
Registros usados: AH, CX

SET_CUR fila, columna
Coloca cursor en fila (DH) y columna (DL).
Registros usados: AH, BH, DX

LEE_POS_CUR fila,columna,l_inicio,l_final
Lee posición y líneas de tamaño del cursor.
fila (DH) = fila del cursor
columna (DL) = columna del cursor
l_inicio (CH) = línea de inicio del cursor
l_final (CL) = línea de final del cursor
Registros usados: DX,CX y AH

LEE_CAR_CUR car,atr
Lee Ascii y atributo del carácter en la posición del cursor
car (AL) = carácter
atr (AH) = atributo
Registros usados: AX
Se han predefinido los siguientes colores para el carácter:
negro
azul
verde
cyan
rojo
magenta
cafe
blanco
amarillo
Se han predefinido los siguientes colores para el fondo:
f_negro
f_azul
f_verde
f_cyan
f_rojo
f_magenta
f_cafe
f_blanco
Se han predefinido las siguientes propiedades
intenso
parpadeo

Los atributos se construyen sumando colores y propiedades.
Por ejemplo, amarillo es cafe + intenso

ESC_CAR car,atr,num
Escribe caracteres en la posición del cursor.
car (AL) = carácter
atr (BL) = atributo. Si no se indica, asume blanco sobre fondo negro.
num (CX) = número de repeticiones. Si no se indica, asume 1.
Registros usados: AX,BL y CX.

LEE_PIX fil,col,color
Lee el color de pixel en pantalla
fil (DX) = fila en pantalla
col (CX) = columna en pantalla
color (AL) = color del pixel.
Registros usados: AX,BH,CX y DX.

ESC_PIX fil,col,color
Escribe un pixel en pantalla.
fil (DX) = fila
col (CX) = columna
color (AL) = color. Blanco si no se indica.
Si no se dan parámetros usa DX,CX y 7 respectivamente.
Registros usados: AX,BH,CX y DX.

Asociadas a INT 16H para teclado

LEE_CAR car,scan
Lee tecla.
car (AL) = carácter Ascii
scan (AH) = scan code
Se han predefinido las siguientes constantes.
Registro usado : AX.

scan (ASCII en AL es 0)
_f1 : tecla de función 1
_f2 : tecla de función 2
_f3 : tecla de función 3
_f4 : tecla de función 4
_f5 : tecla de función 5
_f6 : tecla de funcion 6
_f7 : tecla de funcion 7
_f8 : tecla de funcion 8
_f9 : tecla de funcion 9
_f10 : tecla de funcion 10
_ins : insert
_hom : home
_pgu : Page Up
_del : Del
_end : End
_pgd : Page Down
_up : Up
_dn : Down
_rig : Right
_lef : Left

_c_hom : control home (va al principio de todo)
_c_pgu : control page up
_c_end : control end (va al final de todo)
_c_pgd : control page down
_c_rig : control right
_c_lef : control left

car (AL diferente de 0)

_cr : Enter
_bac : backspace
_c_bac : control backspace (borra todo lo anterior)

La tecla Alt combinada con letras, deja el ASCII en O
La tecla Control genera controles; Control A deja en ASCII el valor 1 (Control-A)

INF_CAR car,scan
    Lee un carácter
    car (AL) = ascii
    scan (AH) = scan code
    bandera Z = 1 si no hay carácter. 0 en otro caso.
    Registro usado : AX.

IF_CAR etq
    Bifurca a etq (etiqueta) si hay un carácter
    Registro usado : AX.

IF_NO_CAR etq
    Bifurca a etq (etiqueta) si no hay un carácter
    Registro usado : AX.

LEE_SHIFT shift
    Lee estado de shift en AL
    Registro usado : AX.
    También se ha predefinido el registro EST_SHIFT con los campos:
        insert
        caps_lock
        num_lock
        scroll_lock
        alt
        control
        shift_izq
        shift_der

IF_INS etq
    Bifurca a etq (etiqueta) si se ha presionado Ins
    Registro usado : AX.

IF_SCROLL etq
    Bifurca a etq (etiqueta) si se ha presionado scroll lock
    Registro usado : AX.

IF_CONTROL etq
    Bifurca a etq (etiqueta) si se ha presionado control
    Registro usado : AX.

IF_NO_INS etq
    Bifurca a etq (etiqueta) si no se ha presionado Ins
    Registro usado : AX.

IF_NO_SCROLL etq
    Bifurca a etq (etiqueta) si no se ha presionado scroll lock
    Registro usado : AX.

IF_NO_CONTROL etq
    Bifurca a etq (etiqueta) si no se ha presionado control
    Registro usado : AX.

EJEMPLOS

1.- Trazar una diagonal de 100 puntos

```
.MODEL TINY
.CODE
ORG 100H
INCLUDE BIOS.MAC
INICIO: SET_VIDEO GRAF_256
        MOV CX,100
P0:     ESC_PIX CX,CX
        LOOP P0
        LEE_CAR
        SET_VIDEO TEX_BN_80
        INT 20H
END INICIO
```

2.- Leer caracteres y mostrar ASCII y Scan code

```
.model tiny
.code
.386
include bios.mac
org 100h
inicio: mov dx,offset mensaje   ; mensaje final
   mov ah,9
   int 21h
p0:lee_Car
   cmp al,13
   jz fin
   mov dh,ah
   call imp_al
   mov al,dh
```

```
   call imp_al
   mov al,0dh
   call p_char
   mov al,0ah
   call p_char
   jmp p0
fin: int 20h


imp_al: mov cl,al
    shr al,4
    call p_hex
    mov al,cl
    and al,0fh
    call p_hex
ret
;*****
p_hex:
;*****
cmp al,0ah
jb p1
add al,'A'-0ah
jmp p_char
p1: or al,'0'
;******
p_char:
;******
esc_C al
ret
mensaje db 'Salga con Enter',0dh,0ah,'$'
end inicio
```

Ejercicios (Grupo 11)

1.- Instalar una interrupción 50h los siguientes servicios:

AH=1 Leer un número decimal en AL y llevarlo a binario
AH=2 Desplegar en decimal el número en AL

Probarla con un programa que la invoque.

2.- Investigar los servicios del DOS que instalan y recuperan direcciones de interrupciones.

III.3.8.- SINCRONIZACION

ESC b,r/m

Código de instrucción para el coprocesador; b es un número en seis bits y r/m un operando en memoria. Se traduce como una instrucción Fffff si existe en un juego del coprocesador y en realidad todas las Ffff son ESC.

WAIT

Espera que la señal TEST del coprocesador se active, que con ello indica que ha terminado sus tareas y se encuentra listo.
Es aconsejable colocarla luego de un grupo de instrucciones del coprocesador y antes de una instrucción del procesador para asegurar que el coprocesador ha terminado. Solo algunas instrucciones la usan obligatoriamente

LOCK

Es un prefijo que bloquea los buses durante la ejecución de la siguiente instrucción. Tiene sentido en un ambiente multiprocesador.

III.4.- PROBLEMAS CON ESTRUCTURAS DE DATOS

Se han mencionado varias estructuras de datos básicas:
Cadenas de caracteres: de longitud fija (F)
estructuradas (E)
ASCIIZ (A)
Números: Enteros en binario puro: de 1 a 8 bytes, con y sin signo
Fraccionarios de punto fijo en binario
Reales de punto flotante
Fechas: en 2 bytes con a¤o desde 1980
Horas: en 2 bytes

Las operaciones que se necesitan para cada una y se proponen como ejercicios son:

Cadenas:
Editar
Escribir
Llenar con repeticiones de otra
Concatenar
Buscar una dentro de otra
Obtener partes izquierda, derecha y central
Reemplazar un parte por otra

Se debe tratar cada una con todas las combinaciones posibles. Idealmente se debe considerar el entorno de un compilador donde existen cadenas constantes, variables en memoria y cadenas que se crean temporalmente para la evaluacion de expresiones.

Números: Editar en binario, decimal y hexadecimal cuando sea aplicable

Escribir
Convertir a cadena; de longitud fija o variable, rellena con ceros o espacios a la izquierda, con o sin inserción de comas separadoras de miles.
Convertir de cadena a número

Fechas:
Editar
Escribir
Obtener días entre dos fechas
Sumar días a una fecha
Sumar meses a una fecha
Sumar años a una fecha
Obtener dia, mes y a¤o
Componer a partir de dia, mes y a¤o

Horas:
Editar
Escribir
Obtener tiempo entre dos horas
Sumar segundos a una hora
Sumar minutos a una hora
Sumar horas a una hora
Obtener hora, minutos y segundos
Componer a partir de hora, minutos y segundos

Dos algoritmos generales muy importantes son ordenar y buscar en registros ordenados o desordenados.
La edición implica mostrar el valor inicial de la variable y permitir usarlo con todos los recursos de teclado; cursores, inserción, eliminación y avance por caracteres y palabras.

Además, se debe considerar pase de parámetros por pila y desarrollar cada uno como rutinas externas.

En el siguiente ejemplo se trata la edición básica de números. El número se supone en formato real y se usa el coprocesador para llevarlo a un formato intermedio (area_bcd) en BCD empaquetado y de allí se lleva a ASCII. Puede usarse como
modelo para otras entradas de datos.

1.- Desarrollar una rutina que edite con cursores a derecha e izquierda una cadena. Tomar una copia de la cadena en el área de la pila para trabajar en ella y no alterar necesariamente la cadena original.

```
.model tiny
.code
include bios.mac
public lee_cad,esc_cad

r struc                 ; estructura en pila
        dw ?            ; BP empilado
  area_cad db 255 dup (?) ; área de copia de la cadena
  c_loc    db           ; columna en pantalla durante edición
  i        db           ; posición relativa en el área de edición
  p_area   dw           ; puntero sobre el área en edición
  dir_ret  dw ?         ; dirección de retorno
  lon      db ?,?       ; longitud de la cadena
  di_emp   dw ?         ; dirección de cadena
  fila     db           ; fila
  col      db           ; columna
  atr      db ?,?       ; atributo
ends
lee_cad: sub sp,dir_ret-area_cad
        push bp
        mov bp,sp
        mov cl,[bp.lon]
        xor ch,ch
        cld
        lea di,[bp.area_cad]
        mov si,[bp.di_emp]
        rep movsb
        call desp       ; la despliega
        mov al,[bp.col] ; apunta a la primera columna
        mov [bp.c_loc],al
        mov [bp.i],1    ; corresponde también a la primera columna
        lea ax,[bp.area_cad] ; apunta a la segunda columna, la parte entera
        mov [bp.p_area],ax
ciclo:  SET_CUR [bp.fila],[bp.c_loc]    ; se coloca en la fila y la columna
        LEE_CAR         ; lee un carácter
        or al,al        ; es un carácter especial ?
        jnz letr_nums_? ; si no es, salta a verificar que carácter es
        cmp ah,_lef     ; es cursor a izquierda ?
        jnz right_?
        cmp [bp.i],1    ; si es cursor izquierda, no puede retroceder si esta
        jz ciclo        ; en la primera columna.
        call retro      ; Retrocede
```

```
        jmp ciclo
right_?: cmp ah,_rig
        jnz del_?
cur_Der: mov al,[bp.lon]
        cmp [bp.i],al   ; si es cursor derecha, no puede avanzar mas si esta en
        jz ciclo        ; la ultima columna
        call avanc      ; avanza
        jmp ciclo
del_?:  cmp ah,_del
        jnz ciclo       ; borrado de un carácter
        mov di,[bp.p_area]      ; si esta antes del punto decimal, debe
        lea si,[di+1]           ; correr los anteriores caracteres de la
        mov cl,[bp.lon]         ; izquierda sobre el e insertar un espacio
        sub cl,[bp.i]
        xor ch,ch               ; a la izquierda.
        cld
        rep movsb
        mov byte ptr [di],' '
fin_co: call desp
        jmp ciclo

letr_nums_?: cmp al,_cr
        jz enter_
caract: mov si,[bp.p_area]
        mov [si],al
        ESC_CAR ,[bp.atr],      ; lo escribe y avanza a la derecha
        jmp cur_der

enter_: call desp
fin:    mov cl,[bp.lon]
        xor ch,ch
        cld
        lea si,[bp.area_cad]
        mov di,[bp.di_emp]
        rep movsb

fin0:   pop bp
        add sp,dir_ret-area_cad
        ret 8                   ; descarta parámetros empilados

retro:  dec [bp.i]
        dec [bp.p_area]
        dec [bp.c_loc]
ret
avanc:  inc [bp.i]
        inc [bp.p_area]
        inc [bp.c_loc]
ret

esc_cad: mov si,[bp.di_emp]     ; despliega la cadena en la dirección empilada
        call desp1
        ret 8

desp:   lea si,[bp.area_cad]    ; despliega la cadena en el area_cad
desp1:  mov cl,[bp.lon]
        xor ch,ch
        mov bl,[bp.col]
        cld
c_desp: set_cur [bp.fila],bl
        lodsb
        push cx
        push bx
        esc_car  ,[bp.atr],
        pop bx
        pop cx
        inc bl
        loop c_desp
ret
end
```

2.- Usar la rutina del ejercicio 1

```
.model tiny
.code
extrn lee_cad:near
org 100h
.386
inicio: push 70h                ; empila atributo
        push 1010h              ; empila fila y columna
        push offset cadena      ; empila dirección del número
        push 20                 ; empila tamaño
        call lee_cad            ; lee la cadena
        int 20h
cadena db 'abcdefghi01234567890'
end inicio
```

3.- Escribir una macro que llame a la rutina del ejercicio 1

En el archivo STRINGS.MAC

```
LEE_CAD MACRO atr, fila, col, c ,siz
        mov ax,atr
```

```
        push ax
        mov ah, fila
        mov al,col
        push ax
        mov ax,offset c
        push ax
        push siz
        extrn lee_cad:near
        call lee_cad    ; lee la cadena
ENDM
```

4.- Usar la macro del ejercicio 3

```
.model tiny
.code
include STRINGS.MAC
org 100h
inicio: LEE_N 7,10h,10h,cadena,10
        int 20h
cadena 'a b c d e '
end inicio
```

Ejercicios (Grupo 12)

Sobre el programa de lectura de cadenas números implementar:

1.- Que la tecla Home lleve al inicio de la cadena
2.- Que ^M (control M) convierta todas las minúsculas en mayúsculas.
3.- Que ESC (Escape) deje la cadena sin cambios.
4.- Que Back space (retroceso) borre un retroceda y borre un carácter.
5.- Activar la inserción con Ins.
6.- Avanzar y retroceder por palabras con la combinación de control y los cursores derecha e izquierda.
7.- Ir al final de lo escrito con End
8.- Borrar palabras con ^Del y ^Back space
9.- Borrar todo lo anterior y todo lo siguiente con combinación de control y los cursores arriba y abajo.

III.5.- INTERFAZ CON LENGUAJES DE ALTO NIVEL

El presente punto es debido al aporte del universitario Steven Rojas Lizarazu, redactado durante la gestión I/1999 en la que fue Ayudante de esta materia en la carrera de ingeniería de sistemas de la Universidad Católica Boliviana.

Las herramientas de programación cada vez se distancian mas de la maquina y se acercan progresivamente al modo de pensar y al lenguaje de las personas. El uso del propio lenguaje de la maquina, o en su defecto el mas cercano a él, que es el Ensamblador o Assembler esta justificado:

* Cuando se desea realizar una operación no disponible en el lenguaje de alto nivel e incluirla en su repertorio de instrucciones.
* Cuando se desea alcanzar una velocidad de ejecución mayor a la de una funcion implementada en alto nivel.

Para el segundo punto debemos considerar que para optimizar un determinado proceso, debemos escribir un código ensamblador que supera al código generado por el propio lenguaje de alto nivel.

Una mayoría de los lenguajes de alto nivel soportan llamadas a módulos realizados en ensamblador. Para la comunicación entre lenguajes de alto nivel y ensambladores intervienen tres elementos:

* El modulo llamador.
* El modulo llamado.
* La interfaz de comunicación entre ambos módulos

La llamada, que puede ser NEAR (mismo segmento) o FAR (segmento distinto), varia según el lenguaje de alto nivel que se esta utilizando. Esta es la interfaz de control.

El método mas seguro para el paso de variables es utilizar la pila, pero existen muchas maneras de pasar información variando también según el lenguaje de alto nivel. Esta es la interfaz de datos. El paso de parámetros sobre la pila puede ser de manera directa (pasando el valor) o bien de manera indirecta (pasando la dirección).

Respecto a los registros se puede decir en general 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.

En el modulo llamado se debe guardar los registros utilizados en la pila y recuperarlos antes de devolver el control (instrucción RET) al modulo llamador. En el caso de que se pierda la dirección de retorno (IP o IP y CS, dependiendo del tipo de llamada NEAR o FAR respectivamente) por un mal manejo de la pila lo normal es que el programa se 'cuelgue'.

El modulo llamado puede utilizar la pila pero 'a ciegas' ya que no es posible saber el espacio con el cual dispone. En el caso de que la pila no sea suficiente el modulo deberá reservar su propio espacio en memoria.

Interfaz con C, C++ y C++ for Windows.- (C++)

Con InLine Assembler de Borland se puede escribir código ensamblador directamente dentro de un programa en C o C++ el cual soporta el uso de registros de 8 y 16 bits.

Sentencias ensamblador:

Podemos introducir sentencias ensamblador en cualquier punto de una funcion C++. Para ello se utiliza un bloque que empieza con la palabra reservada Asm y a continuación las sentencias ensamblador que deberán estar
entre llaves ({ }).

Asm {Declaraci¢n_Ensamblador [separador]}

Donde 'Declaraci¢n_Ensamblador' puede ser una o varias líneas de código y 'separador' es un punto y coma (;), una nueva línea o un comentario. Se puede poner dos o más sentencias en una línea si cada una es separada por un punto y coma. No es necesario utilizar ';' si cada sentencia se escribe en una línea diferente.

```
Asm
{
      Mov CX,DX;Sub AX,CX {resta CX de AX}; Mov CX,AX
}
```

Es equivalente a:

```
Asm
{
  Mov CX,DX
  Sub AX,BX {resta CX de AX}
  Mov CX,AX
}
```

Directivas ensamblador:

En C++ se puede trabajar solo con tres directivas ensamblador, DB, DW, y DD (definir byte, word, y doble word). Para la declaración de variables debe utilizarse la sintaxis de C++. Los datos de estas directivas son almacenados en el segmento de código. Para utilizar el segmento de datos, se debe utilizar las declaraciones de C++.

La directiva DB genera una secuencia de bytes, que puede ser una constante con un valor entre -127 y 255 o una cadena de caracteres. DW genera una secuencia de words, que puede ser una constante con un valor entre -32768 y 65535 o una dirección del tipo NEAR. DD genera una secuencia de words, que puede ser una expresión constante que tiene un valor entre -2147483648 y 4294967295 o una dirección del tipo FAR.

A continuación una lista de ejemplos para directivas ensamblador:

```
DB      0FFH    Un byte
DB      0,99    Dos bytes
DB      'A'     Ordinal('A')
DB      'Hola',0DH,0AH  Cadena + CR/LF
DB      12,"C++ "       Cadena estilo C++

DW      0FFFFH  Una word
DW      0,9999  Dos words
DW      'A'     Similar a DB 'A',0
DW      'BA'    Similar a DB 'A','B'
DW      MiVar   Offset de MiVar

DD      0FFFFFFFFH      Un doble-word
DD      0,999999999     Dos doble-words
DD      'A'     Similar a DB 'A',0,0,0
DD      'DCBA'  Similar a DB 'A','B','C','D'
DD      MiVar   Puntero a MiVar
```

Operandos:

Los operandos son una combinación de constantes, registros, símbolos y operadores. Las siguientes palabras reservadas tienen un significado propio para InLine Assembler:

AH AL AND AX BH BL BP BX BYTE CH CL CS CX DH DI DL DS DWORD DX ES FAR HIGH LOW
MOD NEAR NOT OFFSET OR PTR SEG SHL SHR SI SP SS ST TYPE WORD XOR

Estas palabras reservadas tienen preferencia sobre los identificadores declarados por el usuario. InLine Assembler evalúa todas las expresiones como valores enteros de 32 bits.

Registros ensamblador:

Una declaración en InLine Assembler debe guardar los registros SP, BP, SS y DS, pero puede modificar libremente los registros AX, BX, CX, DX, SI, DI ,ES y las banderas(flags) Al entrar a

una declaración ensamblador, BP apunta a la estructura de la pila actual, SP apunta a la cima de la pila, SS contiene el segmento de la pila, y DS contiene el segmento de datos actual.

Etiquetas ensamblador:

Una etiqueta (label) puede estar formada por una o mas letras (A. Z), los dígitos (0. .9), el símbolo de subrayado (_), seguida por dos puntos (:). La declaración debe ser realizada en la parte de declaraciones de etiquetas de los procedimientos (después del encabezado).

Procedimientos y Funciones ensamblador:

C++ permite introducir código ensamblador en cualquier parte de un procedimiento o funcion siempre y cuando este código está en un bloque Asm {...}. Por ejemplo:

```
int Multi(int X, int Y):
asm
{
 MOV     AX,X
 MUL     Y
}
```

Acceso a variables desde ensamblador:

Se puede acceder a las variables por su nombre, ya sea de 8, 16 o 32 bits. Asignar una variable del tipo int a DL, por ejemplo, se producirá un error en tiempo de compilación ya que el tipo de datos no es compatible.

El acceso a los parámetros de un procedimiento o funcion se realiza de manera similar a una variable local, a excepción de aquellos parámetros a los cuales se los paso anteponiendo '&', ya que se pasa el valor por referencia. En este caso se utiliza los registros base e índice para direccionar la memoria.

Ejemplo: Escribir una funcion que acepte un número introducido en tiempo de ejecución, para luego obtener como resultado la suma de ese número con todos los enteros positivos que le preceden.

```
int sumatoria(int n, int &res);
label b;        {Etiqueta para el ciclo de repetición}
{
  asm
  {
     mov ax,0          {AX en cero}
     mov cx,n                {ECX actuara de contador}
  b: add ax,cx         {Se va acumulando en AX los valores de CX}
     loop b            {Se repite hasta que CX sea cero}
     mov bx,word ptr res {Poner en BX la dirección de la variable Res}
     mov [bx],ax       {Devolver el resultado en la dirección apuntada por Res}
  }
}
```

Devolución de valores:

Cuando escribimos una sentencias ensamblador dentro de un procedimiento o funcion, lo único que debemos hacer es asignar el valor de retorno a una variable o a la funcion, pero si escribimos un procedimiento completamente en lenguaje ensamblador, el valor necesitara un registro apropiado para poder direccionar una posición de memoria y depositar dicho valor ahí para que el modulo llamador pueda acceder a ,l.

Para retornar un valor a través de una funcion se debe utilizar la variable @Result con la que cuentan todas las funciones de manera implícita (no es necesario declararla).

Ejemplo: El ejemplo anterior realizar con un procedimiento y una funcion, la funcion adicional realizara el cálculo al ser llamada por el procedimiento.

```
function suma(n: integer): integer;
label b;
begin
  asm
     mov ax,0
     mov cx,n
  b: add ax,cx
     loop b
     mov @Result,ax     {Se devuelve el resultado en @Result}
  end;
end;


procedure Sumatoria;
var num: integer;
begin
   read(num);          {Leer un número}
   write(suma(num));    {Llamar a la funcion Suma y devolver el resultado}
end;
```

Uso de procedimientos ensamblador externos:

Se puede utilizar un procedimiento o funcion de un programa ensamblador del cual se haya generado un archivo .OBJ enlazándolo con el programa Pascal. Este proceso se consta de dos pasos:

* Declarar en Pascal la funcion o procedimiento como External
* Indicar a Delphi en que archivo se encuentra el código, utilizando para ello,

la directiva {$L nombre}, donde nombre es la ruta y el nombre del archivo objeto generado por el ensamblador (.OBJ)

La funcion o procedimiento deberá cumplir con todas las características y convenciones ya descritas.

Un buen ejemplo de utilización de la interfaz es el siguiente:

Un juego realizado en un lenguaje de alto nivel, por ejemplo Pascal, que pueda ser jugado entre dos computadoras conectadas a través del puerto paralelo, veamos el caso.

Programas un juego de 3 en raya en Pascal (lo clásico). A este programa tendrás que realizarle algunas modificaciones para que puedas jugar con dos computadoras. Estas computadoras tendrán que estar conectadas a través del puerto paralelo de manera que intercambien información sobre las posiciones de las fichas, las jugadas, el jugador actual y si quieres puedes
hacer un Chat. Para hacer esto necesitas saber que el puerto paralelo puede ser manejado por 3 direcciones hexadecimales a las cuales puedes acceder con las instrucciones IN y OUT del ensamblador. Estas direcciones varían según la computadora; en realidad son direcciones base y tienen un determinado número de bits cada una. Resumiendo, la lógica que debes seguir para comunicar las dos computadoras y jugar es la siguiente:

1. Determinar cual será la computadora cliente y cuál será servidor.
2. Verificar si existe una conexión valida entre las computadoras a través de códigos enviados al puerto paralelo.
3. El cliente solicita empezar un juego nuevo o recuperar alguno (si tu programa en Pascal te lo permite), y espera un respuesta.
4. El servidor debe esperar a recibir un solicitud y atenderla, mandando a continuación un código al cliente.
5. Determinar el jugador que empieza la partida.
6. Enviar las nuevas coordenadas de cada ficha cada vez que se realiza una jugada.
7. Repetir el paso seis hasta que el juego termine.

IV.- PROGRAMACION CON SERVICIOS DEL BDOS

IV.1.- Servicios para dispositivos

Como se vio, las funciones mas primitivas de la INT 21H del BDOS tratan de escritura y lectura de caracteres. Tienen la ventaja de simplificar algunas tareas como por ejemplo el avance del cursor y el cambio de línea , que las funciones
del BIOS no hacen automáticamente. Estas operaciones se llaman de 'servicios para dispositivos de caracteres'.
Los servicios mas importantes del BDOS son una relativos a disco como ser abrir archivos, leer, escribir , renombrar, crear directorios y eliminar directorios entre otros. Estas operaciones se llaman de 'servicios para dispositivos de
bloque'.
El esquema vigente es el del llamado 'file handle' en oposición al antiguo FCB que no soporta adecuadamente el manejo de directorios.
En este esquema se suministra al BDOS un nombre de archivo (o directorio si corresponde) como cadena ASCIIZ en la que pueden indicarse drive, trayectoria de directorios, nombre y extensión y se obtiene como respuesta un 'file handle', una word que identifica al archivo para posteriores operaciones.

Existen 5 file handles propios de sistema operativo que están también disponibles para el usuario:

0 Entrada standard (teclado, CON:)
1 Salida standard (pantalla, CON:)
2 Error standard (pantalla, CON:)
3 Auxiliar (puerto serial, AUX)
4 Impresora standard (PRN:, LPT1)

Como en el BIOS, las interrupciones reciben sus parámetros y entregan sus resultados mediante registros. AH se usa normalmente para elegir un servicio de una interrupción. Por ejemplo, AH=1 es la lectura de un carácter para INT 21H.

Para los servicios siguientes, si no se indica lo contrario los errores se reportan en el bit de acarreo C. Si esta activo después de una operación, ha ocurrido un error en ella. Por ejemplo, si se quiere abrir un archivo que no existe, se reporta un error. La identificación del error queda en AX y se provee un modulo con los diferentes errores.

Cuando se ejecuta un archivo, se copia de disco a memoria y se forma una área llamada PSP (Program Segment Prefix) exactamente antes del programa.
Tiene lo siguiente:

Offset en PSP	Contenido
0	Instrucción INT 20H
2	Tamaño de la memoria en paragrafos
4	Reservado
5	Acceso al sistema operativo como CALL ssss:oooo
0A	Dirección de terminación
0E	Dirección de tratamiento de ^C
12	Dirección de tratamiento de error
16	Reservado
2C	Segmento de cadenas de entorno
2E	Reservado
50	INT 21H y RETF
55	Extension FCB 1
5C	FCB No.1
65	Extension FCB 2
6C	FCB No.2
80	Cantidad de caracteres pasados como parámetros de línea de comandos
81	Parámetros de línea de comandos
FF	Fin del PSP

Cuando se carga un programa .COM, todos los registros de segmento apuntan al inicio del PSP. De aquí que el programa deba empezar en 100H

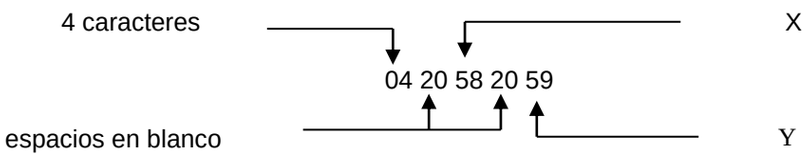
Cuando se carga un programa .EXE, SS apunta a la pila definida con .STACK, CS:IP al punto definido con END etq y DS apunta al PSP. Por esto, un programa .EXE no puede terminar con INT 20H que necesita que CS apunte a PSP para tomar
de allí la dirección de terminación. Se debe usar el servicio 4CH, que no necesita de esa condición. En estilo de escribir programas .EXE es el siguiente:

```
etq PROC FAR    PUSH DS             ; Se declara procedimiento lejano
                XOR AX,AX           ; Se empila el segmento del PSP
                PUSH AX             ; Se empila el desplazamiento O
                MOV AX,_data        ; Se carga DS con el segmento de datos
                MOV DS,AX
                . . . . .
                RET                 ; se desempila segmento y desplazamiento
          por ser procedimiento lejano y
                                    ; se retorna a PSP:0 donde esta INT 20H
                                    ; que termina el programa correctamente.
ENDP
```

Dos posiciones son especialmente importantes:

2C : Segmento de cadenas de entorno. Las cadenas de entorno se crean con el comando SET X = Z. El nombre X queda con el valor Z y se usa con diversos fines por el propio sistema operativo. Por ejemplo, PATH sirve para indicar los directorios de rastreo de programas. COMMSPEC indica la ubicación de COMMAND.COM para recargarlo.
Al ejecutarse un programa recibe sus propias cadenas de entorno. Si no se va a usarlas se las puede descartar con un servicio de administración de memoria.

80 : Longitud de parámetros y parámetros. Cuando se invoca un programa con parámetros como COPY X Y, en el área 80 queda:



Se proporciona el archivo BDOS.MAC con macros que llaman a interrupciones con servicios de teclado, pantalla, disco y otras. Los parámetros de estas macros corresponden a registros.

Para parámetros de entrada:
- si no se indican, no se altera el correspondiente registro. En ese caso, el registro ya debe tener el parámetro.
- si se indican, se altera el correspondiente registro.

Para parámetros de salida:
- si se indican, reciben el contenido del correspondiente registro.
- si no se indican, no se realiza ninguna acción.

En todo caso, los registros implicados son alterados.

Las macros desarrolladas son:

LEE_C c
Lee un carácter y lo despliega en pantalla.
c (AL) = carácter leído
Registros usados: AX

LEE_CE c
Lee una cadena estructurada apuntada.
c (DX) = cadena estructurada
Registros usados: AH,DX

ESC_C c
Despliega un carácter en pantalla.
c (DL) = carácter
Registros usados: AH,DL

ESC_C$ c

Escribe una cadena apuntada por DX y delimitada por $.
c (DX) = cadena
Registros usados: AH,DX

SET_DRIV u,n

Fija el disco por omisión.
u (DL) = disco. 0 es A:, 1 es B:, 2 es C: etc...
n (AL) = número de unidades disponibles.
Registros usados: AX,DL

GET_DRIV u

Informa el disco por omisión.
u(AL) = disco. 0 es A:, 1 es B:, 2 es C: etc...
Registros usados: AX

Ejemplo 1 : Obtener el drive actual y desplegarlo.

```
.model tiny
.code
org 100h
include bdos.mac
inicio:
get_driv
add al,'A'
esc_c al
esc_c ':'
esc_c 0dh
esc_c 0ah
int 20h
end inicio
```

Otra forma de este programa puede ser:

```
.model tiny
.code
org 100h
include bdos.mac
inicio:
get_driv p
add p,'A'
esc_c$ p
int 20h
p db ' :',0dh,0ah,'$'
end inicio
```

Ejercicio: Leer una letra de drive y fijar el drive en ella.
Probar casos de error.

MKDIR n

Crea un directorio

n (DX) = directorio.
Registros usados: AH,DX

CHDIR n

Fija el directorio actual.
n (DX) = directorio
Registros usados: AH,DX

RMDIR n

Elimina un directorio.
n (DX) = directorio
Registros usados: AH,DX

GETDIR u,n

Obtiene el directorio actual.
u (DL) = unidad. 0 es el drive actual ,1 es A, 2 es B, etc ..
n (SI) = directorio
Registros usados: AH,DX

CREATE n,a,h

Crea y abre un archivo
n(DX) = archivo
a(CL) = atributo
Los atributos disponibles son:

_file, archivo normal (se usa si no se indica)
r_o_file, archivo de solo lectura
hide_file, archivo oculto
sys_file, archivo del sistema

Se pueden construir combinaciones sumándolos entre si.
h(AX) = handle
Si el archivo existe previamente, es eliminado.
Registros usados: AX,CX y DX

SAFE_CREATE n,a,h

Similar a CREATE pero funciona solo cuando no existe previamente el archivo
n(DX) = archivo
a(CL) = atributo
h(AX) = handle
Registros usados: AX,CX y DX

OPEN n,m,h

Abre un archivo.
n (DX) = nombre
m (AL) = modo. Los disponibles son:
r_o_mod, solo lectura
w_o_mod, solo escritura
r_w_mod, lectura y escritura (se usa si no se indica)
h (AX) = file handle al retorno
Registros usados: AX y DX

READ h,a,q,r

Lee de un archivo abierto a memoria.
h (BX) = handle
a (DX) = área
q (CX) = cantidad a leer
r (AX) = bytes realmente leídos
Registros usados: AX,BX,CX y DX

WRITE h,a,q,r

Escribe de memoria a un archivo abierto.
h (BX) = handle
a (DX) = área
q (CX) = cantidad a escribir
r (AX) = bytes realmente escritos
Registros usados: AX,BX,CX y DX

GO_TOP h

Coloca el puntero de lectura/escritura al inicio de un archivo.
h(BX)= handle
Registros usados: AX,BX,CX y DX

GO_BOTT h,h_pos2,l_pos2

Coloca el puntero de lectura/escritura al final de un archivo.
h (BX)= handle
h_pos2 (DX) = parte alta de la posición absoluta del puntero
l_pos2 (AX) = parte baja de la posición absoluta del puntero
Registros usados: AX,BX,CX y DX

GO_TO h,h_pos1,l_pos1,

Coloca el puntero de lectura/escritura en una posición absoluta
h_pos1 (CX) = parte alta de la posición absoluta del puntero
l_pos1 (DX) = parte baja de la posición absoluta del puntero
h (BX) = handle
Registros usados: AX,BX,CX y DX

SKIP h,h_pos1,l_pos1,h_pos2,l_pos2

Coloca el puntero de lectura/escritura en una posición relativa a la actual
h_pos1 (CX) = parte alta del saldo relativo del puntero
l_pos1 (DX) = parte baja del saldo relativo del puntero
h (BX) = handle
h_pos2 (DX) = parte alta de la posición absoluta del puntero
l_pos2 (AX) = parte baja de la posición absoluta del puntero
Registros usados: AX,BX,CX y DX

CLOSE h

Cierra un archivo.
h (BX) =handle
Registros usados: AH y BX

ERASE n

Borra un archivo.
n(DX) = archivo
Registros usados: AX y DX

GET_ATR n,a

Lee los atributos de un archivo.
n(DX) nombre del archivo
a(CL) atributo
Registros usados: AX, CL y DX

Los atributos son los que se indicaron para la creación de archivos y dos mas:
_file, archivo normal (se usa si no se indica)
r_o_file, archivo de solo lectura
hide_file, archivo oculto
sys_file, archivo del sistema
vol_label, etiqueta de volumen

directory, directorio

SET_ATR n,atr

Fija los atributos de un archivo.
n(DX) nombre del archivo
a(CL) atributo
Registros usados: AX, CL y DX
Los atributos de etiqueta y directorio no se pueden fijar.

FIRST n,atr

Busca un primer elemento de directorio
n (DX) = archivo/directorio buscado. Puede tener símbolos ambiguos como * y ?.
atr(CL) = atributo del elemento buscado.
El resultado de la búsqueda queda en el DTA (Disk Transfer Area), un área de 43 bytes normalmente localizada en PSP:80h.

Offset en DTA		Contenido
0	21	bytes de uso por el DOS
21	1	byte de atributo del archivo
22	2	bytes de hora de creación/ultima actualización del archivo
24	2	bytes de fecha de creación/ultima actualización del archivo
26	2	bytes de parte baja de tamaño del archivo en bytes.
28	2	bytes de parte alta de tamaño del archivo en bytes.
30	13	bytes de nombre del archivo en ASCIIZ

Fecha y hora siguen los formatos expuestos anteriormente
Con un error (Carry = 1) se anoticia que la búsqueda ha sido fallida
Registros usados: DX,AX y CL

NEXT

Busca un siguiente elemento de directorio
Con un error (Carry = 1) se anoticia que la búsqueda ha concluido
Registros usados: AX

GET_DTA seg,off

Obtiene el segmento y desplazamiento de la DTA
seg (ES) = segmento
off (BX) = offset
Registros usados: ES,AH y BX

SET_DTA seg,off

Fija el segmento y desplazamiento de la DTA
seg (DS) = segmento
off (DX) = offset
Registros usados: DS,AH y DX

RENAME fil1,fil2

Renombra un archivo
fil1 (DS:DX) nombre original
fil2 (ES:DI) nuevo nombre
En la macro se asume que DS y ES apuntan a las direcciones correctas.
Registro usados: DX,DI,AX

ON_ERROR_21 etq

Bifurca a etq cuando se presenta un error (C=1) en muchos servicios de la INT 21h

Ejemplos

1.- Escribir un programa similar a TYPE que muestre el contenido de un archivo.

```
; T.ASM Despliegue del contenido de un archivo
; Se recibirá un nombre de archivo pasado como parámetro en el procesador
; de comandos.
; Se usara la macro OPEN abrir archivos;
;
; La lectura se hará con la macro READ existe un puntero de archivo que
; apunta al siguiente byte a leer/escribir; se puede colocar con otra
; funcion y cada vez que se lee o escribe, avanza al siguiente carácter.
;
; La INT 20H libera los handles abiertos, pero no actualiza tamaños ni
; fecha y hora de actualización. En este caso de solo lectura, no es
; necesario hacerlo; por ello no se cierra el archivo.

.model tiny
.code
org 100h
include bdos.mac
inicio: mov di,80h      ; busca en el PSP:80
        mov cl,[di]     ; verifica la cantidad de caracteres pasados
        xor ch,ch       ; como parámetro
        jcxz fin        ; si no existen caracteres, no hay nada que hacer
        mov al,' '      ; busca un carácter no blanco
```

```
        inc di
        cld                     ; proceso hacia adelante
        repz scasb
        jz fin                  ; si no existen caracacteres no blancos no hay nada que hacer
        lea dx,[di-1]                   ; dx apunta al primer no blanco
        repnz scasb             ; busca el final de los caracteres no blancos
        jcxz p0                 ; si termino la cadena, estamos en el 1er.
        dec di                  ; no blanco; en otro caso REPNZ avanzo uno mas
p0:     mov byte ptr [di],0
        open ,0,bx
        on_error_21 error  ; error

lee:    read ,area,512,cx
        jcxz fin                ; termina.
        lea si,area             ; Si no, escribe los caracteres leídos
escribe: esc_C [si]
        inc si
        loop escribe
        jmp lee
error:  shl ax,1                ; multiplicar el número de error por dos
        mov bx, offset t_err            ; para extraer de la tabla t_err
        add bx,ax               ; el mensaje de error
        esc_c$ [bx]
fin:    int 20h
        ; 0 1     2       3         4       5       6  7 8 9 10 11    12
 t_err dw ?,?,offset er1,offset er2,offset er3,offset er4,?,?,?,?, ?,offset er5
   er1 db 'No existe el archivo$'
   er2 db 'No existe camino$'
   er3 db 'Muchos archivos abiertos$'
   er4 db 'Acceso denegado$'
   er5 db 'Codigo de acceso invalido$'
área:

end inicio
```

2.- Escribir un programa similar a DIR que muestre el contenido de un directorio.

```
; D.ASM : Listado de Directorio según un parámetro en la línea de comando
; Se procederá a buscar los elementos del directorio que satisfagan las
; condiciones del nombre de archivo ambiguo pasado como parámetro en el
; procesador de comandos.
; Se usaran las macros FIRST, ESC_C y NEXT
;
.model tiny
.code
include bdos.mac
org 100h
inicio:
        cld                     ; procesamiento de cadenas hacia adelante
        mov bx,80h                      ; busca en el PSP:80
        mov cl,[bx]             ; verifica la cantidad de caracteres pasados
        xor ch,ch               ; como parámetro
        jcxz no_existe_p        ; si no existen caracteres, fija *.*  como parámetro
        mov al,' '              ; busca un carácter no blanco
        lea di,[bx+1]
        repz scasb              ; repetir si blanco
        jz no_existe_p          ; si no existen caracacteres no blancos,fija *.* como parámetro

        lea bx,[di-1]                   ; busca el final de los caracteres no blancos
        repnz scasb
        jcxz p0                 ; si termino la cadena, estamos en el 1er.
        dec di                  ; no blanco; en otro caso REPNZ avanzo uno mas
p0:     mov byte ptr es:[di],0  ; convierte en cadena ASCIIZ
        jmp fin_p

no_existe_p:
        mov word ptr [bx],'.*'  ; coloca '*.*',0 en el área apuntada por bx
        mov word ptr [bx]+2,'*' ; Se pone al revés porque es una word

fin_p:  first bx,_file          ; ds:BX apunta a una cadena ASCIIZ con el nombre del
archivo
ciclo_1:on_error_21 fin         ; Si no hay mas, CF=1 y termina búsqueda

        mov si,80h+30           ; SI apunta al nombre del archivo en DTA + 30

ciclo:  mov dl,[si]             ; despliegue del nombre del archivo
        or dl,dl
        jz cr_lf
        esc_c
        inc si
        jnz ciclo

cr_lf:  esc_c 0dh                       ; despliega CR/LF; cambia de línea
        esc_c 0ah
        next                    ; busca siguiente y repite
        jmp ciclo_1
fin:    mov ah,4ch
        int 21h
end inicio
```

Ejercicios (Grupo 13)

1.- Crear un directorio usando el parámetro de la línea de comando
2.- Cambiar de directorio usando el parámetro de la línea de comando
3.- Eliminar un directorio usando el parámetro de la línea de comando
4.- Escribir macros para escribir y leer cadenas ASCIIZ en una cadena de longitud fija donde se tenga un byte adicional con 0. Usarlas en lugar de los parámetros de las líneas de comandos en los siguientes ejercicios.
5.- Averiguar el directorio actual
6.- Copiar un archivo a otro. Considerar que el nombre de un archivo puede tener disco, trayectoria, nombre y extensión. Los caracteres permitidos son letras, números y los símbolos _ ^ $ ~ ! # % & - { } ( ) @ '
7.- Cambiar las minúsculas por mayúsculas en un archivo
8.- Cambiar las minúsculas por mayúsculas en un conjunto de archivos
9.- Eliminar los archivos bajo un nombre ambiguo. Las funciones de búsqueda en directorio no admiten llamadas a otros servicios en medio de ellas, por lo que deben formarse tablas de archivos con todos los que cumplan el nombre ambiguo dado.
10.- Renombrar archivos bajo un nombre ambiguo. Ver el ejercicio 9.
11.- Escribir una macro para obtener el espacio libre de un disco
12.- Investigar y verificar los siguientes servicios:
- bloqueo de archivos (5C)
- crear archivos de trabajo (5A)
- obtener y modificar fecha y hora de un archivo (57)

Escribir macros y ejemplos
13.- Documentar el servicio 59h que trata del manejo de errores y escribir una macro adecuada para usarlo.
14.- Escribir macros para los servicios de verificación de escritura en disco.
15.- Probar el uso de los file handles 0 a 4 propios del sistema operativo.

16.- Diseñar una estructura para el resultado de la búsqueda de los servicios de directorio y complementar el programa de listado del directorio mostrando el tamaño, la fecha y hora de creación. Mostrar los archivos ordenados por nombre.
Definir registros para manipular adecuadamente fecha y hora.
17.- Escribir macros para los servicios de manejo de fecha y hora del sistema.

IV.2.- Servicios para administración de memoria

GET_INT n,s,o

Obtener un vector de interrupción
n(AL) = interrupción
s(ES) = segmento
o(BX) = offset
Registros usados: AX,ES,BX

SET_INT n,s,o

Fijar un vector de interrupción
n(AL) = interrupción
s(DS) = segmento ; si se indica debe ser un registro o memoria
o(DX) = offset
Registros usados: AX,DS,DX

Estos servicios pueden ser fácilmente programados, pero ya están disponibles.

La administración de memoria sigue la disposición de bloques contiguos de control y de datos. En los bloque de control esta el estado del bloque de datos contiguo (libre/ocupado) y su tamaño.

GET_MEM p,s,t

Obtener memoria
p (BX) = paragrafos deseados
s (AX) = segmento asignado, si es posible
t (BX) = tamaño disponible si no es posible la asignación
Registros usados AX,BX

FREE_MEM s

Liberar memoria
s (ES) = segmento previamente asignado
Registros usados AX,ES

RESET_MEM s,n,t

Modificar el tamaño asignado a un bloque
s(ES) = segmento del bloque actual
n(BX) = paragrafos deseados
t(BX) = máximo tamaño disponible si no posible la modificación
Registros usados AX,BX,ES

Cuando un programa pasa a ejecutarse, recibe como asignación toda la memoria. Para quedar residente o ejecutar otro programa, debe modificar esta asignación al tamaño suficiente y necesario para el. Se lo puede hacer con RESET_MEM usando como segmento del bloque actual al PSP del programa. Se deben incluir los 100H bytes propios del PSP.
También puede liberar completamente el segmento de cadenas de entorno si no se va a usar.

EXEC ns,no,bs,bo
Carga y ejecuta un programa
ns(DS) = segmento de una cadena con el nombre del programa a ejecutar en formato ASCIIZ
no(DX) = offset de la cadena con el nombre del programa a ejecutar en formato ASCIIZ

bs(ES) = segmento de un bloque de memoria con los parámetros del programa a ejecutar
bo(BX) = offset de un bloque de memoria con los parámetros del programa a ejecutar
Registros usados: AX,DS,DX,ES,BX
El bloque de parámetros debe tener:

offset	tamaño	contenido
0	2	Segmento de cadenas de entorno. Con 0 se obtiene una copia del actual entorno
2	2	Offset de línea de comando
4	2	Segmento de línea de comando
6	2	Offset de FCB 1
8	2	Segmento de FCB 1
10	2	Offset de FCB 2
12	2	Segmento de FCB 2

14 bytes

EXEC_COM no,b

Desde un programa .COM carga y ejecuta otro programa pasando como parámetros las áreas que el mismo usa
no(DX) = offset de la cadena con el nombre del programa a ejecutar en formato ASCIIZ
b = área de bloque de parámetros de 14 bytes
Registros usados: AX,DS,DX,ES,BX

TERMINATE c

Termina un programa y deja código de terminación
c(AL) = Codigo de retorno
Registros usados: AX

GET_RET_CODE m,c

Obtiene modo y código de terminación de un programa
m(AH) = modo de terminación

00 = Normal
01 = ^C
02 = Error critico
03 = Termino y quedo residente
c(AL) = código

Registros usados: AX

TSR n,c

Termina un programa, queda residente y deja código de terminación
n(DX) = Longitud reservada
c(AL) = Codigo de terminación
Registros usados: AX,DX

TSR_COM etq,c

Termina un programa .COM ,queda residente y deja código de terminación
etq = etiqueta de terminación
c(AL) = código de terminación
Registros usados: AX,DX

Ejemplo 1 : Dejar residente un programa reloj que muestre la hora en la esquina superior derecha permanentemente.

```
;
; Reloj residente el la esquina superior derecha
; Basado en el programa REPRELOJ de M.A. Rodriguez R.
; (8088-8086 /8087 Programación ensamblador en entorno DOS)
; Descripción:
;   Instalación de rutina residente para escribir la hora en la
;   esquina superior derecha de la pantalla.
;   El temporizador (timer) emite una interrupción tipo 1Ch con la
;   frecuencia f = 1193180/65536 = 18.206481 veces/seg.
;   Por minuto: f x 60    = 1092.3889
;   Por hora  : 1092.3889 x 60 = 65543 = 32761 x 2 (aprox.)
;
; Símbolos:
   pantalla equ 0b800h  ; segmento memoria de pantalla
;
;-----------------------------------------------------------------------
;
.model tiny
.code
include bdos.mac
        org     100h          ; origen por ser fichero tipo COM
inicio :
        jmp     instalar      ; bifurca a la rutina de inicialización
;
;-----------------------------------------------------------------------
;
; variables
; ---------
;
```

```
hora     db   8 dup ('?')              ; hora en formato hh:mm:ss
;
;-------------------------------------------------------------------
;
; rutina reloj
; ------------
;
reloj    proc
         cli          ; inhibir interrupciones
;
; empilar   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,32761
         div  bx            ; dx = resto, ax = horas*2
         shr  ax,1          ; ax = horas (dividir entre 2)

; las siguientes líneas son una corrección necesaria; la división deja su
; residuo en el acarreo; si existe se debe sumar el divisor (el residuo es uno)
; al residuo que queda en DX. Sea x = f*d + r ; si f= 2*k+s; s es 0 o 1
; x = k*2*d+s*d+r = k * (2*d) + ( s*d +r). Así, el verdadero residuo es s*d+r
;
         jnc p0
         add dx,bx

p0:      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            ; dx = resto, ax = segundos
         mov  dh,al         ; dh = segundos
;
; poner la hora en formato decimal sobre campo hora
;
         mov  bh,10              ; bh = 10 (constante)
;
         mov  ah,0               ; ah = 0
         mov  al,ch              ; ax = horas
         div  bh                 ; ax = horas en decimal
         add  ax,3030h
         mov  word ptr hora,ax   ; mover
         mov  hora+2,':'
;
         mov  ah,0               ; ah = 0
         mov  al,cl              ; ax = minutos
         div  bh                 ; ax = minutos en decimal
         add  ax,3030h
         mov  word ptr hora+3,ax  ; mover
         mov  hora+5,':'
;
         mov  ah,0               ; ah = 0
         mov  al,dh              ; dh = segundos
         div  bh                 ; ax = segundos en decimal
         add  ax,3030h
         mov  word ptr hora+6,ax  ; mover
         mov  hora+8,':'
;
         mov  ah,0               ; ah = 0
         mov  al,dl              ; dl = cent,simas
         div  bh                 ; ax = segundos en decimal
         add  ax,3030h
         mov  word ptr hora+9,ax  ; mover
;
; direccionar zona memoria de pantalla con es:[di]
;
         mov  ax,pantalla
         mov  es,ax
         mov  di,2*(80 - length hora)  ;length devuelve el tamaño de la cadena
;
; mover la hora
```

```
;
        mov  cx,length hora
        mov  si,0                   ; desplazamiento inicial
mover:
        mov  al,hora[si]             ; car cter campo hora
        mov  es:[di],al              ; moverlo a memoria de 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 interrupción debe terminar con iret
reloj   endp
;
;--------------------------------------------------------------------
;
; instalación
; ----------
;
instalar:
;
; direccionar segmento de vectores de interrupciones con es
;
        SET_INT 1Ch,,reloj
        TSR_COM <offset instalar-offset inicio+100h>
;
;--------------------------------------------------------------------
;
end  inicio
```

Ejemplo 2: Ejecutar un comando interno de COMMAND o un programa .BAT
Para este tipo de problemas se ejecuta el programa COMMAND.COM /C con comando interno o programa como parámetro.


```
.model tiny
.code
include bdos.mac
org 100h
inicio: mov ax,cs
        reset_mem ax,30        ; se reduce el tamaño del programa
        exec_com command,bloque ; se ejecuta COMMAND pasando como parámetro
                                ; la misma cadena que este programa recibió.
        terminate
command db 'c:\command.com',0
bloque dw 7 dup (?)
end inicio
```

Si se ensambla como X.COM, se ejecuta un comando DIR con X /C DIR

Ejercicios (Grupo 14)

1.- Escribir un programa que desinstale el reloj residente. Se debe liberar el PSP residente y restituir el vector de interrupción original 1Ch.

2.- Dejar residente un programa con servicios de lectura/escritura de números binarios de un byte en la INT 50h. Escribir un programa que pruebe los servicios residentes.

3.- Escribir un programa de administración dinámica de memoria. Requerir un espacio 64K y administrarlo con bloques de datos y control con la siguiente estructura:

Bloque de Control | Bloque de datos

E	S1	S0	

E = byte de estado : 0 = libre 1 = ocupado
S1,S0 = tamaño del bloque de datos en una word

En principio el área debe tener el siguiente aspecto:

0	64K – 3	

Debe recibir dos tipos de requerimientos:

Obtener memoria; parámetros de entrada: CX = Tamaño requerido
Salidas : BX = 0FFFFH. Asignación imposible

otro valor indicara el offset asignado

Liberar memoria; parámetros de entrada: BX = Offset a liberar

La liberación de memoria debe considerar la fusión de bloques contiguos.

4.- Investigar la carga y ejecución de overlays. Modificar los programas del ejercicio 2 para no dejar residente los programas de servicio y mas bien cargarlos en tiempo de ejecución. Revisar la posibilidad de pasar parámetros por la pila considerando inclusive entregar la dirección de la pila en el área de parámetros de línea de comando del programa cargado en tiempo de ejecución.

V.- DISPOSITIVOS

V.1.- Disco y diskette

Los discos estan compuestos por varias 'superficies' de forma circular donde cada una anverso o reverso es tambien llamada 'cara'('side'). En el centro de estas superficies esta un eje impulsado por un 'motor de pasos' que les da movimiento rotatorio.
Sobre ellas, las circunferencias concentricas centradas en el eje son llamadas 'pistas'. Existiendo muchas caras, tambien se denominan 'cilindros', que es la figura que aparece cuando se considera la misma pista en varias caras.
Los sectores de cada pista son genericamente denominados 'sectores fisicos' del disco. Contienen 512 bytes (normalmente) que es la mas peque¤a cantidad de informacion que puede almacenarse en un disco y por ello son llamados la 'unidad fisica de almacenamiento'.
Los sectores son leidos y escritos por 'cabezas lecto/escritoras' situadas a una infima distancia de las superficies que se mueven sobre un radio comun a todas ellas. El contacto de estas cabezas con las superficie se denomina 'aterrizaje de cabezas' y normalmente da¤a permanentemente los sectores involucrados.
Tambien se denomina 'cabezas' a las caras ya que a cada cara corresponde una cabeza exclusivamente.
En un diskette de alta densidad (1.4 Mb.) la estructura usual es la siguiente

Número de caras : 2 (numeradas 0 y 1)
Número de pistas : 80 (numeradas 0 a 79)
Número de sectores por pista : 18 (numerados 1 a 18)

Existen 2*80*18=2880 sectores
Considerando el tama¤o de 512 bytes por sector, en el disco estan disponibles 1440 Kb, algo mas de 1.4 Mb (pero 35 Kb menos que 1.44 Mb).
Se puede manipular directamente los sectores con la INT 13H del BIOS con los siguientes parametros de entrada:

AL = cantidad de sectores a manipular
CL,CH = pista en 10 bits (2 bits altos de CL y 8 bits de CH)
CL = sector (6 bits bajos)
DH = cara
DL = unidad 0 = A 1 = B 2 = C. Suma 80H para discos duros
ES:BX = area de lectura

Las funciones codificadas en AH, entre otras, son:
2 = lectura
3 = escritura
4 = verificacion
5 = formateo (solo pistas y no sectores individuales)

Al termino, reportan en AH y C (carry flag):

0 = operacion sin errores. (C=0)
otro = codigo de error (C=1)

Notese con el siguiente calculo que se puede disponer de hasta 8 Gb con este esquema :

1 Kb (pistas) * 64 (sectores) * 256 (caras) * .5 Kb (bytes/sector) = 8 Gb

Esta cantidad es suficiente para diskettes pero no asi para discos duros que tienen 256 caras y mayor capacidad, que deben ser divididos en varias unidades logicas. El primer sector de un disco duro (nunca un diskette) contiene una 'tabla de particion' que informa de esta division en varias unidades logicas. En un diskette existe una sola particion.

El orden natural en que se accede los sectores varia sector, cara y pista en ese orden; de un sector a otro es deseable cambiar primero de sector, lo que resulta muy logico ya que los sectores de una misma pista estan moviendose bajo su cabeza correspondiente. Luego, cuando se han recorrido todos los sectores de una pista, como todas las cabezas se mueven simultaneamente, es razonable pasar a la siguiente cabeza que esta sobre la misma pista.
Cuando se han recorrido todos los sectores de todas las pistas, recien se puede cambiar de pista moviendo las cabezas lecto/escritoras a la siguiente pista.
Asi, el mejor orden de acceso a los sectores de un diskette de 1.4 Mb es el siguiente:

Pista	Cara	Sector
0	0	1
0	0	2
	....	
0	0	18
0	1	1
0	1	2
	....	
0	1	18
1	0	1
1	0	2
	....	
1	0	18
1	1	1
1	1	2
	....	
1	1	18

En la tabla de particion se indican el primer y el ultimo sector que le corresponden. En ellas se definen 'sectores logicos' bajo una secuencia correlativa simple desde 0 correspondiendo al primer sector fisico de la particion.

Para almacenar informacion se considera el 'cluster', una coleccion de sectores consecutivos como 'unidad logica de almacenamiento', sobre el que se crean 'archivos', colecciones de clusters no necesariamente consecutivos organizados bajo cierta forma de 'lista encadenada' en las denominadas 'FAT'.

La estructura de un disco para almacenar informacion contempla:

Boot Record : un sector con parametros propios del disco con el siguiente formato:
- JMP al inicio del programa cargador = 3 bytes
- nombre Original Equipment Manufacturer (OEM) = 8 bytes
- bytes por sector = 2 bytes
- sectores por cluster = 1 byte
- sectores reservados = 1 byte (sectores usados por B.R.)
- cantidad de FATs = 1 byte
- cantidad de entradas del directorio raiz = 1 byte
- cantidad total de sectores = 2 bytes aplicable en discos peque¤os o diskettes
- descriptor de soporte = 1 byte
  - F8 = disco duro
  - F0 = diskette alta densidad
- sectores por FAT = 2 bytes
- sectores por pista = 2 byte
- cantidad de caras = 2 byte
- cantidad de sectores ocultos = 4 bytes : sectores no visibles en la particion
- cantidad total de sectores = 4 bytes
- número de unidad = 1 byte: 80h = primer disco del sistema
- marca de Boot Record ampliado = 1 byte
  - 0 para diskettes
  - 29 para discos peque¤os
- número de serie = 4 bytes
- etiqueta = 11 bytes
- tipo de sistema de archivo = 5 bytes: FAT16 o FAT32 o FAT12

FAT 1 : File Allocation Table
- Existen tres tipos caracterizados por el tama¤o de sus entradas:
  - 12 bits : usada en diskettes
  - 16 bits : usada en discos menores a 1 Gb
  - 32 bits : usada en discos mayores a 1 Gb
- Cada entrada corresponde a un cluster tiene el siguiente significado:
  - 0 = cluster libre para asignacion
  - F...F7 = cluster con un sector malo
  - F...F8 = fin de un archivo
  - otros valores = siguiente cluster dentro de un archivo
- Los clusters se inician en 0 y el cluster 2 corresponde al inicio del Area de datos que se indica luego.

FAT 2: idealmente copia de FAT 1

Directorio: por cada archivo del directorio raiz se tiene:
nombre del archivo : 8 bytes. En el primer byte se indica
0 : la entrada esta libre
E5H : la entrada esta borrada
Extension : 3 bytes
atributo : 1 byte identico al usado en los servicios de directorio
reservados : 10 bytes
hora de creacion: 2 bytes en formato ya visto
fecha de creacion : 2 bytes en formato ya visto
cluster inicial : 2 bytes. Indica el cluster en que se tiene la cadena de clusters que contienen al archivo
tama¤o del archivo : 4 bytes. La serie de clusters no es suficiente para dar el tama¤o de un archivo.

En Windows los nombres largos anteceden a los nombres cortos en una entrada anterior en otro formato.

Area de Datos : contenido de los archivos

Los subdirectorios son archivos cuyas entradas son similares a las del directorio y se a¤aden dos entradas . y .. que apuntan a si misma (para bajar a sub-directorios dentro de si mismos) y al directorio padre (para subir al el) en el cluster inicial.

El utilitario DISKEDIT 8.0 de Symantec Corporation muestra todos estos aspectos visualmente. No es capaz de reconocer discos con FAT 32 en el modo logico como son los actuales y debe usarse solo con diskettes o discos antiguos.

El acceso a sectores logicos es posible con la interrupciones 25H (lectura) y 26H (escritura) que usean los siguientes parámetros:
AL= unidad (0=A: 1=B: 2=C: etc. no es necesario sumar 80h para discos)
DS:BX = puntero a area de memoria
CX = número de sectores a leer DX = primer sector a leer

Para discos se debe usar CX= 0FFFFH y en el area de memoria se debe tener:

primer sector a leer (4 bytes)
número de sectores a leer (2 bytes)
puntero a area de memoria (4 bytes)

Para leer el sector logico 0 , que es el boot record de un disco C: se puede hacer:

```
.model tiny
.code
org 100h
inicio:
mov al,2
mov bx, offset area
mov dx,0
mov cx,0ffffh
mov s,ds
int 25h
int 20h
area dd 0
    dw 1
    dw 200h    ; el sector leido queda en ds:200H
s   dw
end inicio
```

Para el diskette B:

```
.model tiny
.code
org 100h
inicio:
mov al,1h
mov bx, 200h
mov dx,0
mov cx,1
int 25h
int 20h
end inicio
```

Ejercicios (Grupo 15)

1.- Mostrar el contenido de un boot record interpretando debidamente sus campos
2.- Escribir un programa que muestre el directorio accediendo por sectores logicos a area del directorio.
3.- Escribir un programa que muestre el contenido de un archivo accediendo por sectores logicos.
4.- Escribir un programa que revise los sectores de un diskette
5.- Analizar las posible anomalias en la estructura
6.- Verificar el efecto de un borrado de archivo en la cadena de clusters en la FAT y escribir un programa que recupere archivos borrados en un diskette.
7.- Investigar el servicio 44H de INT 21H que reemplaza a INT 25H e INT 26H.

V.2.- Pantalla

Como se indico en el Capitulo 2 , la pantalla esta almacenada en un area de memoria que en modo texto esta localizada en B800:0 y contiene el caracter y el atributo para cada posicion en ella. Esta area es permanentemente desplegada por el tubo de rayos catodicos pasando por un generador de caracteres que convierte cada caracter en una serie de puntos que forman la imagen en pantalla.

En los modos graficos este generador de caracteres usa dos tablas de mapas de bits para formar caracteres. El tipo de caracteres mas primitivo '8*8' usa ocho bytes para formar un caracter. Cada uno de los bits implica con 1 el dibujo de un punto y con 0 su ausencia. Por ejemplo el número 0 se puede presentar asi:

```
 bits  76543210

         XXXXX      7C
       X       X X   86
       X      X    X   8A
       X   X       X   92   se almacena 7C 86 8A 92 A2 C2 7C 00
       XX          X   A2
        X XX XX     7C
                    00
```

Los caracteres 0 a 127 estan definidos desde f000:fa6e y los caracteres 128 a 255 estan apuntados por la interrupcion 1fH (0:7ch). Esta segunda tabla permite un mecanismo de cambio de forma a los caracteres mediante con el siguientes esquema:

1.- Construir una tabla con nuevas formas para todos los caracteres. Puede basarse en las tablas f000:fa6e y la de int 1fH.

2.- Escribir una rutina de despliegue de caracteres que:

- para caracteres en el rango 0 a 127, incremente en 80h el caracter a desplegar y apunte la interrupcion 1f al inicio de la tabla.

- para caracteres en el rango 128 a 255, apunte la interrupcion 1f al medio de la tabla (donde esta la forma para el caracter 128).

- en ambos casos debe procederse a desplegar el caracter reformado.

Se puede inclusive conseguir el despliege de dos caracteres en uno solo creando caracteres "angostos" basados en los caracteres originales; por ejemplo el 0 anterior podria reducirse asi:

```
 bits  76543210        76543210        Operaciones OR que convierten 7C en E0 :
       XXXXX            XXX       E0
      X     X  X        X  X  X    B0              7C
      X   X    X        X   XX     B0         11 10 11 00
      X  X     X        X  X  X    D0            1  1   1   0
      X X      X        XX   X     D0              E0
      XX       X        X     X    90
       XXXXX            XXX       E0
                                    00
```

Esto puede conseguirse mediante un procedimiento que realize una operacion OR entre cada par de bits de los bytes de la forma original y genere un byte donde la parte alta tenga los cuatro resultados y la parte baja sea 0.

Tambien pueden conseguirse caracteres "chatos". Un resultado asi puede ser util para visualizar archivos de texto con cien o doscientas columnas en toda su anchura pero sin detalles precisos.

Las areas donde se almacenan las pantallas y su formato son las siguientes :

texto a 16 colores de 80 columnas: b800:0 , caracter y atributo
grafico de 16 colores, 640 filas x 480 columnas a000:0, mapa de bits
grafico de 256 colores, 320 filas x 200 columnas: a000:0, color del pixel

A continuacion se presentan un utilitario compuesto por tres programas que copia pantallas en modo texto a un archivo. Debe ser usado en ambiente DOS ya que usa como conexion la tecla de impresion de pantalla (Print Screen) que no esta habilitada en Windows.

PRTSCMEM: (Print Screen copia a memoria) queda residente, guarda y reemplaza los servicios de la tecla Print Screen y hace que cada vez que se presione, la pantalla sea copiada a un area de memoria dentro del mismo programa. El cambio
se realiza en la INT 5 que esta asociada a Print Screen.

PRTSCFIL: Lleva todas las pantallas copiadas al archivo C:\PANTALLA.

PRTSCPRN: Devuelve Print Screen a su funcion original de impresion de la pantalla.

```
;*****************************************************************
; PRTSCMEM: Pantalla a memoria
;
; Copia n pantallas a un area de memoria reservada
;
; PRTSCFIL recupera las pantallas copiadas en el archivo
; C:\Pantalla
;
; PRTSCPRN reasigna Prt Sc a impresora
;*****************************************************************
include bdos.mac
n = 30   ; Número maximo de pantallas a copiarse

.model tiny
.code
org 100h
p0: jmp inicio  ; al principio queda la parte residente y salta al instalador
              ; del programa residente.

seg_5   dw      ;Se guardan el segmento y offset de la INT 5 original (259)
off_5   dw      ; (261)

cr_lf   db 0dh,0ah  ;(263) Marca del programa residente. Se usa para detectar si
```

```
                ; este programa ya esta o no en memoria. Como la INT 5
                ; apuntara a este segmento, se verificara que en la po-
                ; sicion de CR_LF este efectivamente 0d0ah.
                ; 256 + 3+ 2+2 = 263 (100h= 256, JMP usa 3 y seg_5 y
                ; off_5 usan cada una uno 2 bytes.

s_ss    dw ?        ;(265) Se salva el segmento y puntero de la pila actual
s_sp    dw ?        ;(267) para que el programa en ejecucion quede invariable

p_area  dw offset fin_residente+14  ; (269) puntero al area de proxima copia
p_inic  dw offset fin_residente+14  ; (271) punto inicial copia. 14 bytes
                        ; corresponden a la pila de este programa

inicio_rutina:          ; nueva direccion de INT 5. No olvidar que debe
                ; ejecutarse en cualquier momento, cada vez que se
                ; presione Print Screen.

cmp cs:[p_area],0+offset fin_residente+n*80*25 + 14 ;se copiaron n pantallas ?
je retorno              ; si, entonces terminar

mov word ptr cs:[s_ss], ss      ; usando CS, se salva SS y SP del programa
mov word ptr cs:[s_sp], sp      ; que se esta ejecutando.

mov sp,cs                       ; se hace que apunten al final de este residente
mov ss,sp                       ; y dejen 14 bytes que se van a empilar,
mov sp,offset fin_residente + 14

push ax                 ; se empilan 7 registros, que son 14 bytes,
push bx
push cx
push si
push di
push es
push ds

mov ax,0b800h           ;ax     proceso de copia y registros usados
mov ds,ax               ;ds
mov si,0                ;si
mov cx,80 * 25          ;cx
mov bx,cs               ;bx
mov es,bx               ;es
mov di,cs:[p_area]      ;di
cld
ciclo: movsb
inc si
loop ciclo

mov word ptr cs:[p_area],di     ; se actualiza p_area

pop ds  : se desempilan los registros y se restituyen ss:sp originales
pop es
pop di
pop si
pop cx
pop bx
pop ax
mov ss, word ptr cs:[s_ss]
mov sp, cs:[s_sp]
retorno:iret            ; se termina con IRET

fin_residente:
ya_esta db 'Prt Sc ya fue asignada a memoria anteriormente !!$'
se_hizo db 'Prt Sc asignada a memoria exitosamente$'

inicio:                 ; Instalador del residente
mov ah,35h              ; Se recupera la direccion de la interrupcion 5
mov al,5
int 21h
cmp word ptr es:[263],0a0dh   ;Comparar con cr_lf (0d_0a)
jnz no_esta
mov dx, offset ya_esta  ; Mensaje de que el programa ya esta en memoria
mov ah,9
int 21h
int 20h

no_esta: mov ax,es  ; el programa no esta residente en memoria
mov seg_5,ax        ; PrtSc original es salvado
mov off_5,bx

mov dx,offset inicio_rutina     ; DS:DX es la direccion de la interrupcion
                        ; la nueva INT 5
mov ah,25H   ; Se asigna la interrupcion 5 al inicio de este programa.
mov al,5H
int 21H

free_mem [ds:2CH]       ; se libera la copia de las cadenas de ambiente
                ; creadas con SET y se crea para cada programa.
                ; (tal vez sea util mantener en Windows)
mov dx, offset se_hizo              ;Mensaje de realizacion exitosa
mov ah,9
int 21h
l_resid = (offset fin_residente- offset p0) +100h+ n*80*25 +14
                ;n*80*25 (pantallas salvadas)
```

```
                    ;14 bytes para la pila del residente (local)
tsr_com l_resid         ; macro para quedar residente

end p0

;*****************************************************************
; PRTSCFIL recupera las pantallas copiadas en el archivo
; C:\PANTALLA
;*****************************************************************
.model tiny
.code
org 100h

p0:mov ah,35h   ; se recupera en ES el segmento de la interrupcion 5
mov al,5
int 21h
cmp es:[263],0a0dh      ; se revisa si se instalo antes el residente
jz ya_esta
mov dx, offset no_esta
mov ah,9
jmp final

ya_esta:
mov dx, offset n ;asciiz de la pantalla
mov cx,0h       ; se crea el archivo C:\PANTALLA
mov ah,3ch
int 21h

mov bx,ax
mov ax,es
mov ds,ax
mov si,ds:[271]         ; se empieza en el principio del area de pantallas
mov bp,si               ; copiadas
std
ciclo: cmp si,ds:[269]  ; se verifica no haber llegado al final
je cierre
mov dx,si               ; se empieza por el final de cada linea
add si,80
lea di,[si-1]
mov cx,80
mov ax,4020h
repz scasb              ; para saltar los espacios en blanco al final
jz cr_lf
inc cx
int 21h                 ; se graban los caracteres de la pantalla
cr_lf: mov dx,263       ; se incluye un fin de linea (que es precisamente la
```



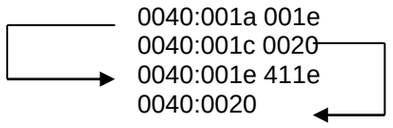
```
mov cx,2                ; marca del programa residente
mov ah,40h
int 21h

jmp ciclo

cierre: mov word ptr ds:[269],bp        ; se vuelve a apuntar al principio
                                        ; del area de pantallas
mov ah,3eh      ; se cierra el archivo creado
final: int 21h
       int 20h

n db 'C:\pantalla',0
no_esta db 'Prt Sc no fue asignada a memoria anteriormente !!$'
end p0

;*****************************************************************
; PRTSCPRN reasigna Prt Sc a impresora
;*****************************************************************
.model tiny
.code
org 100h
p0:mov ah,35h   ; se recupera en ES el segmento de la interrupcion 5
mov al,5
int 21h
cmp es:[263],0a0dh  ; se verifica que el programa residente este presente
jz ya_esta
mov dx, offset no_esta
mov ah,9
jmp final

ya_esta:
mov ax,es:[259]
mov ds,ax
mov dx,es:[261]
mov ah,25H              ; Se asigna la interrupcion 5 a su valor original
mov al,5H
int 21H
mov ah,49h              ; se libera el programa residente. ES contiene el
final: int 21h          ; segmento donde se cargo
       int 20h
no_esta db 'Prt Sc no fue asignada a memoria anteriormente !!$'
end p0
```

V.3.- Teclado

Desde el primer PC de 1982, se implento el funcionamiento del teclado mediante interrupciones, a diferencia de la generacion anterior de micro-computadores basados en el 8080 que operaba mediante comunicacion programada.
En este modo, el teclado esta conectado al controlador de interrupciones 8259x y cada vez que se presiona o libera una tecla, requiere una interrupcion.
El controlador interrumpe al procesador y lo envia a procesar la interrupcion 9.
El mecanismo basico es una cola circular en la que la interrupcion 9 inserta caracteres al final y los servicios del BIOS leen caracteres desde la cabeza.
El teclado suministra un byte identificador de la tecla denominado 'scan code' que debe ser traducido a un caracter ASCII. En la cola circular se mantienen ambos bytes; el caracter ASCII que se asocia a la tecla y el 'scan code'.
La localizacion y estructura de la cola circular son las siguientes:

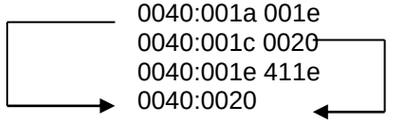
0040:001a PC;puntero de 16 bits a la cabeza de la cola
0040:001c FC;puntero de 16 bits al final de la cola
0040:001e cola circular de 16 words.

Ejemplos: PC FC



0040:001a 001e
0040:001c 0020
0040:001e 411e se puede leer una letra 'A'
0040:0020 el proximo caracter se escribira aqui

411e es: 41 ASCII de 'A'
1e 'scan code' de 'A'



0040:001a 001e se leyo la letra 'A'
0040:001c 0020
0040:001e 411e la cola esta vacia cuando ambos
0040:0020 punteros apuntan al mismo lugar

Los 'scan codes' se leen del puerto 60h y cuando el bit mas alto es 1, se trata de la liberacion de una tecla y se puede ignorar. El puerto 64h tiene el estado del teclado y en el bit 2 (demonimado 'input buffer full') informa con 1 de la presencia de un caracter. El siguiente programa desactiva la operacion por interrupcion, lee caracteres y muestra su 'scan code' hasta que se presione Esc.

```
;
; Prueba de puertos del teclado
;
.model tiny
.code
org 100h

;constantes
status_port     equ 64h
data_port       equ 60h
in_buf_full     equ 02h

inicio:
        cli     ; inhabilitar interrupciones
ciclo:  in al, status_port
        test al , in_buf_full
        jnz ciclo

        in al,data_port
        test al,80h     ; liberacion de tecla ?
        jnz ciclo

        mov dl,al
        mov cl,4
        shr al,cl
        call p_hex
        mov al,dl
        and al,0fh
        call p_hex
        mov al,0dh
        call p_char
        mov al,0ah
        call p_char

        cmp dl,1        ; la tecla presionada es Esc ?
        jnz ciclo

        sti             ; re-habilitar interrupciones
        int 20h

p_hex:  cmp al,0ah      ; escribe AL en hexadecimal
        jb p0
        add al,'A'-0ah
        jmp p_char
        p0: or al,'0'

p_char: xor bh,bh       ; escribe
        mov ah,0eh
        int 10h
        ret
end inicio
```

El siguiente programa muestra la operacion dentro de la interrupcion 9.
Al final de cada lectura, el controlador de interrupciones debe ser reactivado con el comando 'eoi' (end of interruput) en su puerto de control (20h)

```
;
; Prueba de puertos del teclado
;

.model tiny
.code
org 100h

;constantes
status_port     equ 64h
data_port       equ 60h
in_buf_full     equ 02h
eoi             equ 20h
i8259_control_port equ 20h

inicio: mov al,9                ; recuperar direccion de interrupcion de teclado
        mov ah,35h
        int 21h

        mov tec_dos,bx          ; salvarla en tec_dos
        mov tec_dos+2,es

        mov dx, offset new_tec  ; fijar nueva interrupcion de teclado
        mov al,9
        mov ah,25h
        int 21h

        mov dx,offset mensaje   ; desplegar mensaje para terminar con Esc
        mov ah,9
        int 21h

ciclo:  cmp finish,1            ; espera que finish cambie a 1
        jnz ciclo               ; originalmente esta en 0

        mov dx,tec_dos          ; restituir direccion original de INT 9
        mov ds,tec_dos+2
        mov al,9
        mov ah,25h
        int 21h

        mov ax,cs               ; como DS se altero, hay que recuperarlo de CS
        mov ds,ax
        int 20h
tec_dos dw ?,?
finish db 0
```



```
mensaje db 'Interrupcion instalada. Salga con Esc',0dh,0ah,'$'

new_tec: in al, data_port       ; nueva direccion de interrupcion de teclado

        test al,80h             ; liberacion de tecla ?
        jnz fin_i

        mov dl,al               ; despliegue de tecla
        mov cl,4
        shr al,cl
        call p_hex
        mov al,dl
        and al,0fh
        call p_hex
        mov al,0dh
        call p_char
        mov al,0ah
        call p_char

        cmp dl,1                ; se presiono Esc ?
        jnz fin_i
        mov finish,1

fin_i:  mov al,eoi              ; reactivar el controlador de interrupciones
        out i8259_control_port,al
iret

p_hex:  cmp al,0ah
        jb p0
        add al,'A'-0ah
        jmp p_char
        p0: or al,'0'
p_char: xor bh,bh
        mov ah,0eh
        int 10h
        ret
end inicio
```

El siguiente programa complementa la interrupcion 9 del sistema operativo con la creacion de minusculas acentuadas y letras eñe en mayuscula y en minuscula.

```
.model tiny
.code
org 100h

inicio:
```

```
        mov ax,3509h            ; Se recupera en ES:BX el vector de la
        int 21h                 ; INT 9

        mov ax,2550h            ; Se fija la INT 50  al mismo lugar que la
        mov si,es               ; INT 9
        mov ds,si
        mov dx,bx
        int 21h

        cli                     ; No se aceptan interrupciones a partir de
                                ; este momento. En este momento se cambiara la
                                ; direccion en la que opera la INT 9.

        mov ax,2509h            ; Se desvia la INT 9 a la etiqueta int_9
        mov si,cs               ; de este programa
        mov ds,si
        mov dx,offset int_9
        int 21h

        mov ax,40h              ; Se recupera el FC.
        mov ds,ax               ; El ultimo FC se almacenara en s_fc
        mov ax,ds:word ptr [1ch]
        mov cs:word ptr [s_fc],ax

        sti                     ; Se vuelven a aceptar interrupciones
                                ; para que el teclado vuelva a funcionar.

        mov ds,si               ; Termina residente con ah=31.
                                ; DS debe apuntar al PSP
        mov dx,20h              ; Se reservan 200H bytes. DX tiene la cantidad
                                ; de paragrafos (20H). 100H para el PSP y
                                ; 100 para el programa residente.

        mov ah,31h
        int 21h


int_9:  mov word ptr cs:[s_ss], ss    ;Salvar SS y SP para habilitar una
        mov word ptr cs:[s_sp], sp    ;pila local

        mov sp,cs               ; La pila local se habilita en el mismo
        mov ss,sp               ; segmento de codigo CS.
        mov sp,200h             ; Se localiza al final del area reservada

        push ax                 ; Se salvan todos los registros de proposito
        push cx                 ; general, indices y segmentos extra y de
        push si                 ; datos.
        push di
        push es
        push ds

        int 50h                 ; Se ejecuta la INT 9 original

        mov ax,40h              ; Se recupera FC en SI
        mov ds,ax
        mov si,ds:word ptr [1ch]
        cmp si,cs:word ptr [s_fc] ;Actual FC con ultimo FC          **
        je fin_int_9            ; Terminar porque la cola no se ha afectado;
                                ; ha llegado una interrupcion por que se ha
                                ; soltado una tecla

        mov cs:word ptr [s_fc],si ; Se salva el actual FC para ver si se altera
                                ; o no en la proxima INT 9

        cmp si,ds:word ptr [1ah] ;Actual FC con actual PC           **
        je fin_int_9            ; Terminar porque la cola esta vacia    **

                                ; restar dos al FC pra apuntar al ultimo
                                ; caracter introducido; Como 20h=1eh+2
        sub si,20h              ; sub .. resta 1e y 2; asi el puntero queda
        and si, 001fh           ; como si empezara en 0; con and ...
                                ; queda como un número entre 0 y 31 por que
                                ; son 32 bytes en el buffer (16*2)
        add si,1eh              ; Se suma por que en realidad el buffer empieza
                                ; en 1eh y no en 0h.

        mov ah,byte ptr cs:sw_acento; el estado del acento se copia a AH

        mov al,byte ptr [si]    ; el caracter pulsado pasa a AL

        cmp al,39               ; es un acento ?
        jne otro                ; si no es, salta a la etiqueta "otro"

acento: or ah,ah                ; Se tiene un acento antes?
                                ; la variable sw_acento esta en 1 cuando
                                ; se ha presionado un acento

        jz no_acento            ; si no, salta a la etiqueta "no_acento"

        mov byte ptr cs:sw_acento,0     ; si ya hay un acento, se acepta; de esta
                                        ; manera se tiene que el acento como tal
                                        ; se produce con presionando acento dos veces
```

```
        jmp fin_int_9                   ; terminar la interrupcion

no_acento: mov byte ptr cs:sw_acento,1  ; se fija el acento
        mov ds:word ptr [1ch],si        ; el FC se fija en 2 posiciones antes
        mov cs:word ptr [s_fc],si       ; Se salva el actual FC para ver si se altera
                                        ; o no en la proxima INT 9
        jmp fin_int_9                   ; terminar la interrupcion

otro:   or ah,ah                        ; si no se ha acentuado, no hay nada que
        jz fin_int_9                    ; hacer y termina la interrupcion

        mov byte ptr cs:sw_acento,0     ; En otro caso, si hay acento, se retira
                                        ; el acento.

        mov cx,cs               ; exploracion de la tabla t_org
        mov es,cx               ; en el segmento cs
        mov di,offset t_org

        mov cx,7                ; 7 letras

        cld                     ; proceso hacia adelante

        repne scasb             ; mientras el caracter sea diferente de los de
                                ; la tabla

        jnz fin_int_9           ; si no es ninguna de las letras, terminar sin
                                ; cambiar la letra; de esta manera, con dos veces
                                ; acento se tiene un solo acento
        mov al,es:[di+6]        ; se cambia por la letra acentuada o alterada
        mov [si],al             ; (n o N)

fin_int_9:                      ; Se recuperan los valores anteriores
        pop ds                  ; con la misma logica con que se guardaron
        pop es                  ; pero en el sentido inverso
        pop di
        pop si
        pop cx
        pop ax
        mov ss, word ptr cs:[s_ss]
        mov sp, cs:[s_sp]
iret
t_org db 'aeiounN'      ; tabla de vocales minusculas, n y N
      db ' ,¡¢£¤¥'

s_fc dw ?               ; anterior FC; si no cambiam se trata de un interrupcion
                        ; por soltarse una tecla
```



```
s_ss dw ?               ; ss y sp anteriores
s_sp dw ?

sw_acento db 0          ; 0=no se acentuo 1=se acentuo previamente
fin_residente label near

end inicio
```

V.4.- Impresora

La interfaz mas comun para impresora es el puerto paralelo, aunque tambien puede usarse el puerto serial con el hardware apropiado. El puerto paralelo tiene mayor velocidad de transmision ya que en una sola operacion envia 8 bits mientras el puerto serial envia 1 bit por vez. De todas maneras, la baja velocidad propia de la impresora no hace muy ventajosa esta caracteristica. En el puerto paralelo se pueden ver claramente los tres puertos de control, datos y estado. Las direcciones mas usuales para LPT1 son:

ESTADO 379h
DATOS 378h
CONTROL 37Ah

Los bits mas altos del puerto de estado en 1 (0C0h) informan que la impresora ha impreso aun el ultimo caracter enviado. Si esta condicion no se cumple, se debe esperar a que lo haga. Una vez que estos bits estan ambos en 1, se puede
proceder a imprimir un caracter enviandolo por el puerto de datos.
Una se¤al complementaria debe subir y bajar el bit cero del puerto de control activando a la vez los bits 2 y 3.
El siguiente programa realiza una impresion de lineas con el alfabeto en orden inverso mientras no se presione una tecla.

```
.model tiny
.code
.386
INCLUDE BIOS.MAC
org 100h
out_p equ     378h;3bch ;378h
status_p equ  379h;3bdh ;379h
control_p equ 37ah;3beh ;37ah

inicio:

p0:     mov bl,'Z'
ciclo1: call car
        dec bl
```

```
        CMP BL,'A'
    jae ciclo1
    mov bl,0dh
    call car
        mov bl,0aH
        call car
        IF_NO_CAR p0
fin:  int 20h

car:    mov dx,status_p
        in al,dx
        and al,0c0h
        cmp al,0c0h
        JnZ car
    mov dx,out_p
    mov al,bl
    out dx,al
    mov dx,control_p
        mov al,1101b
    out dx,al
    mov al,1100b
    out dx,al
ret
end inicio
```

V.5.- Mouse

El mouse se conecta por el puerto serial. Este dispositivo realiza la transmision de datos bit por bit usando una sola linea para transmision, en oposicion al puerto paralelo, que envia o recibe un byte completo (8 bits) por ocho lineas.
Para transmision de datos por linea telefonica, el puerto serial es el unico dispositivo util ya que las redes telefonicas no tienen el juego de ocho lineas para la transmision paralela. Aun mas, requiere de un modem (modulador-demodulador) que convierte la se¤al del digital del computador a la se¤al analogica de la red telefonica.
Por lo comun que resulta el dispositivo de apuntamiento que es el mouse, se han incluido en los servicios del sistema operativo una serie de servicios relacionados con el. Ademas, el movimiento del mouse y la presion de teclas genera la interrupcion 0Fh.

La interrupcion asignada es la 33h. Algunos de sus servicios son los siguientes:

AX=0 Reinicia el raton e informa su estado. Retorna:
    AX=-1 : Raton instalado. En BX se tiene el número de botones
    otro valor : Raton no instalado.

AX = 1 Activa el cursor del mouse que esta en principio invisible.

AX = 2 Desactiva el cursor del mouse.

AX = 3 Informa de la posicion del mouse y los botones pulsados/liberados
    CX = coordenada x
    DX = coordenada y
    BX = bit 0 para boton izquierdo 0 = liberado
        bit 1 para boton derecho 1 = pulsado
        bit 2 para boton central

AX = 4 Fija posicion del cursor de mouse
    CX = nueva coordenada x
    DX = nueva coordenada y

El siguiente programa muestra el estado de los botones y la fila y columna del mouse.
Las unidades en que se expresana las coordenadas son los “mickeys”, 1/200 de pulgada.

```
.model tiny
.code
include bios.mac
include bdos.mac
org 100h
inicio:
mov ax,1
int 33h
ciclo:  mov ax,3
    int 33h

    mov ah,bh
    mov si,offset b_x
    call cnv

    mov ah,bl
    call cnv

    mov ah,ch
    mov si,offset c_x
    call cnv

    mov ah,cl
    call cnv

    mov ah,dh
    mov si,offset d_x
    call cnv

    mov ah,dl
```

```
        call cnv
        set_cur 24,0
        esc_c$ cad

        if_no_car ciclo
; Escribir aqui el programa

fin: int 20h

cnv:    mov al,ah
        shr ah,1
        shr ah,1
        shr ah,1
        shr ah,1
        call p1
        mov ah,al
        and ah,0fh
        call p1
ret
p1:     cmp ah,9
        ja p0
        add ah,'0'
        jmp p2
p0:     add ah,'A'-10
p2:     mov byte ptr [si],ah
        inc si
ret
; Escribir aqui los datos

cad  db 'BX = '
b_x db ?,?,?,?
  db ' CX = '
c_x db ?,?,?,?
  db ' DX = '
d_x db ?,?,?,?
  db '$'
end inicio
```


VI.- EL COPROCESADOR MATEMATICO

VI.1.- Tipos de datos del coprocesador

Poco tiempo después de la aparición del 8088, INTEL lanzo el "coprocesador matemático" 8087. Este es otro procesador orientado a operaciones con números de punto flotante y además muchas funciones matemáticas complejas como logaritmos, exponentes, funciones trigonométricas y otras.

La utilidad de este coprocesador se hace evidente en aplicaciones científicas y también en computación gráfica, donde las rotaciones y el dibujo de gráficas de funciones requieren de estas funciones matemáticas.

Los cálculos de este tipo, con la ayuda del coprocesador se logran hacer de 10 a 100 veces mas rápidos respecto lo mejor que puede hacerse con el procesador.
Este hecho radica en que las instrucciones del coprocesador están implementadas en la propia electrónica, a nivel de micro-codigo, de manera mas eficiente que el mejor programa para el procesador.
El coprocesador analiza la instrucción a ejecutar en CS:IP simultáneamente al procesador. Al aparecer una instrucción ESC, el procesador pone sus operandos en el bus y continua, en tanto el coprocesador toma los operandos, decodifica
su instrucción y la ejecuta.

Como no se llego a difundir lo suficiente, desde la aparición del 80286 se escribieron emuladores del coprocesador que se instalaban como programas residentes accesibles por la interrupción 7 y se incorporo un grupo de flags (MP,EM y TS,ET) destinados a controlar esta posibilidad.

Posteriormente, junto con los procesadores siguientes, aparecieron sus correspondientes co-procesadores; 80287, 80387SX, 80387DX, 80487SX y finalmente, desde el 80486DX se incorporo el coprocesador dentro del mismo procesador.
En el proyecto del 80486DX sobro espacio y se pudo colocar, además del 487, el controlador de 'memoria cache' que tiene copias de los lugares mas accedidos de la memoria dentro del mismo procesador.
Se lo denomina "coprocesador inter-construido".

Los tipos de datos que maneja son los siguientes:

Enteros : 2,4 y 8 bytes de binario puro con signo. Se pueden definir con DW, DD y DQ.

Binario con signo en el bit mas alto

BCD empaquetado: 10 bytes, 18 dígitos en nueve bytes y un byte de signo donde solo el bit mas alto indica el signo. Se define con DT.

Byte de signo 00/80h	BCD sin signo

Reales; corto : 4 bytes, 1 bit de signo, 8 bits de característica (sumada a 7Fh) y 23 bits de mantisa con bit escondido.
Se define con DD.
largo : 8 bytes, 1 bit de signo, 11 bits de característica (sumada a 3FFh) y 52 bits de mantisa con bit escondido.
Se define con DQ.
temporal: 10 bytes, 1 bit de signo, 15 bits de característica
(sumada a 3FFFh) y 64 bits de mantisa SIN bit escondido.
Se define con DT.

Signo	Mantisa	Característica

Estos convenios han sido establecido por el IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) y han sido adoptados por INTEL. En oposición, Microsoft tenía otro convenio que no es soportado por el coprocesador.

Como se dijo en el capitulo I, existen algunas interpretaciones especiales:

Característica	Mantisa	Signo	Interpretación
0....0	0.....0	0	+ 0
0....0	0.....0	1	- 0
0....0	<>0		número desnormalizado con la mantisa mas baja; no se supone el bit escondido.
1....1	0.....0	0	+ infinito
1....1	0.....0	1	- infinito
1....1	10....0	1	indefinido, generado por operaciones inusuales como 0/0. Es NAN (no número)
1....1	otro valor		NAN de interpretación libre

Con otros valores en la característica, se trata de un número normalizado.

VI.2.- Registros del coprocesador

Existe una pila de 8 registros de 10 bytes cada uno. Todos los tipos de datos ingresados al coprocesador son convertidos al formato real temporal que tiene exactamente ese tamaño.
Como pila, el ultimo en ingresar es el primero en ser utilizado.
Este diseño corresponde el tratamiento clásico de expresiones matemáticas como lenguajes libres de contexto donde los operandos son empilados y las operaciones toman a los últimos empilados como operandos. Existe un puntero que apunta al ultimo empilado que recibe la denominación ST (= stack top) o ST(0) (=stack top 0). Los registros anteriormente empilados reciben la denominación ST(1),ST(2),ST(3),ST(4),ST(5),ST(6) y ST(7) según el orden en que ingresaron. El penúltimo es ST(1), el anterior ST(2) y así sucesivamente.
Internamente tienen números invariables y ST es solo el registro apuntado por un puntero.

79 bits 0

ST(0) o ST
ST(1)
ST(2)
ST(3)
ST(4)
ST(5)
ST(6)
ST(7)

Por ejemplo, la suma 2+3 se realiza de la siguiente forma:

1.- Empilar 2

ST = 2

2.- Empilar 3

ST (1) = 3
ST = 2

3.- Dejar la suma en el tope de la pila

ST = 5

Se utiliza la nomenclatura "interrupción" para eventos externos que afectan a un procesador y "excepción" para eventos excepcionales dentro de la ejecución de un procesador. Cuando el coprocesador encuentra situaciones anormales como divisiones por 0, raíces cuadradas de negativos genera "excepciones" que pueden derivar en la interrupción 2, NMI (NMI = Non-maskable interrupt, interrupción no enmascarable) y además se empila un operando indefinido.

Existen 6 excepciones:
- Precisión (PE)
- Underflow (UE)
- Overflow (OE)
- División por 0 (ZE)

Desnormalización (DE)
- Operación invalida (IE)

Sus estados y otros se reflejan en un entorno (environment) de 7 words con los siguientes bits y significado:

Palabra de Control CW

			CI	CR	CP	IEM		PM	UM	OM	ZM	DM	IM

Palabra de Estado SW

B	C3	ST	C2	C1	C0	ES		PE	UE	OE	ZE	DE	IE

Tags

TAG 7	TAG 6	TAG 5	TAG 4	TAG 3	TAG 2	TAG 1	TAG 0

En modo real:

Direccion de la instruccción

16 bits bajos		
4 bits altos	0	Código de operación

Direccion del operando

16 bits bajos	
4 bits altos	

En modo protegido:

Direccion de la instruccción

Offset
Selector

Direccion del operando

Offset
Selector

B = Busy : 1 cuando el procesador esta ejecutando una instrucción.

ST = Stack top: Indica el número de registro que es actualmente ST.

C3,C2,C1,C0 = Condiciones: Reflejan los resultados de las instrucciones FTST, FUCOM,FCOM,FPREM y FXAM.

Excepciones : Con 1 indican que ha existido una excepción de:

PE = precisión: Resultados u operandos exceden la precisión seleccionada en CP.
UE = underflow: El resultado es muy pequeño como para representarse.
OE = overflow : El resultado es muy grande como para representarse.
ZE = división por cero: El divisor es 0 y el dividendo no es cero ni infinito.
DE = desnormalización: Al menos un operando esta desnormalizado.
IE = operación invalida: Intento de empilar mas de 8 elementos en la pila o desempilar cuando no existen elementos u otros como raíces cuadradas de negativos, etc.

Mascaras de excepciones: Con 1 indican que la excepción esta enmascarada y no tiene ningun efecto. Con 0, en caso de presentarse, se genera INT 2.

PM,UM,OM,ZM,DM,IM corresponden a PE,UE,OE,ZE,DE,IE , respectivamente.

IR = Interrupt Request; se activa cuando se presenta una excepción y no esta enmascarada. Puede servir en una rutina de la INT 2 para detectar si se ha invocado por el coprocesador u otro evento.

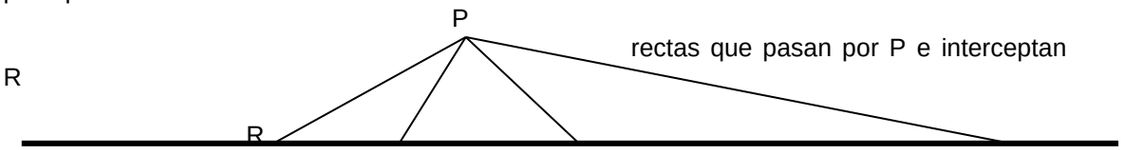
IEM = Mascara de habilitación de interrupciones: Con 0, habilita interrupciones no enmascaradas y con 1 las inhabilita. Desde el 387 no puede ser 1.

CI = Control de infinito;
- 0 Infinito sin signo (proyectivo).
- 1 Infinito con signo (afín).

La geometría afín es la que normalmente manejamos, por ello se llama afín.

En ella dos rectas paralelas como las vías del tren no se cortan. La geometría proyectiva es la que observamos en fotografías; aquí, una fotografía de dos vías del tren muestra que ambas convergen a un punto.

Dada una recta R horizontal y un punto P fuera de ella, punto de R define una recta que pasa por P.



En la geometría proyectiva, el punto "infinito" corresponde a la recta paralela a R que pasa por P. Existen dos formas de llegar al "infinito" ;

una avanzando por R a la derecha de P hasta alcanzar la paralela y otra avanzando a la izquierda, con lo que se alcanza también la paralela; de esta manera la recta se comporta como si fuera cerrada y el infinito no tiene signo.

Se podría esperar que tan (90) y tan (-90) den el mismo valor si se ha elegido fijado IC=0, pero no ocurre así.

El uso de este bit no parece estar documentado ya que inclusive publicaciones en Internet indican no conocer su uso.

CR = Control de redondeo

0 Redondear al mas próximo o al par. (2.5 se redondea a 2 y 3.5 a 4)
1 Redondear hacia el lado de - infinito.
2 Redondear hacia el lado de + infinito.
3 Redondear hacia el lado donde esta 0.

CP = Control de precisión

0 24 bits; corresponde al real corto
1 Reservado
2 53 bits; corresponde al real largo
3 64 bits; corresponde al real temporal

TAG0 a TAG7 son los estados de los registros de la pila. Sus posibles valores son:

0 = Contiene un número valido diferente de 0.
1 = Contiene cero.
2 = Contiene un No número o indefinido o infinito.
3 = No contiene, se retiro su valor.

Corresponden a las posiciones fijas de los registros y no a las relativas al tope del pila como ST(i).

Cada vez que el procesador encuentra una instrucción ejecutable, registra su dirección, su código de operación y la dirección de su operando.

VI.3.- Instrucciones del coprocesador

Todas las instrucciones empiezan con F (floating). La instrucción FINIT es equivalente a un activar la línea RESET del procesador y tiene el siguiente efecto en las palabras de estado y control:

B = 0
ST = 0
C3,C2,C1,C0 = ?,?,?,?
PE,UE,OE,ZE,DE,IE = 0,0,0,0,0,0 (sin excepciones).
PM,UM,OM,ZM,DM,IM = 1,1,1,1,1,1 (excepciones inhabilitadas).
IEM = 0 (no se requiere interrupción).
CI = 0 (infinito proyectivo).
CR = 0 (redondear al mas próximo o al par).
CP = 3 (64 bits).
TAG0 a TAG7 = 11 (vacíos).

FWAIT (o WAIT que es su equivalente) aconsejablemente debe ser usada para asegurar la sincronización entre el coprocesador y el procesador escribiéndola antes de una instrucción del procesador que requiera que el coprocesador haya terminado su tarea. Solo las instrucciones FSAVE y FSTENV requieren WAIT y el ensamblador siempre lo coloca antes de ellas.

Algunas instrucciones tienen una forma alterna que incluye N como segunda letra con lo que no se genera nunca una instrucción WAIT previa; por ejemplo FNINIT no genera un previo WAIT mientras que FINIT si lo hace.

Vale el siguiente esquema general:

Instrucción	Destino	Fuente
Foper	ST(1)	ST Si no se indican operandos, destino es ST(1) y origen ST(0)
Foper ST(i),ST(0)	ST(i)	ST
Foper ST(0),ST(i)	ST	ST(i)
Foper mem	ST	mem (real)
FIoper mem	ST	mem (entero)

La letra "I" indica "integer" entero.

Instrucciones de carga al coprocesador

Para cargar un elemento a la pila, se asume ST como destino:

FILD <mem>
Dependiendo del atributo del operando <mem>; word, dword o qword, asume una cantidad de bytes como entero con signo y lo empila convertido a real temporal.

FLD <mem> Dependiendo del atributo del operando <mem>; dword, qword o ten /ST(i) byte, asume una cantidad de bytes como real y lo empila convertido a real temporal.

FBLD <mem> El atributo de <mem> debe ser tbyte y se asumen los 10 bytes como BCD empaquetado con signo y se empilan.

FLD1 Empila 1.

FLDZ Empila 0.

FLDPI Empila PI.

FLDL2E Empila $\log_2 e$.
FLDL2T Empila $\log_2 10$.
FLDLG2 Empila $\log_{10} 2$.
FLDLN2 Empila $\log_e 2$.
FXCH Intercambia ST(0) con ST(1).

FXCH ST(x) Intercambia ST(0) con ST(x).

Instrucciones de recuperación del coprocesador

El operando fuente es ST. La letra "P" al final indica desempilar adicionalmente a recuperar. En algunas instrucciones se asume.

FST <mem>/ST(i) Lleva ST a memoria o a un registro de la pila.
FSTP <mem> Lleva ST a memoria y desempila. Se asumen números reales y el tamaño depende del atributo dword,qword o tbyte (solo FSTP) de <mem>. FST se traduce como FST ST(1).

FIST <mem> Lleva ST a memoria.
FISTP <mem> Lleva ST a memoria y desempila. Se asumen números enteros y el tamaño depende del atributo word,dword, qword (solo FISTP) de <mem>.

FBSTP <mem> Lleva ST a memoria y desempila. <mem> debe tener atributo tbyte. ST es previamente redondeado a entero según CR.

Sumas

FADD <mem>	Suma al tope de la pila un real de 4 u 8 bytes.
FADD/FADDP	Suma ST a ST(1) y desempila.
FADD ST(i),ST	Suma ST a ST(i).
FADDP ST(i),ST	Suma ST a ST(i) y desempila.
FADD ST,ST(i)	Suma ST(i) a ST.
FIADD <mem>	Suma al tope de la pila un entero de 2 o 4 bytes.

Restas

FSUB <mem>	Resta al tope de la pila un real de 4 u 8 bytes.
FSUB/FSUBP	Resta ST a ST(1) y desempila.
FSUB ST(i),ST	Resta ST a ST(i).
FSUBP ST(i),ST	Resta ST a ST(i) y desempila.
FSUB ST,ST(i)	Resta ST(i) a ST.
FISUB <mem>	Resta al tope de la pila un entero de 2 o 4 bytes.

La letra R al final indica que la operación en "reversa":
destino = fuente-destino.

FSUBR <mem>	Resta el tope de la pila de un real de 4 u 8 bytes.
FSUBR/FSUBRP	Resta ST(1) a ST y desempila.
FSUBR ST(i),ST	Resta ST(i) a ST.
FSUBRP ST(i),ST	Resta ST(i) a ST y desempila.
FSUBR ST,ST(i)	Resta ST a ST(i).
FISUBR <mem>	Resta el tope de la pila de un entero de 2 o 4 bytes.

Por ejemplo, el calculo 2+3-1 se puede realizar de las siguientes formas:

1) Cargando los operandos a la pila:

```
fild n_2
fild n_3
faddp
fild n_1
fsubp
fist n_r
fwait
```

2) Operando directamente con el tope de la pila:

```
fild n_2
fiadd n_3
fisub n_1
fist n_r
fwait
```

3) Usando la carga del operando 1:

```
fild n_2
fiadd n_3
fld1
fsubp
fist n_r
fwait
```

Se supone: n_2 dw 2
n_3 dw 3
n_1 dw 1
n_r dw ? ; aquí queda del resultado

Multiplicaciones

FMUL <mem> Multiplica el tope de la pila por un real de 4 u 8 bytes.

FMUL/FMULP Multiplica ST por ST(1) y desempila.

FMUL ST(i),ST Multiplica ST por ST(i).

FMULP ST(i),ST Multiplica ST por ST(i) y desempila.

FMUL ST,ST(i) Multiplica ST(i) por ST.

FIMUL <mem> Multiplica al tope de la pila un entero de 2 o 4 bytes.

Divisiones

FDIV <mem> Divide el tope de la pila por un real de 4 u 8 bytes.

FDIV/FDIVP Divide ST(1) por ST y desempila.

FDIV ST(i),ST Divide ST(i) por ST.

FDIVP ST(i),ST Divide ST(i) por ST y desempila.

FDIV ST,ST(i) Divide ST por ST(i).

FIDIV <mem> Divide el tope de la pila por un entero de 2 o 4 bytes.

La letra R al final indica que la operación en "reversa":
destino = fuente/destino

FDIVR <mem> Divide un real de 4 u 8 bytes por el tope de la pila.

FDIVR/FDIVRP Divide ST por ST(1) y desempila.

FDIVR ST(i),ST Divide ST por ST(i).

FDIVRP ST(i),ST Divide ST por ST(i) y desempila.

FDIVR ST,ST(i) Divide ST(i) por ST.

FIDIVR <mem> Divide un entero de 2 o 4 bytes por el tope de la pila.

Comparaciones

Comparan ST con un operando. Resta ST del operando.

FCOM/FCOM ST(1) Compara el tope de la pila con ST(1).

FCOM ST(i) Compara el tope de la pila con ST(i).

FCOM <mem> Compara el tope de la pila con un real de 4 u 8 bytes.

FCOMP Compara el tope de la pila con ST(1) y desempila.
FCOMP ST(1)

FCOMP ST(i) Compara el tope de la pila con ST(i) y desempila.

FCOMP <mem> Compara el tope de la pila con un real de 4 u 8 bytes y desempila.

FCOMPP Compara el tope de la pila con ST(1) y desempila dos veces.
FCOMPP ST(1)

FUCOM , FUCOMP , FUCOMPP (Unordered Comparations) son similares a FCOM , FCOMP, FCOMPP con la diferencia que no generan excepción de operación invalida cuando uno de los operandos no es número (NAN).

FICOM <mem> Compara el tope de la pila con un entero de 2 o 4 bytes.

FICOMP <mem> Compara el tope de la pila con un entero y desempila.

FTST Compara el tope de la pila con +0.

Los resultados de la comparación quedan en C0, C2 y C3 :

	C3	C0
ST > operando	0	0
ST < operando	0	1
ST = operando	1	0
ST no es comparable con operando	C2=1 (alguno no es número)	

La palabra de estado puede ser llevada a AX con FSTSW AX y como las posiciones de los bits en AH coinciden con el registro de banderas; C3 con Z, C2 con P y C0 con C; con SAHF se pueden fijar condiciones que pueden ser usadas por las bifurcaciones de números sin signo; JB, JBE, JA, JAE, JE y JNE.
Solo se debe tener cuidado que sean comparables con C2=0.

FXAM Actualiza las banderas C0,C1,C2 y C3 analizando el operando del tope de la pila:

C3	C2	C1	C0	
0	0	0	0	+ no normalizado
0	0	0	1	+ no número
0	0	1	0	- no normalizado
0	0	1	1	- no número
0	1	0	0	+ normalizado
0	1	0	1	+ infinito
0	1	1	0	- normalizado
0	1	1	1	- infinito
1	0	0	0	+ 0
1	0	0	1	vacío
1	0	1	0	- 0
1	0	1	1	vacío
1	1	0	0	+ desnormalizado
1	1	0	1	vacio
1	1	1	0	- desnormalizado
1	1	1	1	vacio

No se debe olvidar que el tope de la pila esta en formato real temporal de 10 bytes sin bit escondido. El número desnormalizado es aquel que tiene la característica en 0 y la mantisa diferente en 0. El número no normalizado es aquel que tiene el bit mas significativo de la mantisa en 0.

Otras operaciones aritméticas

FSQRT Calcula la raíz del tope de la pila en el mismo tope.

FSCALE

Suma la parte entera de ST(1) al exponente del tope de la pila: X = X *2^[Y] ; X es el tope de la pila e Y es ST(1).
El resultado final debe estar en el rango 2^-15 y 2^15. Para números positivos, la parte entera es el entero menor o igual y para números negativos es el entero mayor o igual. Es el redondeo hacia 0.

FPREM

Deja el residuo de ST / ST(1) en el tope de la pila por restas sucesivas. El resultado tiene el mismo signo que ST.

FPREM1

Calcula ST-n*ST(1) en el tope de la pila. Donde n es el entero mas próximo al valor exacto ST/ST(1). Esta es la definición IEEE de FPREM.

FRNDINT Redondea el tope de la pila a un entero según los bits CR.

FABS Extrae el valor absoluto del tope de la pila en el mismo tope.

FCHS Cambia del signo al tope de la pila en el mismo tope.

FXTRACT Descompone el tope de la pila y deja la mantisa en ST y la característica en ST(1).

Funciones trascendentes

FPTAN

Calcula la tangente del tope de la pila como una razón X/Y donde X queda en ST e Y en ST(1). Alguna bibliografía indica que tope de la pila debe estar en el rango 0 a pi/4 pero fuera de ese rango la instrucción sigue brindando resultados correctos.

FPATAN

Calcula el arco tangente de X/Y en el tope de la pila. X debe estar en ST e Y en ST(1). Alguna bibliografía indica que debe darse 0 < X < Y pero fuera de ese rango la instrucción sigue brindando resultados correctos.

FSIN Calcula sen(ST) en ST. ST puede ser cualquier número.

FCOS Calcula cos(ST) en ST. ST puede ser cualquier número.

FSINCOS Calcula cos(ST) en ST y sen(ST) en ST(1). ST puede ser cualquier número.

F2XM1

Calcula (2 ^ ST) -1 en el tope de la pila. Alguna bibliografía indica que debe darse 0 < ST <=0.5 pero en el rango -1 <=ST<= 1 la instrucción sigue brindando resultados correctos. El dominio resulta ser -1/2 <= ST <= 1, que coincide con el rango de FYL2XP1.

FYL2X
Calcula Y * LOG2(X) donde X esta en ST e Y en ST(1).X debe ser positivo. Si X=0 da el resultado infinito negativo.

FYL2XP1

Calcula Y * LOG2(X+1) donde X esta en ST e Y en ST(1). Alguna bibliografía indica que debe darse 0 <³ST³ <=1-û2/2 pero en el rango -1/2 <ST< 1 la instrucción brinda resultados correctos.
Si X=-1 da como resultado un no número negativo. Toma menos ciclos de ejecución que FYL2X en todos los co-procesadores excepto en el 387, en el que toma mas ciclos.

Instrucciones de control

FSETPM Cambia a direccionamiento a modo protegido.

FLDCW <mem> Carga la palabra de control desde <mem>, asumida word.

FSTCW <mem> Lleva la palabra de control a <mem>, asumida word.
/FNSTCW
FSTSW AX Lleva la palabra de estado a AX.
/FNSTSW AX

FCLEX/FNCLEX Borra las banderas de excepciones y el estado de ocupado.

FLDENV<mem> Carga las palabras de entorno en el orden descrito desde <mem>.

FSTENV <mem> Lleva las palabras de entorno en el orden presentado a <mem>.

FRSTOR <mem> Carga las palabras de entorno y los registros de la pila (en ese orden) desde <mem>.

FSAVE <mem> Lleva las palabras de entorno y los registros de la pila (en ese orden) a <mem>.
/FNSAVE

FINCSTP Incrementa el puntero de pila.

FDECSTP Decrementa el puntero de pila.

FFREE ST(x) Cambia a vacío el estado TAG del registro ST(x).

FNOP Idéntica al NOP del procesador.

EJERCICIOS RESUELTOS

1.- Calcular el área de un circulo de radio 1.5

```
FINIT
FLD RADIO
FMUL ST(0),ST(0)
FLDPI
FMUL
FWAIT

RADIO  DD 1.5
```

2.- Calcular las raíces de la ecuación de 2o. grado x^2 - 16x + 39 = 0

```
FILD DOS
FMUL A
FILD CUATRO
FMUL A
FMUL C
FLD B
FMUL B
FSUBR
FSQRT
FLD B
FCHS
FLD ST(0)
FSUB ST,ST(2)
FDIV ST,ST(3)
FSTP R1
FADD
```

```
        FDIVR
        FSTP R2
        FWAIT

DOS     DW 2
CUATRO  DW 4
A       DD 1.0
B       DD -16.0
C       DD +39.0
R1      DD ?
R2      DD ?
```

3.- Calcular sen(x) ≅ x - x ^ 3 / 3 ! +x ^ 5 / 5!-x ^ 7 / 7!. x es un entero expresado en grados

```
        FLDPI           ; convertir a radianes
        FIMUL X
        FIDIV N180
        FLD ST(0)       ; cargar otra vez
        FMUL ST(0),ST(0) ; calcular x^2
        FLD ST(1)       ; empilar otra vez x. La pila será :
                        ; ST(0) = termino que se calcula en base al anterior
                        ; ST(1) = x^2
                        ; ST(2) = sen(x)
        MOV CX,3        ; repetir 3 veces
P0:     FMUL ST(0),ST(1) ; se nota que :            x^3/3! = x* x^ 2 / 2 / 3
        INC I           ;                           x^5/5! = (x^ 3 / 3!) * x ^2 / 4 / 5
        FIDIV WORD PTR I ;                          x^7/7! = (x^ 5 / 5!) * x ^2 / 6 / 7
        INC I
        FIDIV WORD PTR I
        FCHS            ; los signos de los términos se alternan
        FADD ST(2),ST(0)
        LOOP P0
        FFREE ST(0)     ; se liberan dos elementos de la pila
        FFREE ST(1)     ; por que el resultado esta en ST(2)
        FINCSTP
        FINCSTP
        FWAIT

X       DW 30
N180    DW 180
I       DW 1
```

4.- Generar una interrupción 2 por division 1/0. Usar formato .EXE

```
; INT2.EXE
; Muestra que la division por cero genera la interrupción 2.
; Muestra también que IEM no puede ser cambiado y es un bit de solo lectura
;
.MODEL SMALL

.DATA
MINT2 DB "EJECUTO INT 2$"
MNORM DB "TERMINO NORMALMENTE$"
C_W DW

.STACK 10

.CODE
.387
CW RECORD DM1:3,CI:1,CR:2,CP:2,IEM:1,DM2:1,PM:1,UM:1,OM:1,ZM:1,DM:1,IM:1
            ; CW es un registro con la estructura de la palabra de
            ; control del coprocesador
INICIO: MOV AX,CS           ; se fija en DS:DX la dirección de la
MOV DS,AX                   ; etiqueta INT_2 y se asigna a ella
MOV AX,2502H                ; la interrupción
MOV DX,OFFSET INT_2
INT 21H

MOV AX,_DATA                ; se fija en DS el segmento de datos
MOV DS,AX
FINIT                       ; inicializar el coprocesador
FSTCW C_W                   ; leer la palabra de control
AND C_W, NOT MASK ZM        ; fijar en 0 ZM
OR C_W,MASK IEM             ; fijar en 1 IEM para inhabilitar
FLDCW C_W                   ; interrupciones. Resulta no tener efecto

FLD1                        ; cargar 1
FLDZ                        ; cargar 0
MOV DX,OFFSET MNORM         ; mensaje de terminación normal
FDIV                        ; dividir 1 / 0
FWAIT                       ; sincronizar. EN ESTE PROGRAMA RESULTA
                            ; INDISPENSABLE

FIN: MOV AH,9               ; desplegar y terminar
INT 21H
MOV AH,4CH
INT 21H

INT_2: MOV DX,OFFSET MINT2  ; mensaje EJECUTO INT 2
IRET

END INICIO
```

5.- Calcular la potencia 10^n usando el hecho $10^n = 2^{n*\log 10}$

```
.MODEL TINY
.CODE
ORG 100H
CW RECORD DM1:3,CI:1,CR:2,CP:2,IEM:1,DM2:1,PM:1,UM:1,OM:1,ZM:1,DM:1,IM:1
            ; CW es un registro con la estructura de la palabra de control
INICIO: FLDL2T
FIMUL WORD PTR N              ; el tope de la pila tiene n*log 10
FLD ST(0)                                          2
FLD ST(0)                   ; se almacena dos veces
FSTCW C_W                       ; leer la palabra de control
OR C_W,MASK CR                  ; fijar redondeo hacia 0
FLDCW C_W
FRNDINT                      ; se obtiene la parte entera
FSUB                        ; queda la parte fraccionaria en el tope
F2XM1                        ; 2 se eleva a esa fracción
FLD1                        ; es necesario sumar 1
FADD
FSCALE                       ; se suma la parte entera a la potencia
FBSTP DECIMAL
FWAIT

C_W DW
N DW 10
DECIMAL DT
END INICIO
```

6.- Desarrollar una rutina que edite con cursores a derecha e izquierda un número con cuatro dígitos decimales almacenado multiplicado por 10000 en formato real largo.

```
.model tiny
.code
.386
.387
include bios.mac
public lee_num,esc_num

r struc                 ; estructura en pila
      dw ?              ; BP empilado
  area_bcd dt           ; area de conversión a BCD empaquetado
  c_loc db              ; columna en pantalla durante edición
  i     db              ; posicion relativa en el area de edición
  p_area dw             ; puntero sobre el area en edición
  signo db              ; signo
  p_ent db 14 dup(?)    ; parte entera en ASCII
  punto db              ; punto decimal
  p_dec db 4 dup(?)     ; parte decimal en ASCII
  dir_ret dw ?          ; dirección de retorno
  di_emp dw ?           ; dirección de número
  fila  db              ; fila
  col   db              ; columna
  atr   db ?,?          ; atributo
ends

lee_num: sub sp,dir_ret-area_bcd
       push bp
       mov bp,sp
       call e_to_a     ; convierte el número a ASCII
       call desp       ; lo despliega
       mov al,[bp.col] ; apunta a la segunda columna
       inc al
       mov [bp.c_loc],al
       mov [bp.i],1    ; corresponde también a la segunda columna
       lea ax,[bp.p_ent] ; apunta a la segunda column, la parte entera
       mov [bp.p_area],ax

ciclo:  SET_CUR [bp.fila],[bp.c_loc]    ; se posiciona en la fila y la columna
       LEE_CAR         ; lee un caracter
       or al,al        ; es un caracter especial ?
       jnz letr_nums_? ; si no es, salta a verificar que caracter es
       cmp ah,_lef     ; es cursor a izquierda ?
       jnz right_?
       cmp [bp.i],1    ; si es cursor izquierda, no puede retroceder si esta
       jz ciclo        ; en la segunda columna. La primera se deja para el signo
       call retro      ; Retrocede
       mov si,[bp.p_area]      ; si es punto decimal, retrocede una vez mas.
       cmp byte ptr [si],'.'
       jnz ciclo
       call retro
       jmp ciclo
right_?: cmp ah,_rig
       jnz del_?
cur_Der: cmp [bp.i],19  ; si es cursor derecha, no puede avanzar mas si esta en
       jz ciclo        ; la ultima columna
       call avanc      ; avanza
       mov si,[bp.p_area]
       cmp byte ptr [si],'.'   ; si llega al punto decimal, sigue avanzando
       jnz ciclo
       call avanc
```

```
        jmp ciclo
del_?:  cmp ah,_del
        jnz ciclo       ; borrado de un caracter
        cmp [bp.i],15
        ja a_der
        mov di,[bp.p_area]      ; si esta antes del punto decimal, debe
        lea si,[di-1]           ; correr los anteriores caracteres de la
        mov cl,[bp.i]           ; izquierda sobre el e insertar un espacio
        xor ch,ch               ; a la izquierda.
        dec cl
        std
        rep movsb
        mov byte ptr [di],' '
        jmp fin_co
a_der:  mov di,[bp.p_area] ; si el cursor esta mas alla del punto decimal, debe
        lea si,[di+1]      ; correr los siguientes caracteres de la derecha
        mov cl,[bp.i]      ; sobre el y insertar un 0 al final
        sub cl,15
        neg cl
        add cl,4
        xor ch,ch
        cld
        rep movsb
        mov byte ptr [di],'0'
fin_co: call desp
        jmp ciclo

letr_nums_?: cmp al,_cr
        jnz sign_?
        lea si,[bp.punto-1]     ; si es Enter,
        mov di,si
        mov cx,14
        mov dx,14
        std
ver_forma:lodsb
        cmp al,' '
        jz p0
        stosb
        dec dx
p0:     loop ver_forma
        mov cx,dx
        mov al,' '
        rep stosb
        call desp
        jmp fin
```



```
sign_?: cmp al,'-'      ; si es signo, afecta la primera columna
        jne nums_?
        cmp [bp.signo],'-'
        je negativo
        mov al,'-'
        jmp esc_sign
negativo: mov al,' '
esc_sign: mov [bp.signo],al
        SET_CUR [bp.fila],[bp.col]
        ESC_CAR ,[bp.atr],
        jmp ciclo

nums_?: cmp al,'0'      ; verifica que sea número
        jb ciclo
        cmp al,'9'
        ja ciclo
        mov si,[bp.p_area]
        mov [si],al
        ESC_CAR ,[bp.atr],      ; lo escribe y avanza a la derecha
        jmp cur_der

fin:    call a_to_e             ; convierte de ASCII a real
fin0:   pop bp
        add sp,dir_ret-area_bcd
        ret 6                   ; descarta parametros empilados

esc_num: sub sp,dir_ret-area_bcd        ; rutina de solo escritura
        push bp
        mov bp,sp
        call e_to_a
        call desp
        jmp fin0

retro:  dec [bp.i]
        dec [bp.p_area]
        dec [bp.c_loc]
ret
avanc:  inc [bp.i]
        inc [bp.p_area]
        inc [bp.c_loc]
ret

e_to_a: mov si,[bp.di_emp]      ; convierte el número real a ASCII
        finit                   ; primero se convierte a BCD empaquetado
        fld qword ptr [si]
        fbstp tbyte ptr [bp.area_bcd]
```

```
        lea si,[bp.area_bcd]
        cld
        lea di,[bp.signo+19]    ; desempaqueta los dígitos menos significativos
        call des_emp
        call des_emp
        mov al,'.'              ; coloca un punto
        stosb
        mov cx,7                ; desempaqueta los restantes
p_des:  call des_emp
        loop p_des
        lodsb
        or al,al                ; coloca signo o espacio en blanco
        jz sig_pos
        mov al,'-'
        jmp fin_ea
sig_pos: mov al,' '
fin_ea: stosb
        lea di,[bp.p_ent]
        mov cx,13
        mov al,'0'
        cld
p1:     scasb
        jnz fin_ea1
        mov byte ptr [di-1],' '
        loop p1
fin_ea1: ret

des_emp: cld                    ; convierte de BCD empaquetado a ASCII
        lodsb
        mov ah,al
        and al,0fh
        or al,30h
        std
        stosb
        mov al,ah
        and al,0f0h
        shr al,4
        or al,30h
        stosb
ret

a_to_e: lea si,[bp.signo+19]    ; convierte de ASCII a real
        cld
        lea di,[bp.area_bcd]
        call emp
        call emp
        dec si
        mov cx,14
p_emp:  call emp                ; elimina espacios en blanco intermedios
        loop p_emp
        lodsb
        cmp al,' '
        jz sig_pos1
        mov al,0ffh
        jmp fin_ae
sig_pos1: xor al,al
fin_ae: stosb
        finit
        fbld tbyte ptr [bp.area_bcd]
        mov si,[bp.di_emp]
        fstp qword ptr [si]
ret

emp: std                        ; empaqueta
    lodsb
    mov ah,al
    and ah,0fh
    lodsb
    shl al,4
    or al,ah
    cld
    stosb
ret

desp:   lea si,[bp.signo]       ; depliega cadena desempaquetada
        mov cx,20
        mov bl,[bp.col]
        cld
c_desp: set_cur [bp.fila],bl
        lodsb
        push cx
        push bx
        esc_car  ,[bp.atr],
        pop bx
        pop cx
        inc bl
        loop c_desp
ret
end
```

7.- Usar la rutina del ejercicio 6

```
extrn lee_num:near
org 100h
inicio: mov ax,70h      ; empila atributo
        push ax
        mov ax,1010h    ; empila fila y columna
        push ax
        mov ax,offset número    ; empila dirección de la cadena
        push ax
        call lee_num    ; lee el número
        int 20h
número dq 13231000000.0
end inicio
```

8.- Escribir una macro que llame a la rutina del ejercicio 6

En el archivo ARIT.MAC

```
LEE_N   MACRO atr, fila, col, n
        mov ax,atr
        push ax
        mov ah, fila
        mov al,col
        push ax
        mov ax,offset n
        push ax
        extrn lee_num:near
        call lee_num    ; lee el número
ENDM
```

9.- Usar la macro del ejercicio 8

```
.model tiny
.code
include ARIT.MAC
org 100h
inicio: LEE_N 7,10h,10h,número
        int 20h
número dq 13231000000.0
end inicio
```


Ejercicios (Grupo 16)

1.- Calcular el area ($4\pi R^2$) y volumen ($4/3\pi R^3$) de una esfera de radio 2.25

2.- Calcular e^x según la aproximación $x+x/2! + x/3! + x/4! + x/5! + x/6!$

Escribir macros que calculen:

3.- e^x aprovechando que es $2^{x * \lg e}$

4.- x^y aprovechando que es $2^{y * \lg x}$

5.- $\lg x$, $\log x$, $\ln x$.

6.- $\lg_x y$.

7.- tan (x), arcsen (x), arccos (x) y arctan (x)

Para la conversión del formato de punto flotante a ASCII y viceversa se usara la capacidad del procesador para almacenar y depositar números en BCD empaquetado (FBLD y FBSTP). Se usara el siguiente formato intermedio para el
número $M * 10^C$.

Mantisa (M) : BCD empaquetado en 9 bytes y un byte de signo
Característica (C) : word con signo.

8.- Dado un número en el formato intermedio localizado en el area de trabajo de una rutina almacenarlo en el tope de la pila. Obtener las potencias de 10 de una tabla o calcularlas con el procedimiento del ejercicio resuelto 5.

9.- Dado un número en el tope de la pila llevarlo al formato intermedio en el
area de trabajo. Si esta en el rango $10^n <= x < 10^{n+1}$, depositar $x * 10^{17-n}$ y dejar n-17 como característica. Posteriormente eliminar los dígitos mas bajos en 0 sumando 1 a la característica y corriendo a la derecha los dígitos del número depositado. Calcular n=[log x]. (redondeo hacia 0).