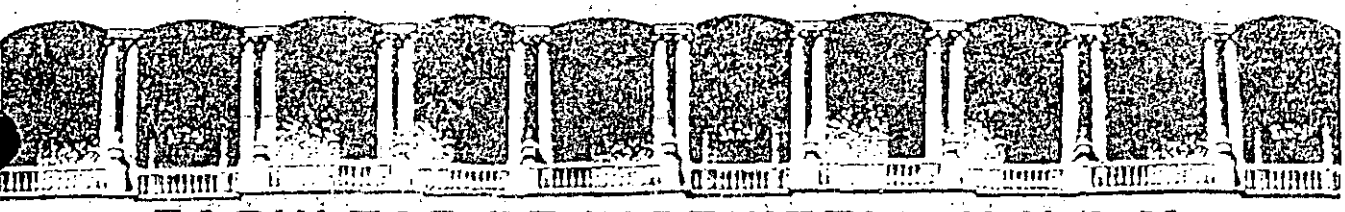


FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

PROGRAMACION DE SISTEMAS PARA COMPUTADORAS PERSONALES USANDO C

Universidad Nacional Autónoma de México
División de Educación Continua de la Facultad de Ingeniería
Centro de Cálculo de la Facultad de Ingeniería
Autor: José Ramón Gallardo Hernández

OBJETIVO: Proporcionar al asistente los elementos necesarios para que pueda explotar los recursos que proporcionan las computadoras personales y así mejorar la eficiencia y presentación de sus aplicaciones.

PRESENTACION:

Para que una computadora pueda realizar o ejecutar una operación es necesario que sea dirigida por una serie de instrucciones contenidas en memoria. Estas instrucciones, llamadas programas, acompañadas con un medio de almacenamiento de datos, son referidas como software. Así, el software es un conjunto de programas que gobiernan la operación de la computadora.

El software puede ser generado y dividido en dos grandes grupos: el software de sistemas y el software de aplicación. El software de sistemas está constituido por los procesos del sistema de control y la captura general de datos. Estos sistemas de software abarcan generalmente las áreas de sistema operativo, utilerías asociadas, manejadores de disco y compiladores de lenguajes de alto nivel. Existen para la PC una gran variedad de lenguajes de alto nivel (Fortran, Basic, Pascal, Cobol, Prolog, C, etc.), de tal manera que la PC es uno de los instrumentos más prácticos para la enseñanza de la programación.

Cualquier programa que puede ser usado en una computadora para resolver un problema es un programa de aplicación. El software de aplicación pueden ser dividido en diversas áreas:

a) Los paquetes de contabilidad general,
b) la aplicación a las áreas industriales (como la médica, la automovilística, financiera, etc.),
c) las hojas electrónicas de cálculo,
d) las aplicaciones de procesadores de palabras,
e) los sistemas manejadores de base de datos,
f) los paquetes gráficos y, g) los paquetes de integración.

——Muchas son las compañías que producen software, pero son pocas las que realmente tienen éxito. Entender el por qué de este éxito no es tan complicado, sólo basta con observar las características de algunos de estos productos: tienen elegantes y atractivas interfases con el usuario, son "amigables" y/o sencillos de usar, son eficientes, son flexibles y, sobre todo, son innovadores. Todas estas características se deben a que sus programadores no sólo tienen un firme dominio de la aplicación específica, sino un completo conocimiento del entorno de la computadora, incluyendo el sistema operativo y hardware.

A lo largo de este curso se presentarán algunos conceptos, métodos y técnicas para desarrollar programas que realmente llamen la atención, con detalles que se puedan señalar como profesionales. Así mismo, se revisarán algunos ejemplos que permitan formar una pequeña biblioteca de funciones para posteriores aplicaciones.

Contenido

Introducción

1. Repaso del lenguaje de programación C

2. Conceptos generales de estructuras de datos
   a) Arreglos y apuntadores
   b) Estructuras en C

3. Recursos de la Computadora Personal
   a) Procesadores 8086, 8088, 8286
   b) Memoria
   c) Periféricos

4. Programación de bajo nivel usando Turbo C
   a) Registros de la máquina
   b) Tipos de apuntadores
   c) Estructura de las interrupciones
   d) Empleo de las interrupciones
   e) Manejadores de interrupciones

5. Temas especiales
   a) Manipulación de la memoria de video
   b) Creación y despliegue de pantallas de captura
   c) Manejo de funciones de graficación
   d) Programas residentes en memoria
   e) Programación de periféricos

Introducción

El término "programación de sistemas" es usado frecuentemente para describir la producción de programas que hacen uso de los recursos de la computadora a bajo nivel. Explotar estos recursos, independientemente de elevar la velocidad de ejecución de los programas, permite al programador una mayor libertad en la creación de algoritmos nuevos, más simples y eficientes.

C fue diseñado originalmente para el sistema operativo Unix en la PDP-11 (DEC) e implantado en ella por Dennis Ritchie en 1972 en los laboratorios Bell. Es relativamente un lenguaje de bajo nivel pues trabaja sólo con caracteres, números y direcciones; no contiene operaciones para manejar directamente objetos compuestos como cadenas, conjuntos, arreglos o listas; tampoco cuenta con operaciones de entrada/salida ni métodos propios para el acceso a archivos. Sin embargo, C provee de construcciones de control de flujo adecuadas a la programación estructurada y es un lenguaje modular, ya que las funciones de un programa pueden ser compiladas por separado.

C es un lenguaje de uso general y aunque su aplicación más usual ha sido la escritura de sistemas operativos, también ha sido empleado con éxito en la programación de procesadores de texto, manejadores de bases de datos y programas numéricos.

Es por lo anterior que C ha tenido gran aceptación entre los programadores de sistemas, que cada vez más aprovechan las ventajas que este lenguaje ofrece.

La estructura general de un programa en C es la siguiente:

```
[ Directivas al preprocesador
  redefinición de nombres de tipos
  declaración y/o definición de
  funciones ]

main([argumentos])
{
    [ Proposiciones ]
}
[ Directivas al preprocesador
  redefinición de nombres de tipos
  declaración y/o definición de
  funciones ]
```

Directivas al preprocesador

Una directiva al preprocesador tiene la siguiente forma:

#comando parámetros

Por ejemplo, la directiva

#include <stdio.h>

indica al preprocesador que el archivo stdio.h (standard input output header) será incluido para compilarse junto con el programa.

Redefinición de nombres de tipos

Sólo hay unos cuantos tipos básicos en C:

char	**Puede almacenar un caracter.**
int	**Entero.**
float	**Punto flotante de precisión sencilla.**
double	**Punto flotante de doble presición.**

Existen algunos calificadores que pueden ser aplicados al tipo int: **short, long y unsigned**. Los dos primeros sirven para diferenciar el tamaño de los enteros, los números sin signo serán siempre positivos; short int, long int y unsigned int pueden ser reemplazados por short, long y unsigned, respectivamente. El calificador long puede ser también usado con double, para especificar un número de punto flotante de alta presición.

Para la definición de nuevos nombres de tipos de datos, C dispone de una declaración denominada typedef.

Sintaxis:

typedef <tipo> <nuevo nombre>;

Ejemplo:

```
typedef int   LENGHT;
typedef char  *STRING;
typedef int (*PFI)();
LENGTH len, maxlen;
LENGTH *lengths[];
STRING p,lineptr[LINES], alloc();
PFI strcmp,numcmp,swap;
```

Hay dos razones importantes para el uso de typedef:

- Para proteger un programa contra problemas de portabilidad. Si se utiliza typedef con los tipos de datos que pueden ser dependientes de la instalación, sólo se tendrá que cambiar los typedef cuando se lleve el programa a otra computadora.
- Para facilitar la documentación de un programa.

Proposiciones y bloques

Una proposición es una expresión válida seguida de un punto y coma (;). Opcionalmente, se puede preceder a una proposición por una etiqueta. En el lenguaje C, el punto y coma es un terminador y no un separador, como en lenguajes de tipo Pascal.

Un conjunto de proposiciones delimitadas entre { } forman un bloque o proposición compuesta; en ella se pueden incluir, a su vez, otras proposiciones (simples o compuestas).

Definición de funciones

Las funciones dividen un problema de computación en pequeñas piezas más fáciles de manejar y entender y aprovechan el trabajo hecho por otras personas en lugar de partir de cero. Además, las funciones permiten abstraer detalles de operación en las partes del programa que no necesitan conocerlas y facilitan un diseño "top-down".

El lenguaje C está diseñado de tal forma que el manejo de funciones es eficiente y sencillo. Un programa consiste en un conjunto de funciones distribuidas en uno o más archivos. La ejecución del programa empieza en la función llamada main.

La forma general de una función es:

```
tipo nombre ( [ lista de parámetros ] )
{
      [ Declaraciones y
        definiciones ]

      [ Proposiciones
        ejecutables ]
}
```

algunas de esas partes son opcionales.

La función más sencilla que se puede definir es

nada () { }

Si el tipo se omite, se asume que la función regresa un entero; si la función no regresa un valor, puede usarse el tipo void. La proposición return se usa para devolver el valor de la función y regresar el control a la función que hizo la llamada, aunque puede omitirse. Si el valor regresado por return no es del mismo tipo que la función, se realiza una conversión automática a él. El valor que regresa una función puede ser ignorado por la función que la llame. Si una función regresa un valor no entero, debe ser declarada antes de usarse.

Si algún parámetro no se declara, es asumido como entero, aunque es una buena práctica declarar todos los parámetros. Cuando un argumento es un arreglo, no es necesario especificar el número de elementos.

Las variables definidas dentro del cuerpo de la función son locales ella.

Precedencia y evaluación.

La siguiente tabla resume la precedencia y asociatividad de todos los operadores del lenguaje C.

Operador	Asociatividad
() [] -> .	->
! ~ ++ - (tipo) * & sizeof	<-
* / %	->
+ -	->
<< >>	->
< <= > >=	->
== !=	->
&	->
^	->
|	->
&&	->
||	->
?:	<-
= op=	<-
,	->

Conversiones de tipos.

Los operandos de diferentes tipos en una misma expresión se convierten a un mismo tipo de acuerdo a una cuantas reglas:

- Los caracteres y enteros se mezclan libremente en las expresiones aritméticas. Los caracteres (char) se convierten siempre a enteros. Cuando se fuerza la conversión de un entero a un caracter, se trunca su parte izquierda.
- char y short se convierten a int y float a double.
- Si algún operando es double, el otro se convierte a double.
- Si algún operando es long, el otro se convierte a long.
- Si algún operando es unsigned, el otro se convierte a unsigned.

La conversión de float a int se hace mediante truncamiento. De double a float, mediante redondeo.

Las conversiones anteriores pueden forzarse mediante la construcción

(tipo)expresión

llamada **cast.**

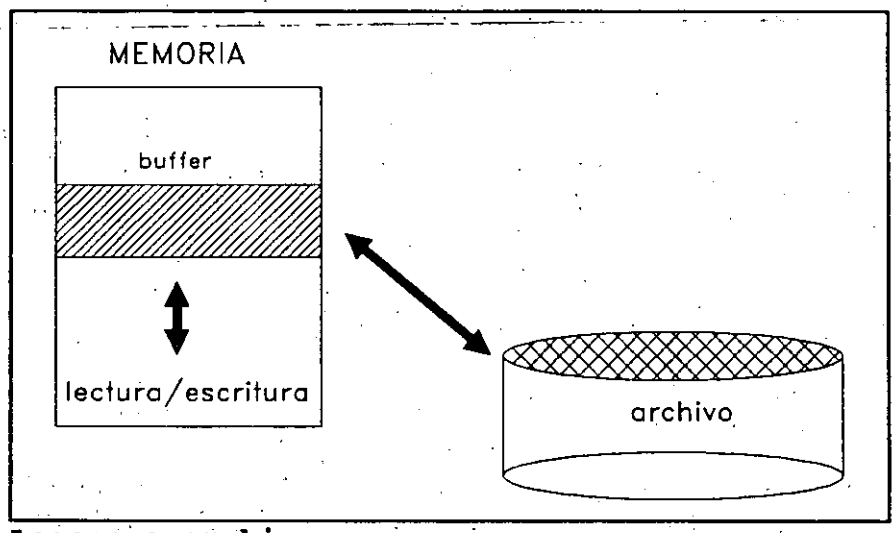
Acceso a archivos

Como ya se ha mencionado, C es un lenguaje que originalmente carece de proposiciones de entrada/salida. Sin embargo, se han desarrollado una serie de funciones que permiten tener acceso a archivos.

Cuando se accesa un archivo no se hace directamente a disco, sino que se declara un apuntador a un "buffer", en el cual se almacenan los datos ya sea para escritura o lectura. Una vez lleno el buffer, se procede a vaciar o leer más información del disco según sea el caso.

Para tener acceso a un archivo, en C se tiene que definir un apuntador al buffer:

```
FILE *fp;
```



Acceso a archivos

donde FILE es un tipo (definido mediante typedef) de una estructura declarada para soportar al buffer; y fp es una variable apuntador a esa estructura.

Para accesar un archivo es necesario usar una función para solicitar un buffer al mismo. Esto se hace mediante la función fopen():

```
FILE * fopen(char *nombre_de_archivo, char *modo);
```

fopen() es una función que recibe dos argumentos y que regresa un apuntador a FILE.

El primer argumento **char *nombre_de_archivo** es una cadena de caracteres (un apuntador) que contiene el nombre del archivo en disco. El segundo argumento, **char *mode**, es una cadena de caracteres que describe la operación a realizar sobre el archivo. Se pueden realizar tres operaciones básicas sobre el archivo:

"r" (read): Abrir el archivo para lectura
"w" (write): Abrir el archivo para escritura
"a" (append): Abrir el archivo para agregar

Algunos sistemas distinguen entre archivo tipo texto y archivos binarios, por lo que se podría agregar otro modo más:

"b" (binary): Abrir el archivo para agregar

```
FILE fopen(), *fp;

fp = fopen("RAMON.DAT", "r");

/* Abre el archivo RAMON.DAT para lectura */
```

```
#include <stdio.h>

 FILE *fp;
char *archivo = "A:\\usr\\curso\\datos.dat";
fp = fopen(archivo,"w");
 /* abre el archivo datos.dat para escritura */
```

Ejemplos

Si un archivo que no existe es abierto con los modos de escritura y agregado, este se creará en disco. Pero si se trata de abrir un archivo inexistente para lectura, esto causará un error. Cuando se producen errores al tratar de abrir un archivo con la función fopen(), se retornará el valor NULL.

Otras funciones para el manejo de archivos son:

int getc(FILE *fp)

que lee un carácter del buffer apuntado por fp, y

int putc(int c, FILE *fp)

que escribe un carácter al buffer apuntado por fp.

Conceptos generales de estructuras de datos

Arreglos y apuntadores.

En C hay una estrecha relación entre apuntadores y arreglos. Cualquier operación realizada con arreglos puede ser hecha también con apuntadores, la versión con apuntadores es generalmente más rápida, aunque puede resultar más difícil de entender.

Un apuntador no es más que la dirección de algo que está en memoria, es decir, se trata de una variable que contiene la dirección de otra variable. Debido a que el nombre de un arreglo es en sí mismo un apuntador, los arreglos y apuntadores mantienen una estrecha relación.

Puesto que un apuntador contiene la dirección de un objeto, se puede acceder al objeto "indirectamente" a través de él.

Si px es un apuntador y x una variable, entonces :

```
px = &x;
```

asigna la dirección de x a la variable px, por lo que se dice que px "apunta" a x.

El operador unario * toma su operando como una dirección y accesa a esa dirección para obtener su contenido.

```
y = *px;
```

asigna el contenido de cualquier parte a donde apunte px. Por lo que la secuencia:

```
px = &x;
y  = *px;
```

es lo mismo que hacer:

```
y = x;
```

La declaración

```
int a[10];
```

De hecho, getchar() y putchar() están definidas a partir de las funciones anteriores:

```
#define  getchar()   getc(stdin)
#define  putchar(c)  putc((c),stdout)
```

donde stdin (standard input) y stdout (standard ouput) son los apuntadores a los buffers de la entrada estándar (teclado) y salida estándar (video) respectivamente.

Otras funciones para manejo de archivos son

```
int fscanf(FILE *fp, char *format, ...)
int fprintf(FILE *fp, char *format, ...)
```

que realizan funciones análogas a scanf y printf, sólo que a disco.

Otra función importante es fclose() cuya sintaxis es:

```
int fclose(FILE *fp)
```

que sirve para liberar el buffer asignado al archivo en disco.

```
char fgets(char *line, int maxline, FILE *fp)
int  fputs(char *line, FILE *fp)
```

son funciones que leen y escriben una cadena de caracteres. En fgets(), maxline indica el tamaño máximo de la línea a leer en caso de no encontrar el carácter '\n' (nueva línea).

Ejemplo:

```
#include <stdio.h>
main(int argc,char *argv[])  /* cat: concatenador de archivos */
{
    FILE *fp, *fopen();

    if (argc == 1)  /* ningún argumento */
        filecopy(stdin);
    else
        while (--argc > 0)
            if ((fp = fopen(*++argv, "r")) == NULL) {
                printf("cat: No se puede abrir %s\n",*argv);
                break;
            }else {
                filecopy(fp);
                fclose(fp);
            }
}

filecopy(FILE *fp) {
    int c;
    while ((c = getc(fp)) != EOF)
        putc(c,stdout);
}
```

Arreglos y apuntadores.

En C hay una estrecha relación entre apuntadores y arreglos. Cualquier operación realizada con arreglos puede ser hecha también con apuntadores, la versión con apuntadores es generalmente más rápida, aunque puede resultar más difícil de entender.

Un apuntador no es más que la dirección de algo que está en memoria, es decir, se trata de una variable que contiene la dirección de otra variable. Debido a que el nombre de un arreglo es en sí mismo un apuntador, los arreglos y apuntadores mantienen una estrecha relación.

Puesto que un apuntador contiene la dirección de un objeto, se puede acceder al objeto "indirectamente" a través de él.

Si px es un apuntador y x una variable, entonces :

```
px = &x;
```

asigna la dirección de x a la variable px, por lo que se dice que px "apunta" a x.

El operador unario * toma su operando como una dirección y accesa a esa dirección para obtener su contenido.

```
y = *px;
```

asigna el contenido de cualquier parte a donde apunte px. Por lo que la secuencia:

```
px = &x;
y  = *px;
```

es lo mismo que hacer:

```
y = x;
```

La declaración

```
int a[10];
```

define un arreglo de enteros, que es un bloque de diez objetos consecutivos a[0], a[1], ..., a[9].

Si pa es un apuntador a entero

```
int *pa;
```

la expresión

```
pa = &a[0];
```

indica que **pa** apuntará al elemento cero de **a**, esto es, **pa** contiene la dirección de **a[0]**.

Si **pa** apunta a un elemento de un arreglo, por definición, **pa+1** apunta al siguiente elemento, **pa-1** apunta al elemento anterior, **pa+i** apunta **i** elementos después de **a** y **pa-i** apunta **i** elementos antes de **a**. Así, si **pa** apunta a **a[0]**,

```
*(pa+1)     es equivalente a  pa[1]
*(pa+i)     es equivalente a  pa[i]
```

La correspondencia entre los índices de un arreglo y la aritmética de apuntadores es muy cercana, de hecho, una referencia a una arreglo es convertida por el compilador en una expresión entre apuntadores.

Ahora, las expresiones

```
a[i]
*(a+i)
pa[i]
*(pa+i)
```

son equivalentes. La única diferencia entre pa y a es que a es una constante, no una variable; así, aunque pa++ es una expresión válida, a++ no lo es.

Siguiendo el razonamiento anterior se puede decir que la expresión

```
*(i+a)
```

donde **i** es el índice y **a** el arreglo, es equivalente a

```
a[i]
```

Dado que en una suma el orden de los sumandos no importa, entonces se puede afirmar que la expresión

```
i[a]
```

es equivalente a

```
*(i+a)
```

que a su vez es equivalente a

```
a[i]
```

Así mismo, el nombre de un arreglo es un apuntador al primer elemento del arreglo. Entonces la asignación

```
pa = &a[0];
```

puede ser sustituida por

```
pa = a;
```

También es posible pasar sólo parte de un arreglo a una función:

```
f(&a[2]);
```

C garantiza que cualquier dirección no contendrá un valor de cero. La constante **NULL**, definida en stdio.h como 0, puede ser usada para inicializar apuntadores que no contengan una dirección por el momento.

Los apuntadores pueden ser también comparados: **p < q** es verdadero si **p** apunta a un elemento de un arreglo anterior al apuntado por **q**. Los operadores == y != pueden ser usados también para comparar a un apuntador con **NULL**. Si **p** y **q** apuntan a miembros del mismo arreglo, **p-q** es el número de elementos entre **p** y **q**:

Si un arreglo bidimensional se pasa a una función, la declaración del argumento debe incluir la dimensión de la columna, la dimensión de los renglones es irrelevante, puesto que lo que se pasa es un apuntador.

Estructuras en C

Una estructura es un conjunto de variables de, posiblemente, diferentes tipos. Las estructuras ayudan a organizar datos complicados, pues permiten tratar datos relacionados como unidad, en vez de tratarlos como objetos independientes. Por ejemplo, una fecha puede tener tres componentes:

```
struct fecha {
        int dia;
        int mes;
        int anio;
};
```

la palabra reservada **struct** especifica la declaración de una estructura; el nombre o etiqueta de la estructura, fecha, se puede emplear en declaraciones posteriores como abreviatura de la estructura:

```
struct fecha nacimiento, hoy;
```

declara las variables nacimiento y hoy como estructuras fecha.

Los elementos de la estructura son llamados miembros; para referenciar a un miembro de una estructura se usa el operador .

```
nacimiento.anio=65;

switch (hoy.mes) {
        case 1 : printf("Enero\n"); break;
        case 2 : printf("Febrero\n"); break;
        ...
        case 12: printf("Diciembre\n");
}
```

Las estructuras externas y estáticas se pueden inicializar

```
struct fecha navidad = {25,12,1988};
```

Las estructuras se pueden anidar

```
struct personas{
     char nombre[30];
     char direccion[50];
     long cp;
     double salario;
     struct fecha nacimiento;
}
```

La etiqueta de la estructura es opcional, las declaraciones anteriores se pueden abreviar como

```
struct {
     int dia;
     int mes;
     int anio;
} nacimiento, hoy;
```

El operador . es asociativo de izquierda a derecha:

```
struct persona empleado;
empleado.nacimiento.mes=6;
```

En versiones anteriores de C, las estructuras no se podían asignar ni pasar como argumentos a funciones, sin embargo, estas restricciones han sido eliminadas en las nuevas versiones del lenguaje.

La declaración

```
struct personas{
     char nombre[42];
     char direccion[50];
     char telefono[10];
}

struct personas *persona;
```

especifica que persona es un apuntador a una estructura del tipo personas. El miembro nombre se puede referenciar mediante

```
(*persona).nombre
```

pero los apuntadores a estructuras son tan frecuentes que se ha introducido la notación

p->nombre

que es más breve que la anterior.

Si se deseara tener el registro de un grupo de personas, cada una con su nombre, su dirección y su teléfono, podría hacerse la siguiente declaración

struct personas grupo[20];

Esta definición nos permitiría tener almacenados los datos de hasta 20 personas, muy útil cuando se sabe exactamente el número de personas cuya información será almacenada.

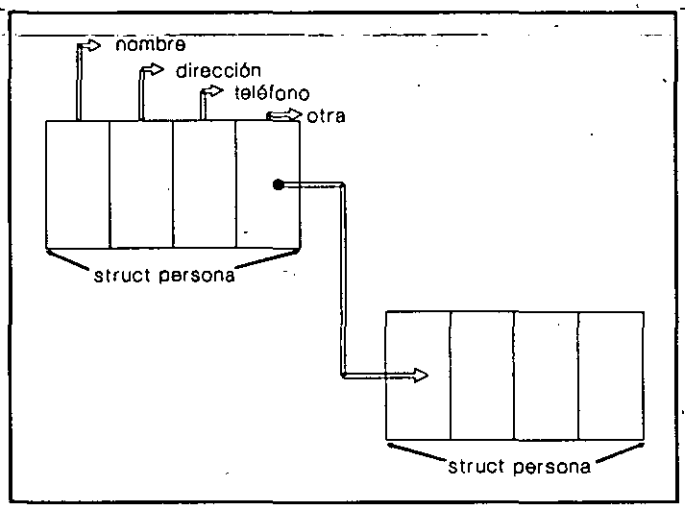
Sin embargo, es probable que durante la ejecución del programa, surja la necesidad de agregar una persona más a la lista. Modificar el programa cada vez que suceda esta situación, puede ser molesto -sobre todo cuando el código a compilar es muy grande. Una forma eficiente de solucionar este problema, es usando estructuras autoreferenciadas. Por ejemplo, la declaración

```
struct personas{
     char nombre[42];
     char direccion[50];
     char telefono[10];
     struct personas *otra;
};
struct personas *persona;
```

nos indica que el tipo personas tiene un campo más, que es un apuntador a una estructura del mismo tipo.

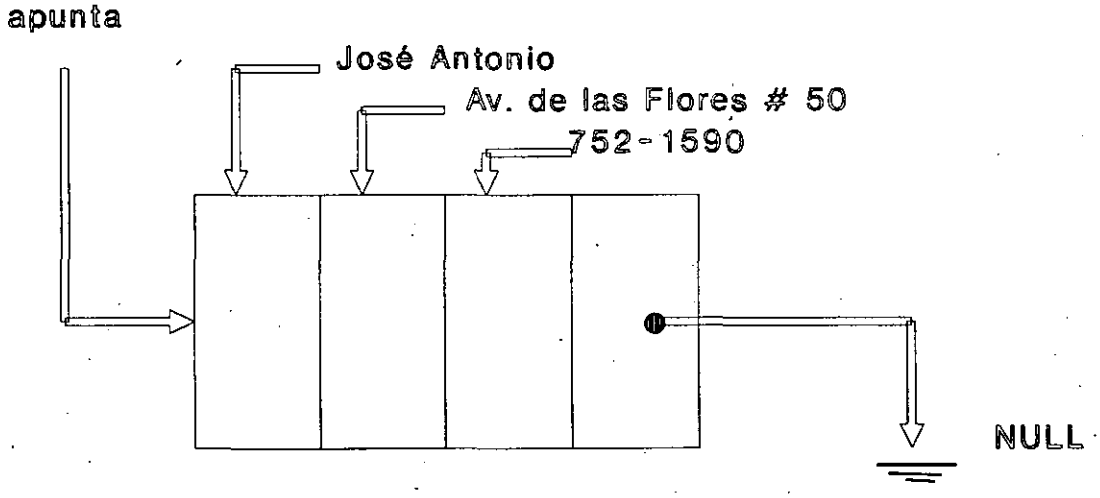
Si se tiene

```
struct personas persona =  {
   {"José Antonio"},
   {"Av. de las flores #50"},
   {"752-15-90"},
   {NULL}
};
struct personas *apunta;
apunta = persona;
```



Estructuras autoreferenciadas

Esto indica que **apunta** es una variable de tipo **struct persona**, que se le asigna la dirección de persona, con lo cual es posible tener acceso a sus miembros.



Acceso a campos

La proposición

```
printf("%s\n",apunta->nombre);
```

tendrá como salida

José Antonio

Una unión es un tipo derivado. Las uniones tienen la misma sintaxis que las estructuras, pero sus miembros comparten el almacenamiento.

Una variable unión define a un conjunto de valores alternos que pueden almacenarse en una porción compartida de memoria. Una unión tiene por objeto proporcionar una variable que contenga legítimamente diferentes tipos. Ejemplo:

```
union  valor {
     int     n;
     float   x;
} temp;
```

```
union  <nombre> {
       declaración
       de miembros;
} <variables>;
```

El compilador asigna una porción de almacenamiento que pueda acomodar al más grande de los miembros especificados.

Sintácticamente se tiene acceso a los miembros de una unión como

<nombre>.<miembro>

y también

<apuntador a union> -> <miembro>

Procesadores 8086, 8088 y 80286

Hablar del microprocesador 8086 es como hablar del 8088; en realidad, se podría decir que éstos son gemelos, La mayor diferencia entre ellos es el tamaño del bus de datos (16 bits para el 8086 y 8 bits para el 8088).

El 8088 es un microprocesador de 16 bits que conforma el CPU de la computadora. El 8088 realiza las operaciones básicas de transferencia de datos y direcciones a través de una red electrónica de intercomunicaciones llamada bus. Localizados a lo largo del bus están los puertos de entrada y salida, que conectan memorias y chips de soporte al bus. Dentro del 8088, 14 registros proporcionan una área de trabajo para transferencia y proceso de datos. Con éstos registros, el 8088 puede accesar hasta 1 millón de bytes de memoria y hasta 64K de puertos de entrada y salida. El 8088 puede trabajar sólo con números enteros, los reales o de punto flotante se manejan de manera especial en subrutinas, pero a un gran costo de eficiencia y velocidad.

A principios de 1985 IBM presentó el procesador 80286 diseñado para soportar al sistema operativo XENIX (UNIX system 3.0) con multiproceso y capacidades de multiusuario. El 80286 tiene un incremento en el número de funciones internas respecto al 8088, haciendo su ejecución mucho más eficiente y rápida (casi 2.5 veces).

A diferencia de los anteriores, el microprocesador 80286 corre con una velocidad de 12 MHz. Como el 8086, el 80286 es un verdadero microprocesador de 16 bits (con un bus de datos interno y externo

MICROPROCESADOR	80286	8086/88
VELOCIDAD	12 MHz	4.77 - 10 MHz
MEMORIA RAM	Hasta 16 Mbytes	640 Mbytes
DISCO DURO	Hasta 110 Mbytes	20 Mbytes

Tabla comparativa

de 16 bits). También ha avanzado en un manejador de memoria con capacidad de accesar una memoria virtual con un espacio de direcciones de un Gigabyte. El 80286 puede correr bajo dos modos: bajo el modo PC DOS, el 80286 corre en direcciones reales que simulan al 8086 o al 8088; en este modo el 80286 puede accesar directamente sólo 1 Mbyte de memoria en sistema. En el segundo modo, bajo XENIX, el 80286 puede direccionar hasta 16 Mbytes de memoria física.

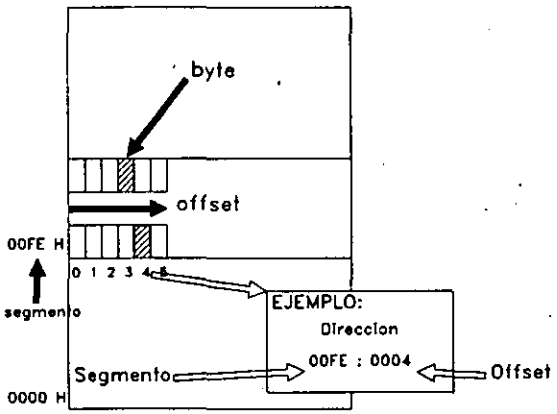
En cuanto a la programación, el 80286 permitió agregar una serie de interrupciones al ROM-BIOS:

* **Manejo de unidades de disco**
* **Manejo del teclado**
* **Manejo del reloj interno.**

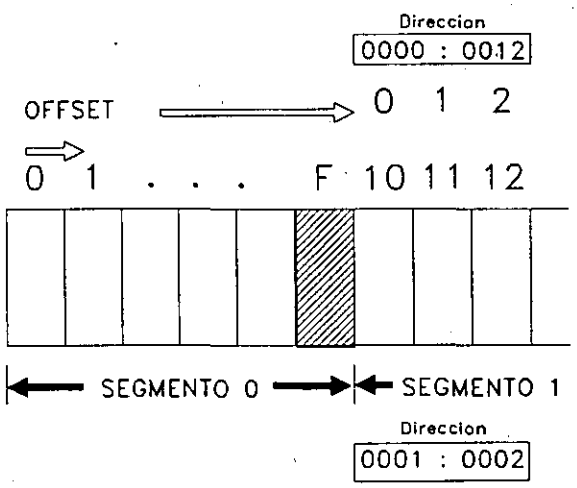
Estas nuevas interrupciones pueden darle mayor velocidad a la ejecución de los programas.

La memoria de la PC

La organización lógica de la memoria de las computadoras personales (PC) bajo DOS divide el total de la memoria en segmentos. Para poder direccionar un valor dentro de cada segmento se requiere un desplazamiento (offset). Por lo tanto, una dirección en memoria esta formada por dos partes: un segmento y un offset, que usualmente se representan como SEGMENTO:OFFSET.



EL tamaño de un segmento puede ir de 16 bytes hasta 64 K de memoria, por lo que el offset puede variar en el rango de 0 a 64 Kbytes. Este esquema de direccionamiento (segmentación) permite que una dirección de memoria puede ser representada con diferentes pares segmento offset. Por ejemplo, la direccion 0000:0012 también se puede representar como 0001:0002 (ver figura).



Representación de direcciones

La máxima capacidad de memoria que puede direccionar una PC es de 1024 Kbytes (1 Mbyte, con la dirección F000:FFFF). Sin embargo, sólo 640 Kbytes están disponibles para el usuario, ya que el resto esta dedicado al sistema operativo, el manejador de interrupciones, memoria de video, etc.

SEGMENTO	
F000	Area permanente de ROM: ROM BIOS
E000	Area de ROM
D000	Area de ROM
C000	Extensiones del BIOS
B000	Memoria convencional de video
A000	Extension de memoria de Video
9000	RAM
8000	RAM
7000	RAM
6000	RAM
5000	RAM
4000	RAM
3000	RAM
2000	RAM
1000	RAM
0000	RAM Usada por los manejadores de interrupciones

Diagrama de la memoria

Generalmente el bloque 0000 es usado para el almacenamiento de información propia del sistema.

Parte de la filosofía de diseño de la familia de las PC se centra en las rutinas de servicio del BIOS, que son las encargadas de proporcionar todas las funciones de control y operaciones que se consideran necesarias. La filosofía de la familia PC es: Dejar que el BIOS lo haga. Esta idea tiene diversos resultados benéficos: Usando las rutinas del BIOS se fomentan prácticas de buena programación y se previenen algunos "trucos malos" que han sido la base de otras computadoras. El uso del BIOS incrementa la oportunidad de que un programa corra en todas las PC sin problemas de compatibilidad.

Parte de BIOS es fija y se proporciona en ROM. El papel de este ROM-BIOS es proporcionar los servicios fundamentales que son necesarios para la operación de la computadora. En gran parte el BIOS controla los mecanismos periféricos de la computadora como son el video, teclado, manejadores de discos,etc. Cuando se usa el término BIOS, nos referimos a un "mecanismo" que controla los programas. Un programa que traslada realiza un comando (como leer algo de el disco) y revisa cada paso necesario para realizar el comando, incluyendo errores de detección y corrección. El BIOS no sólo se refiere a las rutinas que contienen información o realizan tareas que son fundamentales para otros aspectos de operación de la computadora como control de la hora del dia.

Conceptualmente, los programas BIOS actúan como una interfase entre los programas de aplicación y el hardware, esto significa que el BIOS trabaja en dos direcciones. Por un lado recibe llamadas de programas para realizar ciertos servicios y , por otro comunica la computadora con los periféricos y maneja las interrupciones que genera el hardware.

Controladores de video

Existe una gran variedad de tarjetas controladoras o adaptadoras de video encargadas de realizar el manejo de la pantalla de despliegue de una computadora. Dependiendo de la tarjeta que se use, se tendrá la posibilidad de manejar diferentes características de pantalla como son: manejo de colores, subrayado, video inverso, desplegado de gráficos, manejo de la resolución, brillantez, etc.

A continuación se proporcionará una descripción de los principales adaptadores de video existentes: Monochrome Display Adapter (MDA), Hercules Graphics Card(Hercules), Color Graphics Adapter (CGA), Enhanced Graphics Adapter(EGA) y Video Graphics Array(VGA).

El adaptador de video es una tarjeta que provee toda la circuiteria de control necesaria para convertir los comandos de visualización de la computadora en imágenes visibles sobre la pantalla de despliegue. En esencia, el adaptador de video funciona como un transductor, convirtiendo las señales de la computadora en señales de voltaje que controlan el barrido de la pantalla de despliegue.

El adaptador de video tiene dos partes fundamentales: un circuito integrado conocido como "controlador de video", el cual contiene los comandos que regulan la operación de la pantalla de despliegue, y una "memoria" asociada. La información que aparece en pantalla necesita ser almacenada, para lo cual se dispone, dentro de la misma tarjeta, de circuitos de almacenamiento especialmente destinados para este fin. Aunque esta memoria opera igual que la memoria principal de la computadora, la cantidad y direccionamiento dependen del tipo de adaptador de video.

Los dos adaptadores de video que han dominado el mercado son los que fueron introducidos con las PCs originales: el monochrome display adapter (MDA) y el color graphics adapter (CGA). El MDA es el más popular, dado que viene como componente original de las computadoras personales, aunque es el que más limitadas capacidades tiene. Sólo puede visualizar caracteres y únicamente en un color. El CGA fue diseñado para mostrar tanto textos como gráficos, dibujándolos como una secuencia finita de puntos. Tiene la posibilidad de manejar hasta 16 colores diferentes y varios modos de video, los cuales le proveen de una variedad de combinaciones de color y resolución. Ambos adaptadores de video fueron diseñados por IBM.

Después de los dos adaptadores de video originales, otros mas fueron diseñados con el propósito de proporcionar mayores cualidades, principalmente mayor resolución. El más popular, por su bajo costo, es el adaptador gráfico Hercules, diseñado por la Hercules Computer Technologies. La tarjeta Hercules, aunque únicamente se puede conectar a videos monocromáticos, proporciona la alta calidad de imagen de los textos del MDA y, además, provee de un modo gráfico de alta resolución.

Posteriormente, IBM desarrolló dos nuevos adaptadores para sus modelos avanzados de computadoras y sistemas personales. El primero conocido como EGA, es el esfuerzo de IBM para unificar la gran variedad de adaptadores de video existentes y utilizar mejor las nuevas posibilidades tecnológicas de los modelos avanzados de las PC. El EGA se ha convertido en el estándar de las PC XT, AT y clones. Es un controlador gráfico que permite un control directo tanto en monitores a color como monocromáticos en una gran variedad de modos. El segundo, conocido como VGA, es esencialmente una versión ampliada del EGA. Es mucho más flexible en el aspecto de la resolución en los modos de video que se pueden desplegar, particularmente cuando se usan monitores de frecuencia variable.

Posiblemente no existe otro dispositivo periférico estándar que de más problemas al programador que el puerto serie asíncrono. A diferencia del puerto paralelo (cuya transmisión se realiza a través de bytes), en el puerto serie los datos se transmiten bit a bit.

Cada byte de datos que se transmite a través del puerto serie usa la siguiente secuencia de datos:

1. Un bit inicial.
2. Ocho bits de datos (siete en algunos casos).
3. Un bit de paridad.
4. Uno o dos bits de fin de transmisión.

La transmisión se denomina asíncrona porque el tiempo entre la transmisión de un byte de los datos (bit a bit) es indiferente, es decir, puede transcurrir cualquier tiempo entre la transmisión de un par de bytes cualquiera.

El estado de la línea de transmisión cuando no está ocupada es en alto. Un bit puesto a cero hace que la línea esté en bajo; un bit puesto a uno hace que la línea esté en alto. El bit inicial que marca el principio de la transmisión de un nuevo byte lleva la línea a cero por un ciclo. Se transmiten entonces los bits de datos. Finalmente se envían uno o dos bits fin de transmisión, que corresponden a estados de la línea en nivel bajo. Los bits de fin de transmisión determinan el tiempo más corto entre el envío de dos bytes. En general, no tiene importancia el usar uno o dos bits de fin de transmisión, siempre y cuando los dos puertos, tanto el que envía como el que recibe, utilicen el mismo número.

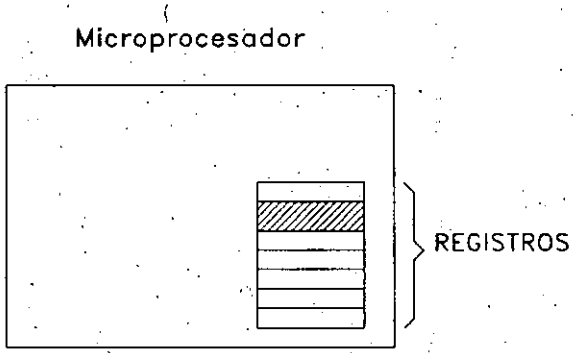
El bit de paridad, si se usa, sirve para detectar los errores de transmisión. La paridad puede ser par o impar. Si se elige la paridad par, al bit de fin de transmisión se le da un valor de forma que el byte transmitido más el bit de paridad de un número

par. Si se usa la paridad impar, el byte más la paridad darán un número impar.

La velocidad a la que se transmiten los bits se mide en bauds (bits por segundo). La velocidad más baja a la que se usa es de 300 bauds, que se utiliza fundamentalmente en los modems más viejos. Una PC es capaz de transmitir hasta 9600 bauds.

Registros de la máquina

Los registros pueden ser vistos como pequeños dispositivos de almacenamiento de alta velocidad dentro de la unidad central de proceso (CPU, por sus siglas en inglés).



Registros de la computadora

Muchas instrucciones emplean estos registros de una u otra manera para operaciones aritméticas, de control y acceso a memoria. La familia del 8086 tiene 14 registros, cada uno de 16 bits, divididos en registros de segmento, de índice, de datos y de banderas.

Para que se puedan ejecutar los programas, la PC maneja áreas de segmento especiales conocidas como de código, de datos, de stack y una área extra. Las direcciones de estas áreas se encuentran contenidas dentro del microprocesador en los "registros de propósito especial" o "direcciones base":

CS (Code Segment
DS (Data Segment)
SS (Stack Segment)
ES (Extra Segment)

Así mismo, dentro de estos segmentos se manejan offsets (desplazamientos) para obtener direcciones efectivas de memoria.

Existen cuatro registros, conocidos como "apuntadores" e "índices", asociados a los registros de propósito especial:

IP (Instruction Pointer)

BX (Base register)

SP (Stack Pointer)

DI (Destination Index)

Estos registros se combinan con los registros de propósito especial de la siguiente manera:

CS + IP = Dirección de una instrucción (como la dirección de una función)

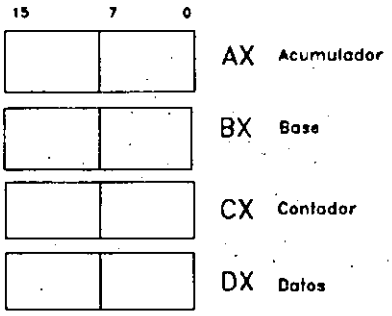
DS + BX = Dirección de un dato en RAM (como la de una variable estática)

SS + SP = Dirección de un dato en el stack (como la de una variable local)

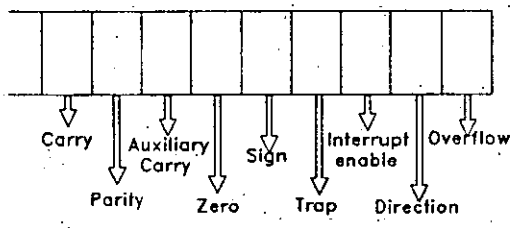
ES + DI = La dirección de una cadena de caracteres en RAM.

También el registro DS se combina con otros registros de índice, el SI (Source Index) ó el DI (Destination Index). Así mismo, el registro SS también puede ser combinado con el registro BP (Base Pointer).

La familia del 8086 posee cuatro registros, llamados de propósito general, cada uno de los cuales pueden ser manejados como dos registros de 8 bits.



Registros de propósito general



Registro de banderas

El registro de banderas de "status" usa 9 de sus 16 bits para marcar status de estados del procesador u operaciones aritméticas.

Tipos de apuntadores

En Turbo C, así como en muchos otros compiladores de C, clasifican a las funciones y los apuntadores de datos en near, far y huge.

En la PC la dirección base del segmento de datos está dada por el registro DS y el máximo tamaño de esta área está dada por el máximo valor de su offset (16 bits), o sea 64 Kbytes. Si todos los datos en un programa están contenidos en le mismo segmento, no es necesario mantener la dirección del segmento para cada variable, por lo que los apuntadores de datos necesitan sólo un tamaño de 16 bits (para el offset) y las llamadas a función no necesitan volver a cargar el registro de segmento. A este tipo de apuntadores de datos y funciones se les conoce como near.

Sin embargo, si un programa necesita almacenar sus datos en más de un segmento, los apuntadores a estos datos deben contener la dirección del segmento además del offset, por lo que necesitan una longitud de 32 bits. De manera similar, si las funciones en un programa están contenidas en más de un segmento, la longitud de la dirección de las funciones también debe ser de 32 bits, ya que un llamado a estas funciones necesita cargar el registro del segmento de código. Este tipo de apuntadores a datos y funciones se conoce como far, y son usados en programas que contienen más de 64 Kbytes de código o datos.

Dado que direcciones tipo far consisten de componentes de segmento y offset diferentes, dos variables con la misma dirección, formados por diferentes segmentos y offsets, al ser comparados (==, >) pueden producir resultados inesperados. Por ejemplo, la dirección (segmento:offset) 0000:0012 y la dirección 0001:0002 realmente son la misma, pero como una dirección es un número éstas representan valores diferentes. Es por esto que Turbo C agrega un tipo más conocido como huge, que tiene un formato que permite la comparación entre apuntadores, así como accesar estructuras de más de 64 Kbytes.

Estructura de las interrupciones.

Las interrupciones son señales enviadas al CPU para suspender lo que está haciendo y transferir el control a un programa especial llamado manejador de interrupciones. La transferencia es forzada por mecanismos especiales de hardware que son cuidadosamente diseñados para alta velocidad. El manejador de interrupciones es el responsable de determinar la causa de la interrupción, tomando la acción apropiada y regresando el control al proceso que originalmente fue suspendido.

Las interrupciones son causadas generalmente por eventos externos al CPU que requieren atención inmediata, tales como:

+ **Completar un proceso de Entrada/Salida.**
+ **Detección de una falla de hardware.**
+ **Catástrofes.**

La familia de microprocesadores Intel soporta 256 tipos de interrupciones, o niveles, que pueden ser agrupados en tres categorías básicas:

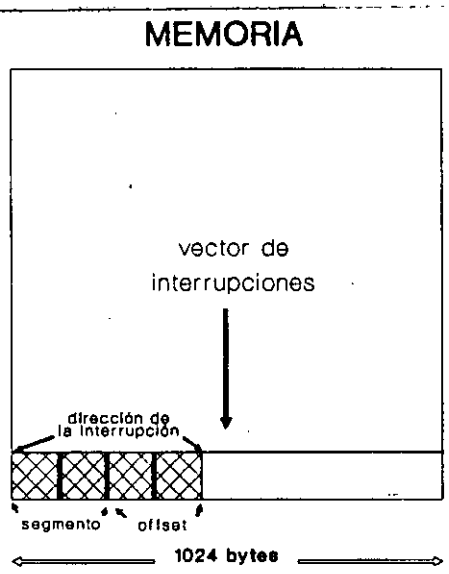
Interrupciones internas de hardware.— Son generadas por ciertos eventos encontrados durante la ejecución de un programa, tal como una división por cero.

Interrupciones externas de hardware.— Son lanzadas por los controladores de dispositivos periféricos o por coprocesadores como el 8087/80287.

Interrupciones de software.— Son realizadas de manera asíncrona por cualquier programa al ejecutar la instrucción INT.

Para cada interrupción se reserva un elemento del llamado "vector de interrupciones", que especifica dónde se encuentra localizado el programa manejador de interrupciones para ese tipo de interrupción.

La parte más baja (los primeros 1024 bytes) de la memoria es llamada "vector de interrupciones". Cada 4 bytes en la tabla corresponde a un tipo de interrupción (0H a 0FFH) y contiene el segmento y el offset del manejador de interrupciones para ese nivel. Los niveles más bajos, es decir, las interrupciones 0H a 1FH, son usadas para interrupciones internas de hardware; las interrupciones 20H a 3FH son usadas por DOS y todas las demás están disponibles para posteriores aplicaciones.



Vector de interrupciones

Manejo de las interrupciones

Para poder ejecutar una interrupción de software, se requiere especificar el tipo de interrupción y cargar en los registros del procesador ciertos datos que son solicitados por el sistema. Turbo C tiene variables que representan a cada uno de los registros del procesador, por lo que no es necesario programar en ensamblador. A estas variables se les conoce como pseudovariables, y son

```
_AH  (acumulador, parte alta)
_AL  (acumulador, parte baja)
_AX  (acumulador)
_BH  (base, parte alta)
_BL  (base, parte baja)
_BX  (base)
_CH  (contador, parte alta)
_CL  (contador, parte baja)
_CX  (contador)
_DH  (datos, parte alta)
_DL  (datos, parte baja)
_DX  (datos)
_CS  (segmento de código, "code segment")
_DS  (segmento de datos, "data segment")
_SS  (segmento de stack, "stack segment")
_ES  (segmento extra, "extra segment")
```

Con ellos es posible realizar casi cualquier actividad sobre las interrupciones de la PC.

Existe, sin embargo, un gran inconveniente al usar las pseudovariables; dado que se modifican directamente los registros de la computadora y éstos son accesados por el procesador, la ejecución del programa puede llegar a interferir los valores finales que regresa la interrupción. Además, muchas interrupciones regresan un valor de éxito o error en el registro de banderas y no existe una pseudovariable para este.

Es conveniente, por lo anterior, hacer uso de funciones que invoquen a estas interrupciones, usando un registro de datos intermedio.

Ejemplos:

La siguiente función nos permite posicionar el cursor en pantalla

```
#include <dos.h>
void GoToXY(int col, int ren)
{
     _AH = 02;              /* Función 2, servicios de video */
     _BH = 00;              /* página de despliegue (usar 0) */
     _DH = ren;             /* renglon                       */
     _DL = col;             /* columna                       */
     geninterrupt(0x10);    /* función para ejecutar INT     */
}
```

La interrupción 10 hexadecimal, permite tener funciones del sistema básico de entrada/salida (BIOS, basic input output system), y la función 2 sirve para posicionar el cursor en pantalla.

La misma función se puede escribir con base en un registro de datos.

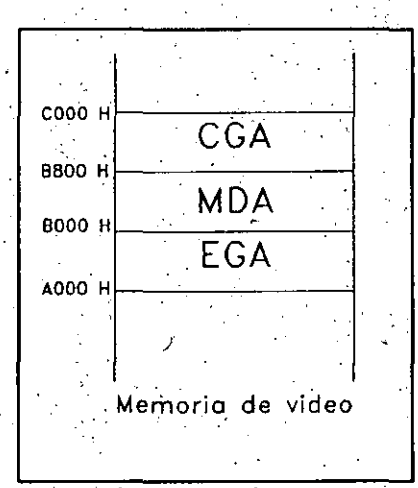
```
#include <dos.h>
void GoToXY(int col, int ren)
{
     REGPACK reg;

     reg.r_ax = 2 << 8;              /* servicios de video   */
     reg.r_bx = 0;                   /* página de despliegue */
     reg.r_dx = (ren << 8) | col;    /* renglon y columna    */
     intr(0x10,&registro);           /* ejecuta interrupción */
}
```

El uso del registro intermedio resulta más seguro, pero la función no es tan rápida en su ejecución.

Memoria de video

Cuando un programa despliega un carácter en pantalla, lo logra escribiendo a una área de la memoria RAM conocida como *memoria de video*. El sistema operativo reserva las direcciones absolutas A000:0000 a C000:0000 para la memoria de video; las direcciones exactas dependen de los diferentes tipos de resoluciones.

En modo texto se tienen 80 columnas y 25 renglones en la pantalla (80 x 25 = 2000 bytes). Si se considera que cada carácter de video está representado por 2 bytes (un byte para el ASCII del carácter y otro para sus atributos), entonces se tienen 4000 bytes (80 x 25 x 2 = 4000) para representar a toda la pantalla.



Memoria de video

El byte de atributo tiene el siguiente formato:

bits 7 6 5 4 3 2 1 0

BLINK	FONDO	INTENSIDAD	LETRA

Los bits 0-2 están reservados para el color de la letra; los bits 4-6, para el color del fondo; el bit 3 indica la intensidad del color, y el bit 7 permite el efecto de intermitencia.

En cuanto a los colores, se pueden establecer las siguientes combinaciones:

Para monitor monocromático

FONDO	LETRA	VIDEO
000	000	No hay video
000	001	Underline
000	111	Video normal
111	000	Video inverso

Para monitor a color (fondo o letra)

I: Intensidad
R: Rojo
V: Verde
A: Azul

I	R	V	A	NUMERO	DESCRIPCION
0	0	0	0	0	Negro
0	0	0	1	1	Azul
0	0	1	0	2	Verde
0	0	1	1	3	Cyan
0	1	0	0	4	Rojo
0	1	0	1	5	Magenta
0	1	1	0	6	Café
0	1	1	1	7	Gris intenso
1	0	0	0	8	Verde oscuro
1	0	0	1	9	Azul intenso
1	0	1	0	10	Verde intenso
1	0	1	1	11	Cyan intenso
1	1	0	0	12	Rojo intenso
1	1	0	1	13	Magenta intenso
1	1	1	0	14	Amarillo
1	1	1	1	15	Blanco

Existen dos maneras de manipular la memoria de video: utilizando una matriz de acceso o un vector de mapeo.

Para poder representar la memoria de video, tal como se la imagina el usuario, es necesario utilizar una estructura de datos que permita accesarla como una matriz (80 x 25). Sabemos que cada posición de la pantalla está formada por dos bytes (carácter y atributo), de tal forma que se puede definir un elemento que tenga dos campos:

```
typedef struct {
    unsigned char ch;
    unsigned char atributo;
} letra;
```

Ahora se el tipo *letra* para definir una matriz que permita manipular cada carácter de la pantalla y sus atributos.

```
typedef letra matriz[25][89];
```

Con esto se tiene un nombre de tipo que puede mapear una matriz de 25 x 80 de dos bytes cada uno. Si se define una variable como un apuntador far a este tipo de matriz, se tendrá una representación más adecuada.

La siguiente función permite colocar una carácter con ciertos atributos en una posición de renglón-columna conocidas en la memoria de video.

```
#include "video.h"
/*  Escribe un carácter en la memoria de video dados un renglón,
columna y atributos */
void caracter(unsigned char c, unsigned char atributo, int ren,
int col) {
    matriz far *pantalla;     int VideoSeg;
    if (*(char far *)0x00400049 == 7)
        VideoSeg = 0xb000;
    else
        VideoSeg = 0xb800;
    pantalla = (matriz far *) MK_FP(VideoSeg,0x0000);
    (*pantalla)[ren][col].ch = c;
    (*pantalla)[ren][col].atributo = (char)atributo;
}
```

La declaración

```
matriz far *pantalla;
```

indica un tipo especial de apuntador a carácter. El calificador far nos dice que el apuntador no va a ser ordinario (de 2 bytes), sino que va a tener 4 bytes. Este modelo de apuntador es propio de Turbo C, por lo que no es estándar.

Observe que para conocer el tipo de tarjeta que se tiene (lo cual determina la dirección de inicio de la memoria de video), se utiliza la estructura de decisión

```
    if (*(char far *)0x00400049 == 7)
        VideoSeg = 0xB000;
    else
        VideoSeg = 0xB800;
```

En la dirección 0040:0049 se almacena el tipo de la tarjeta que tiene la computadora; si se trata de un monitor monocromático, se asignará al segmento el valor hexadecimal de B000H (ver mapa de

memoria de video); en caso contrario, se asume que el monitor es a color, asignándose al segmento el valor de B800H.

Con 4 bytes es posible direccionar de manera absoluta toda la memoria de la PC, por lo que la expresión

```
pantalla = MK_FP (VideoSeg,0x0000);
```

permite armar un valor que represente la dirección de la memoria real.

MK_FP (MaKe Far Pointer) es una macro que se encuentra definida en dos.h, de tal forma que si se da un segmento y offsets adecuados, se puede formar un apuntador a cualquier parte de la memoria.

La segunda forma de acceso a memoria de video es el de mapeo por vector; se trata de una representación un poco más complicada, debido a que se tiene que realizar un mapeo de la matriz que representa a la pantalla al vector que utiliza realmente la computadora.

A continuación se describe un función que permite usar la memoria de video para desplegar cadenas de caracteres, utilizando esta segunda forma de acceso:

```
/* Función Put String */
void PutS(char *String, int Attr, int Row,int Col) {
    register int Atributo;
    char far *Video;     /* Declara un apuntador de 4 bytes */
    int VideoSeg,        /* Segmento de la memoria de video */
    VideoOff;            /* Offset de la memoria de video   */

    Atributo = Attr;
    if (*(char far *)0x00400049 == 7)
        VideoSeg = 0xb000;
    else
        VideoSeg = 0xb800;
    VideoOff = Row * 160 + Col * 2;
    Video = MK_FP (VideoSeg,VideoOff);
    while (*String) {          /* Mientras no sea fin de cadena */
        *Video++ = *String++; /* Coloca carácter en pantalla   */
        *Video++ = Atributo;  /* Coloca atributo del carácter  */
    }
}
```

La expresión

```
VideoOff = Row * 160 + Col * 2;
```

determina la posición inicial (offset) en donde se debe colocar el primer carácter de la pantalla, según el algoritmo de mapeo de matriz a vector.

Usando este principio, se pueden accesar importantes áreas de memoria y realizar actividades particulares que a las funciones estándares les llevaría más tiempo.

Existen dos macros definidas en C, para el manejo de apuntadores de tipo far

```
C = peek(int segmento, int offset)
poke(int segmento, int offset, char C)
```

peek permite leer un carácter (C) en la dirección dada por el segmento y el offset, y poke permite almacenar un carácter (C) en la dirección dada por el segmento y offset.

Como ejercicio, podría ser modificar la función PutS para emplear la macro poke.

Gráficos

Muy de moda se ha puesto en el ámbito de la programación comercial las llamadas interfases gráficas, elegantes y versátiles. Sin embargo, realizar una aplicación con interfase gráfica no es fácil. Debido a que no existe un estándar en las tarjetas manejadoras de video, las aplicaciones deben ser capaces de detectar los diferentes modos gráficos y así poder configurarse automáticamente.

Una buena aplicación con interfase gráfica tiene todos sus módulos manejando coordenadas relativas; esto es, no importa el número de pixeles del modo de video ya que se adaptarán a las dimensiones de éste.

Antes de que cualquier función gráfica sea usada, la computadora debe estar en el modo de video apropiado. Para el IBM PC significa seleccionar el modo y la paleta apropiados.

La IBM PC cuenta con varios modos de video diferentes:

Modo	Tipo	Dimensiones	Adaptadores
0	texto, b/n	40x25	CGA, EGA
1	texto, 16 colores	40x25	CGA, EGA
2	texto, b/n	80x25	CGA, EGA
3	texto, 16 colores	80x25	CGA, EGA
4	gráficos, 4 colores	320x200	CGA, EGA
5	gráficos, 4 tonos grises	320x200	CGA, EGA
6	gráficos, b/n	640x200	CGA, EGA
7	texto, b/n	80x25	monocromo
8	gráficos, 16 colores	160x200	PCjr
9	gráficos, 16 colores	320x200	PCjr
10	gráficos, 4 colores PCjr, 16 colores EGA	640x200	PCjr, EGA
13	gráficos, 16 colores	320x200	EGA
14	gráficos, 16 colores	640x200	EGA
15	gráficos, 4 colores	640x350	EGA

EGA permite mayor resolución en otros modos, este modo es usado porque puede trabajar con ambos adaptadores gráficos (CGA y EGA). Para utilizar el modo EGA, sólo se tiene que modificar la función que escribe un punto.

La interrupción 16 del BIOS, función 0 selecciona el modo de video y es usada por la función modo() que se muestra a continuación.

```
/* selecciona el modo de video */
void modo(int codigo_modo) {
    union REGS r;
    r.h.al = codigo_modo;
    r.h.ah = 0;
    int86(0x10, &r, &c);
}
```

Dos tipos de paletas son permitidas en el modo de gráficos 4. La paleta determina que cuatro colores son visualizados. Sobre el IBM PC, la paleta 0 tiene los colores rojo, verde y amarillo; la paleta 1 da blanco, magenta y el cyan. Para cada paleta, el cuarto color es el color de fondo, usualmente el negro. La interrupción 16, función 11 del BIOS inicializa la paleta. La función paleta(), mostrada aquí, selecciona la paleta especificada en su argumento.

```
/* selecciona la paleta */
void paleta(int nump) {
    union REGS r;
    r.h.bh = 1;     /* código del modo de gráficos 4 */
    r.h.bl = nump; /* numero de paleta              */
    r.h.ah = 11;    /* función poner paleta          */
    int86(0x10, &r, &r);
}
```

La rutina fundamental de gráficos es la que dibuja un pixel, la unidad más pequeña direccionable de la pantalla de un monitor. Sin embargo, para nuestro propósito el término pixel será usado para describir el más pequeño punto direccionable en un modo gráfico específico. Debido a que la función que escribe un pixel será usada por otras rutinas de mayor nivel, su eficiencia es muy importante a la hora de que las funciones gráficas operen con rapidez. En el IBM PC y compatibles, hay dos maneras para escribir

La función que dibuja un pixel directamente en la memoria de video debe tener en cuenta una característica bastante interesante de la función de la ROM-BIOS que realiza la misma función: si esta interrupción es llamada con el séptimo bit (el más significativo) del código de color puesto a 1, entonces al color especificado se le hace un XOR con el color existente en la localización especificada, en lugar de simplemente sustituir al color anterior. La ventaja de esta característica será explicada con mayor detalle en ejemplos posteriores. Debido a que cada byte contiene cuatro pixeles, se debe preservar el valor de los otros tres cuando se cambie el valor de uno. La mejor forma de hacer esto es crear una máscara de bits con todos los bits encendidos, excepto aquellos en los que la localización del pixel deba ser cambiada. Este valor se opera utilizando un AND lógico con el byte original y un OR con la nueva información. Sin embargo, la situación es algo diferente si lo que se quiere es hacer una XOR entre el nuevo valor y el antiguo; en este caso, simplemente se tiene que hacer una OR del byte original y 0, y después hacer una XOR del resultado con el nuevo color.

La dirección de un byte se obtiene multiplicando primero la coordenada X por 40 y asignándole después el valor de la coordenada Y dividida entre 4. Para determinar qué pixel está en el banco de memoria par y cuál en el impar, sólo hay que ver el residuo de dividir la coordenada X entre dos. Si es 0, el número es par y el primer banco es utilizado; en el caso de que el número fuera impar se utilizaría el segundo banco. Los bits del byte son calculados mediante una división entera entre 4. El resto de la división será el número, en paquetes de dos bits, que contiene la información para el pixel deseado. Las operaciones de desplazamiento de bits son usadas para fijar el byte del código de color y el bit de máscara en sus propias posiciones. Aunque las manipulaciones de bits que se usan en la función mempunto() pueden resultar difíciles

información en un pixel. El primer método, a través del uso de las interrupciones de la ROM-BIOS, es la manera más fácil pero también la más lenta, de hecho demasiado lenta para nuestros propósitos.

La segunda y más rápida es colocar la información directamente en la memoria RAM de vídeo; este método es el que examinaremos a continuación. La memoria RAM de video del CGA siempre comienza en la dirección B800:0000. El EGA usa la misma dirección para esos modos y son compatibles con los modos CGA (para una descripción completa del hardware de video, consulte el manual Referencia de IBM). En el modo 4, cada byte contiene la información de cuatro pixeles, usando cada pixel dos bits. Por tanto, se necesitan 116K para una resolución de 320 x 200. Debido a que dos bits pueden tener sólo cuatro valores diferentes, sólo puede haber cuatro colores en el modo 4. El valor de cada paquete de dos bits determina el color que es visualizado de acuerdo con esta tabla:

Valor	Color de la paleta 0	Color de la paleta 1
0	color de fondo	color de fondo
1	amarillo	azul
2	rojo	púrpura
3	verde	blanco

Una característica extraña de la tarjeta EGA es que los pixeles pares se almacenan en memoria comenzando en la dirección B800:0000, mientras que los impares se almacenan 2000h bytes más altos (8152 en decimal), en la B800:2000. Cada fila de pixeles requiere 80 bytes: 40 para los pares y 40 para los impares. En cada byte, los pixeles son almacenados de izquierda a derecha de acuerdo a como aparecen en la pantalla. Esto significa que el pixel número 0 ocupa los bits 6 y 7, mientras que el pixel número 3 usa los bits 0 y 1.

de entender, resultan muy comunes en programas que utilizan características de bajo nivel; por ello, es recomendable que estudie el siguiente ejemplo con cuidado.

```
/* Escribe un punto directamente en el CGA/EGA. */
void puntomem(int x,int  y, int codigo_color) {
    union máscara {
        char c[2];
        int i;
    } bit_mascara;
    int i, index, posicion_bit;
    unsigned char t;
    char xor; /* hace un xor con color o lo reemplaza */
    /* apuntador a la memoria de CGA */
    char far *ptr *(char far *) 0xB8000000;
    bit_mascara.i= 0xFF3F; /* 11111111 00111111 en binario */
    /* comprueba el rango para el modo 4 */
    if (x < 0 || x > 199 || y < 0 || y > 319) return;
    /* ve si el modo xor esta activado */
    xor = codigo_color & 128;
    codigo_color = codigo_color & 127; /* quita la mascara del
                                          bit más alto */
    /* pone bit_mascara y los bits codigo_color
       en la posición correcta */
    posicion_bit = y % 4; /* calcula su localización
                             dentro del byte */
    codigo_color <<= 22 * posicion_bit ; /* desplaza el
                                            codigo_color a su
                                            posición */
    bit_mascara.i >>= 2*posicion_bit; /* despláza el bit_mascara
                                         a su posición */
```

```
    /* encuentra el byte correcto en la memoria de pantalla */
    index = x*40 +(y/4);
    if(x % 2) index += 8152; /* si es par usa el 2 banco */
    /* escribe el color */
    if (!xor) { /* modo de sobre escritura */
        t = *(ptr+index) % bit_mascara.c[0];
        *(ptr+index) = t | codigo_color;
    } else { /* modo xor */
         t = *(ptr+index) | (char) 0;
         *(ptr+index) = t ^ codigo_color;
    }
}
```

Note que el apuntador a la memoria de video es declarado como far; esto es necesario si usted está compilando con un modelo de datos pequeño. Usted debería darse cuenta también de que el modo especial de escritura, XOR, también disponible en la función de la ROM-BIOS, ha sido preservado en mempunto().

La siguiente función, una rutina gráfica fundamental, es usada para dibujar una línea con un color determinado especificando las coordenadas de los extremos de la línea. Aunque es realmente fácil dibujar líneas horizontales y verticales, es más difícil crear una función que dibuje líneas en diagonal. Por ejemplo, para dibujar una línea desde la posición (0,0) a la (80,120). ¿Cuáles son los puntos de la recta entre esas coordenadas?

Un primer paso para crear esta función es usar la relación entre el cambio de las dimensiones de X e Y. Para ver esto, consideremos una línea desde la coordenada (0,0) a la (5,10). El cambio en X es de 5 y en Y de 10. La relación es 1/2 y se usa para determinar la relación de cómo cambian las coordenadas X e Y a 10 largo de la línea. En este caso lo que significa es que la

coordenada X es incrementada la mitad de veces que la coordenada Y. Aunque este método es fácil de entender, tiene la desventaja de que debe usar variables de punto flotante. Esto significa que la función que dibuja líneas se ejecutará lentamente a menos que tenga instalado un coprocesador matemático, como el 8087. Por esta razón, es raramente usado.

El mejor método usado para dibujar una línea es conocido como algoritmo de Bresenham. Aunque está basado conceptualmente en las relaciones entre las distancias X e Y, no se requiere ningún cálculo de punto flotante. En su lugar, la relación entre el cambio en los valores de X e Y se maneja implícitamente a través de suma y resta de series. La idea básica sobre la que se apoya este algoritmo es la de ir registrando en cada punto el error entre donde se encuentra el punto y su posición ideal. El error entre la posición real y la ideal es debido a las limitaciones del hardware (el hecho de que la pantalla no tenga una resolución infinita). En cada iteración, las variables llamadas xerr e yerr se incrementan por los cambios en la magnitud de las coordenadas X e Y, respectivamente. Cuando el error llega a un límite predeterminado, se coloca en cero y el contador de la coordenada se incrementa. Este proceso continúa hasta que la línea se dibuja completamente. La función línea( ), mostrada a continuación, implementa este método. Note que usa la función mempunto(), desarrollada antes, para dibujar un punto en la pantalla.

```
/* Dibuja una línea usando el algoritmo de Bresenham. */
void línea(int comienzax,int comienzay,int finx,int finy,int color)
{
    register int t, distancia;
    int xerr=0, yerr=0, delta_x, delta_y;
    int incx, incy;

    /* calcula las distancias en ambas direcciones */
    delta_x = finx - comienzax;
    delta_y = finy - comienzay;
    /* Calcula la dirección del incremento, un incremento de 0
       significa una línea vertical u horizontal */
    if (delta_x > 0)
         incx = 1;
    else
         if (delta_x == 0)
              incx = 0;
         else
              incx = -1;
     if (delta_y > 0)
           incy = 1;
     else if(delta_y == 0)
           incy = 0;
     else
           incy = -1;
     /* determina que distancia es mayor */
     delta_x = abs(delta_x);
     delta_y = abs(delta_y);
     if (delta_x> delta_y)
         distancia = delta_x;
     else
         distancia = delta_y;
     /* dibuja la línea */
     for(t=0; t != distancia + 1; t++) {
         puntomem(comienzax, comienzay, color);
         xerr += delta_x;
         yerr += delta_y;
         if (xerr> distancia) {
              xerr -= distancia;
              comienzax + incx;
         }
         if (yerr > distancia){
              yerr -= distancia;
              comienzay += incy;
         }
    }
}
```

A partir de estas dos funciones es posible realizar una librería de graficación para realizar operaciones como dibujo de círculos, polígonos, gráficas de barras, etc. Es posible, además combinar las interfases gráficas con manejo de ratón y diseñar programas de gran calidad.

Puerto serie

Otro ejemplo interesante es la posibilidad de establecer comunicación entre dos computadoras por el puerto serie (RS-232).

La interrupción que permite realizar la comunicación por el puerto serie es la 20 (14 hexadecimal) y tiene 4 servicios:

servicio	descripción
0	Inicializa el puerto
1	Envía carácter
2	Recibe carácter
3	Obtiene el status

```
/* Función para inicializar el puerto serie */

#include <dos.h>

int InicializaPuerto(char Parametro,int NumPuerto)
{
    int status;

    _AH = 0;          /* servicio 0, inicializa puerto  */
    _AL = Parametro;  /* Parámetros del puerto serie    */
    _DX = NumPuerto;  /* Numero de puerto (se usa 0)    */
    geninterrupt(20); /* también geninterrupt(0x14);    */

    status = _AX;
    return(status);
}
```

La variable parametro, permite establecer las características de la comunicación entre dos terminales:

Bit 7 6 5 4 3 2 1 0	Uso
x x x	Velocidad de trasmisión
. . . x x . . .	Código de paridad
. . . . . x . .	Código de bit de paro
. . . . . x x	Tamaño de carácter

A continuación se muestran las diferentes combinaciones para cada punto de la tabla.

Velocidad de transmisión		
Bit encendido 7 6 5	Valor	Bits por segundo
0 0 0	0	110
0 0 1	1	150
0 1 0	2	300
0 1 1	3	600
1 0 0	4	1,200
1 0 1	5	2,400
1 1 0	6	4,800
1 1 1	7	9,600

Código de paridad			
Bit 4	Bit 3	valor	significado
0	0	0	Ninguna
0	1	1	Paridad Non
1	0	2	Ninguna
1	1	3	Paridad Par

Código de bit de paro		
Bit 2	valor	significado
0	0	1 bit de paro (stop)
1	1	2 bits de paro

Tamaño de carácter			
Bit 4	Bit 3	valor	significado
0	0	0	No usado
0	1	1	No usado
1	0	2	7 bits
1	1	3	8 bits

Existen sólo 128 caracteres ASCII estándar, por lo que algunas máquinas sólo transmiten a este tamaño (7 bits). Sin embargo, en la PC es más usado el código de 8 bits (código ASCII extendido).

```
/* Función para enviar un carácter por el puerto serie */
#include <dos.h>
int EnviaCaracter(char C, int NumPuerto) {
    int status;
    _AH = 01;
    _AL = C;
    _DX = NumPuerto;
    geninterrupt(0x14);
    status = _AH;
    return(status);
}
```

```
/* Función para recibir un carácter por el puerto serie */
#include <dos.h>
int RecibeCaracter(char *C,int NumPuerto) {
    int status;

    _AH = 02;
    _DX = NumPuerto;
    geninterrupt(0x14);
    status = _AH;
    *C = AL;
    return(status);
}
```

```
/* Función para obtener el status del puerto serie */
#include <dos.h>
int ObtieneStatus(int NumPuerto) {
    int status;
    _AH = 03;
    _DX = NumPuerto;
    geninterrupt(0x14);
    status = _AX;
    return(status);
}
```

Ahora bien, es posible modificar la primera función para que el parámetro no sea armado por el usuario, simplemente dando los valores correspondientes:

```
/* Función para inicializar el puerto serie, versión 2*/
#include <dos.h>
int InicializaPuerto(int Bauds,     /* velocidad de transmisión */
                     int paridad,   /* código de paridad        */
                     int paro,      /* código de bits de paro   */
                     int tam,       /* tamaño de carácter       */
                     int NumPuerto)
{
    int status;
    char Parámetro;

    Parametro = ((Bauds & 7) << 5);
    Parametro |= ((paridad & 3) << 3);
    Parametro |= ((paro & 1) << 2);
    Parametro |= (tam & 3);

    _AH = 0;           /* servicio 0, inicializa puerto  */
    _AL = Parametro;   /* Parámetros del puerto serie    */
    _DX = NumPuerto;   /* Numero de puerto (se usa 0)    */
    geninterrupt(20); /* también geninterrupt(0x14);     */

    status = _AX;
    return(status);
}
```

También se puede usar un archivo de declaraciones:

```
#include <dos.h>

/* definiciones para velocidad de transmisión */

#define  BPS9600   7
#define  BPS4800   6
#define  BPS2400   5
#define  BPS1200   4
#define  BPS600    3
#define  BPS300    2
#define  BPS150    1
#define  BPS110    0


/* definiciones para código de paridad */

#define  PAR       3
#define  NON       1
```

```
/* definiciones para código de paro */
#define PARO1       0
#define PARO2       1

/* definiciones para tamaño de carácter */

#define ASCII7      2
#define ASCII8      3

int InicializaPuerto( int, int, int, int, int );
int RecibeCaracter( char *, int );
int EnviaCaracter( char, int );
int ObtieneStatus( int );

/* fin de archivo serie.h */
```

Para conocer el significado de la variable status, se presenta la siguiente tabla de referencia:

Status de línea para el registro AH

Bit 7	6	5	4	3	2	1	0	Significado
1	.	.	.	.	.	.	.	Time-out error
.	1	.	.	.	.	.	.	Transfer shift register empty
.	.	1	.	.	.	.	.	Transfer holding register empty
.	.	.	1	.	.	.	.	Break-detect error
.	.	.	.	1	.	.	.	Framing error
.	.	.	.	.	1	.	.	Parity error
.	.	.	.	.	.	1	.	Overrun error
.	.	.	.	.	.	.	1	Data ready

Status de modem para el registro AL

Bit 7	6	5	4	3	2	1	0	Significado
1	.	.	.	.	.	.	.	Received line signal detect
.	1	.	.	.	.	.	.	Ring indicator
.	.	1	.	.	.	.	.	Data-set-ready
.	.	.	1	.	.	.	.	Clear-to-send
.	.	.	.	1	.	.	.	Delta received line signal detect
.	.	.	.	.	1	.	.	Trailing-edge ring detector
.	.	.	.	.	.	1	.	Delta data-set-ready
.	.	.	.	.	.	.	1	Delta clear-to-send

Manejo de ratón

Uno de los dispositivos periféricos más populares hoy en día es el ratón (mouse). Con el desarrollo acelerado de interfases gráficas se ha necesitado que el usuario sienta más transparente la interacción con la máquina (la filosofía moderna en el ámbito de la programación es que el ambiente de trabajo sea lo más amigable posible). El ratón ha sido un gran auxiliar en ese aspecto.

El ratón fue inventado hace casi treinta años por Douglas Engelbart, para el diseño de las computadoras Star de Xerox, Apple Macintosh y Lisa, y más tarde en el diseño de poderosas estaciones de trabajo.

En el diseño original de las computadoras personales (PC) no se comprendía la inclusión de este dispositivo. Charles Simonyi, un ex-empleado de Xerox, trabajando para la empresa Microsoft fue quién sugirió el uso del ratón para el paquete Microsoft Word. Microsoft diseñó entonces un ratón pequeño, fácil de utilizar y con bajo consumo de energía. Fue puesto a la venta en junio de 1983 y se le conoció como el ratón Microsoft de primera generación. Era color crema con botones verdes, y el mecanismo deslizante era un balín de acero puro con una resolución de 100 ppi (puntos por pulgada). Actualmente el ratón Microsoft es el más estándar, ya que solo ha sufrido cambios en su exterior, mientras que su interfase de programación ha permanecido casi sin cambios, permitiendo la compatibilidad del software con versiones más recientes de dicho ratón.

Casi todos los fabricantes de ratones se están haciendo compatibles con el de Microsoft; por esta razón, los ejemplos que se presentan aquí estarán basados en ese producto.

Para poder usar el ratón Microsoft es indispensable contar con el programa manejador del ratón (Mouse Driver); una versión 2.11 o mayor de DOS y un compilador de lenguaje que soporte llamadas directas al sistema operativo a través de interrupciones (como Turbo C).

Con las funciones de manejo de ratón se podrá esconder o desplegar el cursor (en modo texto o gráfico); detectar qué botón ha sido presionado, así como cuántas veces; saber en qué posición se encuentra; su desplazamiento relativo,etc. Todas estas funciones están presentes en el manejador del ratón (MOUSE.SYS), y son llamadas al ejecutar la interrupción 033h con diferentes valores como argumentos. Estas llamadas se hacen en forma muy similar a las llamadas al BIOS o al sistema operativo DOS. El ratón permite incluso definir algunas interrupciones como parte de su programación.

Una función sirve para activar el cursor del ratón. Algunos adaptadores de video solamente funcionan en modo texto o tienen modos gráficos no soportados por IBM (como MDA y Hércules). Sin embargo, las consideraciones de programación para ambos tipos de adaptadores son muy similares.

El ratón Microsoft tiene dos tipos de cursor por default: el rectangular (para modo texto) y el apuntador (que es una flecha y se activa en modo gráfico). Sin embargo, es posible sustituirlo por uno propio o por ninguno, según sea el caso. Para el ratón existe una *pantalla virtual* sobre la cual está desplegado el cursor correspondiente. La pantalla virtual simplifica además la programación para los diferentes adaptadores, ya que su resolución es independiente del adaptador. Se puede pensar que es una especie de rejilla subdividida en posiciones y se sobrepone a la pantalla normal. Un programa que usa el ratón solamente requiere conocer las

coordenadas de esta pantalla virtual. El tamaño mínimo de una pantalla virtual es de 640 x 200 puntos (para una pantalla de texto cada columna y renglón ocupan 8 puntos). El manejador del ratón intercepta todas las llamadas que pudieran modificar el modo de video, ajustándose automáticamente a los cambios de resolución o al adaptador empleado. Independientemente del tipo de cursor que se está desplegando en la pantalla, el manejador del mouse tiene una bandera o indicador interno para determinar si el cursor deberá aparecer en la pantalla o no. Esta bandera siempre valdrá cero o menos. Cuando vale cero, el ratón despliega el cursor. Dado que el ratón es un objeto, los programadores no tienen acceso directo a esta bandera, en su lugar se proveen algunas funciones para su manejo.

Por último, el movimiento del ratón se traduce en valores que expresan directamente la dirección y duración del mismo. Estos valores se dan en unidades de distancia denominadas "mickeys", y cada una tienen un valor aproximado de 1/200 de pulgada. El manejador del ratón utiliza la cuenta de mickeys para desplazar el cursor por la pantalla.

Programas Residentes

Un concepto aparentemente simple, como es la creación de programas residentes, es realmente uno de los trabajos de programación más difíciles en el entorno DOS.

Intrínsecamente unido al desarrollo y uso de programas residentes está la modificación del vector de interrupciones, por lo que los programas ejemplo que se verán pueden no funcionar bien con compiladores diferentes.

Un programa residente termina su ejecución mediante una llamada a la función 49 del DOS, que causa la salida a DOS mientras el propio programa permanece en memoria. Con este método, el programa puede activarse instantáneamente sin tener que ser cargado de nuevo. Uno de los ejemplos de programas residentes más conocidos es el Sidekick de Borland.

La mayoría de los programas residentes se activan con el uso de una interrupción, que puede ser generada de varias formas. Las interrupciones más comunes son la de reloj, teclado e imprimir pantalla. Un ejemplo muy famoso de un programa residente que emplea la interrupción de reloj es el llamado "virus de Turín" o de la "pelotita".

Cuando se produce una interrupción, ésta hace que todas las interrupciones sean inhabilitadas. La rutina de servicio de interrupciones debe habilitar de nuevo las interrupciones sobre la entrada para evitar la caída del sistema. La rutina de servicio de interrupciones debe acabar con una instrucción IRET. Una característica importante de DOS es que no es reentrante, es decir

mientras está siendo usado por un programa, el DOS no puede ser usado por un segundo programa (es una de las razones por las que DOS no es multitarea). Así pues, la rutina de servicio de interrupciones no puede usar cualquier función del DOS, y si intenta hacerlo puede causar la caída del sistema. Por tanto, deberá adaptarse para controlar directamente muchas funciones para las cuales normalmente se habrían usado llamadas al DOS.

La BIOS permite cierta reentrada. Por ejemplo, la interrupción 16, entrada de teclado, posible utilizarse sin efectos colaterales. Otros servicios, sin embargo, pueden no resultar seguros; desgraciadamente, el único camino para averiguarlo es a través de la experimentación. Si una función no puede ser usada, seguramente el sistema quedará paralizado; por esta razón se debe tener cuidado especial al escribir programas residentes.

Debido a que muchas de las funciones de biblioteca estándar del C usan el DOS o el BIOS, muchas veces es necesario sustituirlas por funciones equivalentes que no utilicen llamadas al DOS o al BIOS. Hay que tener presente que las funciones de E/S no son las únicas que utilizan el DOS o el BIOS. Por ejemplo, la función de asignación en memoria malloc() normalmente utiliza una llamada al DOS para determinar cuánta memoria libre está disponible en el sistema.

Esta es la principal razón de por qué los programas residentes son tal difíciles de escribir y transportar a nuevos entornos y por qué son creados tan pocos programas residentes a pesar de ser tan populares.

A pesar de que no es parte del estándar ANSI, Turbo C incluye un modificador especial llamado "interrupt" que permite a una función de C ser usada como una rutina de servicio de interrupción (la mayor parte de los fabricantes de compiladores de C incluirán probablemente este modificador en sus siguientes versiones porque es una importante ampliación). Por ejemplo, supongamos que la función una función test() que va a ser utilizada como una interrupción. La definición de la función debe incluir al calificador interrupt:

```
void interrut test(bp, di, si, ds, es, dx, cx, bx,
                   ax, ip, cs, bandera)
unsigned bp, di, si, ds, es, dx, ax, ip, cs, bandera;
{
 ...
 ...
 ...
}
```

Una función interrupt salva automáticamente el contenido de todos los registros y los restituye antes de regresar. La función usa la instrucción IRET en vez de la instrucción usual RET para regresar al punto en que fue llamada.

Para los ejemplos, el modificador interrupt es aplicado sólo a aquellas funciones que sirven como punto de entrada a la rutina de servicio de interrupciones de los programas residentes.

Si un compilador no soporta los modificadores interrupt, es necesario escribir un pequeño módulo en ensamblador que salve todos los registros, rehabilite las interrupciones y llame a la función apropiada. Para salir, debe usar una IRET. La forma de crear una función en ensamblador varía para cada compilador.

Todos los programas residentes se escriben en dos partes. La primera se usa para inicializar el programa y debe salir mediante una llamada a la función 49 del DOS. Esta parte del programa no se vuelve a ejecutar hasta que el programa residente deba ser cargado nuevamente.

La segunda parte del programa es la aplicación real. Casi sin excepción estas aplicaciones usan ventanas de modo tal que la pantalla no se modifique cuando termine la aplicación. Recuerde que la mayor parte de las aplicaciones residentes son utilerías como calculadoras y editores, los cuales deben dejar la pantalla igual que se la encontraron (VideoPush, VideoPop).

Una de las interrupciones más utilizadas para activar programas residentes es la número 5. Esta interrupción es llamada cuando se pulsa la tecla PRINT SCREEN. Si se está dispuesto a renunciar a la función de imprimir pantalla, se puede sustituir la dirección de esta rutina con la dirección de la aplicación residente. De esta forma, cuando se pulse la tecla PRINT SCREEN, la aplicación residente será activada.

Una forma general para la parte de inicialización de una aplicación residente usando la tecla PRINT SCREEN puede ser la siguiente:

```
void interrupt Residente(); /* Función residente */
main()
{
     struct direccion {
        char far  *p;
     };

/* dirección de la interrupción */
struct direccion far *addr = (struct direccion far *) 20;

addr->p = (char far *) Residente;
termina(2000); /* termina pero queda residente */
}
```

El programa residente debe cambiar la dirección de la interrupción 5 para apuntar a la función especificada por el programa residente. Hay varias formas de cambiar una dirección en la tabla de vectores de interrupción. Una de ellas es usar una llamada al DOS. La dificultad con las funciones DOS es que requieren que el segmento de direcciones esté en el registro ES, el cual no se puede modificar con int86(). Algunos compiladores, como Turbo C, disponen de funciones especiales para obtener o seleccionar una dirección en la tabla de interrupciones. Sin embargo, el método mostrado aquí debería trabajar prácticamente con cualquier compilador. La función Residente() es el apuntador de entrada en la aplicación residente. Usa un apuntador a la dirección de la interrupción 5 de la tabla de vectores. Recuerde: la dirección de la interrupción 5 está en 20 (4x5), ya que todas las direcciones tiene 4 bytes. Algunos programas residentes tienen una opción que reestablece las direcciones previas de la interrupción.

Finalmente, main() sale mediante una llamada a termina(), mostrada a continuación.

```
/* termina pero permanece residente */
termina(unsigned tamaño);
{
    union REGS r;
    r.h.ah = 49;  /* termina y permanece residente */
    r.h.al = 0;   /* codigo de retorno */
    r.x.dx = tamaño;
    int86(0x21, &r, &r);
}
```

El parámetro tamaño, especificado en DX, se usa para informar al DOS de cuánta memoria requiere el programa residente (se especifica en bloques, un bloque tiene 16 bytes). A veces es difícil determinar la cantidad de memoria que necesita nuestro programa, pero puede aproximarse si se divide el tamaño del archivo ejecutable (.EXE) entre 16 y lo multiplica por 2. La cantidad exacta de memoria es difícil de determinar porque los archivos .EXE están parcialmente ligados cuando se cargan en memoria y no se colocan necesariamente en áreas contiguas. Una forma de calcular este parámetro es experimentando. El código de retorno especificado en AL es devuelto al sistema.

Después de que sea ejecutado main(), el programa está ya en la memoria RAM y no puede ser sobrescrito por otro programa. Esto significa que la aplicación está lista para ser ejecutada presionando la tecla PRINT SCREEN. Para la aplicación residente, es importante recordar que si se vuelve a presionar la tecla PRINT SCREEN se invocará nuevamente, por lo que es necesario tener una variable estática que permita mantener el estado de invocación y así evitar ese problema.

La interrupción imprimir pantalla es fácil de utilizar, pero tiene tres grandes desventajas. Primero, permite sólo a una aplicación residente estar presente en el sistema. Segundo, no podrá imprimir pantallas. Un camino mejor para activar un programa residente es usar la interrupción 9, la interrupción de tecla pulsada. Cada vez que se pulsa una tecla en el teclado, se ejecuta la interrupción 9.

Para usar la interrupción 9 para activar un programa residente, se requiere la siguiente aproximación básica. En primer lugar, se necesita mover la dirección actual en la tabla de vectores asociada con la interrupción 9 a una interrupción no utilizada por el DOS. Se podría usar la interrupción 60. A continuación, se colocará la dirección del punto de entrada del programa residente en la posición de la interrupción 9 en la tabla de interrupciones. El nuevo manejador de la interrupción 9 deberá invocar al antiguo controlador de entrada del teclado; después deberá checar el carácter leído para ver si es la "tecla clave" usada para activar la aplicación residente. Si es así, la aplicación residente es ejecutada; en caso contrario, no se realiza ninguna acción y el programa es desactivado. Por lo tanto, cada vez que se pulsa una tecla, se activa la función de entrada residente, pero la aplicación sólo se ejecuta si se pulsa la tecla adecuada.

Este método tiene dos ventajas: no se pierde funcionalidad y permite mantener varias aplicaciones residentes simultáneamente; además de que cada aplicación puede activarse con diferentes teclas clave.

Las versiones estándar del buffer del DOS permiten almacenar hasta 15 caracteres desde el teclado, lo que permite teclear por adelantado. Cada vez que una tecla se pulsa, se genera la

interrupción 9. La rutina de servicio de entrada de teclado lee el carácter desde el puerto y lo coloca en el buffer. Cuando se invoca a una función de lectura de teclado del DOS o del BIOS, sólo se examina el contenido del buffer, y no el puerto real. Es posible hacer que las rutinas residentes examinen el contenido del buffer del teclado de igual forma como lo hacen las rutinas del DOS y el BIOS; así permitirá a la función de entrada del programa residente determinar si ha sido pulsada una tecla clave o no sin tener que trasladar los caracteres desde el buffer.

El buffer del teclado está localizado en 0000:041E (1054 en decimal). Debido a que todas las teclas generan 16 bits, éste requerirá 30 bytes para los 15 caracteres. Sin embargo, se necesitan realmente 32 bytes, puesto que el código de la tecla de RETURN también se almacena al final del buffer.

El buffer está organizado como una cola circular, a la cual se accede a través de un apuntador de cabeza y un apuntador de cola. El apuntador de cabeza señala al último carácter tecleado. El apuntador de cola señala al siguiente caracter a enviar por un requerimiento de entrada del DOS o el BIOS. El apuntador de cabeza es almacenado en la posición 0000:041A (1050 en decimal), y el apuntador de cola en 0000:041C (1052 en decimal). Los valores de los apuntadores de cabeza y cola están realmente indexados en la cola, lo que significa que sus valores son índices de la posición actual más 30 (esto es posible por la forma en que el 8086 procesa el direccionamiento indirecto). Los valores de los apuntadores de cabeza y cola tienen el mismo valor cuando la cola está vacía.

Para la aplicación residente, se requiere una pequeña inicialización; una forma general podría ser la función main() mostrada a continuación.

```
main() {
        struct direccion {
            char far *p;
        } temp;

    /* apuntador a la direccion de la interrupción 9 */
    struct direccion far *addr = (struct direccion far *) 36;

    /* apuntador a la direccion de la interrupción 60 */
    struct direccion far *int9 = (struct direccion far *) 240;

    /* Desplaza la direccion de la rutina de interrupción de
       teclado a int 60.
       Si int 60 e int 61 contienen las mismas direcciones, el
       programa residente no ha sido instalado.
    */

    if (int9->p == (int9 + 1)->p) {
       int9->p = addr->p;
       addr->p = (char far *) Residente;
    }else{
       printf("programa residente ya inicializado \n");
       exit (1);
    }

    termina(2000);
}
```

Observe que en esta versión del programa residente, se previene de que sea instalado más de una vez. Esto es necesario porque una segunda instalación podría poner una copia del punto de entrada del programa residente en la posición de la tabla de vectores de la interrupción 60. La función trabaja checando para ver que los valores de la tabla de interrupciones sean los mismos para las interrupciones 60 y 61 (la interrupción 61 tampoco es utilizada). El DOS recorre todas las interrupciones no usadas para la misma rutina nula de interrupción. Por tanto, antes de que el programa residente haya sido instalado, estas direcciones pueden ser las mismas, pero después de la instalación serán diferentes.

La función residente de entrada es más compleja que la usada con el método que se presentó anteriormente (interrupción para imprimir pantalla). Debe generar primeramente una interrupción 60, puesto que la tecla es leída por la rutina de entrada estándar. La mayoría de los compiladores C tienen una función que genera una interrupción. En Turbo C es geninterrupt(), la cual es llamada con el número de interrupción que desea generar. Después de que la interrupción 60 retorne, su función debe examinar el contenido de la cola y, mediante el apuntador de cabeza, ver si la tecla clave fue presionada. Si es el caso, la tecla presionada debe ser leída para que la aplicación no la "vea". Se ocupa un variable global para prevenir que ambas aplicaciones se activen simultáneamente. Si una aplicación está activa, a la otra le es denegada la activación. Una ejemplo de una función Residente() se muestra a continuación.

```
/* Este es el punto de entrada al codigo de la aplicación
   residente
*/
void interrupt Residente(void) {
    char far *t = (char far *) 1050; /* dirección del apuntador
                                        de cabeza */
    geninterrupt(60); /* Ejecuta la interrupción del teclado
                         como interrupción 60 */
    if (*t != *(t+2)) { /* si no esta vacío */
         t += *t-30+5; /* avanzar a la disposición del caracter */
         if(*t == tecla1 || *t == tecla2) {
              bioskey(0); /* limpia las teclas F2/F3 */
              if (!ocupado) {
                   ocupado = !ocupado;
                   ... /* invocación a función, instrucciones,
                   ...  ejecutables, etc., usando el valor de
                   ...  tecla1 y tecla2 como disyuntiva */
                   ocupado = !ocupado;
              }
         }
    }
}
```

Observe que la condición

```
if (*t == tecla1 || *t == tecla2)
```

nos permite seleccionar entre dos teclas diferentes para efectuar actividades. Es posible, entonces tener la selección de más de dos teclas (pudiendo usarse un switch).

Actualmente, los programas residentes son de gran utilidad. Dadas las características de MS-DOS no es posible la multitarea, por lo que se recurre a los programas residentes para que efectúen actividades tales como monitoreo, spoolers de impresión, aplicaciones ejecutivas, capturadores gráficos, etc.

Turbo C user's guide.
Borland International, 1989

The C programming language
Kernighan/Ritchie
Prentice Hall, 1988

C Guía para usuarios expertos
Herbert Schildt
Mc Graw Hill

Systems Programming in Turbo C
Michael J. Young
Sybex, 1989

Programer's guide to the IBM PC
Peter Norton
Microsoft Press, 1986

Advanced MS-DOS
Ray Duncan
Microsoft Press, 1986

Local Area Network Arquitectures
David Hutchison
Addison-Wesley, 1988

Gráficas por computadora
Donald Hearn / Pauline Baker
Prentice Hall, 1988

Lenguaje C, Introducción a la programación
Kely / Pohl
Interamericana

Adaptadores de video

MONOCHROME DISPLAY ADAPTER (MDA)

El MDA (Adaptador de despliegue monocromático) es una tarjeta que generalmente viene integrada como equipo original de fábrica en las PC y que trabaja en conjunción con una unidad de despliegue monocromática. Este adaptador produce una señal de barrido de video menor que el de la televisión estándar, por lo cual no puede utilizarse directamente como controlador de video si se utiliza como monitor un televisor. El adaptador monocromático de IBM barre 50 veces por segundo la pantalla.

Este adaptador tiene un sólo modo de video, el cual únicamente permite el despliegue de textos. La pantalla puede desplegar hasta 80 caracteres horizontalmente y 25 caracteres verticalmente. Cada caracter está formado por una matriz de puntos de 9X14, lo cual produce textos de gran calidad fáciles de leer. Dado que utiliza un monitor monocromático sólo puede desplegar en un color, sin embargo, su modo de video en conjunción con los atributos de despliegue del monitor permiten en el manejo de textos características tales como: video inverso, subrayado y manejo de la "brillantez". El MDA sólo puede trabajar, directamente, con monitores de despliegue monocromáticos IBM o equivalentes. El monitor de despliegue monocromático es de color verde claro, aunque también existen algunos de color ámbar.

MODO	TIPO	COLOR	RESOLUCION VER.	RESOLUCION HOR.	DESCRIPCION
Unico (7)	Texto	No	25	80	Textos monocromáticos (con atributos de pantalla)

COLOR GRAPHICS ADAPTER (CGA)

El CGA (Adaptador Gráfico a Color) fue diseñado para permitir compatibilidad con televisores y monitores compuestos, los cuales en su tiempo eran los que más prevalecían. Esto limitaba la resolución vertical a 200 líneas distinguibles. La resolución horizontal fue diseñada para 320 columnas, siendo posible tener hasta 640 columnas en monitores RGB o compuestos.

El CGA permite trabajar tanto en modo texto como en modo gráfico manejando, en cada uno de ellos, las opciones de edición en color o sin él en varias resoluciones, con un total de siete modos diferentes. Puede operar con cuatro diferentes monitores de despliegue en contraste con el MDA que sólo puede manejar el monocromático. Uno de ellos, el cual es poco usado, es el monitor de televisor, la calidad de la imagen usando éste es pobre y su uso requiere de un dispositivo adicional conocido como modulador RF el cual transforma las señales del CGA a señales de televisión.

El monitor más recomendable para éste adaptador es el Monitor a Color de IBM(el cual es del tipo RGB) que proporciona una gran calidad de imagen. Los otros tipos de monitores que acepta este adaptador son del tipo compuesto. Ellos trabajan con la señal de

más baja calidad del CGA, por lo que no producen imágenes tan claras como un monitor RGB. Puede controlar monitores compuestos a color y monocromáticos; el monitor compuesto monocromático acepta señales de color y muestra los colores en forma de sombras o tonalidades del único color que maneje el monitor.

En modo texto se tiene disponibles 2 anchos (40 y 80 columnas). El modo de 40 columnas fue creado para hacer más legibles los textos cuando se usa un aparato de TV como monitor de despliegue; sin embargo, la mayoría de los programas escritos para PC no están diseñados para trabajar en modo de 40 columnas. Todos los modos texto tienen 25 líneas de caracteres en la pantalla. En modo gráfico existen 2 resoluciones: alta y media. El modo de alta resolución tiene 640 puntos a lo ancho, mientras que el modo de resolución media tiene sólo 320 puntos. En ambos casos se tienen 200 líneas de visualización. El CGA requiere de 16K de memoria para manejar 4 colores en modo de resolución media, sin embargo, con esta cantidad sólo puede manejar 2 colores en modo de alta resolución y uno de ellos deberá ser el negro.

MODO	TIPO	RESOLUCION HOR.	RESOLUCION VER.	COLOR	DESCRIPCION
0	Texto	40	25	No	Texto a 40 columnas monocromático.
1	Texto	40	25	16	Texto a 40 columnas con 16 posibilidades de color.
2	Texto	80	25	No	Texto a 80 columnas monocromático.
3	Texto	80	25	16	Texto a 80 columnas con 16 posibilidades de color.
4	Gráfico	320	200	4	Gráficas de resolución media con color.
5	Gráfico	320	200	No	Gráficas de resolución media monocromáticas.
6	Gráfico	640	200	2	Gráficas de alta resolución en blanco y negro.

HERCULES GRAPHICS CARD

La tarjeta gráfica HERCULES es esencialmente del mismo tipo del MDA, pero con la característica adicional del manejo de gráficos. Posee un modo especial de video para manejo de gráficas en alta resolución. Al igual que el MDA, HERCULES sólo trabaja con monitores monocromáticos o equivalentes. A pesar de que su modo gráfico no es un modo de despliegue estándar (según los canones de IBM), este adaptador ha sido bien recibido por los desarrolladores de software, de tal manera que los sistemas orientados a gráficas más importantes pueden trabajar perfectamente con esta tarjeta adaptadora.

El adapatador HERCULES puede reemplazar directamente al MDA en el mismo slot, esto es, en el que controla la unidad de despliegue, produciendo una imagen idéntica a la del monocromático. Sin embargo, requiere de otro slot adicional para conectarse a la interfase paralela de impresión.

En su modo gráfico se puede manejar una resolución de 720 puntos horizontales por 348 puntos verticales. Además, posee dos páginas gráficas. Dado que se puede escribir en una página mientras se despliega la otra, es posible realizar algunos tipos de animación.

La tarjeta HERCULES contiene un set de 15 primitivas gráficas, las cuales pueden ser llamadas desde BASIC, ensamblador u otro lenguaje de alto nivel. Usando estas primitivas como punto de partida se pueden escribir sofisticados programas para manipular las imágenes gráficas en la pantalla.

La siguiente tabla explica dichas primitivas:

PRIMITIVA	DESCRIPCION
GMODE	Accesa a modo gráfico.
TMODE	Accesa a modo texto.
CLRSCR	Limpia la pantalla actual de gráficos.
GPAGE	Habilita la página actual de gráficos para escritura,.pero no afecta la pantalla.
LEVEL	Habilita el nivel de intensidad para subsecuentes escrituras.
XOR	Aplica la función OR exclusiva a lo que esta en la pantalla.
DISP	Habilita la pantalla actualmente desplegada de gráficos.
PLOT	Escribe un pixel sobre la pantalla.
GETPT	Toma un pixel de la pantalla.
MOVE	Habilita un endpoint para ser usado en la línea de dibujo.
DLINE	Dibuja una línea desde el punto habilitado por el último MOVE hasta el punto actual.
BLKFIL	Llena un rectángulo de acuerdo con el patrón actual de intensidad.
TEXT	Coloca un caracter o caracteres en la pantalla.
ARC	Dibuja un cuarto de circulo.
CIRCLE	Dibuja un circulo.
FILL	Llena un polígono irregular.

La forma más fácil de accesar las primitivas es usando el "BASIC" de HERCULES (HBASIC). El programa llamado HBASIC no es un nuevo BASIC, este simplemente carga BASICA y modifica su mapa de memoria para así poder realizar las gráficas, siendo esta la única diferencia con respecto a la versión de BASIC de IBM (BASICA). Con respecto a los caracteres, HBASIC los forma a partir de una matriz de 9X14 puntos(BASICA lo hace con una de 8X8).

HERCULES incluye software para permitir la impresión de lo que se visualiza en pantalla, pero sólo trabaja con impresoras Epson equipadas con chips Graftrax.

Existe una versión plus de este adaptador, cuya principal diferencia, con respecto a la versión estándar es la posibilidad de manejar diferentes conjuntos de caracteres de edición en el modo texto. Esto lo realiza a partir de un hardware especial llamado RamFont. Anteriormente la única forma de accesar a un nuevo conjunto de caracteres de edición era a través de la instalación de un nuevo chip de ROM. RamFont viene disponible en dos formas: 4K y 48K bytes. En modo 4K un nuevo conjunto reemplaza al original. Este modo permite la compatibilidad con todo el software para manejo de textos y permite, por ejemplo, el uso de letras itálicas con símbolos matemáticos de procesadores de palabras. En modo de 48K, un programa permite el acceso simultáneo de 12 conjuntos de caracteres (itálico, script, medieval, hebreo, arábigo, etc).

Por otra parte, la versión plus permite variar las dimensiones de la matriz de puntos usada para representar los caracteres. La dimensión vertical puede variar desde 1 pixel hasta 14. La dimensión horizontal, sin embargo, está limitada a 8 o 9 pixeles. Por lo anterior, el número máximo de caracteres por línea es de 90.

MODO	RESOLUCION HOR.	VER.	COLOR	DESCRIPCION
Texto	80	25	No	Textos monocromáticos (con igual calidad a los de MDA).
Gráfico	720	348	No	Gráficas de alta resolución monocromáticas.

ENHANCED GRAPHICS ADAPTER (EGA)

El EGA (Adaptador Gráfico Realzado) combina todas las características del MDA, el CGA y el adaptador HERCULES, adicionando nuevos modos de video que pueden ser usados con monitores de color de alta calidad, tal como el monitor a color realzado de IBM, el cual es una versión avanzada de un monitor RGB. Igualmente puede manejar el monitor a color estándar de IBM y monitores monocromáticos. El EGA puede reemplazar directamente ambos adaptadores de IBM (MDA y CGA), sin embargo su costo es del doble. Este adaptador se ha convertido en el estándar para aplicaciones administrativas, ofreciendo una resolución de 640 por 350 pixeles (en forma estándar) con 16 posibilidades de color y textos de buena calidad visual.

Generalmente este adaptador viene con 256K bytes de memoria, los cuales son suficientes para implementar hasta 16 colores diferentes de los 64 de que disponen los monitores a color. Con esta expansión de memoria la EGA puede manejar hasta 2 páginas diferentes de gráficas a color en modo de alta resolución. Con el modo de baja resolución (o gráficas monocromáticas) puede manejar hasta 8 páginas de texto o gráficas. Esto hace posible que se despliegue una página mientras la otra empieza a ser modificada y

desplegarlas alternadamente. La memoria extra también puede ser usada para almacenar conjuntos de caracteres para edición creados por el usuario. Generalmente posee 2 conjuntos de caracteres instalados en ROM con un total de 1024 caracteres diferentes, 512 de los cuales pueden ser utilizados al mismo tiempo en modo texto. En modo gráfico sólo se pueden utilizar 256 caracteres.

La EGA tiene 5 modos texto y 7 gráficos. Los modos texto del 0 a 3 son idénticos al los modos 0-3 de CGA y el modo texto 7 es prácticamente el mismo modo(7) del MDA. Igualmente los modos gráficos 4-6 son idénticos con CGA. Los nuevos modos gráficos(los cuales proveen de más colores y/o mayor resolución) son numerados de 0Dh-10h.

MODO	TIPO	RESOLUCION HOR.	RESOLUCION VER.	COLOR	DESCRIPCION	PAGINAS
	MODOS COMPATIBLES CON CGA:					
0	Texto	40	25	No (B/N)	Texto a 40 columnas monocromático.	8
1	Texto	40	25	16	Texto a 40 columnas con 16 posibilidades de color.	8
2	Texto	80	25	No (B/N)	Texto a 80 columnas monocromático.	8
3	Texto	80	25	16	Texto a 80 columnas con 16 posibilidades de color.	8
4	Gráfico	320	200	4	Gráficas de resolución media con color.	1
5	Gráfico	320	200	No (B/N)	Gráficas de resolución media monocromáticas.	1
6	Gráfico	640	200	2	Gráficas de alta resolución en blanco y negro.	1
	MODO COMPATIBLE CON MDA:					
7	Texto	80	25	No	Textos monocromáticos con atributos de pantalla.	8

MODO	TIPO	RESOLUCION HOR.	RESOLUCION VER.	COLOR	DESCRIPCION	PAGINAS
	OTROS:					
0Dh	Gráfico	320	200	16	Gráficas de resolución media con 16 posibilidades de color.	8
0Eh	Gráfico	640	200	16	Gráficas de alta resolución con 16 posibilidades de color.	4
0Fh	Gráfico	640	350	No	Gráficas de alta resolución con 350 líneas. Sólo se puede usar en combinación de monitor monocromático.	2
10h	Gráfico	640	350	16	Gráficas de alta resolución con 350 líneas. Sólo se puede usar en combinación de monitor de color realzado de IBM.	2

Observación: Existen versiones avanzadas del EGA(Enhanced EGA), las cuales establecen en los modos 0Fh y 10h una resolución de 640 por 480 pixeles. Incluso algunos fabricantes llegan a manejar hasta 800 por 560.

VIDEO GRAPHICS ARRAY (VGA)

El VGA (Arreglo Gráfico de Video) apareció en escena en abril de 1987 con la introducción de las computadoras IBM PS/2, siendo actualmente equipo de serie en sus modelos 50, 60 y 80. También se puede usar como adaptador de video en los equipos IBM PC AT y XT. Es básicamente una versión ampliada y mejorada del EGA. Las mejoras de VGA se centran en los modos gráficos.

En general VGA no acepta cualquier tipo de monitor, como anteriormente lo hacia EGA. El aumento en el ancho de banda de la señal requerida para el manejo de color provocaba que el diámetro de los cables en los monitores TTL usados tradicionalmente se incrementara en forma excesiva, por lo cual IBM decidió utilizar monitores analógicos. Los monitores analógicos RGB transmiten sus señales de color a través de 3 cables controlando directamente los niveles de voltaje en ellos para así manipular las intensidades de despliegue. Cuando se requiere de una gran variedad de colores, el control de los niveles de voltaje es mucho mas sencillo en monitores analógicos que en los tradicionales digitales (TTL). Por lo anterior, frecuentemente no se puede utilizar este adaptador con cualquier monitor a color o monocromático con excepción de aquellos que manejen frecuencia variable, los cuales tienen un mecanismo que les permite conmutar de entrada TTL a analógica.

La habilidad de modificar fonts, trabajar con caracteres adicionales y seleccionar colores de más de 250,000 diferentes indican la potencia y flexibilidad adicionada a VGA. Casi cualquier atributo del display puede ser modificado. Los caracteres pueden aumentar o disminuir sus dimensiones (con los mismos patrones de EGA) y el número de líneas en la pantalla puede ser cambiado. Los cambios pueden ser temporales (solo afectar a la aplicación actual) o permanentes (afectar a todos los procesos que se efectúen en la

sesión o hasta que se reprograme el sistema). A diferencia de EGA, este adaptador posee tablas de fonts con hasta 1024 caracteres diferentes, 512 de los cuales pueden ser utilizados al mismo tiempo.

Los 3 nuevos modos gráficos de VGA son: 11h con 640 por 480 pixeles y 2 colores, 12h con 640 por 480 pixeles y 16 colores y 13h con 320 por 200 pixeles y 256 posibilidades de color. Los colores de los modos 12h y 13h pueden ser elegidos de una paleta de 264,144 opciones. En general, estos modos no son compatibles con los modos 0Fh y 10h del Enhanced EGA.

MODO	TIPO	RESOLUCION HOR.	RESOLUCION VER.	COLOR	DESCRIPCION
0	Texto	40	25	No	Texto a 40 columnas monocromático.
1	Texto	40	25	16	Texto a 40 columnas con 16 posibilidades de color.
2	Texto	80	25	No	Texto a 80 columnas monocromático.
3	Texto	80	25	16	Texto a 80 columnas con 16 posibilidades de color.
4	Gráfico	320	200	4	Gráficas de resolución media con color.
5	Gráfico	320	200	No	Gráficas de resolución media monocromáticas.
6	Gráfico	640	200	2	Gráficas de alta resolución en blanco y negro.
7	Texto	80	25	No	Textos monocromáticos con atributos de pantalla.
0Dh	Gráfico	320	200	16	Gráficas de resolución media con 16 posibilidades de color.

MODO	TIPO	RESOLUCION HOR.	RESOLUCION VER.	COLOR	DESCRIPCION
0Eh	Gráfico	640	200	16	Gráficas de alta resolución con 16 posibilidades de color.
0Fh	Gráfico	640	350	No	Gráficas de alta resolución con 350 líneas.
10h	Gráfico	640	350	16	Gráficas de alta resolución con 350 líneas.
	MODOS COMPATIBLES CON EGA:				
11H	Gráfico	640	480	2	Gráficas de alta resolución con 480 líneas y 2 colores (monocromáticas).
12h	Gráfico	640	480	16	Gráficas de alta resolución con 480 líneas y 16 posibidades de color.
13h	Gráfico	320	200	256	Gráficas de resolución media con 256 posibilidades de color.

ANALISIS COMPARATIVO.

El costo de los adaptadores va en proporción directa a los modos de video que manejan. En orden creciente de costo se tiene: MDA, HERCULES, CGA, EGA y VGA. Así por ejemplo, la HERCULES tiene un costo de 320 dólares mientras que la VGA cuesta 595 dólares (costos promedios).

El mismo orden se observa en cuestión de sus facilidades gráficas. Así se tiene que la que presenta más limitantes es el MDA, siendo la VGA la más avanzada.

Dado que el MDA sólo maneja modo texto, la única posibilidad de manejo de figuras es utilizando puntos, guiones y caracteres especiales ASCII como son figuras de esquinas, cruces, etc. De esta manera sus figuras tienen una resolución limitada a 80 por 25.

El adaptador HERCULES presenta la ventaja de su modo de alta resolución (720 por 348 pixeles), lo que le permite crear figuras de alta calidad. Su principal desventaja es que sólo puede usarse para controlar monitores monocromáticos, por lo que se ve limitado a manejar 2 colores. Así mismo, otra limitante es que no pude utilizarse en combinación con otro adaptador de video, a excepción de la versión de CGA de la Hercules Computer Technologies.

El CGA es una buena opción en cuestión de costos y opciones gráficas cuando se usa para controlar monitores monocromáticos o de color tradicionales. Por otro lado, a pesar de poseer 3 modos gráficos, uno de los cuales maneja una resolución alta de 640 por 200 pixeles, su posibilidad de utilización de colores es sumamente limitada. El número máximo de colores que permite manejar es de 4 y, lo anterior, únicamente en modo de resolución media con 320 por 200 pixeles. Y dado que su modo de alta resolución solo permite manejar 2 colores(blanco y negro), éste se ve superado por el correspondiente modo de alta resolución de HERCULES. Sin embargo este adaptador si puede trabajar en combinación con otro adaptador como es el caso del MDA o el EGA.

Con respecto a EGA, su principal ventaja es la de poder utilizarse directamente para controlar cualquier monitor monocromático o de color. Su modo de alta resolución aumenta el

número de líneas a 350(resolución de 640 por 350). Posee modo de alta resolución para monitor monocromático y a color, característica que no presenta CGA, aunque para utilizar éste último se requiere de un IBM Enhanced Color Display o equivalente. Se puede utilizar en combinación con alguno de los 2 adaptadores de IBM: el CGA o el MDA. Sólo algunas versiones de EGA son compatibles para su uso junto con HERCULES.

Finalmente, el VGA representa la elección obligada para las aplicaciones gráficas avanzadas y de alta calidad. Puede manejar varias páginas gráficas(por default, el mismo número manejado por EGA), lo cual, adicionado a la facilidad de programar sus diferentes modos de video y posibilidades de color, permite desarrollar adecuadamente aplicaciones con animación. Posee la paleta de colores con la mayor variedad (264,144 posibilidades de color), el mayor número de colores disponibles a la vez (256) y el modo de mayor resolución con color (640 por 480) de los adaptadores anteriormente mencionados. La versión original de IBM se puede combinar para trabajar junto con HERCULES, MDA, CGA o EGA. La principal desventaja es su requerimiento de utilización de monitores analógicos. Por otra lado, es el de mayor costo, el cual fácilmente puede triplicar el precio de un MDA.