Práctica 6. Periféricos Avanzados.
Programación del Disco Duro a través de la interfaz IDE.
Lectura de la tabla de particiones.

Introducción.

En esta práctica accederemos al disco duro a través de la funcion Bios 13h. En la práctica se proporciona un codigo fuente C que se ha de completar para poder leer toda la informacion disponible de la tabla de particiones.

¿Qué es la tabla de particiones?

La tabla de particiones es una estructura que ocupa 64 bytes y que define la forma en que un disco duro está dividido en secciones lógicas, conocidas como particiones.

Offset	Naturaleza	Size
+OOh	Status de partición 00h = no activa 80h = Partición de Boot	1 byte
+01h	Inicio de partición : Cabecera	1 byte
+02h	Inicio de partición : Cilindro - Sector	1 palabra
+04h	Tipo de partición	1 byte
+05h	Fin de partición : Cabecera	1 byte
+06h	Inicio de partición : Cilindro - Sector	1 palabra
+08h	Número de sectores entre el MBR y el 1er sector de la partición	4 bytes
+0Ch	Número de sectores en la partición	4 bytes

Estructura de la tabla de particiones.

A pesar de que puede haber mas de una particion lógica en el disco, la más importante es aquella que se almacena dentro del MBR (Master Boot Record) la misma que contiene información adicional tal como desde qué particion se inicia el sistema.

La tabla de particiones es exclusivamente una estructura de información, es común confundirla con el MBR.

MBR

Esto es válido para todas las versiones DOS, y por tanto para todos los sistemas operativos que quieran convivir con el DOS deben respetar este formato. Podría haberse escogido otro..., pero por suerte o por desgracia, este es el que nos ha tocado. Todos los sistemas operativos actuales de PC respetan este formato.

El llamado sector de particiones es creado por FDISK en su primera llamada (con un disco recién adquirido y sin preparar) o cuando ejecutamos el comando FDISK /MBR.

Es el primer sector del disco duro (cabeza 0, cilindro 0, sector 1). Este es el sector que siempre arranca la BIOS primeramente antes de cargar ningún sistema operativo. La bios lo carga en la posición de memoria 0000:7C00 siempre que no encuentre un disquete en la unidad A:, y le cede el control para la ejecución del programa que espera encontrar allí.

Si los dos ultimos bytes de los 512 de este sector contienen el codigo 55h,AAh (hexadecimal) considera este sector como ejecutable y comienza la ejecución de programa en el primer byte de ester sector una vez se ha cargado en la posición de memoria anterior.

El código de programa que hay en este sector de arranque, tiene como tarea el reconocer la partición "activa" y con ello, el sistema operativo a ejecutar, cargar su sector de arranque y comenzar la ejecución del código de programa que allí está contenido. Ya que este código de programa, por definición, se ha de encontrar en la posicion de memoria 0000:7C00, el código de partición, primeramente, se desplaza a la posición de memoria 0000:0600 y con ello deja espacio para el sector de arranque.

Dirección	Contenido	Tipo
+000h	Código de la partición	Código
+1BEh	1ª entrada en la tabla de particiones	16 Bytes
+1CEh	2ª entrada.....	16 Bytes
+1DEh	3ª entrada.....	16 Bytes
+1EEh	4ª entrada.....	16 Bytes
+1FEh	Identificación AA55h	2 Bytes

Longitud= 200h = 512 Bytes.

Veamos cada entrada de 16 Bytes que define una partición, qué es lo que contiene:

Dirección	Contenido	Tipo
+00h	Estado de la particion 00h = Inactiva 80h = Partición de arranque	1 BYTE
+01h	Cabeza de lectura/escritura donde comienza la partición.	1 BYTE
+02h	Sector y Cilindro donde comienza la partición (formato WORD - palabra)	2 BYTES
+04h	Tipo de particion 00h = Libre 01h = DOS con la vieja 12-bit FAT 02h = XENIX 03h = XENIX 04h = DOS FAT 16	1 BYTE

05h = Partición extendida
06h = Partición DOS 4.0 > 32 Megas
DBh = Concurrent DOS
.... etc

+05h	Cabeza de lectura/escritura donde termina la partición.	1 BYTE
+06h	Sector y cilindro donde termina la partición.	2 BYTES
+08h	Distancia del primer sector de la partición (Sector de arranque)	4 BYTES
+0Ch	Numero de sectores de esta partición	4 BYTES

Longitud = 10h = 16 Bytes

Luego las funciones del programa de boot (MBR) del disco duro son:

1) Localizar el sector de arranque de la partición activa, para esto se recorre las 4 entradas de las 4 posibles particiones para ver cual es la activa.

2) Posicionar la cabeza de lectura escritura en dicha partición.

3) Volver a cargar los 512 primeros bytes de esa partición en memoria y ceder el control (este es el verdadero sector de arranque del sistema operativo. En el caso de MSDOS o WINDOWS, es creado al dar un FORMAT a la partición)

****

Con la descripción anterior nos podemos dar cuenta de una cosa:

* Sólo es posible un máximo de 4 particiones (no hay espacio para más, en discos que sean compatibles MSDOS, esto por definición).

Y otra cosa MUY importante. Lo fácil que lo tienen los virus para destrozar un disco. Únicamente con *modificar* un poquito este sector desde la posicion +1BEh, podemos dar por perdidos los datos en el disco.

** Y otra cosilla, en un programa de media docena de instrucciones, podremos cambiar la "marca" de partición activa a la que nos interese, para reiniciar desde ella.

Guia para la elaboración de la práctica

Lo que debeis hacer en esta práctica es rellenar el código que se os proporciona para poder ver las particiones del disco duro, para ello además de comentarios en el código fuente, disponeis de la siguiente aclaración y de un extenso apéndice en el que se explica con todo detalle lo mas importante sobre los discos.

Información útil.

1.- La interrupción a invocar es la 13h.

2.- Estructuras a usar.

Para realizar la práctica disponemos de tres estructuras "**struct c"** que describen las partes del disco a las que vamos a acceder, estas son :

1.- SECPOS: Describe la posición de un sector.

2.- PARTENTRY: Una entrada en la tabla de particiones.
- 1 byte de estado.
- 1 estructura SECPOS para localizar al primer sector.
- 1 byte PartTyp indica el tipo de particion.
- 1 estructura SECPOS para localizar el final de la partición.
- 1 unsigned long para localizar al sector de arranque.
- 1 unsigned long para indicar el número de sectores.

3.- PARTSEC. Describe el sector.
- 1 byte para el codigo Boot.
- 1 estructura PARTENTRY
- 1 word que indica el Id de codigo.

Para comprender esta última estructura :

La tabla comienza en un offset 1BEh del sector (al principio está el código ejecutable); cada partición de las 4 posibles ocupa 16 bytes; al final de las cuatro está la marca 0AA55h, ubicada en el offset 1FEh, que indica que la tabla es válida.

Este texto está extraido del apéndice. Epígragfe
"CABEZA 0. PISTA 0. SECTOR 1" Leer para más detalle.

3.- Detalles para completar la implementación de la función LeerPartSec

LeerPartSec : Leer 1 sector de la partición del disco duro y almacenarlo en un buffer.

Inciso :
** Recordad que cuando se invocaba con int86 o int86x a una funcion , en este caso, la función interrupción13h, existían una serie de funciones que se debían cargar en el registro ah o en el ax (según corresponda) para que se lleve a cabo la acción adecuada.

En este caso, lectura de un sector de particion del disco duro, la informacion que deben llevar los registros de entrada a la interrupcion son :

Funcion para leer el primer sector : 0x0201
(**ojo dado el número de bits, que registro usaremos ah o ax)**

El resto de registros debe contener la siguiente informacion :

dl : la unidad.
dh: la cabeza
cx: numero de sector y cilindro en formato BIOS.

4.- Código a rellenar en la funcion MostrarParticion:

Se debe completar el código de la partición activa.

Se debe implementar un switch para que en función del valor que tenga la variable AP.PartTyp imprima el tipo de partición (ver tabla pagina 2 – página3)

Ejercicios :

1.- Rellenar el código que falta para que el programa funcione.
2.- Comentar brevemente cada función del código, incluidas las macros.

Apéndice 1 . Código fuente

Nota : El modelo de memoria seleccionado para esta aplicación ha de ser "small", esto se puede cambiar en Option->Compiler->Code Generation.

```
#include <dos.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

/*== Constantes ==========================================================*/

#define TRUE  ( 1 == 1 )
#define FALSE ( 1 == 0 )

/*== Macros ==============================================================*/

#define HI(x) ( *((BYTE *) (&x)+1) )          /*devuelve byte alto de un WORD*/
#define LO(x) ( *((BYTE *) &x) )              /*devuelve byte bajo de un WORD*/

/*== Declaración de tipos ================================================*/

typedef unsigned char BYTE;
typedef unsigned int WORD;

typedef struct {          /*describe la posición de un sector*/
                          BYTE Head;          /*cabezal de lectura/escritura*/
                          WORD SecZyl;        /*n§ de sector y cilindro*/
                         } SECPOS;

typedef struct {          /*entrada en la tabla de particiones*/
                          BYTE      Status;     /*estado de partición*/
                          SECPOS    StartSec;   /*primer sector*/
                          BYTE      PartTyp;    /*tipo de partición*/
                          SECPOS    EndSec;     /*£ltimo sector*/
                          unsigned long SecOfs; /*offset del sector de arr.*/
                          unsigned long SecAnz; /*n§ de sectores*/
                         } PARTENTRY;

typedef struct {          /*describe el sector de part.*/
                          BYTE     BootCode[ ?????? ];
                          PARTENTRY PartTable[ 4 ];
                          WORD     IdCode;          /*0xAA55*/
                         } PARTSEC;

typedef PARTSEC far *PARSPTR; /*puntero al sector de partición en mem.*/
```

```
/*************************************************************************/
 ReadPartSec :  Lee un sector de particiˇn del disco duro a   un buffer
 Entrada:
        - Unidad : Cˇdigo BIOS de la unidad (0x80, 0x81 etc.)
        - Head   : N£mero del cabezal de lectura/escritura
        - SekZyl : N§ de sector y cilindro en formato BIOS
        - Buf    : buffer, en el que se carga el primer sector
 Salida : TRUE, si se pudo leer el sector sin problemas
          sino FALSE
/*************************************************************************/

BYTE LeerPartSec( BYTE LaUnidad, BYTE Head, WORD SekZyl, PARSPTR Buf )

{
 union REGS   Regs;    /*reg. de procesador para la llamada de interr.*/
 struct SREGS SRegs;

 Regs.x.ax = ??????;                  /*N§ fun.c para "Read", 1er sector*/
 Regs.h.dl = ¿?????;                        /*cargar los otros*/
 Regs.h.dh = ¿????;                           /*a los diferentes*/
 Regs.x.cx = ¿?????;                               /*registros*/
 Regs.x.bx = FP_OFF( Buf );
 SRegs.es  = FP_SEG( Buf );

 int86x( ¿???, ¿???, ¿?????, ¿?????);     /*llamar int. de disco duro*/
 return !Regs.x.cflag;
}

/*************************************************************************/
  GetSecZyl: obtiene de la codificaciˇn combinada de sector/cilindro  de la BIOS
             el n§ de sector y cilindro

            Entrada:
                -SekZyl   : el valor a decodificar
                -Sector   : referencia a la variable de sector
               -Cylinder : referencia a la variable del cilindro
           Salida : ninguna
/*************************************************************************/

void GetSecZyl( WORD SekZyl, int *Sector, int *Cylinder )

{
 *Sector   = SekZyl & 63;                    /*apagar bits 6 y 7*/
 *Cylinder = HI(SekZyl) + ( ( (WORD) LO(SekZyl) & 192 ) << 2 );
}
```

```
/********************************************************************/
MostrarParticion : muestra las particiones de un disco duro  en la pantalla
           Entrada:
              LW : Número de la unidad de disco duro direccionada
                   (0, 1, 2 etc.)
           Salida : ninguna
/********************************************************************/

void MostrarParticion( BYTE LW )
{
 #define AP ( ParSec.PartTable[ Entry ] )

 BYTE      Head,                  /*cabezal de la partici¢n actual*/
           Entry;                          /*contador de bucle*/

 int       Sector,                        /*acogen n§ de sector y*/
           Cylinder;                                /*cilindro*/

 PARTSEC    ParSec;                  /*el sector de partici¢n actual*/
 union REGS Regs;      /*reg. de procesador para la llamada de interr.*/

 printf("\n");
 LW = LW | 0x80;                  /*preparar nº de unidad para la BIOS*/


 if ( LeerPartSec( LW, 0, 1, &ParSec ) )    /*leer sector de partici¢n*/
  {                                          /*sector se pudo leer*/
         Regs.h.ah = 8;                     /*obtener datos de la unidad*/
         Regs.h.dl = LW;
         int86( 0x13, &Regs, &Regs );           /*llamar int. de disco duro*/
         GetSecZyl( Regs.x.cx, &Sector, &Cylinder );

         /*Imprime una  tabla en pantalla*/

         printf( "ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ"
                      "ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»\n");

         printf( "º Unidad   %2d:    %2d Cabezales con %4d"
                      " Cilindros a %3d Sectores         º\n",
                      LW-0x80, Regs.h.dh+1, Cylinder, Sector );
         printf( "º Tabla de part. en el sector de partici¢n  "
                      "                               º\n");
         printf( "ÌÍÍÑÍÍÍÍÑÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÑÍÍÍÍÍÍÍÍÍÍÍÍÍÍÑ"
                      "ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÑÍÍÍÍÍÍÍÑÍÍÍÍÍÍÍ¹\n");
         printf( "º  ³    ³                 ³    Inicio   ³"
                      "    Final     ³Distanc³       º\n");
         printf( "ºNr³Boot³Tipo             ³Cbzl Cil. Sec.³"
                      "Cbzl Cil. Sec.³SecArrq³N£mero º\n");
         printf( "ÌÍÍØÍÍÍÍØÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍØÍÍÍÍÍÍÍÍÍÍÍÍÍÍØ"
                      "ÍÍÍÍÍÍÍÍÍÍÍÍÍÍØÍÍÍÍÍÍÍØÍÍÍÍÍÍÍ¹\n");

         /*-- Switch para recorrer la tabla de particiones -----------------------------*/
         for ( Entry=0; Entry < 4; ++Entry )
          {
           printf( "º %d³", Entry );
           if ( AP.Status == ?????? )                     /*partici¢n activa?*/
                 printf(" S¡ ");
           else
                 printf (" No ");
```

```
            printf("³");

           /*Implementar el  switch para que imprima el tipo de particion adecuado*/
            switch( AP.PartTyp )                /*evaluar tipo de partici¢n*/
                   {
                          ???????
                          ???????
                   }




            /*-- evaluar datos f¡sicos y l¢gicos ---------------------------*/
            GetSecZyl( AP.StartSec.SecZyl, &Sector, &Cylinder );
            printf( "³%2d %5d  %3d ", AP.StartSec.Head, Cylinder, Sector );
            GetSecZyl( AP.EndSec.SecZyl, &Sector, &Cylinder );
            printf( "³%2d %5d  %3d ", AP.EndSec.Head, Cylinder, Sector );
            printf( "³%6lu ³%6lu º\n", AP.SecOfs, AP.SecAnz );
           }
           printf( "ÈÍÍÏÍÍÍÍÏÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÏÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÏ"
                          "ÍÍÍÍÍÍÍÍÍÍÍÍÍÏÍÍÍÍÍÍÍÍÍÍÏÍÍÍÍÍÍÍÍ¼\n" );
 }
 else
  printf("-Error al acceder al sector de arranque!\n");
}

/*********************************************************************/
/*              P R O G A M A   P R I N C I P A L              */
/*********************************************************************/

int main( int argc, char *argv[] )
{
 int LaUnidad;

 printf( " Práctica 6 – Periféricos avanzados \n" );
LaUnidad = 0;                    /*defecto en el primer disco duro*/


/*Si el numero de argumentos es 2 , entonces el usuario ha especificado en la linea de comandos*/
/*otra unidad, si no indica nada , por defecto la unidad a estudiar será la 0, es decir, C:*/

 if ( argc == 2 )                              /*¨otra unidad?*/
  {                                              /*Si*/
           LaUnidad = atoi ( argv[1] );
           if ( LaUnidad == 0 && *argv[1] != '0' )
            {
            printf("\nN£mero de unidad no v lido!");
            return( 1 );                          /*el programa*/
            }
  }
 MostrarParticion( (BYTE) LaUnidad );      /*mostrar sector de partici¢n*/
 return( 0 );
}
```