Cómo Direccionar Más Allá del Primer MegaByte en el Modo Real de las PC

M. en C. Gustavo A. Mas Levario
Profesor e Investigador del CINTEC-IPN.

La finalidad de este artículo es describir el método a seguir para accesar las direcciones de memoria superiores al primer megabyte en máquinas PC's que estén operando con el MS-DOS y en el modo real, desde un programa en C en interface con otro en ensamblador.

Fundamentos

Antes de entrar en detalles con respecto al método es necesario que se analicen las capacidades de operación del direccionamiento de memoria de los microprocesadores Intel de la familia 80x86, en los modos real y protegido.

Modo Real

El modo real se define **[1]** literalmente como sigue: En el Intel 286 y máquinas más avanzadas, estado operacional en el cual la computadora funciona como si fuera una 8086 u 8088. Se limita a direccionamientos de un megabyte de memoria.

La definición anterior se usará para detallar más algunos aspectos. En los 8088 y 8086 los programadores se enfrentan a una arquitectura de memoria segmentada en bloques de 64KB que dificulta el manejo de ciertas estructuras de datos con longitudes mayores de 64KB. Otra restricción en el direccionamiento de memoria de mayor peso proviene por la cantidad de memoria que los 8088 y 8086 pueden direccionar, y la cual es de 1 MegaByte.

Debido a las restricciones del manejo de la memoria, cuando el MS-DOS se implantó como sistema operativo de las máquinas basadas en los microprocesadores 8088 y 8086, se le limitó a funcionar con un megabyte y como consecuencia el BIOS también se limitó a funcionar en un megabyte.

En la actualidad, aunque los microprocesadores de las computadoras personales sean 80386, 80486 o PENTIUM, cuya capacidad de direccionamiento es 4 Gigabytes, estas se siguen empleando con las restricciones del modo de operación de las máquinas 8088 y 8086.

La situación anterior fue ocasionada por la regla que adoptó Intel, la cual consiste en ofrecer una rígida compatibilidad de sus nuevos productos con los que ya tenía comercializados anteriormente **[3]**. Las máquinas 80286 y superiores ejecutan todo el software desarrollado para las máquinas 8088 y 8086. En este modo real de trabajo la única diferencia aparente consiste en la velocidad del proceso, siendo mucho mayor en las máquinas 80286 y superiores.

El Direccionamiento en Modo Real

Las máquinas de Intel direccionan los objetos en la memoria en bytes y usan un modelo de direccionamiento de memoria conocido como segmentación, la cual consiste en particionar la memoria en segmentos. Dentro de cada segmento cada uno de los bytes tiene un desplazamiento (offset en computación) único.

Para direccionar un byte en la memoria tanto el segmento como el desplazamiento deben ser conocidos. Una dirección completa en modo real se especifica como SSSS:0000, donde SSSS representa el segmento y 0000 es el desplazamiento. La notación SSSS:0000 usa números en notación hexadecimal. Para obtener una dirección lineal o física en el modo real la parte SSSS se desplaza un lugar a la izquierda y en ese lugar vacío insertamos un cero, de modo que tendríamos lo siguiente SSSS0 y finalmente a SSSS0 le sumamos 0000; el resultado es una dirección lineal o física.

El valor de SSSS de la dirección SSSS:0000 es usada por uno de los registros de segmento, ya sea de

código, datos o pila, para direccionar junto con 0000 un campo dentro de un segmento de 64 Kbytes. Las direcciones van desde 0000:0000 pasando por A000:0 000 hasta FFFF:000F, es decir, un rango de un megabyte.

Normalmente se indica con el número inicial de la notación SSSS:0000 (o sea la primera S leida de izquierda a derecha) el número del segmento al que se hace referencia. De este modo, el segmento cero, 0000:0000, es previo al segmento uno, 1000:0000, y así sucesivamente existiendo un espacio de 64 Kbytes entre cada uno.

Un fenómeno que se presenta en el modo real y que tiene que ver con el manejo de direcciones en el límite del megabyte en las máquinas 80286 y superiores es el llamado **"wrap-around" [2]**.

Para el propósito de este articulo **wrap-around** tiene el siguiente significado: Hecho que se presenta en el calculo de una dirección lineal el cual consiste en restar un megabyte (100000 en hexadecimal) a las direcciones que sean mayores del rango de un megabyte en el modo real. Por ejemplo, al calcular la dirección lineal de FFFF:0011 en un 80286 o superiores se presenta el fenómeno wrap-around y se generará la dirección uno, o sea 0000:0001. Los diseñadores de tarjetas madres de las máquinas 80286, 80386, 80486 han agregado una "puerta" para resolver esta restricción de compatibilidad en el modo real. Esta puerta permite que la línea de dirección 20 de las máquinas pueda ser usada para calcular una dirección lineal, en el modo real sólo se usan las líneas 0 a la 19 para un total de un megabyte. De modo que con la "puerta" activada el fenómeno wrap-around no ocurre, con la consecuencia de poder direccionar arriba del primer megabyte. Con la "puerta" desactivada wrap-around ocurre tal y como pasa en el modo real.

Modo protegido

El modo protegido se define **[1]** literalmente como sigue: En el Intel 286 y superiores, un estado operacional que permite a la computadora direccionar toda la memoria. También previene a un programa introducirse en los límites de la memoria de otro, permitiendo de este modo ejecutarse en un entorno "protegido".

Además de la definición anterior necesitamos usar las siguientes definiciones, ya que en el modo protegido se usan términos nuevos con respecto al modo real:

Selector **[5]**.- Una cantidad de 16 bits que especifica un descriptor de un segmento.

Descriptor **[5]**.- Una cantidad de 8 bytes, especificando independientemente un objeto protegido.

Objeto **[1]**.- Un modulo de datos relacionados y autónomos y su proceso asociado.

Tabla de descriptores **[5]**.- Un arreglo de descriptores de segmento. Hay dos clases de tablas de descriptores: La Tabla Global de Descriptores (GDT) y un número arbitrario de Tablas Locales de Descriptores (LDT).

Registro de la Tabla Global de Descriptores (GDTR) **[5]**. - Registro que guarda los 32 bits de la dirección base lineal y los 16 bits del límite de la GDT.

Nivel de Privilegio **[5]**.- Un número en un rango predeterminado indicando el grado de protección del objeto de la memoria.

Nivel de Privilegio del Descriptor (DPL) **[5]**.- Un campo en un descriptor indicando como está protegido el descriptor.

Derechos de Acceso **[5]**.- Una cantidad de ocho bits contenida en algún descriptor, que indica las condiciones del objeto de la memoria. Las condiciones muestran si el objeto está presente en la memoria principal o no, su DPL, si es un objeto de datos o de código, y además si ha sido accesado o no.

Granularidad .- Los incrementos más pequeños que pueden ser diferenciados **[7]**. Con respecto al direccionamiento de memoria en los 80286 y superiores se puede decir que la granularidad es en byte o en páginas de 4 Kbytes **[4]**. La granularidad está relacionada con el campo límite del descriptor, de modo que con los 19 bits del campo límite podemos establecer el tamaño del objeto de la memoria, es decir, podemos tener hasta un megabyte de bytes si el bit G=0, o hasta un megabyte de páginas si el bit G=1 para dar un total de 4 Gigabytes.

Unidad de Segmentación **[5]**.- Traduce direcciones segmentadas a direcciones lineales. La unidad de segmentación contiene una caché que guarda la información de la tabla de descriptores para cada uno de los seis registros de segmento.

La característica principal del mecanismo de protección en el modo protegido es el selector **[4]**. Todos los programas en el modo

protegido usan los mismos registros de segmento (CS, DS, ES y SS) que se usan en el modo real. La diferencia es lo que está contenido en los registros **[8]**. En el modo real los registros de segmento contienen la dirección inicial de un segmento en memoria. En el modo protegido los registros de segmento contienen un selector. Un selector consiste de un índice de 13 bits que apunta a un descriptor de alguna tabla de descriptores, un indicador de una tabla (ya sea GDT o LDT) de un bit y tres bits que indican el nivel de privilegio, que para la finalidad de este artículo no se utilizarán.

El descriptor es el otro nivel de protección, el cual contiene las siguientes características del objeto de la memoria: la dirección base de 32 bits, el límite de 19 bits, los derechos de acceso de 8 bits y otros cuatro bits de control de los cuales el que interesa es el bit llamado G (Granularidad) **[4]**.

Direccionamiento en el modo protegido

Para direccionar un byte, una palabra, o una doble palabra en el modo protegido **[3]**, las máquinas realizan el siguiente proceso usando únicamente la unidad de segmentación : se obtiene el selector del registro de segmento involucrado para saber con que tabla de descriptores está relacionado (ya sea con la GDT o la LDT). Con el propósito de dar más sencillez a este artículo se empleó la GDT. Después de que el selector indica que la GDT se va a usar, se le localiza en la memoria física por medio de GDTR. Ya hecho lo anterior se localiza en la GDT al descriptor asociado con el selector usando su índice de 13 bits desplazado a la izquierda tres bits y así obtener un índice de 16 bits para apuntar a una entrada de la GDT de 8 bytes que es el descriptor deseado. El contenido de este descriptor es usado para cargarlo en la unidad de segmentación en el registro de su caché asociado con el registro de segmento que contiene dicho selector. Finalmente la unidad de segmentación calcula la dirección lineal sumando la dirección base lineal del segmento que se desea accesar con el desplazamiento de 32 bits de alguna instrucción. En la **figura 1** se muestra de un modo gráfico la traducción de dirección segmentada a lineal.

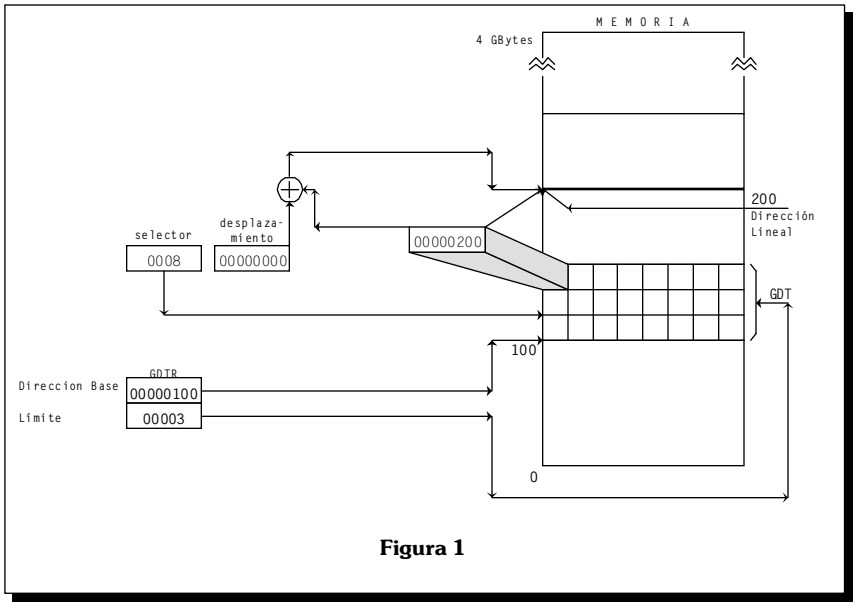

Figura 1

El Modelo Plano de la Memoria [3]

Es una circunstancia en la cual el direccionamiento de la memoria física es visto sin segmentos, es decir la memoria es usada como un espacio lineal de 4 Gigabytes, empezando desde la dirección 00000000h hasta la dirección FFFFFFFFh.

Para lograr lo anterior se construye un descriptor en la GDT con una dirección base lineal de 00000000h usando los bytes 1, 2, 3 y 7, un límite de FFFFFh usando los bytes 0, 1 y los cuatro bits menos significativos del byte 6, con G=1 y un valor de 92h en el byte 5, que son los derechos de acceso. Para la finalidad de este artículo se empleó ese valor en los derechos de acceso para indicar un segmento de datos de lectura/escritura de direccionamiento ascendente.

Para implantar el modelo plano es necesario estar en el modo protegido, ya que los registros caché de la unidad de segmentación únicamente pueden ser reinicializados en su totalidad en ese modo de operación **[6]**.

En el modo real, solamente el campo de la dirección base lineal se modifica directamente en los registros caché, de modo que los campos para el límite y los atributos permanecen fijos; y además, el campo límite se fija a FFFFh y la granularidad es en bytes (G=0) **[6]**.

Plan

Con todos los conceptos anteriores se debe idear un plan para direccionar un modelo plano desde el modo real usando el MS-DOS. El plan es el siguiente:

1.- Estar seguro de que no existe en operación otro programa que haga uso de la memoria extendida.

2.- Crear dos funciones, una que convierta las direcciones segmentadas a direcciones lineales y otra para que haga lo inverso, ya que cuando se involucre en el direccionamiento a GS y ES la unidad de segmentación estará inhibida.

3.- Crear una función que controle la "puerta" de la línea de dirección 20h del bus de direcciones.

4.- Crear una función que conmute al modo protegido desde el modo real e instale el modelo plano en los registros GS y ES, y además regrese al modo real.

5.- Hacer funciones que escriban y lean de cualquier parte del espacio de la memoria.

Resultado

Con el plan ideado se logró un método factible que puede accesar los 4 Gigabytes que es capaz de manejar una máquina de Intel de 32 bits de un modo simple, ya que se sigue usando todo lo conocido de una PC.

Restricciones

El programa anterior únicamente corre cuando no hay otro programa que haga uso de la memoria extendida, de modo que Windows, EMM386.EXE, y otros no deben estar funcionando.

Por último, se debe reinicializar la computadora para prevenir una falla ocasionada por un conflicto de direccionamiento si otro programa (un tiempo después) hace uso de los registros de segmento de datos GS y ES en el modo real.

El programa

El programa está hecho con una interface de lenguaje C y ensamblador de Borland Ver. 4.5 **[2]**. Está formado básicamente de una biblioteca con ocho funciones, las cuales se detallan a continuación.

Las funciones *seg_a_lineal()* y *lineal_a_seg()* se usan para emular la unidad de segmentación en el modelo plano, de modo que con estas funciones se tiene una conexión entre la segmentación del modo real y la linealidad del modelo plano. La expresión y el contenido de las funciones en lenguaje C es como sigue **[2]**:

```
LPTR seg_a_lineal(void far *p)
{
  return( ( (unsigned long) FP_SEG(p) )
  << 4 ) + FP_OFF(p);
}

void far *lineal_a_seg(LPTR lin)
{
  void far *p;
  FP_SEG(p) = (unsigned int) (lin >> 4);
  FP_OFF(p) = (unsigned int) (lin & 0xF);
  return p;
}
```

El término LPTR indica un objeto de 32 bits sin signo, y los términos *FP_SEG()* y *FP_OFF()* son macros del C usados para generar la parte correspondiente al segmento y al desplazamiento de una dirección segmentada.

La función *seg_a_4Gigas()* es la parte principal del programa y efectúa dos acciones básicas:

1.- Conmuta al modo protegido para poner el modelo plano en los registros de segmento GS y ES, y ...

2.- regresar al modo real con los registros caché modificados.

Esta función llama a una rutina en ensamblador nombrada "protsetup", para entrar desde el modo real al modo protegido y regresar de nuevo al modo real.

La función *a20()* desactiva y activa la "puerta" de la línea 20 del bus de direcciones para controlar el fenómeno "wrap-around" del modo real.

Las funciones *escribe_8bits()* y *escribe_32bits()* escriben un byte o una doble palabra respectivamente, en cualquier dirección deseada del rango de 0 a 4 Gigabytes.

Las funciones *lee_8bits()* y *lee_32bits()* leen un byte o una doble palabra, respectivamente.

Cada una de las últimas cuatro funciones anteriores llaman a una rutina escrita en ensamblador.

Documentación del programa

El programa consta de cuatro archivos, tres de ellos están escritos en lenguaje C de Borland Ver. 4.5 y el restante está escrito en lenguaje ensamblador **[2]** para los procesadores de 32 bits.

El archivo BIBLI4G.H contiene las definiciones de todas las funcio-

nes y variables necesarias para la finalidad del artículo.

El archivo BIBLI4G.C contiene la implantación del contenido descrito en el archivo BIBLI4G.H.

El archivo MAIN4G.C contiene una aplicación que consiste en escribir y leer en la dirección FFFFFCh (16 Mb-4 bytes) el dato BBBBAAAAh. Esta aplicación se ejecuta adecuadamente en una máquina 80286 o superior con 16 Mbyte de RAM; si no se cuenta con 16 Mbyte de RAM, entonces se debe cambiar la dirección a otra deseada y que permita correr adecuadamente la aplicación.

El archivo BB4G.ASM contiene las rutinas del lenguaje ensamblador invocadas por las funciones del archivo BIBLI4G.C.

Listados

En los siguientes cuadros se tiene el listado de la biblioteca de funciones desarrolladas para el manejo de la memoria.

```
// Archivo BIBLI4G.H
// Aquí están las definiciones de las funciones
// del archivo BIBLI4G.C.

/* el objeto LPTR es usado para guardar una dirección lineal
*/
typedef unsigned long LPTR;

/* funciones prototipo */
LPTR seg_a_lineal(void far *p);
void far *lineal_a_seg(LPTR lin);
void seg_a_4gigas(void);
void a20(int flag);
unsigned int lee_8bits(LPTR address);
void escribe_8bits(LPTR address, unsigned int byte);
void escribe_32bits(LPTR direccion, unsigned long
                    palabra32);
unsigned long lee_32bits(LPTR direccion);
// Fin del archivo BIBLI4G.H


// Archivo BIBLI4G.C
// Aquí está el desarrollo de las
// funciones del archivo BIBLI4G.H

#include <dos.h>
#include <conio.h>
#include «bibli4g.h»

/* se definen los controladores del teclado */
/* para habilitar la línea 20 de direcciones */
#define RAMPORT     0x70
#define KB_PORT     0x64
#define PCNMIPORT       0xA0
#define INBA20      0x60
#define INBA20ON    0xDF
#define INBA20OFF       0xDD

/* Convertir un apuntador far a una direccion lineal */
LPTR seg_a_lineal(void far *p)
{
   return (((unsigned long) FP_SEG(p)) << 4)
   +FP_OFF(p);
}

/* Convertir una direccion lineal a un apuntador far */
void far *lineal_a_seg(LPTR lin)
{
   void far *p;
   FP_SEG(p) =(unsigned int) (lin >> 4);
   FP_OFF(p) =(unsigned int) (lin & 0xF);
   return p;
}

/* estructura para hacer un descriptor para la GDT */
struct _GDT
{
   unsigned int limit;
   unsigned int base;
   unsigned int access;
   unsigned int hi_limit;
};

/* se inicializan dos descriptores para la GDT */
/* uno llamado nulo que es un default */
/* y el segundo para poner el modelo plano */
static struct _GDT GDT[2] =
{
   // descriptor no usado por el usuario y está por
   // default.
   {0, 0, 0, 0},
   // descriptor para poner un modelo plano en el
   // registro GS Y ES.
   {0x0FFFF, 0, 0x9200, 0x8F}
```

```
};
/* estructura para inicializar el GDTR */
struct fword
{
   unsigned int limit;
   unsigned long linear_add;
};

/* definición del objeto gdtptr que es el */
/* medio para inicializar al GDTR */
static struct fword gdtptr;

/* se ajusta el limite de GS y ES a AGB */
void seg_a_4gigas()
{
   /* se calcula la direccion lineal y el limite de
   GDT y se introduce en el GDTR */
   gdtptr.linear_add = seg_a_lineal((void far *)
   GDT);
   gdtptr.limit = 15;

   /* se deshabilitan interrupciones comunes */
   _disable();

   /* se deshabilitan lasinterrupciones NMI */
   outp(PCNMIPORT, 0);

   /* codigo para llamar al modo protegido */
   // rutina del archivo BB4G.ASM
   // esta rutina pone el modelo plano
   // en GS y ES y regresa de nuevo al
   // modo real
   protsetup(&gdtptr);

   /* se habilitan las interrupciones de nuevo */
   _enable();

   /* reabilitar NMI de nuevo */
   outp(PCNMIPORT, 0x80);
}

/* funcion de proposito general para habilitar  */
/* A20 (flag = 1) o desabilitar A20 (flag = 0) */
void a20(int flag)
{
   outp(INBA20, flag ? INBA20ON:INBA20OFF);
}

/* leer un simple byte de la memoria extendida */
/* dando una dirección líneal */
unsigned int lee_8bits(LPTR address)
{
   // rutina del archivo BB4G.ASM
   leer_1byte(address);
}

/* escribir un simple byte hacia la memoria */
/* extendida dando una direccion lineal */
void escribe_8bits(LPTR address, unsigned int byte)
{
    // rutina del archivo BB4G.ASM
   escribir_1byte(address, byte);
}

/* escribe 32 bits a la memoria extendida */
void escribe_32bits(LPTR direccion, unsigned
long palabra32)
{
    // rutina del archivo BB4G.ASM
   esc_32bits(direccion, palabra32);
}

/* se leen 32 bits de la memoria extendida */
unsigned long lee_32bits(LPTR direccion)
{
   void far * b;
   LPTR r = 0, t = seg_to_linear(b);

    // rutina del archivo BB4G.ASM
   leer_32bits(direccion, t);

   r = _AX;
   t = _DX;
   t = t<<16;
   t = t + r;
   return t;
}
// Fin del archivo BIBLI4G.C



// Archivo MAIN4G.C.
// Aquí está el programa principal, que usa las
// funciones del archivo BIBLI4G.C para accesar
// memoria en cualquier lugar.

#include <stdio.h>
#include <dos.h>
#include «bibli4g.h»

void main(void)
{
   // dirección FFFFFC (16 MBb- 4B]
   LPTR md = 0xFFFFFC;
   unsigned long d = 0;
   int i;
   char string[25];

   /* se hace un segmento de 4Gigas en
   GS y ES*/
   seg_a_4gigas();

   /* poner A20 en on */
   a20(1);

   /* se escribe el dato bbbbaaaa desde la
   dirección FFFFFC a la FFFFFF */
   escribe_32bits(md,0xBBBBAAAA);

   /* se lee el dato de la direccion FFFFFC */
   d = lee_32bits(md);

   // se convierte el dato leido en un string
   ltoa(d, string, 16);

   // se despliega el dato convertido en la pantalla
   printf(«En la direccion 0xFFFFFC Se
   escribio bbbbaaaa Y se leyo = %s\n», string);

   // mantiene el dato desplegado en la pantalla
   // hasta que recibe una interrupción del
   // teclado
   getchar();

   /* pone A20 en off de nuevo */
   a20(0);
}
// Fin del archivo MAIN4G.C



// Archivo BB4G.ASM
    .MODEL LARGE,C

    .386P
    .CODE
    PUBLIC  protsetup, esc_32bits
    PUBLIC leer_32bits, leer_1byte
```

```
; rutina para ir al modo protegido luego poner
; el modelo plano en los registros GS y ES
IF @DataSize
protsetup proc fpointer:dword,c
   push ds
   lds bx,fpointer
ELSE
protsetup proc fpointer:word,c
   mov bx,fpointer
ENDIF
; se guarda el valor de fpointer en GDTR
   lgdt fword ptr [bx]
; Load GDT
IF @DataSize
   pop ds
ENDIF
; se prepara el procesador para meterlo
; al modo protegido
   mov eax,cr0
   or al,1
   mov cr0,eax
; se reinicializa el registro IP
   jmp short nxtlbl
nxtlbl:
; se guarda en bx el selector que apunta
; al segundo arreglo de la estructura GDT[2]
; del archivo BIBLI4G.C
   mov bx,8
; se inicializan los registros GS y ES con el
; selector anterior para poner el modelo plano
; en los registros caché de GS y ES
   mov gs,bx
   mov es,bx
   and al,0feh
; el procesador se regresa al modo real de
; nuevo
   mov cr0,eax
   ret
protsetup endp

; leer un byte desde un LPTR
leer_1byte proc address:dword,c
   xor ax,ax
; aquí se opera en modo real así que
; GS tiene el segmento 0000 que es la
; base del segmento de 4 Gigas
   mov gs,ax
   mov eax,address
; se lee el byte y se pone en al
   mov al,gs:[eax]
   xor ah,ah
   ret
; se regresa el byte leido en ax
leer_1byte endp

// rutina que lee 32 bits en el modelo plano
leer_32bits proc direccion:dword, b:dword,c
   xor ax, ax
; aquí se opera en modo real así que
; GS tiene el segmento 0000 que es la
; base del segmento de 4 Gigas
   mov gs, ax
   mov ebx, direccion
   mov eax, dword ptr gs:[ebx]
   mov gs:[b], eax
   mov ax, word ptr gs:[ebx]
   mov dx, word ptr gs:[ebx+2]

   ret
; se regresa el dato leido en ax y dx
leer_32bits endp

; se escribe un byte en la dirección de LPTR
escribir_1byte proc address:dword, byt:word,c
   xor ax,ax
; aquí se opera en modo real así que
; GS tiene el segmento 0000 que es la
; base del segmento de 4 Gigas
   mov gs,ax
   mov eax,address
   mov bx,byt
   mov byte ptr gs:[eax],bl
   ret
escribir_1byte endp

; escribe una doble palabra a la direccion LPTR
esc_32bits proc direccion:dword,
       palabra32:dword,c
   xor ax, ax
; aquí se opera en modo real así que
; GS tiene el segmento 0000 que es la
; base del segmento de 4 Gigas
   mov gs, ax
   mov eax, direccion
   mov ebx, palabra32
   mov dword ptr gs:[eax], ebx
   ret
esc_32bits endp

   end
// Fin del archivo BB4G.ASM
```

Bibliografía

[1] Freedman Alan. "Diccionario de Computación". Quinta Edición Mc Graw Hill.1993.

[2] Williams Al. "DOS + 386 = 4 Gigabytes". Dr. Dobb's Journal. July 1990.

[3] Angulo Jóse Ma., Funke Enrique M. "386 y 486 Microprocesadores avanzados de 32 bits". Editorial Paraninfo.

[4] Nelson Ross P. "80386/80486 Programming Guide". Second Edition. Microsoft Press. 1991.

[5] Brumm Penn, Brumm Don, Scanlon Leo J. "80486 Programming". Windcrest Books. 1991.

[6] Intel. "Microprocessors: Volume II". 1994.

[7] Turner Rufus P., Gibilisco Stan. "The Illustrated Dictionary of Electronics". Fifth Edition. TAB. 1991.

[8] Syck Gary. "Turbo Assembler Bible". SAMS. 1991.