

## ACCESO A MODO PROTEGIDO

Este programa entra a modo protegido y se mantiene ahí hasta que se presione la tecla escape, y vulva a modo real.

### Programa:

Autor:  
Mariano Cerdeiro  
<[m.cerdeiro@soix.com.ar](mailto:m.cerdeiro@soix.com.ar)>  
<http://www.soix.com.ar/links/mpbyexamples.html>

```
1          use16
2          org 100h
3 00000000 E90000          jmp INICIO
4
5          INICIO:
6
7 00000003 EBFE          jmp $
8 00000005 FA          cli          ;deshabilito interrupciones
9 00000006 0F20C0        mov eax,cr0          ;lee byte de control
10 00000009 0C01        or al,1          ;setea PE=1 PROTECTION ENABLE bit 0 de CRO
11 0000000B 0F22C0        mov cr0,eax          ;lo carga en cro, pasa a modo protegido
12 0000000E E90000        jmp PROTEGIDO          ;vacia la cola de intruccioness
13
14          PROTEGIDO:
15
16 00000011 E460          in al,60h          ;se lee el contenido del puerto 60h
17 00000013 FEC8          dec al          ;se decrementa al para compararlo con 0
18 00000015 75FA          jnz PROTEGIDO          ;si no es ESC vuelve al loop
19
20 00000017 EBFE          jmp $
21 00000019 0F20C0        mov eax,cr0          ;lee byte de control
22 0000001C 24FE          and al,1111110b          ;Pone en 0 PE
23 0000001E 0F22C0        mov cr0,eax          ;vuelve a modo real
24 00000021 E90000        jmp REAL          ;vacia cola instrucciones
25
26          REAL:
27
28 00000024 FB          sti          ;habilita interrupciones
29 00000025 B44C          mov ah,4ch          ;servicio de dos
30 00000027 CD15          int 21          ;llama interrupciones de DOS
31          ;devuelve el control a DOS
```

### Explicación:

1 use16

Indica al compilador que debe interpretar instrucciones de 16 bits

2 org 100h

Nuestro programa se carga a partir de un desplazamiento de 256bytes, este espacio es utilizado por el sistema y se denomina PSP ( Program Segment Prefix)

```
3 00000000 E90000          jmp INICIO
```

Salta al inicio del programa, pues luego de esta instrucción se colocan los datos y no hay que ejecutarlos. En este caso no hay ningún dato y la instrucción puede omitirse, pero se incluye para mantener la forma.

```
4
5          INICIO:
Inicio del programa ( es un label)
```

```
6
7 00000003 EBFE          jmp $
```

Se utiliza para bloquear el programa y depurarlo. En un programa depurado esta instrucción se debe omitirse.

```
8 00000005 FA          cli          ;deshabilito interrupciones
```

Deshabilita las interrupciones ya que pueden interferir en el pasaje de modo real a modo protegido y bloquear el programa.

```
9 00000006 0F20C0      mov eax,cr0      ;lee byte de control
```

Se lee el Registro de Control CR0 para modificar solo el bit PE que indica el pasaje a modo protegido.

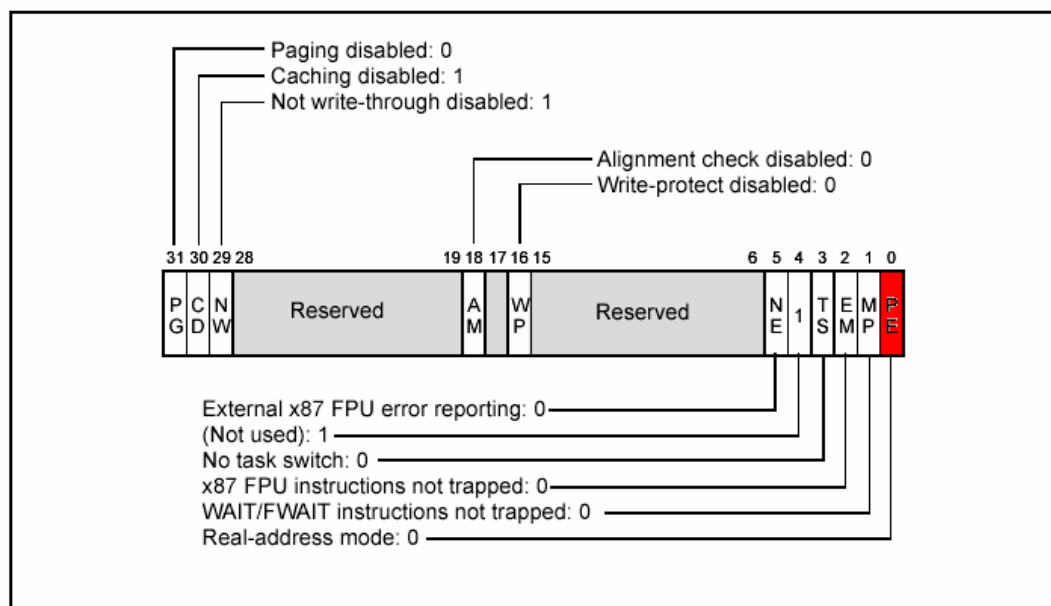


Figure 8-1. Contents of CR0 Register after Reset

```
10 00000009 0C01          or al,1          ;setea PE=1 PROTECTION ENABLE bit 0 de CR0
```

Se modifica el bit de PE del registro de control CR0, lo que indica que se pasa a modo protegido.

**11 0000000B 0F22C0                    mov cr0,eax                    ;lo carga en cr0, pasa a modo protegido**

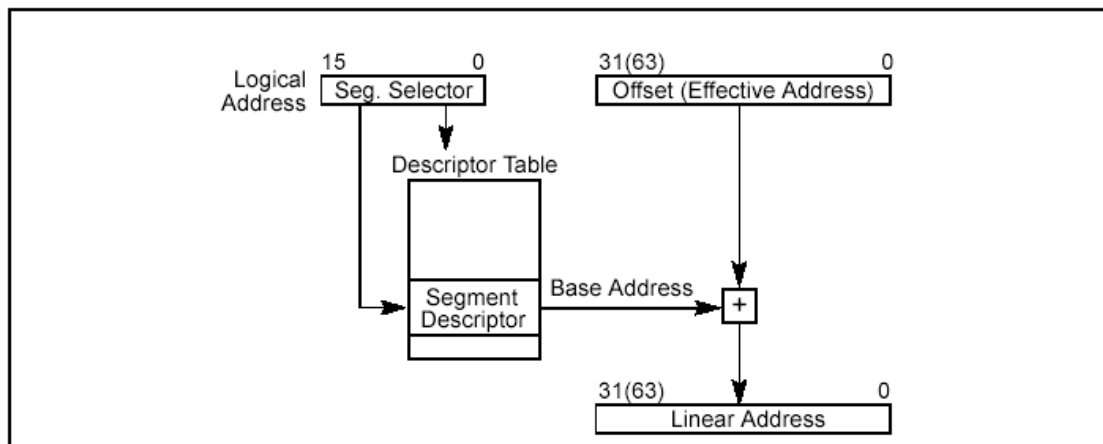
Se carga la nueva palabra de control con PE=1 (Protection Enable). En este punto el procesador pasa a modo protegido.

Como vemos el procesador esta en la posición Real 103e:010b, lo que indica una dirección lineal de 103eH x 10h + 010bH = 104EBH.

El segmento en modo real lo situa el Sistema operativo a partir del primer segmento libre de memoria, donde es carga nuestro programas.

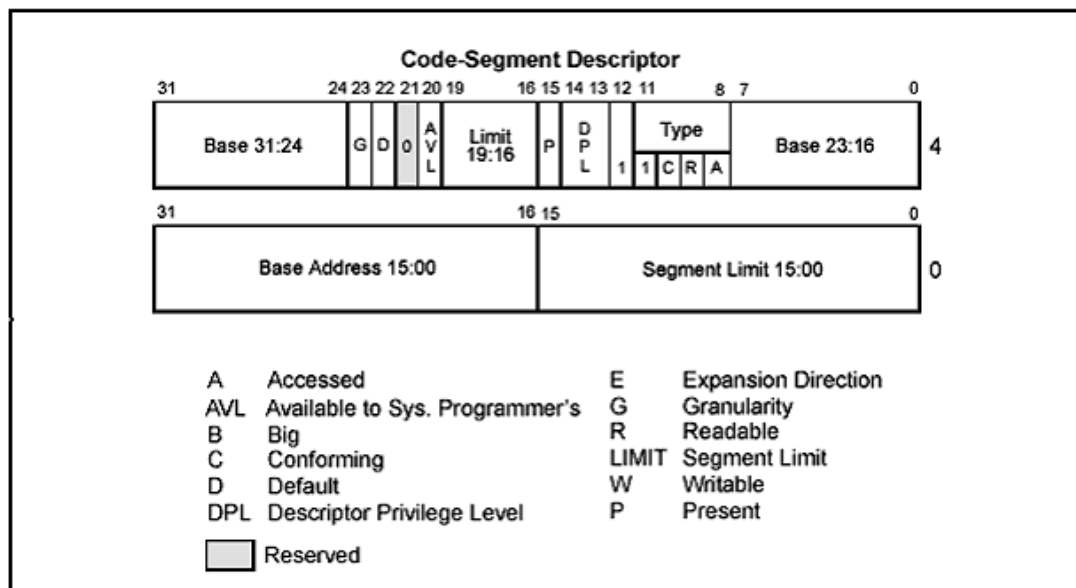
Al pasar a modo protegido, el procesador conmuta el modo de direccionamiento y pasa a 103e:000000000000010b. El 103e que en modo real indicaba el segmento, en modo protegido indica el selector, que por haber omitido su carga en este ejemplo puede ser cualquier dirección física. El desplazamiento que en modo real es de 16bit pasa a tener 32bits.

El direccionamiento en modo protegido se muestra a continuación:



**Figure 3-5. Logical Address to Linear Address Translation**

y el selector de código tiene la siguiente estructura:



**Figure 4-1. Descriptor Fields Used for Protection**

**12 0000000E E90000 jmp PROTEGIDO ;vacía la cola de instrucciones**

Al pasar a modo protegido tenemos almacenadas en el cache interno del procesador la cola de instrucciones en modo real, las cuales pueden traer algún tipo de inconveniente. Intel recomienda vaciar la cola mediante un jmp.

13  
 14  
 15

**PROTEGIDO:**

Inicia el Código en modo protegido.

**16 00000011 E460 in al,60h ;se lee el contenido del puerto 60h**

Leemos el teclado que esta disponible en el puerto de E/S 60H.

**17 00000013 FEC8 dec al ;se decrementa al para compararlo con 0**

Se busca que se presione la Tecla Esc que posee un ScanCode igual a 1. Al decrementar podemos comparar con cero, como se realiza a continuación.

**18 00000015 75FA jnz PROTEGIDO ;si no es ESC vuelve al loop**

19

Si AL es diferente a cero vuelve a leer el teclado. Cuando se presiona Esc sale de loop y continua.

**20 00000017 EBFE jmp \$**

Se vuelve a bloquear el programa para su depuración.

**21 00000019 0F20C0 mov eax,cr0 ;lee byte de control**

Se lee el Registro de Control CR0 para modificar solo el bit PE que indica el pasaje a modo real.

**22 0000001C 24FE and al,11111110b ;Pone en 0 PE**

Se pone PE=0 (Protection Enable) para pasar a modo real.

**23 0000001E 0F22C0 mov cr0,eax ;vuelve a modo real**

Se carga el registro de control con PE=0 y se vuelve a modo real.

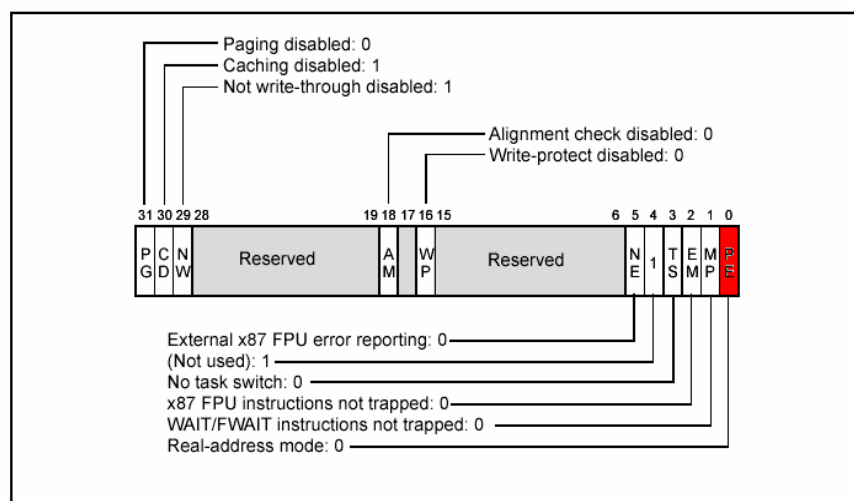


Figure 8-1. Contents of CR0 Register after Reset



## MODO PROTEGIDO

El programa escrito a continuación pasa a modo protegido, carga la Tabla de descriptores Globales, invierte el contenido de pantalla, y luego vuelve a modo real y le da el comando al DOS.

### Programa :

Autor:  
Gustavo Nudelman

```
1          use16          ;instrucciones de 16bits
2          org 100h      ;deja espacio porque es .com
3
4 00000000 E92600          jmp inicio          ;Salta al inicio del programa
5
6 00000003 00000000      gdt: dd 0          ; deja 8 Bytes para el primer descriptor
7 00000007 00000000          dd 0          ; nulo de la GDT
8
9          codsel equ $-gdt          ;desplazamiento del descriptor de código
10 0000000B FFFF          dw 0FFFFh      ;descriptor de código Base=0, limite=64kB
11 0000000D 0000          dw 0
12 0000000F 00          db 0
13 00000010 9A          db 10011010b
14 00000011 00          db 0
15 00000012 00          db 0
16
17          datsel equ $-gdt          ;desplazamiento del descriptor de datos
18 00000013 FFFF          dw 0FFFFh      ;descriptor de datos Base=0, limite=64kB
19 00000015 0000          dw 0
20 00000017 00          db 0
21 00000018 92          db 10010010b
22 00000019 00          db 0
23 0000001A 00          db 0
24
25          vdosel equ $-gdt          ;desplazamiento de otro descriptor de datos (video)
26 0000001B FFFF          dw 0FFFFh      ; descriptor de vides Base=0, limite=64kB
27 0000001D 0080          dw 08000h
28 0000001F 0B          db 0Bh
29 00000020 92          db 10010010b
30 00000021 00          db 0
31 00000022 00          db 0
32
33          gdtsize equ $-gdt          ; tamaño de la tabla de descriptores
34          ;define gdt ( registro de Gdt) como:
35 00000023 1F00          gdt: dw gdtsize-1          ;limite GDT
36 00000025 00000000          dd 0          ;Base
```

```

37
38          INICIO:
39          ; jmp $          ; paso a paso
40 00000029 6631C0      xor eax, eax          ;borra eax
41 0000002C 8CC8        mov ax, cs
42 0000002E A3[AD00]    mov [csreal],ax      ;guarda el valor de cs en la variable csreal
43 00000031 66C1E004    shl eax, 4           ;obtiene el dirección lineal del segmento
44
45 00000035 A3[0D00]    mov [gdt+codsel+2], ax ;carga base del descriptor de código
46 00000038 A3[1500]    mov [gdt+datsel+2], ax ; carga base del descriptor de datos
47
48 0000003B 66C1E810    shr eax, 16          ;parte más significativa de Cs
49 0000003F A2[0F00]    mov [gdt+codsel+4], al ;carga base del descriptor de código
50 00000042 A2[1700]    mov [gdt+datsel+4], al ;carga base del descriptor de datos
51 00000045 8826[1200]  mov [gdt+codsel+7], ah ;carga base del descriptor de código
52 00000049 8826[1A00]  mov [gdt+datsel+7], ah ;carga base del descriptor de datos
53
54 0000004D 6631C0      xor eax,eax          ;pone a cero eax
55 00000050 8CC8        mov ax,cs           ;lee nuevamente el segmento de código (M. real)
56 00000052 66C1E004    shl eax,4           ;obtiene la dirección lineal
57 00000056 6605[03000000] add eax, gdt         ;le suma el desplazamiento de la gdt
58 0000005C 66A3[2500]    mov [gdr+2], eax    ;carga la base del registro de la GDT en la
                                     ;dirección efectiva
59 00000060 0F0116[2300]  lgdt [gdr]         ;carga el registro de la GDT con la tabla que se
                                     ;definió en memoria

60
61 00000065 FA          cli                 ;deshabilita interrupciones
62 00000066 6631C0      xor eax, eax        ;borra eax
63 00000069 0F20C0      mov eax, cr0        ;carga el registro de control
64 0000006C 660D01000000    or eax, 1           ;setea PE ( Protected Enable)
65 00000072 0F22C0      mov cr0, eax        ;Pasa a modo protegido
66 00000075 EA[7A00]0800 jmp codsel:modoprot ;limpia la cola de instrucciones y salta
                                     ; a el primer selector de GDT ( selector código)

67
68          MODOPROT:
69 0000007A FB          sti                 ;habilita las interrupciones en modo protegido
70
71 0000007B B81800      mov ax, vdosel      ;lee descriptor de datos (video)
72 0000007E 8EC0        mov es, ax          ;y lo accede mediante el es
73 00000080 66BF01000000    mov edi,1           ;inicia un contador edi=1
74 00000086 6631F6      xor esi,esi
75 00000089 66B9D1070000    mov ecx,2000        ;la pantalla tiene 80x25=2000 caracteres
                                     ;cada carácter tiene un Byte de atributo
                                     ;y uno del carácter propiamente dicho

76
77          VIDEO:
78 0000008F B070        mov al, 70h         ;carga el modo inversos en el bit de
                                     ;atributos de pantalla
79 00000091 268805      mov [es:di],al     ;invierte el video en la posición de pantalla
80 00000094 6681C702000000 add edi, 2          ;pasa a la siguiente posición

```

```
81 0000009B E2F2      loop VIDES      ;Repite el ciclo
82
83
84          SALIDAMODPROT:
85
86 0000009D FA        cli              ;deshabilita interrupciones
87 0000009E 6631C0     xor eax, eax    ;borra eax
88 000000A1 0F20C0     mov eax, cr0    ;carga eax con el registro de control 0
89 000000A4 25FEFF     and ax, 0FFFEh ;resetea Pe(Protection Enable)=0
90 000000A7 0F22C0     mov cr0, eax    ;pasa a modo real
91
92 000000AA EA        db 0EAh        ;instrucción de salto por código de operación
93 000000AB [AF00]     dw MODOREAL    ;ya que el compilador no lo toma de otra manera
94 000000AD 0000     csreal dw 0    ;equivale jmp [csreal]:modoreal
95
96          MODOREAL:
97
98
99 000000AF A1[AD00]   mov ax, [csreal]
100 000000B2 8ED8     mov ds, ax      ;apunta segmento de datos
101 000000B4 8EC0     mov es, ax      ;apunta extraSegment
102 000000B6 FB        sti             ;Habilito las interrupciones y
103 000000B7 6631C0     xor eax, eax    ;vuelvo al DOS con el servicio
104 000000BA B04C     mov al, 4Ch    ;4Ch de la INT 21h
105 000000BC CD21     int 21h
```

**1 use16**

Indica al compilador que debe interpretar instrucciones de 16 bits

**2 org 100h**

Nuestro programa se carga a partir de un desplazamiento de 256bytes, este espacio es utilizado por el sistema y se denomina PSP ( Program Segment Prefix)

3

**4 00000000 E92600 jmp inicio ;Salta al inicio del programa**

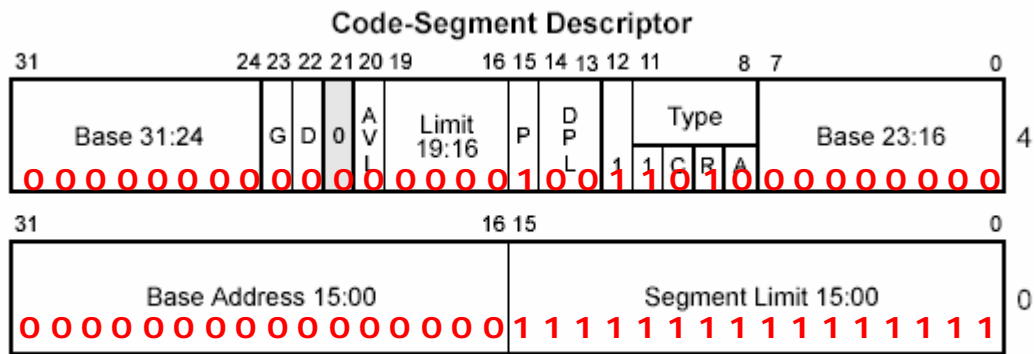
Salta al inicio del programa, pues luego de esta instrucción se colocan los datos y no hay que ejecutarlos.

**5**  
**6 00000003 00000000 gdt: dd 0 ; deja 8 Bytes para el primer descriptor**  
**7 00000007 00000000 dd 0 ; nulo de la GDT**

Carga el primer descriptor nulo en la Tabla de Descriptores Globales.

8			
9	<b>codsel equ \$-gdt</b>		<b>;desplazamiento del descriptor de código</b>
10	<b>0000000B FFFF</b>	<b>dw 0FFFFh</b>	<b>;descriptor de código Base=0, limite=64kB</b>
11	<b>0000000D 0000</b>	<b>dw 0</b>	
12	<b>0000000F 00</b>	<b>db 0</b>	
13	<b>00000010 9A</b>	<b>db 10011010b</b>	
14	<b>00000011 00</b>	<b>db 0</b>	
15	<b>00000012 00</b>	<b>db 0</b>	

Define el descriptor de código como sigue:



Base = 0  
 Limite = 64KB

Tipo:  
 A ( Acceso) =0 significa que el sistema no lo esta utilizando  
 R(read)=1 es de lectura  
 C(conforming)=0 no adapta los niveles de privilegio  
 bit 11 =1 Segmento de código

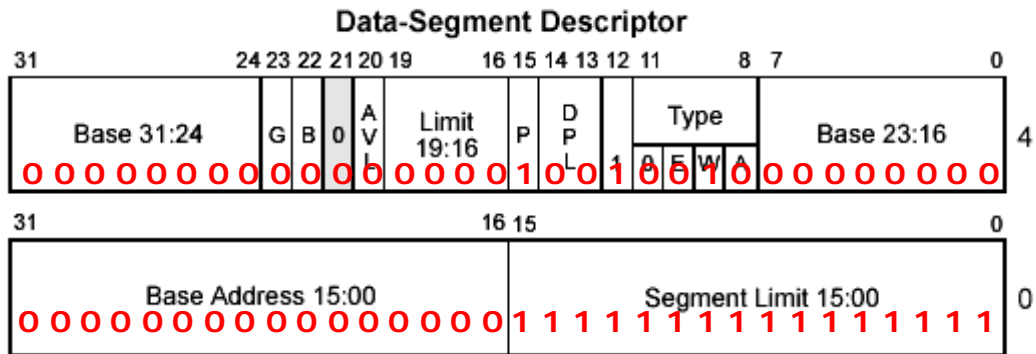
Bit 12 (S)=1 : No es un descriptor de sistema, sino de datos o de código.  
 DPL(Nivel de privilegio del descriptor). DPL=0 indica mayor nivel de privilegio  
 P(Presente)=1 el segmento esta presente

AVL(Disponibile para uso de sistema)=0 No esta disponible.  
 bit 21(Reservado)=0  
 D(default)=0 Indica como debe interpretar las instrucciones el procesador. En este caso como de 16 bits  
 G(granularidad)=0 . Multiplicador del limite. En este caso se multiplica por 1.

```

16
17          datsel equ $-gdt          ;desplazamiento del descriptor de datos
18 00000013 FFFF          dw 0FFFFh   ;descriptor de datos Base=0, limite=64kB
19 00000015 0000          dw 0
20 00000017 00           db 0
21 00000018 92           db 10010010b
22 00000019 00           db 0
23 0000001A 00           db 0
    
```

Define Descriptor de datos como sigue:



Base = 0  
 Limite = 64KB

Tipo:  
 A(Acceso)=0 No esta siendo accedido  
 W(write)=1 Es posible escribir en el segmento  
 E (sentido de Expansion)=0. El segmento se expande hacia arriba porque se trata de datos ( No de una pila).  
 bit 11=0 Segmento de Datos

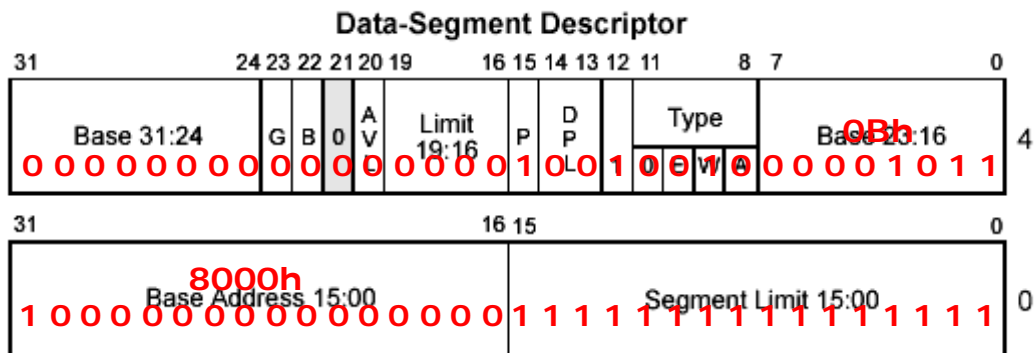
Bit 12 (S)=1 : No es un descriptor de sistema, sino de datos o de código.  
 DPL(Nivel de privilegio del descriptor). DPL=0 indica mayor nivel de privilegio  
 P(Presente)=1 el segmento esta presente

AVL(Disponible para uso de sistema)=0 No esta disponible.  
 bit 21(Reservado)=0  
 B(big)=0. Segmento de datos de 16 o 32bits. En este caso es de 16bits  
 G(granularidad)=0 . Multiplicador del limite. En este caso se multiplica por 1.

```

24
25          vdosel equ $-gdt          ;desplazamiento de otro descriptor de datos (vi
26 0000001B FFFF          dw 0FFFFh    ; descriptor de vides Base=0, limite=64kB
27 0000001D 0080          dw 08000h
28 0000001F 0B           db 0Bh
29 00000020 92           db 10010010b
30 00000021 00           db 0
31 00000022 00           db 0
    
```

Define Descriptor de datos para video como sigue :



Base = 000B8000h  
 Limite = 64KB

Tipo:

A(Acceso)=0 No esta siendo accedido

W(write)=1 Es posible escribir en el segmento

E (sentido de Expansion)=0. El segmento se expande hacia arriba porque se trata de datos ( No de una pila).

bit 11=0 Segmento de Datos

Bit 12 (S)=1 : No es un descriptor de sistema, sino de datos o de código.

DPL(Nivel de privilegio del descriptor). DPL=0 indica mayor nivel de privilegio

P(Presente)=1 el segmento esta presente

AVL(Disponible para uso de sistema)=0 No esta disponible.

bit 21(Reservado)=0

B(big)=0. Segmento de datos de 16 o 32bits. En este caso es de 16bits

G(granularidad)=0 . Multiplicador del limite. En este caso se multiplica por 1.

```

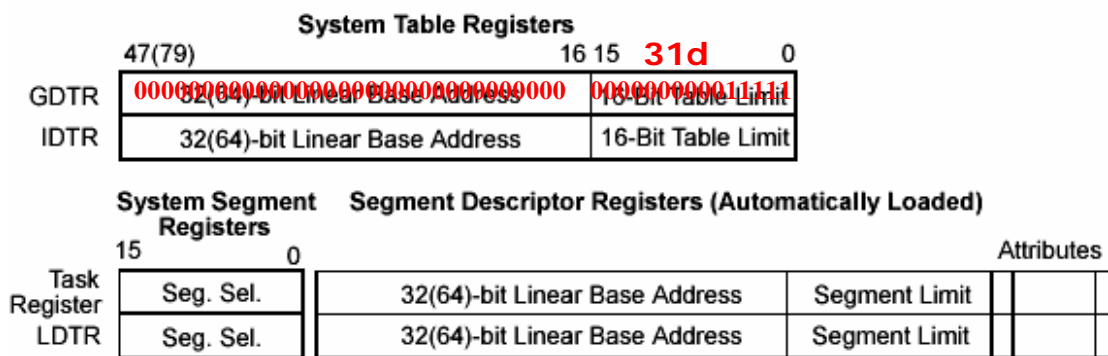
32
33          gdtsize equ $-gdt          ; tamaño de la tabla de descriptores
    
```

Calcula el tamaño de la tabla de descriptores y la coloca en una constante denominada gdtsize.

```

34          ;define gdtr ( registro de Gdt) como:
35 00000023 1F00          gdtr: dw gdtsize-1      ;limite GDT
36 00000025 00000000          dd 0              ;Base
37
    
```

Carga el límite en GDTR ( en el registro de tablas de sistema). En la tabla de descriptores globales (GDT) tenemos un primer descriptor nulo, luego uno de codigos, y dos de datos. En total son 4 descriptores de 8 Bytes cada uno, lo que nos da un total de  $8 \times 4 = 32$  Bytes. Por lo tanto el límite será de  $32 - 1 = 31$ . La base esta inicialmente en cero, ya que se cargará luego.



38 INICIO:  
 Inicio del programa.

```

39          ; jmp $          ; paso a paso
40 00000029 6631C0          xor eax, eax          ;borra eax
41 0000002C 8CC8          mov ax, cs
42 0000002E A3[AD00]          mov [csreal],ax      ;guarda el valor de cs en la variable csreal
    
```

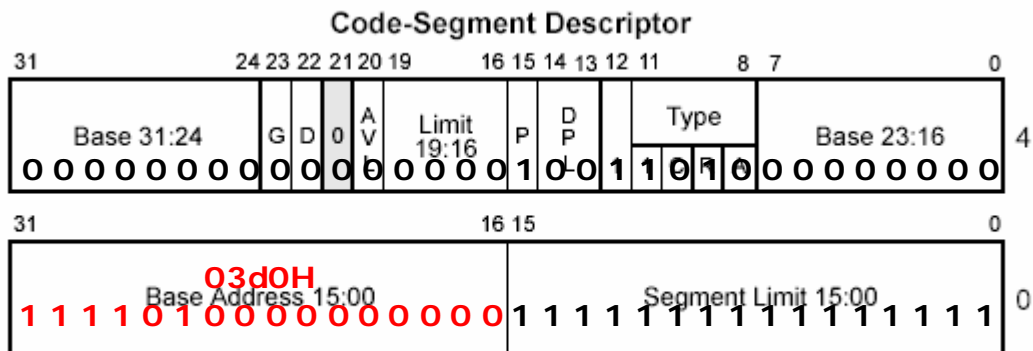
Como vemos el procesador esta en la posición Real  $103e:012e$ , lo que indica una dirección lineal de  $103eH \times 10h + 012eH = 1050eH$ .

El segmento en modo real lo sitúa el Sistema operativo a partir del primer segmento libre de memoria, donde es carga nuestro programas y es  $csreal = 103eH$

**43 00000031 66C1E004 shl eax, 4 ;obtiene el dirección lineal del segmento**

Si rotamos 4 posiciones a la derecha, tenemos  $eax=000103d0H$ , que es la posición lineal del segmento. En  $ax$  tenemos los 2Bytes menos significativos  $ax=03d0H$

**44**  
**45 00000035 A3[0D00] mov [gdt+codel+2], ax ;carga base del descriptor de código**  
**46 00000038 A3[1500] mov [gdt+datsel+2], ax ; carga base del descriptor de datos**

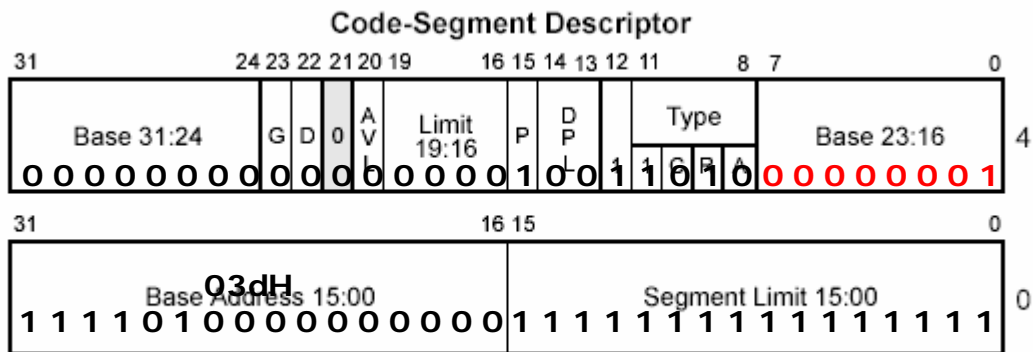


el segmento de datos se lo apunta a la misma posición.

**47**  
**48 0000003B 66C1E810 shr eax, 16 ;parte más significativa de Cs**

En  $eax$  teníamos la posición lineal del segmento de código, es decir,  $eax=000103d0H$ , si lo rotamos 16 bit, o 4 posiciones obtenemos  $eax=00000001H$ , es decir “AX” tenemos los dos Byte más significativo de la dirección lineal, es decir,  $AX= 0001H$ . Como la base esta dividida en dos partes en descriptor, usamos  $AL=01H$  y  $AH=00H$

**49 0000003F A2[0F00] mov [gdt+codel+4], al ;carga base del descriptor de código**  
**50 00000042 A2[1700] mov [gdt+datsel+4], al ;carga base del descriptor de datos**  
 Cargamos el primer Byte de la parte más significativa de la base, como se ve a continuación.

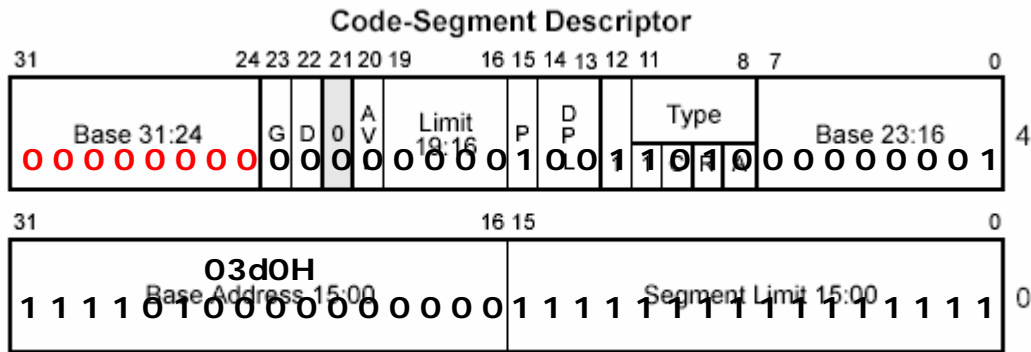


el segmento de datos se lo apunta a la misma posición.

```

51 00000045 8826[1200]    mov [gdt+codel+7], ah    ;carga base del descriptor de código
52 00000049 8826[1A00]    mov [gdt+datsel+7], ah  ;carga base del descriptor de datos
    
```

Cargamos el segundo Byte más significativa de la base:



el segmento de datos se lo apunta a la misma posición.

```

53
54 0000004D 6631C0        xor eax,eax                ;pone a cero eax
55 00000050 8CC8          mov ax,cs                  ;lee nuevamente el segmento de código (M. real
56 00000052 66C1E004       shl eax,4                  ;obtiene la dirección lineal
    
```

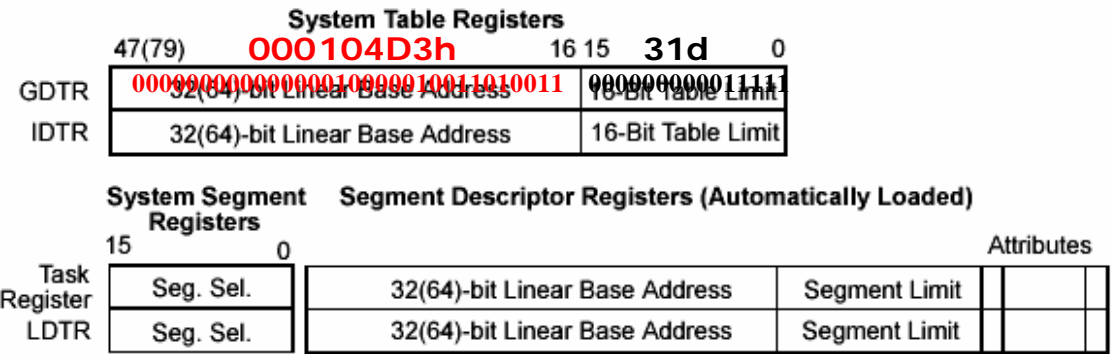
nuevamente cargamos en la dirección efectiva del segmento  $eax=000103d0H$ .

```

57 00000056 6605[03000000]  add eax, gdt                ;le suma el desplazamiento de la gdt
58 0000005C 66A3[2500]      mov [gdt+2], eax           ;carga la base del registro de la GDTR en la
    
```

**;dirección efectiva**

en  $eax=000103d0H$  tenemos la dirección del segmento, si le sumamos el desplazamiento de `gdt` obtenemos la dirección lineal de `gdt`, por lo tanto sumamos:  
 $000103d0H(\text{Segmento}) + 100H(\text{desplazamiento .com}) + 3(\text{desplazamiento gdt}) = 000104d3h$   
 Tenemos  $eax=000104d3H$ . Lo cargamos en la parte del registro de la tabla de descriptores globales (GDTR) en la parte de base, es decir, en `gdt+2`:



```
59 00000060 0F0116[2300]    lgdt [gdtr]                ;carga el registro de la GDT con la tabla que se
                                ;definió en memoria
```

Cargamos el registro de la tabla de descriptores globales(GDTR) como la habíamos definido en memoria

```
60
61 00000065 FA                cli                          ;deshabilita interrupciones
62 00000066 6631C0           xor eax, eax                ;borra eax
63 00000069 0F20C0           mov eax, cr0                ;carga el registro de control
64 0000006C 660D01000000     or  eax, 1                  ;setea PE ( Protected Enable)
65 00000072 0F22C0           mov cr0, eax                ;Pasa a modo protegido
```

Ponemos PE(Protection Enable) =1 del registro de control CR0 para pasar a modo protegido.

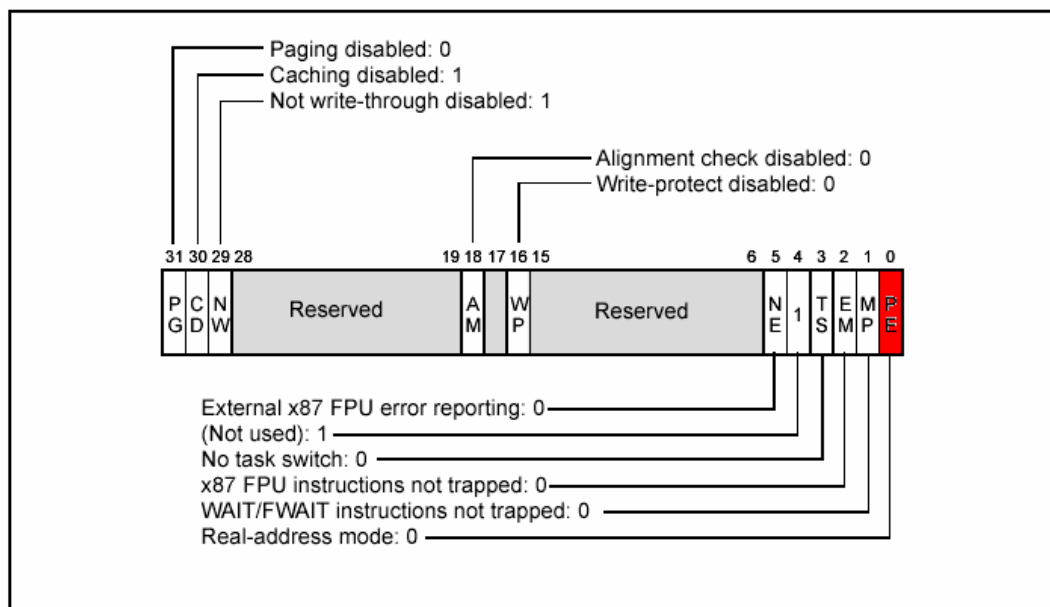


Figure 8-1. Contents of CR0 Register after Reset

En este punto el procesador que debía pasar a la dirección de segmento más desplazamiento de 103d:0175, en realidad pasa a la dirección 103d:0000000000000175, lo que significa que esta dada por el descriptor 103d que en realidad todavía no está definido y se determina en la siguiente instrucción, y por un desplazamiento, que ahora es de 32 bits.

```
66 00000075 EA[7A00]0800     jmp codsel:modoprot        ;limpia la cola de instrucciones y salta
                                ; a el primer selector de GDT ( selector código)
```

Se realiza un salto largo (jmp far) al primer descriptor de la GDT, es decir que el procesador, que interpretaría la dirección siguiente como 103d:000000000000017A, ahora la interpreta como 0008:000000000000017A. La dirección lineal sigue siendo la misma.

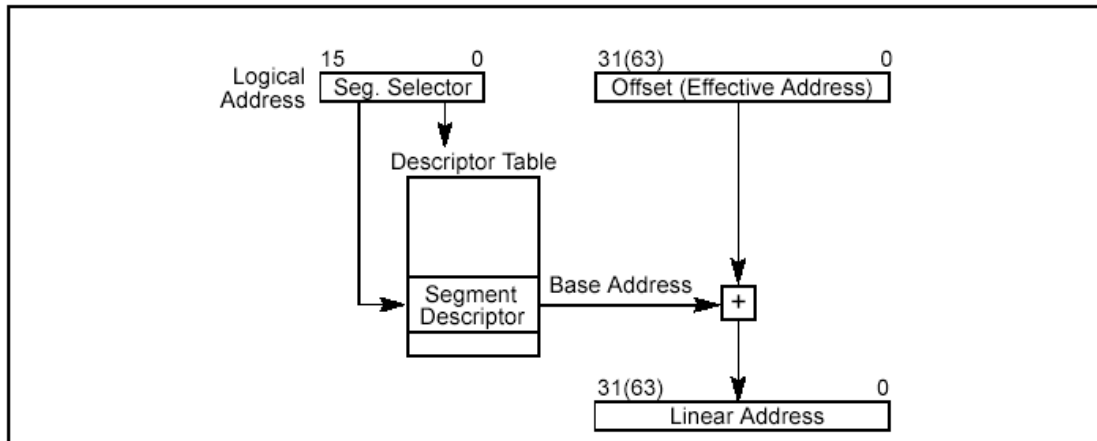


Figure 3-5. Logical Address to Linear Address Translation

El selector apunta al 0008 porque en realidad es el primer descriptor de la tabla de descriptores Globales, dado que los tres bit menos significativos definen la si utiliza la tabla global (GDT) o local(LDT), y el nivel de privilegio (RPL)

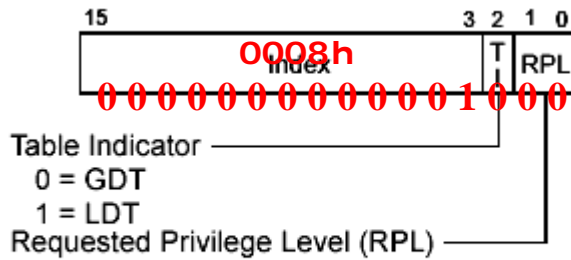


Figure 3-6. Segment Selector

Notar además que el número de selector coincide con el desplazamiento en la Gdt, es decir, si yo quiero acceder al segmento de video y se que esta cargado en el tercer descriptor de la GDT como un selector de datos, coloco en el selector: 3 en el index, TI=0,RPL=0,es decir 0000000000000011000=24 y coincide con el desplazamiento en gdt ya que cada selector ocupa 8 Bytes y como es el tercero hago 8x3=24. Como conclusión decimos que puedo acceder a un segmento por el desplazamiento en la gdt ó por el número de selector porque son el mismo número.

```

67
68          MODOPROT:
Aquí comienza el programa que invierte la pantalla en si. Lo anterior es para pasar a modo protegido.

69 0000007A FB          sti          ;habilita las interrupciones en modo protegido
una vez que paso a modo protegido debo habilitar las interrupciones.
    
```

70

```
71 0000007B B81800      mov ax, vdosel      ;lee descriptor de datos (video)
72 0000007E 8EC0       mov es, ax          ;y lo accede mediante el es
```

con el ExtraSegment apunto a video.

```
73 00000080 66BF01000000  mov edi,1          ;inicia un contador edi=1
74 00000086 6631F6       xor esi,esi
75 00000089 66B9D1070000  mov ecx,2000      ;la pantalla tiene 80x25=2000 caracteres
                                       ;cada carácter tiene un Byte de atributo
                                       ;y uno del carácter propiamente dicho
```

Inicio variables del programa

76

```
77                               VIDEO:
78 0000008F B070       mov al, 70h       ;carga el modo inversos en el bit de
                                       ;atributos de pantalla
```

Cada posición de pantalla en modo texto esta formada por 2 Bytes, el primero es el código del carácter, y el segundo es el atributo. Nosotros tenemos que cambiar este segundo Byte para invertir la pantalla. El 70H pone el bit de atributo 11100000. **El modo inverso se debe observar como se muestra aquí**

```
79 00000091 268805       mov [es:di],al    ;invierte el video en la posición de pantalla
80 00000094 6681C702000000  add edi, 2        ;pasa a la siguiente posición
81 0000009B E2F2       loop VIDES        ;Repite el ciclo
```

Se repite el ciclo para todos los caracteres de la pantalla. Se suma de a dos porque debemos escribir solo el Byte de atributo y no el del código del carácter. La pantalla tiene 80x25=2000 caracteres, por lo tanto cuando ecx llega a c sale del loop.

82

83

```
84                               SALIDAMODPROT:
```

85

Una vez invertida la pantalla volvemos al modo real.

```
86 0000009D FA       cli              ;deshabilita interrupciones
```

Se deshabilitan las interrupciones para que no salte en la conmutación de modo de funcionamiento.

```
87 0000009E 6631C0       xor eax, eax      ;borra eax
88 000000A1 0F20C0       mov eax, cr0     ;carga eax con el registro de control 0
89 000000A4 25FEFF       and ax, 0FFFEh   ;resetea Pe(Protection Enable)=0
90 000000A7 0F22C0       mov cr0, eax     ;pasa a modo real
```

Se pone en cero el bit PE(Protection Enable)=0 para volver al modo real. Este bit esta en el registro de control CR0.

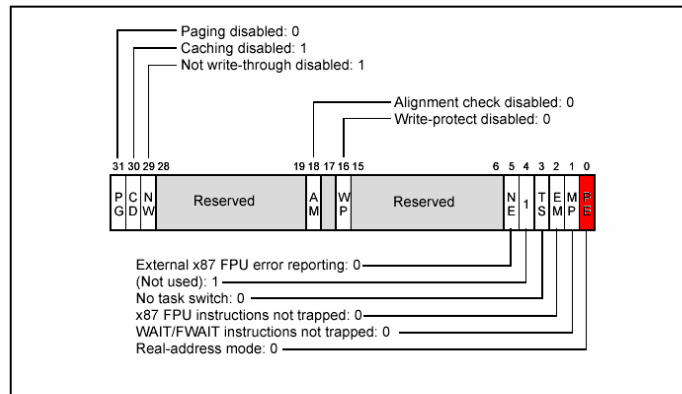


Figure 8-1. Contents of CR0 Register after Reset

En este punto el procesador que tenia que pasar a la posición de memoria dada por la base del selector más el desplazamiento 0008:00000000000001AA pasa a la posición de modo real 0008: 01AA. El procesador ya esta en modo real pero resta redefinir el segmento.

```

91
92 000000AA EA          db 0EAh          ;instrucción de salto por código de operación
93 000000AB [AF00]     dw MODOREAL      ;ya que el compilador no lo toma de otra manera
94 000000AD 0000      csreal dw 0          ;equivale jmp far [csreal]:modoreal
95
96
    
```

El compilador “nasm” no puede interpreta un salto de éste estilo, a no ser que se realice mediante los códigos de operación. Estas tres líneas equivales a **jmp far [csreal]:modórrela**. Recuerde que esta instrucción no puede ser interpretada por el compilador.

Ahora si, en vez de 0008: 01AF pasamos a la posición de modo real 103d:01AF.

De paso también se limpia la cola de instrucciones.

```

97          MODOREAL:
98
99 000000AF A1[AD00]    mov ax, [csreal]
100 000000B2 8ED8      mov ds, ax          ;apunta segmento de datos
101 000000B4 8EC0      mov es, ax          ;apunta extraSegment
102 000000B6 FB        sti                ;Habilito las interrupciones
    
```

Ya en modo real apuntamos el segmento de datos (ds) y el ExtraSegmen(es) a la misma posición que el segmento de codigo (cs) de modo real.

Luego habilitamos las interrupciones

```

103 000000B7 6631C0    xor eax, eax        ;vuelvo al DOS con el servicio
104 000000BA B04C      mov al, 4Ch         ;4Ch de la INT 21h
105 000000BC CD21      int 21h
    
```

Devolvemos el control al DOS. Recién en este punto se invierte la pantalla ya que el DOS se encarga del refresco de pantalla.

# INTERRUPCIONES

## en modo protegido

Este programa pasa a modo protegido, donde permanece, modificando el primer byte de pantalla en cada interrupción del timer tick y leyendo la dirección de E/s de teclado hasta que se presiona la tecla "ESC", y vuelve a modo real devolviendo el control al DOS.

```
1          ; programa con interrupciones
2 3          ;
4          ;
5          ; Autor:
6          ; Mariano Cerdeiro
7 8 9
10
11          use16          ;instrucciones de 16 bits
12          org 100h      ;archivo .com
13 00000000 E9F600      comienzo: jmp inicio          ;salta al inicio del programa
14
15 00000003 456C2070726F636573- modo_virtual_msg db      'El procesador se encuentra en modo virtual, no se puede correr la aplicación.'
16 0000000C 61646F722073652065-
17 00000015 6E6375656E74726120-
18 0000001E 656E206D6F646F2076-
19 00000027 69727475616C2C206E-
20 00000030 6F2073652070756564-
21 00000039 6520636F7272657220-
22 00000042 6C612061706C696361-
23 0000004B 6369F36E2E24
24
25 00000051 00          mask_real_pic1 db      0          ;variable para almacenar el PIC ( controlador
                                     ; de interrupciones programable)
26
27 00000052 <res 00006> real_idtr      resb      6          ; gurada idtr de modo real
28
29 00000058 00          scan_code      db      0          ;variable para teclas presionadas
30
31 00000059 2700          gdtr:          dw      8*5-1      ;limite gdtr (5 descriptores)
32 0000005B 00000000          dd      0
33
34 0000005F 4F00          idtr          dw      8*10-1     ;límite idtr(10 descriptores, hasta la interrupción 9)
35 00000061 00000000          dd      0
```

```

36
37 00000065 <res 00000008> gdt resb 8 ;reserva 8 Bytes para el primer descriptor nulo
38
39
40 sel_cod: ; Selector de código
41 0000006D FFFF dw 0ffffh ;Límite 64K
42
43 0000006F 0000 dw 0
44 00000071 00 db 0
45 00000072 9A db 10011010b ;P PL #S Tipo
46
47 00000073 00 db 00000000b
48 00000074 00 db 0
49
50 sel_datos: ;Selector de datos
51
52 00000075 FFFF dw 0ffffh ;limite 64K
53 00000077 0000 dw 0
54 00000079 00 db 0
55 0000007A 92 db 10010010b ;Atributo
56 0000007B 00 db 0b
57 0000007C 00 db 0
58
59
60 ;Selector de datos de vides
61 0000007D FFFF dw 0ffffh ;límite 64k
62 0000007F 0000 dw 00000h
63 00000081 00 db 000h
64 00000082 92 db 10010010b ;Presente Segmento Datos Read Write.
65 00000083 8F db 10001111b ;G y límite 0Fh
66 00000084 00 db 00h
67
68 00000085 FFFF dw 0ffffh ; recomendado por Intel para compatibilidad modo real
69 00000087 0000 dw 0
70 00000089 00 db 0
71 0000008A 92 db 10010010b ;Atributo
72 0000008B 00 db 0
73 0000008C 00 db 0
74
75 0000008D <res 00000040> idt: resb 8*8 ;se dejan las primeras 8 interrupciones sin inicializar

```

```

76                                     ;la siguiente es Timer Tick tipo 8 ( ya que arranca de 0)
77         irq0:                       ; Selector de IRQ0 ó tipo 8 ( Timer Tick T=55ms)
78
79 000000CD 0000         dw 0
80 000000CF 0800         dw 08h           ;Apunta al primer selector, el de código
81 000000D1 00          db 0
82 000000D2 86          db 10000110b       ;Atributo
83 000000D3 00          db 0
84 000000D4 00          db 0
85
86         irq0_han:                   ;rutina de atención a interrupción timer tick
87
88 000000D5 1E          push ds           ;guarda ds
89 000000D6 6660         pushad          ;guarda registros de propósitos generales
90
91 000000D8 B81800       mov ax,18h
92 000000DB 8ED8        mov ds, ax       ;ds apunta al 3 selector (datos)
93 000000DD 66B800800B00 mov eax,0b8000h ;offset video
94 000000E3 67FE00       inc byte [eax]   ;Se incrementa primer Byte de pantalla
                                     ;lo que produce un cambio de caracteres de la
                                     ;primera posición de pantalla
95
96 000000E6 B81000       mov ax,10h
97 000000E9 8ED8        mov ds, ax       ;ds apunta al 2 selector (datos)
98
99 000000EB E460        in al, 60h       ;dirección de E/S de teclado
100 000000ED 3EA2[5800]  mov [ds:scan_code],al ;guarda el código de tecla en la posición scancode
101
102 000000F1 B020       mov al,20h
103 000000F3 E620       out 20h,al       ; le indica al PIC ( port 20h) que atendió a la interrupción
104
105 000000F5 6661        popad           ;restaura registros de propósitos generales
106 000000F7 1F          pop ds          ;restaura ds
107 000000F8 CF          iret           ; retorno de interrupción
108
109         inicio:                   ; inicio del programa en modo real
110
111         ;jmp $                       ;para debugger
112 000000F9 8C0E[8401]  mov [real_cs],cs ;guarda cs real en variable
113

```

```

114 00000FD 6631C0      xor eax,eax                ;eax=0
115 00000100 8CC8          mov ax,cs                 ;almacena CS en eax.
116 00000102 66C1E004      shl eax,4                 ;dirección lineal de cs
117 00000106 6689C3          mov ebx,eax               ;se guarda en ebx.
118
119 00000109 A3[6F00]        mov [sel_cod+2],ax        ;se carga la base del selector de código
120 0000010C A3[7700]        mov [sel_datos+2],ax     ;se carga la base del selector de datos
121
122 0000010F 66C1E810      shr eax,16                ;gira eax y obtiene parte más significativa de cs real
123 00000113 A2[7100]        mov [sel_cod+4],al        ;se carga la base del selector de código
124 00000116 A2[7900]        mov [sel_datos+4],al     ;se carga la base del selector de datos
125
126 00000119 6689D8          mov eax,ebx               ;vuelve a leer dirección lineal de cs real
127 0000011C 6605[65000000] add eax,gdt                ;obtiene dirección lineal de gdt
128 00000122 66A3[5B00]      mov [gdt+2],eax           ;Base de gdt
129 00000126 0F0116[5900]  lgdt [gdt]                ;se carga el GDTR.
130
131                ;--- Se inicializa la IDT.
132
133 0000012B B8[D500]      mov ax,irq0_han           ;lee offset de la rutina de atención a interrupción
134 0000012E A3[CD00]        mov [irq0],ax             ;lo pone en la base de IDTR
135
136                ;--- Se carga la IDT.
137
138 00000131 0F010E[5200]  sidt [real_idtr]          ;guarda la dirección de idtr real
139 00000136 6689D8          mov eax,ebx               ;lee la dirección lineal de cs real
140 00000139 6605[8D000000] add eax,idt                ;suma offset a idt
141 0000013F 66A3[6100]      mov [idtr+2],eax          ;guarda en la base de IDTR
142 00000143 0F011E[5F00]  lidt [idtr]               ;carga IDTR de modo protegido
143
144                ;--- A modo protegido
145
146 00000148 FA          cli                       ;deshabilita interrupciones
147 00000149 0F20C0          mov eax,cr0               ;lee el registro de control CRO
148 0000014C 0C01          or al,1                   ;setea bit menos significativo PE(protection enable)
149 0000014E 0F22C0          mov cr0,eax               ;carga en el registro de control y pasa a modo protegido
150
151 00000151 EA[5601]0800  jmp 08h:protegido         ;salta al segmento de código de modo protegido
152
153

```

```

154                protegido:                ;modo protegido
155
156 00000156 B81000    mov ax,10h                ;carga selector de datos (segundo selector)
157 00000159 8ED8     mov ds, ax                ;en ds
158 0000015B E421     in al,21h                ;lee el PICmodo real (controlador interrupciones)
159 0000015D A2[5100]  mov [mask_real_pic1],al  ;guarda en variable
160 00000160 B0FE     mov al,011111110b        ;habilita solo Irq0(timer tick T=55ms)
161 00000162 E621     out 21h,al                ;programa PIC modo protegido
162
163 00000164 FB       sti                        ;Setear las interrupciones en modo protegido.
164
165                esperar:                ;label de espera
166
167 00000165 2E803E[5800]01 cmp byte [cs:scan_code],1 ; compara el código de tecla presionada con ESC
168 0000016B 75F8     jne esperar                ; Si no es Esc espera ( notar que salta cada 55ms a
                                ;leer teclado en la rutina de atención a interrupción
169
170 0000016D FA       cli                        ;deshabilita interrupciones. Comienza el retorno a modo real
171
172 0000016E B82000    mov ax,20h
173 00000171 8ED8     mov ds, ax                ;apunta todos los selectores a un segmento de
174 00000173 8EC0     mov es,ax                ;datos como recomienda intel ( tercer selector)
175 00000175 8EE0     mov fs,ax
176 00000177 8EE8     mov gs, ax
177
178
179 00000179 0F20C0    mov eax,cr0                ;lee Registro de control
180 0000017C 24FE     and al,0feh                ;resetea PE(Protection Enable)
181 0000017E 0F22C0    mov cr0,eax                ;pasa a modo real
182 00000181 EA       db 0eah                    ;salta a segmento modo real, equivale a jmp far real
183 00000182 [8601]   dw real                    ;por medio de código de operación, ya que el compilador
184 00000184 0000    real_cs dw 0                ;no lo ensambla de otro modo. Vacía cola de instrucciones
185
186
187                real:                ;retorno a modo real
188
189                ;--- Se carga el selector de modo real en todos los segmentos.
190 00000186 8CC8     mov ax, cs
191 00000188 8ED8     mov ds,ax                ;se recupera el segmento de modo real
192 0000018A 8EC0     mov es,ax

```

```

193 0000018C 8EE0      mov fs,ax
194 0000018E 8EE8      mov gs, ax
195
196                ;--- Se recuperan la idt y las máscaras.
197
198 00000190 0F011E[5200]  lidt [real_idtr]
199 00000195 A0[5100]      mov al,[mask_real_pic1]
200 00000198 E621      out 21h,al
201
202
203                ; Se carga el IDTR de modo real que se guardo antes de pasar a modo real
204                ; y se recupera la máscara del PIC de modo real.
205
206
207 0000019A FB          sti                ;se habilitan interrupciones de modo real
208 0000019B B44C      mov ah,4ch
209 0000019D CD21      int 21h           ;vuelve al DOS

```

## EXPLICACIÓN:

```

1                ; programa con interrupciones
2                ;
3                ; Protected Mode by Examples revision 0.1 - Octubre 2004
4                ;
5                ; Autor:
6                ; Mariano Cerdeiro
7                ;
8                ;
9
10
11                use16                ;instrucciones de 16 bits
12                org 100h            ;archivo .com
13 00000000 E9F600      comienzo:jmp inicio                ;salta al inicio del programa

```

14

```

15 00000003 456C2070726F636573- modo_virtual_msg db 'El procesador se encuentra en modo virtual, no se puede correr la aplicación.S'
16 0000000C 61646F722073652065-
17 00000015 6E6375656E74726120-
18 0000001E 656E206D6F646F2076-
19 00000027 69727475616C2C206E-
20 00000030 6F2073652070756564-
21 00000039 6520636F7272657220-
22 00000042 6C612061706C696361-
23 0000004B 6369F36E2E24

```

24

```

25 00000051 00          mask_real_pic1 db 0          ;variable para almacenar el PIC ( controlador
; de interrupciones programable)

```

26

```

27 00000052 <res 00006> real_idtr resb 6          ; gurada idtr de modo real

```

28

```

29 00000058 00          scan_code db 0          ;variable para teclas presionadas

```

30

```

31 00000059 2700          gdtr: dw 8*5-1          ;limite gdtr (5 descriptores)

```

```

32 0000005B 00000000          dd 0

```

33

Se carga el limite de gdtr para 5 descriptores (Codigo, datos, datos de video, datos para pasar a modor real según recomienda intel). Como los descriptores son de 8 Bites el limite es  $8 * 5 - 1 = 39 = 27h$ .

```

34 0000005F 4F00          idtr dw 8*10-1          ;límite idtr( 8 descriptores, hasta la interrupción 7)

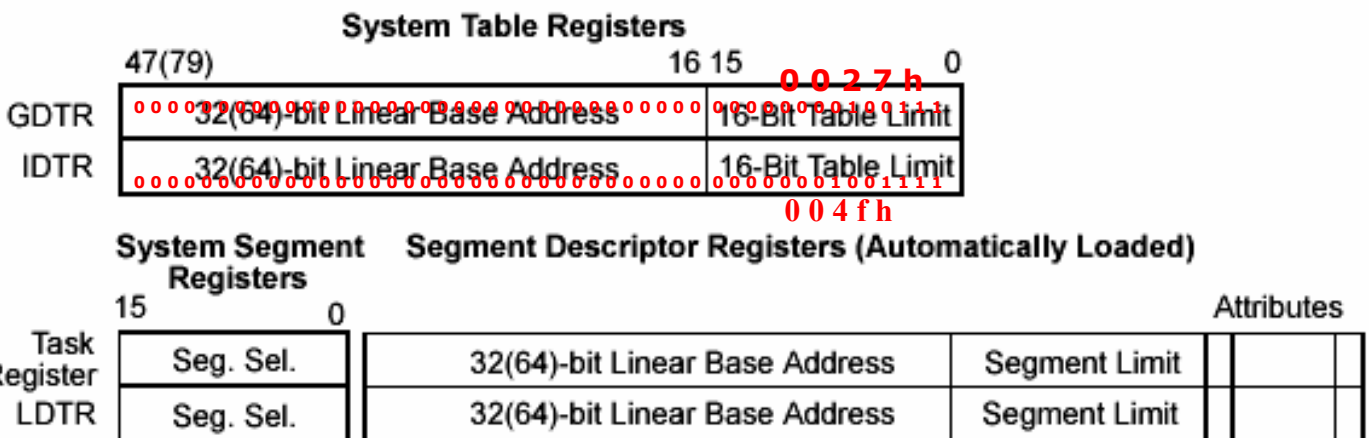
```

```

35 00000061 00000000          dd 0          ;Base em p

```

Cargamos el límite de IDTR. Ya que tendremos 9 interrupciones ( hasta la tipo 8 contando de cero), pondremos el limite en 10. Como cada descriptor de compuerta de interrupción ocupa 8 Bites, tenemos: :



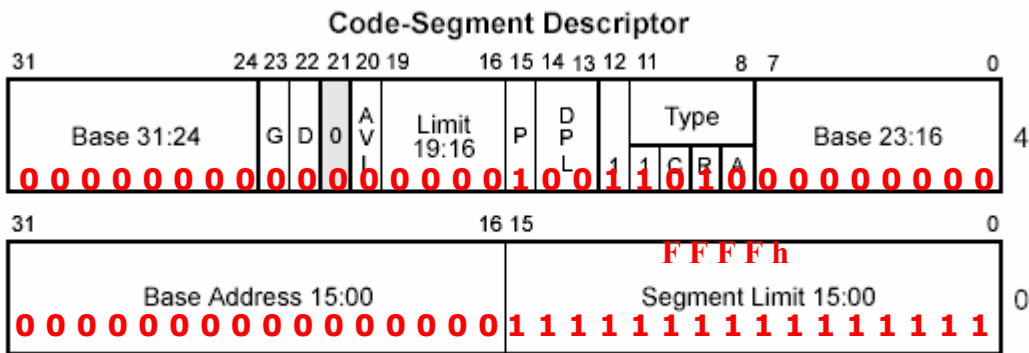
El límite esta cargado en  $8 \times 10^{-1} = 79 = 4fh$ .

```

36
37 00000065 <res 00000008> gdt resb 8           ;reserva 8 Bytes para el primer descriptor nulo
38
39
40                sel_cod:                       : Selector de código
41 0000006D FFFF      dw 0ffffh                 ;Límite 64K
42
43 0000006F 0000      dw 0
44 00000071 00        db 0
45 00000072 9A        db 10011010b             ;P PL #S Tipo
46
47 00000073 00        db 00000000b
48 00000074 00        db 0

```

Define el descriptor de código como sigue GDT + 8H :



Base = 0  
 Limite = 64KB

Tipo:  
 A ( Acceso) =0 significa que el sistema no lo esta utilizando  
 R(read)=1 es de lectura  
 C(conforming)=0 no adapta los niveles de privilegio  
 bit 11 =1 Segmento de código

Bit 12 (S)=1 : No es un descriptor de sistema, sino de datos o de código.  
 DPL(Nivel de privilegio del descriptor). DPL=0 indica mayor nivel de privilegio

P(Presente)=1 el segmento esta presente

AVL(Disponible para uso de sistema)=0 No esta disponible.

bit 21(Reservado)=0

D(default)=0 Indica como debe interpretar las instrucciones el procesador. En este caso como de 16 bits

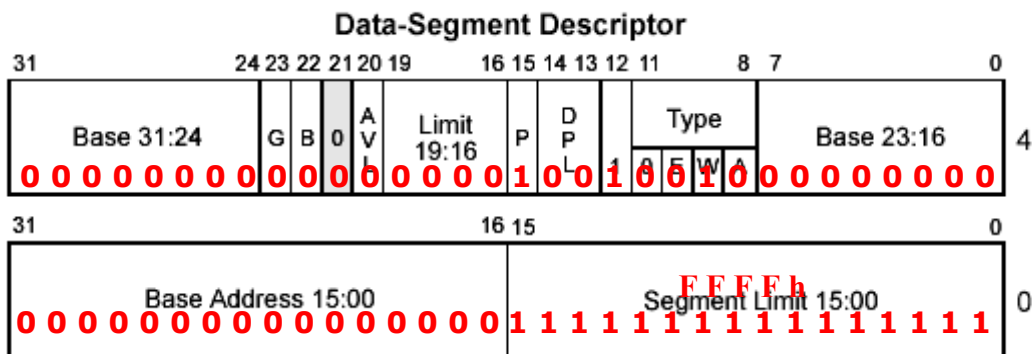
G(granularidad)=0 . Multiplicador del limite. En este caso se multiplica por 1.

```

49
50             sel_datos:                               ;Selector de datos
51
52 00000075 FFFF      dw 0ffffh                       ;limite 64K
53 00000077 0000      dw 0
54 00000079 00        db 0
55 0000007A 92        db 10010010b                   ;Atributo
56 0000007B 00        db 0b
57 0000007C 00        db 0
58
59

```

Define Descriptor de datos como en GDT + 10H sigue:



Base = 0  
 Limite = 64KB

Tipo:

A(Acceso)=0 No esta siendo accedido

W(write)=1 Es posible escribir en el segmento

E (sentido de Expansion)=0. El segmento se expande hacia arriba porque se trata de datos ( No de una pila).

bit 11=0 Segmento de Datos

Bit 12 (S)=1 : No es un descriptor de sistema, sino de datos o de código.  
 DPL(Nivel de privilegio del descriptor). DPL=0 indica mayor nivel de privilegio  
 P(Presente)=1 el segmento esta presente

AVL(Disponible para uso de sistema)=0 No esta disponible.

bit 21(Reservado)=0

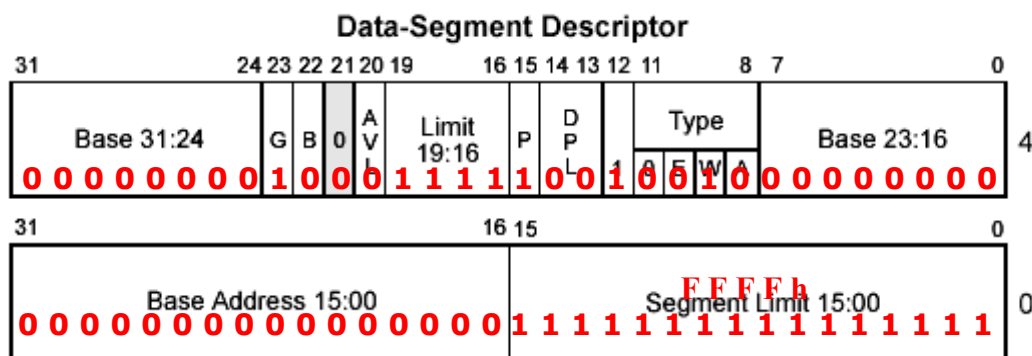
B(big)=0. Segmento de datos de 16 o 32bits. En este caso es de 16bits

G(granularidad)=0 . Multiplicador del limite. En este caso se multiplica por 1.

```

60                                     ;Selector de datos de vides
61 0000007D FFFF          dw 0ffffh          ;límite 64k
62 0000007F 0000          dw                00000h
63 00000081 00          db 000h
64 00000082 92          db 10010010b          ;Presente Segmento Datos Read Write.
65 00000083 8F          db 10001111b          ;G y límite 0Fh
66 00000084 00          db 00h
67
  
```

Define un segundo descriptor de datos en GDT + 18H que apunta a video (flat)



Base = 0  
 Limite = 4GB (1MB \* 4k(G=1))

Tipo:

A(Acceso)=0 No esta siendo accedido

W(write)=1 Es posible escribir en el segmento

E (sentido de Expansion)=0. El segmento se expande hacia arriba porque se trata de datos ( No de una pila).

bit 11=0 Segmento de Datos

Bit 12 (S)=1 : No es un descriptor de sistema, sino de datos o de código.

DPL(Nivel de privilegio del descriptor). DPL=0 indica mayor nivel de privilegio

P(Presente)=1 el segmento esta presente

AVL(Disponible para uso de sistema)=0 No esta disponible.

bit 21(Reservado)=0

B(big)=0. Segmento de datos de 16 o 32bits. En este caso es de 16bits

G(granularidad)=1. Multiplicador del limite. En este caso se multiplica por 4k

```

68 00000085 FFFF      dw 0ffffh      ; recomendado por Intel para compatibilidad modo real

69 00000087 0000      dw 0

70 00000089 00        db 0

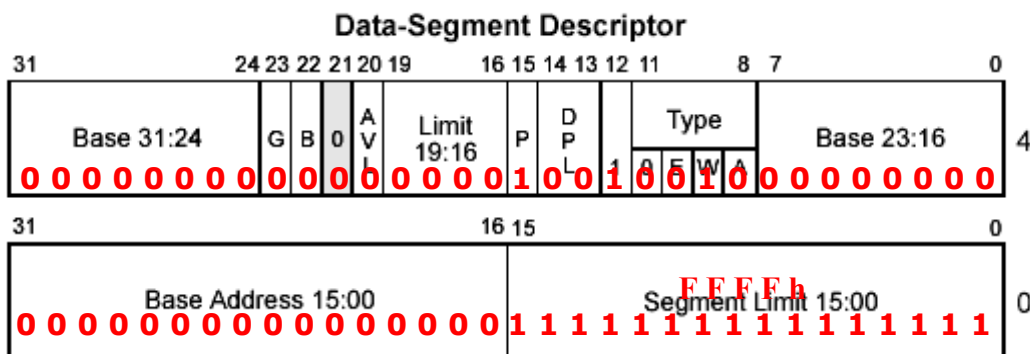
71 0000008A 92        db 10010010b      ;Atributo

72 0000008B 00        db 0

73 0000008C 00        db 0

74
    
```

Define un tercer descriptor de datos en GDT + 20H para apuntar los registros a un mismo segmento al volver a modo real, como recomienda Intel:



Base = 0  
 Limite = 64KB

Tipo:

A(Acceso)=0 No esta siendo accedido

W(write)=1 Es posible escribir en el segmento

E (sentido de Expansion)=0. El segmento se expande hacia arriba porque se trata de datos ( No de una pila).

bit 11=0 Segmento de Datos

Bit 12 (S)=1 : No es un descriptor de sistema, sino de datos o de código.

DPL(Nivel de privilegio del descriptor). DPL=0 indica mayor nivel de privilegio

P(Presente)=1 el segmento esta presente

AVL(Disponible para uso de sistema)=0 No esta disponible.

bit 21(Reservado)=0

B(big)=0. Segmento de datos de 16 o 32bits. En este caso es de 16bits

G(granularidad)=0 . Multiplicador del limite. En este caso se multiplica por 1.

```
75 0000008D <res 00000040>idt: resb 8*8
```

; se dejan las primeras 8 interrupciones sin inicializar

```
76 ;la siguiente es Timer Tick tipo 8 ( ya que arranca de 0)
```

```
76  
77          irq0: ; Selector de IRQ0 ó tipo 8 ( Timer Tick T=55ms)
```

```
78
```

```
79 000000CD 0000          dw 0
```

```
80 000000CF 0800          dw 08h ;Apunta al primer selector, el de código
```

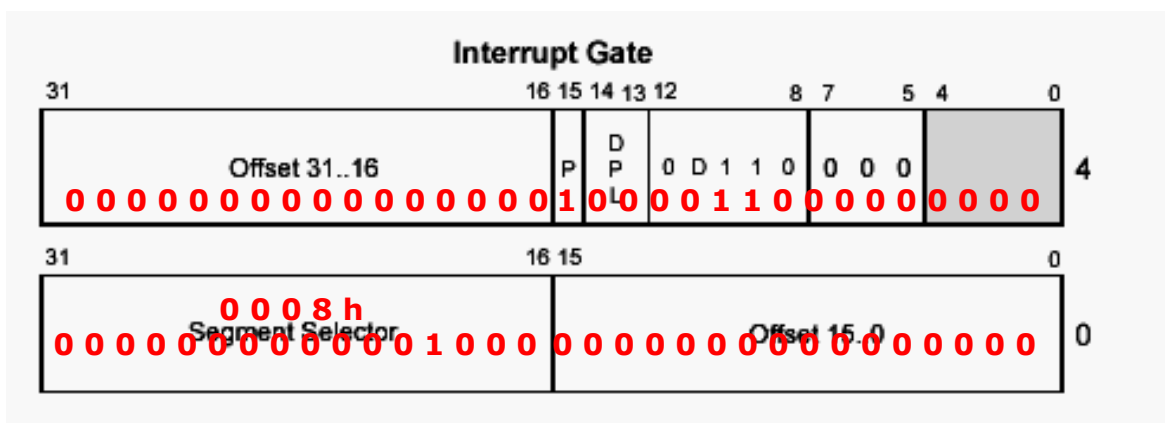
```
81 000000D1 00          db 0
```

```
82 000000D2 86          db 10000110b ;Atributo
```

```
83 000000D3 00          db 0
```

```
84 000000D4 00          db 0
```

Se inicializa la puerta de Interrupción:



P (Bandera de presencia del segmento) = 1 Esta presente

DPL (Nivel de Privilegio del descriptor ) =00b (Máximo privilegio).

D(Tamaño de la puerta de interrupción) = 0 16 bits

El offset se carga en tiempo de ejecución. El selector de segmento apunta a 0008h, que es el primer selector de GDT, ósea es el selector de código.

85

86                   irq0\_han:   ;rutina de atención a interrupción timer tick

La rutina de atención a IRQ0, que es de tipo 8, se ejecuta por Timer tick, es decir, para este caso T=55ms.

87

88 000000D5 1E           push ds   ;guarda ds

89 000000D6 6660        pushad   ;guarda registros de propósitos generales

90

En primer lugar se guardan los registros, para luego antes de salir volver a restaurarlos

91 000000D8 B81800       mov ax,18h

92 000000DB 8ED8        mov ds, ax                                       ;ds apunta al 3 selector (datos)

93 000000DD 66B800800B00 mov eax,0b8000h                               ;offset video

Con ds, que es el segmento de datos, apuntamos al tercer selector ( selector de datos). Con su offset apuntamos a B8000h, es decir, a video. De esta forma estamos posicionados en el primer Byte de pantalla.

94 000000E3 67FE00       inc byte [eax]                               ;Se incrementa primer Byte de pantalla

  ;lo que produce un cambio de caracteres de la

  ;primera posición de pantalla

Luego, cada vez que se interrumpo por timer tick, incrementamos el primer Byte de pantalla, de forma que se vallan sucediendo caracteres.

95

96 000000E6 B81000       mov ax,10h

97 000000E9 8ED8        mov ds, ax                                       ;ds apunta al 2 selector (datos)

Seleccionamos el segundo selector , que es de datos, para poder actuar sobre las variables del programa, de otra forma estaríamos en otro segmento y no podríamos actuar, por ejemplo, sobre ds:scan\_code.

98

99 000000EB E460        in al, 60h                                   ;dirección de E/S de teclado

100 000000ED 3EA2[5800]   mov [ds:scan\_code],al                       ;guarda el código de tecla en la posición scancode

Más tarde, leemos el puerto 60h, que es el scancode de teclado y lo almacenamos en la variable scan\_code.

```

101
102 000000F1 B020      mov al,20h
103 000000F3 E620      out 20h,al          ; le indica al PIC ( port 20h) que atendió a la interrupción

```

Le indicamos al PIC que la interrupción finalizo para que pueda atender a otra.

```

104
105 000000F5 6661      popad              ;restaura registros de propósitos generales
106 000000F7 1F        pop ds            ;restaura ds
107 000000F8 CF        iret              ; retorno de interrupción

```

Restauramos los registros y retornamos al programa principal.

```

108
109          inicio:          ; inicio del programa en modo real
110
111          ;jmp $           ;para debugger
112 000000F9 8C0E[8401]    mov [real_cs],cs  ;guarda cs real en variable

```

Como vemos el procesador esta en la posición Real 103d:01f9, lo que indica una dirección lineal de 103dH x 10h + 01f9H = 105C9H.

El segmento en modo real lo sitúa el Sistema operativo a partir del primer segmento libre de memoria, donde es carga nuestro programas y es csreal=103dH

```

113
114 000000FD 6631C0      xor eax,eax        ;eax=0
115 00000100 8CC8        mov ax,cs          ;almacena CS en eax.
116 00000102 66C1E004    shl eax,4          ;dirección lineal de cs
117 00000106 6689C3      mov ebx,eax        ;se guarda en ebx.
118

```

Si rotamos 4 posiciones a la derecha, tenemos eax=000103d0H, que es la posición lineal del segmento. En ax tenemos los 2Bytes menos significativos ax=03d0H

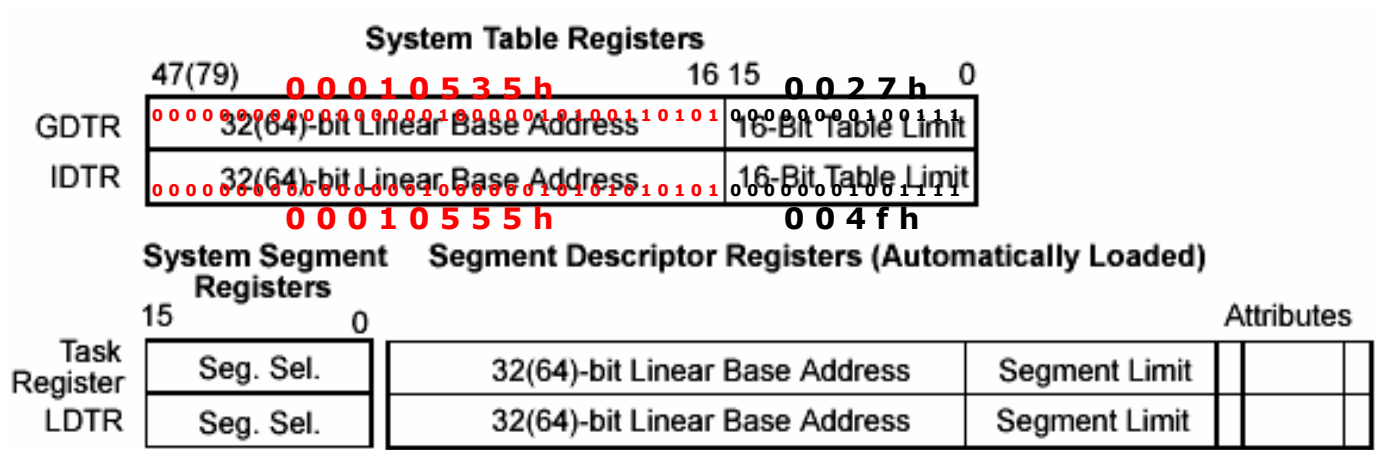


```

136          ;--- Se carga la IDT.
137
138 00000131 0F010E[5200]    sidt [real_idtr]          ;guarda la dirección de idtr real
139 00000136 6689D8        mov eax,ebx              ;lee la dirección lineal de cs real
140 00000139 6605[8D000000] add eax,idt              ;suma offset a idt
141 0000013F 66A3[6100]    mov [idtr+2],eax        ;guarda en la base de IDTR
142 00000143 0F011E[5F00]  lidt [idtr]             ;carga IDTR de modo protegido

```

Lee el offsen de idt 18Dh , y los suma a la dirección lineal del segmento eax=000103d0H. Este valor 000103d0H + 18Dh = 10555H lo carga en base de IDTR [idtr+2],:



```

143
144          ;--- A modo protegido
145
146 00000148 FA          cli          ;deshabilita interrupciones

```

Se deshabilitan las interrupciones para evitar que el procesador salte a un lugar desconocido en el traspaso a modo protegido.

147 00000149 0F20C0

mov eax,cr0

;lee el registro de control CRO

Se lee el registro de control CR0 el cual contiene, en el bit menos significativos, PE( Protection Enable).

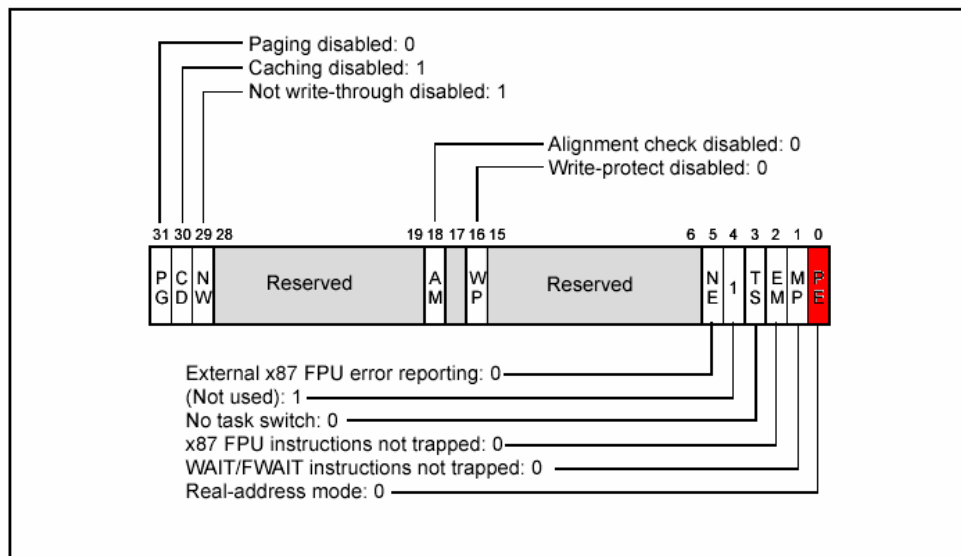


Figure 8-1. Contents of CR0 Register after Reset

148 0000014C 0C01

or al,1

;setea bit menos significativo PE(Protection Enable)

149 0000014E 0F22C0

mov cr0, eax

;carga en el registro de control y pasa a modo protegido

Se pone a uno el bit PE( protection Enable) y se pasa a modo protegido. En este punto el procesador que estaba en la dirección de segmento más desplazamiento 103d:024E pasa a la dirección 103d: 00000000000024E, que en ambos casos corresponde con la dirección lineal  $103d * 10 + 024E = 1061E h$ .

150

151 00000151 EA[5601]0800 jmp 08h:protegido

Salta al segmento de código indicado por el primer selector de la tabla de descriptores globales GDT.

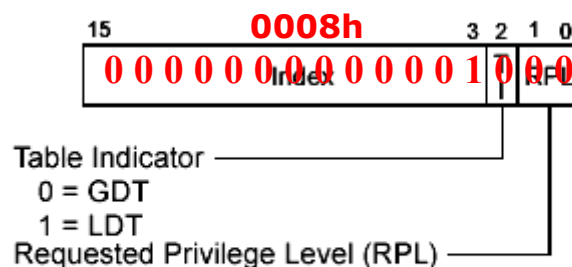


Figure 3-6. Segment Selector

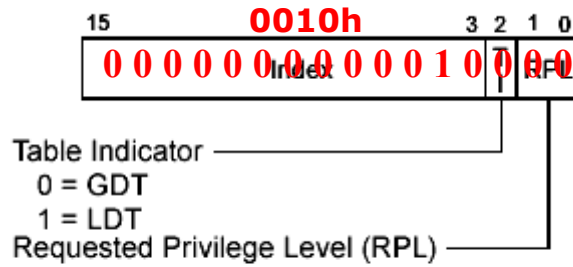
En estas condiciones el procesador entiende la dirección como la dirección de segmento más desplazamiento 0008: 000000000000251. Además la instrucción “jmp” limpia la cola de instrucciones.

```

152
153
154         protegido:                               ;modo protegido
155
156 00000156 B81000     mov ax,10h                   ;carga selector de datos (segundo selector)
157 00000159 8ED8      mov ds, ax                   ;en ds

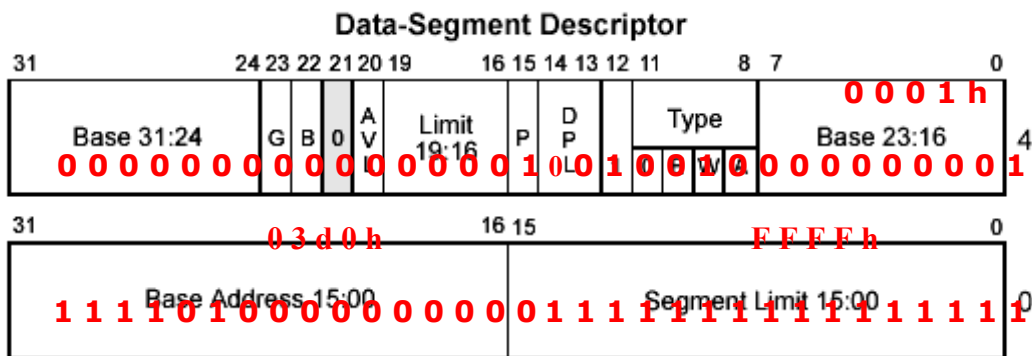
```

Carga el selector de datos ds con el segundo descriptor ( descriptor de datos):



**Figure 3-6. Segment Selector**

y como este descriptor esta definido como:



apunta a la dirección lineal 103d0h, con lo cual podemos acceder a todas las variables este programa mediante el offset.

```

158 0000015B E421     in al,21h                       ;lee el PIC modo real (controlador interrupciones)
159 0000015D A2[5100]  mov [mask_real_pic1],al       ;guarda en variable

```

Lee el controlador de interrupciones programable (PIC) y lo guarda en la posición de memoria definida por el selector ds + el offset de la variable mask\_real\_pic1, es decir, 0010:0151. Como el segundo selector (0010h) tiene un descriptor cuya dirección de base es 103d0, la dirección lineal es 103d0 + 0151 = 10521h.

```
160 00000160 B0FE          mov al,01111110b          ;habilita solo Irq0(timer tick T=55ms)
```

Una vez guardado el valor del PIC de modo real, se pasa a su modificación, de modo de habilitar solo la interrupción de timer tick. Esta modificación se realiza aplicándole una mascara de valor 11111110h = FEh.

```
161 00000162 E621          out 21h,al          ;programa PIC modo protegido
```

Se programa el PIC de modo protegido de forma que solo actúe el timer tick.

```
162
```

```
163 00000164 FB          sti          ;Setea las interrupciones en modo protegido.
```

Se habilitan las interrupciones de modo protegido.

```
164
```

```
165          esperar:          ;label de espera
```

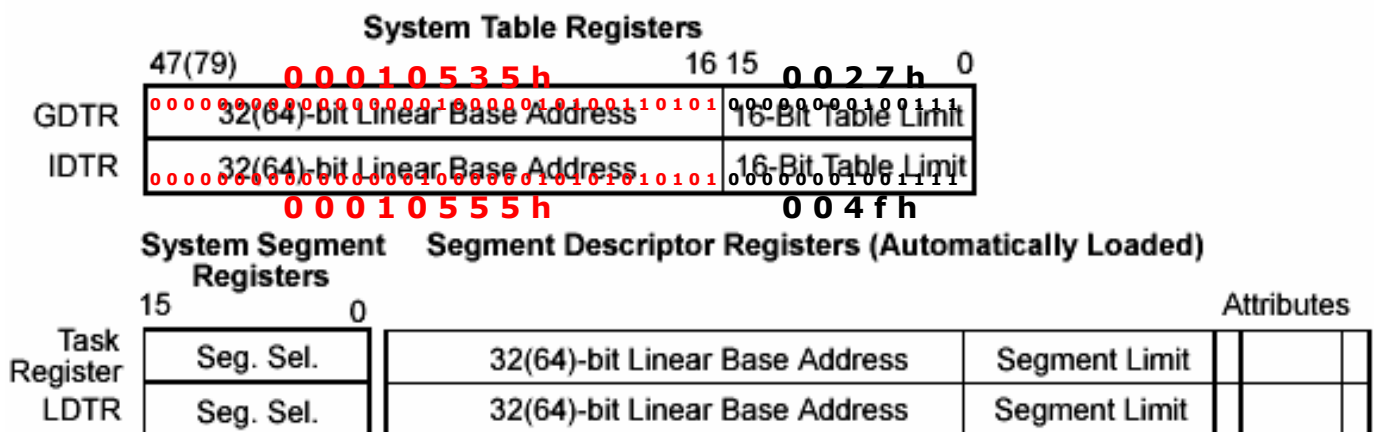
```
166
```

```
167 00000165 2E803E[5800]01 cmp byte [cs:scan_code],1          ; compara el código de tecla presionada con ESC
```

```
168 0000016B 75F8          jne esperar          ; Si no es Esc espera ( notar que salta cada 55ms a leer teclado en la rutina de atención a interrupción
```

Espera hasta que se presione la tecla “Esc”. Este bucle parecería ser infinito si no se tiene presente que el procesador salta al handler de atención a interrupción del timer tick cada 55ms.

Cuando se produce la interrupción por timer tick, el procesador busca el registro de la tabla de descriptores de interrupción (IDT).



Como se observa, el vector de interrupción comienza en la dirección lineal base 10555h. Desde esta posición el procesador busca la interrupción tipo 8 del vector de interrupciones, como se observa a continuación:

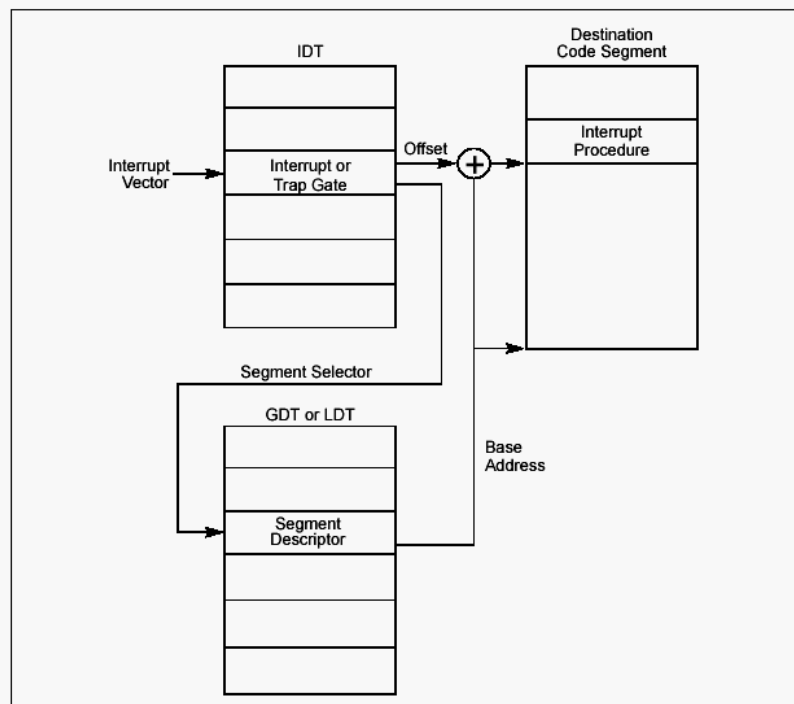
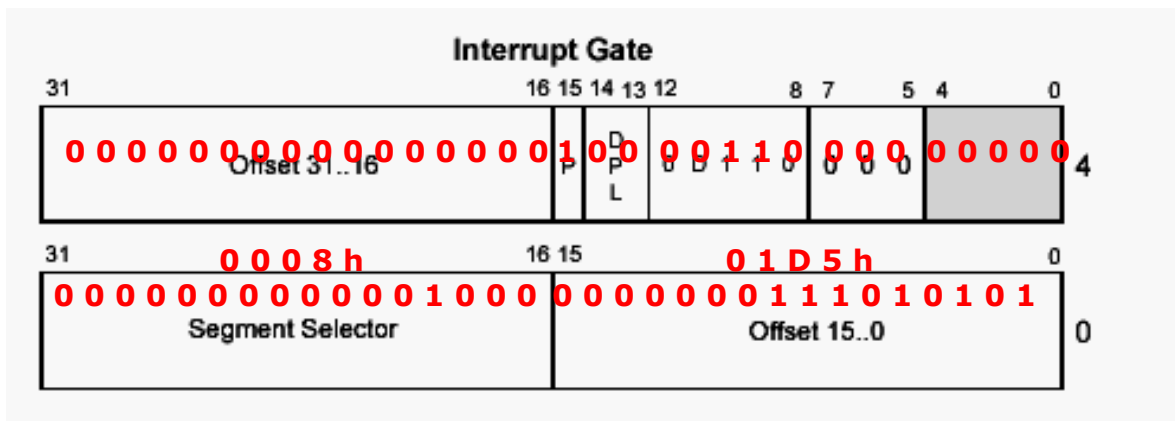


Figure 5-3. Interrupt Procedure Call

La puerta de interrupción contiene el selector de código al que ha de referenciarse, en este caso 0008h, es decir el primer selector de código ( en el cual esta cargado nuestro programa), y el ofset



Como el selector de segmento 0008h, apunta al primer descriptor de código, el cual tiene una dirección lineal de base 103d0h, y el offset 01D5h apunta al handler de interrupción, el procesador salta a la dirección lineal  $103d0h + 01D5h = 105A5h$ . En esa posición esta el la rutina de atención a interrupción definida mediante el label irq0\_han.

En ésta rutina procede a incrementar en un byte a la primera posición de pantalla, y a la lectura de teclado en el puerto de E/S correspondiente. Al apretar la tecla “ESC” el procesador continua con el procedimiento siguiente:

000001D5

169

170 0000016D FA cli ;deshabilita interrupciones. Comienza el retorno a modo real

Se deshabilitan las interrupciones para pasar que el procesador no salte a una posición indefinida en el pasaje a modo real.

171

172 0000016E B82000 mov ax,20h

173 00000171 8ED8 mov ds, ax ;apunta todos los selectores a un segmento de

174 00000173 8EC0 mov es,ax ;datos como recomienda intel ( tercer selector)

175 00000175 8EE0 mov fs,ax

176 00000177 8EE8 mov gs, ax

Se cargan todos los selectores con el 4 descriptor ( descriptor de datos). Este es un procedimiento recomendó por intel, aunque no es obligatorio.

177

178

179 00000179 0F20C0 mov eax,cr0 ;lee Registro de control

180 0000017C 24FE and al,0feh ;resetea PE(Protection Enable)

181 0000017E 0F22C0 mov cr0,eax ;pasa a modo real

Pasa a modo real, y el procesador pasa de interpretar la dirección 0008: 000000000000027E a interpretarla en el formato de modo real como 0008: 027E.

182 00000181 EA db 0eah ;salta a segmento modo real, equivale a jmp far real

183 00000182 [8601] dw real ;por medio de código de operación, ya que el compilador

184 00000184 0000 real\_cs dw 0 ;no lo ensambla de otro modo. Vacía cola de instrucciones

salta al modo real y el procesador interpreta la posición de memoria con el formato de segmento más desplazamiento: 103d:0284 h. Además se vacía la cola de interrupciones de modo protegido.

185

186

187 real: ;retorno a modo real

188

189 ;--- Se carga el selector de modo real en todos los segmentos.

190 00000186 8CC8 mov ax, cs

191 00000188 8ED8 mov ds,ax ;se recupera el segmento de modo real

192 0000018A 8EC0 mov es,ax

193 0000018C 8EE0 mov fs,ax

**194 0000018E 8EE8            mov gs, ax**

Se pasan los segmentos de datos y de propositos generales, al segmento donde reside el programa para tener acceso a las variables de éste.

**195**

**196                            ;--- Se recuperan la idt y las máscaras.**

**197**

**198 00000190 0F011E[5200]    lidt [real\_idtr]**

Se carga el vector de interrupciones de modo real

**199 00000195 A0[5100]        mov al,[mask\_real\_pic1]**

**200 00000198 E621            out 21h,al**

Se carga la mascara de modo real almacenada en la variable mask\_real\_pic1

**201**

**202**

**203                            ; Se carga el IDTR de modo real que se guardo antes de pasar a modo real**

**204                            ; y se recupera la máscara del PIC de modo real.**

**205**

**206**

**207 0000019A FB            sti    ;se habilitan interrupciones de modo real**

Se habilita la tabla de interrupciones de modo real

**208 0000019B B44C        mov ah,4ch**

**209 0000019D CD21        int 21h                                    ;vuelve al DOS**

Se devuelve el control al D.O.S.