

SOLUCIÓN AL FLICKEO DE LA INT 10h

Recomiendo a cualquier equipo de trabajo que solamente encaren proyectos de este tipo si disponen de al menos más de una semana de tiempo y pueden dedicarle 40/60 horas cada uno. Particularmente uno de los integrantes de nuestro equipo dedicó más de 80 horas entre investigación y codeo a solucionar lo que explicaremos a continuación.

Al querer hacer un juego del tipo PONG, Tetris, Arkanoid, Galaga, Galaxy se encontraran que la interrupción 10h no anda como uno quisiera que lo haga, si la perfección no es importante la 10h será suficiente, de lo contrario tendrán que pintar manualmente todo lo que quieran mostrar en pantalla, a continuación dejamos una guía de algunos conceptos necesarios para poder realizar este objetivo.

EL PRINCIPAL PROBLEMA DE LA 10H:

FLICKEA, si flickea y mucho, que es eso? Parpadea, representa fragmentos parciales de la pantalla y hace que el juego se vea poco profesional.

Esto no significa que no es necesaria, al contrario deberemos llamarla para poder darle formato a la pantalla, la cuestión es que si uno desea evitar este problema debe comprender que es lo que lo ocasiona, o al menos tener una idea.

La ram y el cpu tiene disponibles la información para escribir en la placa de video a una velocidad muy alta, aún en el 8086, sin embargo la placa de video esta diseñada para actualizar la pantalla a una frecuencia de 60hz (60 cuadros por segundo, aunque no se si las VGA, CGA y EGA del pasado llegaban a 60hz) y esta diferencia de velocidades no esta controlada o al menos no encontramos que este en la INT10h, por lo que el defasaje entre que esta lista placa y que se envian los datos produce que algunos cuadros se pinten mientras la placa esta escribiendo la mitad de la pantalla y no el principio.

DOBLE BUFFER:

El concepto de doble buffer es la solución a este problema (hay otras formas de hacerlo, como es el flípeo entre dos layers pero no se puede con el modo 13h) pero la forma de lograr implementar esto no incluye a la INT10h directamente, sino por el contrario hay que aprender a pintar la placa de video directamente. En sí la técnica de doble, o triple buffer es preparar lo que va a mostrarse, el "FRAME" en un espacio de memoria separado a la espera de que la placa de video este lista para ser escrita y la misma no este escribiendo la mitad o el principio del barrido sino que por el contrario este en el último pixel del monitor.

QUE MODO DE VIDEO ELEGIR:

Seguramente al empezar a jugar con la int10h van a encontrarse con la parte más básica de esta que es poder setear el modo de video en graficos o bien texto, hay muchos modos y formas de representar en pantalla y aunque no son ni cercanos a los actuales hay que entender que estos modos estaban diseñados para trabajar bajo un máximo de 64k de memoria de video por lo que en su mejor ejemplo vamos a poder mostrar 320x200 pixeles o 640x480 pero siempre que amplíemos la representación perderemos en cantidad de colores.

RECOMENDACIÓN: MODO 13h, es ideal para hacer un juego sencillo por la forma de mapear la memoria de video, los otros modos son complejos por que trabajan en capas y el 13h es suficiente como para hacer casi cualquier proyecto de los mencionados.

CARACTERÍSTICAS DEL MODO 13h:

Principalmente que cuenta con 320 pixeles de ancho por 200 de alto, se divide en 40 columnas por 20 columnas de texto (esto es medio difícil de entender al principio pero básicamente no se puede representar una letra con un pixel por lo que los que diseñaron este modo le dieron N cantidad de columnas que dan la máxima cantidad de pixeles con los cuales se podrá representar un caracter en pantalla).

La forma de pintar un pixel en este modo es escribiendo el bit correspondiente en el address de memoria de video con el valor del color a representar, la memoria de video arranca en el address 0A000h y es necesario un segmento para poder direccionar a la misma, no se puede hacer directamente con un registro.

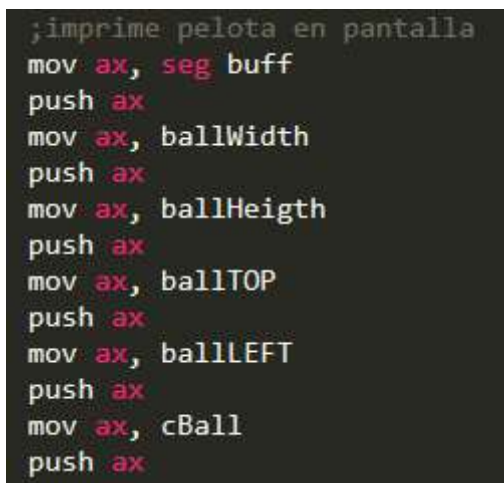
Si escribo el bit 321 con el valor 3 se va a encender un pixel verde en la segunda linea de pixeles y este será el primero, si quiere encender el primer pixel de la fila 3 deberé sumarle a ese valor 320 pixeles más, como si fuese un excel se forma una grilla de 320x200 pixeles siendo el último el 64000-1 (puesto que arranca en 0) .

PROCEDIMIENTO PARA PODER CREAR EL BUFFER:

Es necesario para guardar temporalmente la información que se escribirá en la memoria de video crear un segmento en nuestro código que almacenará cual si fuera una variable un espacio dedicado para los 64kb de video, la forma de hacerlo es por medio de estas líneas de código al final del code segment:

```
.fardata black          ; Segmento donde guardo el double buffer
    buff db 64000 dup (0)
black ends
```

En la siguiente imagen pueden ver como pasar el segmento por stack a la función que dibuja el pixel, el resto de los parametros en nuestro caso eran para dibujar un cuadrado, los archivos del los tienen los profes y estan a disposición del que desee mejorarlo o utilizarlo para su proyecto.



```
;imprime pelota en pantalla
mov ax, seg buff
push ax
mov ax, ballWidth
push ax
mov ax, ballHeight
push ax
mov ax, ballTOP
push ax
mov ax, ballLEFT
push ax
mov ax, cBall
push ax
```

Para escribir en este segmento es necesario utilizar DS o el ES para manejar los address de memoria del mismo.

La siguiente función es un ejemplo en modelo LARGE de como pintar un pixel en pantalla

```

1 drawPIXEL proc far
2 ;imprime un pixel directamente en la memoria          | ss:structure
3 ;   input:  1. double buffer Address                  | bp 12
4 ;           2. x                                     | bp 10
5 ;           3. y                                     | bp 8
6 ;           4. Color                                 | bp 6
7 ;   output: a pixel...
8
9     push bp
10    mov bp, sp
11    push ax bx cx dx si ds
12
13    push ss:[bp+12]                                ; paso el offset del buffer para poder escribirlo
14    pop ds                                           ; como es un segmento lo paso al DS
15    ; para manejar las direcciones
16
17    mov cx, 320                                     ; Bytes per line in mode 13h
18    mov ax, ss:[bp+8]                               ; y
19    mul cx
20    add ax, ss:[bp+10]                              ; x
21    mov si, ax                                       ; paso al registro si el valor de ax que da la
22    xor ah, ah                                       ; posición del pixel
23    mov al, ss:[bp+6]                               ; color
24    mov byte ptr ds:[si], al                       ; le pego el color a la posición de
25    ; memoria que corresponde a X e Y
26    pop ds si dx cx bx ax bp
27    ret 8
28 drawPIXEL endp
29

```

UTILIZACIÓN DEL MODO LARGE:

Simplemente se declara al principio en vez de ".model small" --> ".model large", pero es necesario a cada función colocarle después del proc la palabra "FAR", esto se debe a que contrario al model small los segmentos de memoria no estan cercanos, de no colocarlo el programa no necesariamente mostrará un error pero no podra resolver el address de la función creada. También es necesario al compilar hacerlo con el siguiente esquema: tasm /m miarchivo, de lo contrario arrojará un error.

**La utilización de este modo en el PONG fue consecuencia de probar distintas técnicas para solucionar problemas que tenían que ver con el doble buffer, en model small no pudimos identificar por que aparentemente este segmento siempre pisaba el stack y generaba cuelgues, pero es posible que con un poco de investigación extra el proyecto se pueda realizar en model small.*