# Proyecto 1: The Bootloader
Proyecto 1 del curso de Principios de Sistemas Operativos. El objetivo del proyecto es desarrollar un bootloader que ejecute una versión del juego Lead (Atari 2600). Este proyecto está desarrollado con NASM y C.

## Funcionamiento
El bootloader de este proyecto debe cumplir con ciertas características

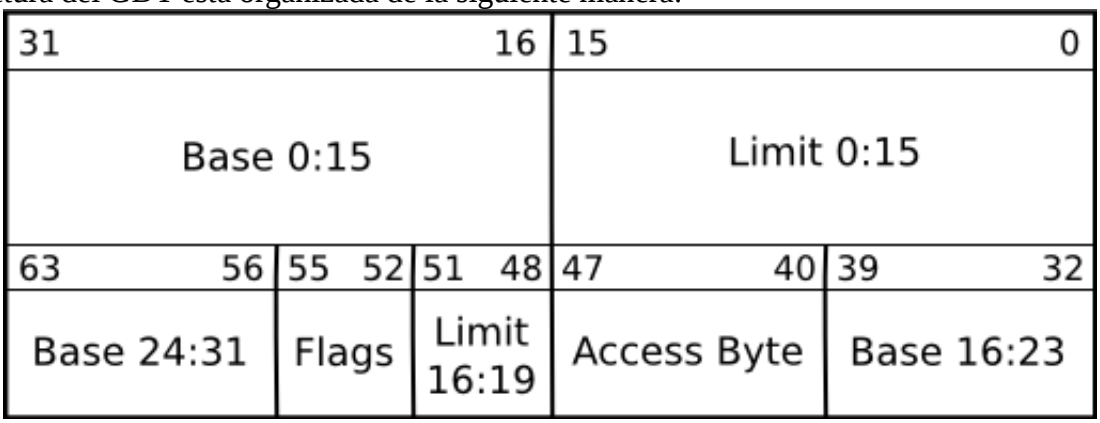
### Ser booteable
Un archivo booteable debe ser de 512 bytes de memoria exactos, lo cuál se logra rellenando con ceros el espacio libre. Para indicarle al BIOS que los 512 bytes marcados son booteables se debe escribir el número hexadecimal aa55 directamente en el ejecutable por medio de `dw 0xaa55`.

### Modo protegido de 32 bits
Si no se le dice al BIOS, solo se pueden usar los registros y las instrucciones de 16 bits. Para poder usar el modo protegido de 32 bits se deben cumplir dos condiciones:
1. Activar las instrucciones de 32 bits.
2. Dar acceso a los registros completos de 32 bits.

Un programa de 512 bytes de memoria no es suficiente para hacer programas muy complejos. Para poder usar más de 1 MB de memoria, se le debe pedir permiso al BIOS para acceder a toda la memoria, para lo cuál debe activar la [línea A20](https://wiki.osdev.org/A20_Line) con la función *A20-Gate activate*.
Para activar las instrucciones de 32 bits y dar acceso a los registros completos se debe activar el bit de modo de la CPU y setear una [*Global Descriptor Table*](https://en.wikipedia.org/wiki/Global_Descriptor_Table), que define un segmento de 32 bits. En el caso de este ejemplo se van a tener 3 GDT, un segmento nulo, un segmento de código y otro segmento de datos. La estructura de cada uno de los GDT es:
* **Base**: 32 bits que definen donde empieza el segmento.
* **Límite**: 20 bits que describen donde termina el segmento. Si la granularidad es 1 entonces se multiplica por 4096.
* **Acceso**: 8 bits sobre los permisos de este segmento.
  * **Presente**: 1 bit que debe ser uno para que sea válido.
  * **Privilegio**: 2 bits que asignan el anillo de seguridad, 0 para el más alto (kernel) y 3 para el más bajo (aplicaciones de usuario).
  * **Tipo de Descriptor**: 1 bit que debe estar activo para los segmentos de código o datos y debería estar limpio para segmentos del sistema.
  * **Ejecutable**: 1 bit que si está activo el segmento puede ser ejecutado, si está limpio es un segmento de datos.
  * **Dirección**: 1 bit el cuál indica si está activo de que el segmento puede ser ejecutado desde niveles inferiores de privilegio. Si es 0 solo puede ser ejecutado en el nivel asignado en el bit de **Privilegio**.
  * **Lectura/Escritura**: 1 bit que define si puede o no leer y escribir en este segmento.
  * **Accedido**: 1 bit que dice si el CPU a accedido el segmento. Siempre se setea en 0, cuando el CPU lo visita se encarga de activarlo.
* **Bandera**: 4 bits que activa banderas para informar de algo al CPU. Originalmente solo tenía 2 banderas, pero en la arquitectura **x86-64** se agrega 1 bandera más.
  * **Granularidad**: 1 bit que si está desactivado marca que el límite está en bloques de 1 byte, si está prendido, el límite es en múltiplos de bloques de 4 KB.

  * **Tamaño**: 1 bit que define el modo del segmento, cuando está apagado está en modo de 16 bits y cuando está prendido está en modo de 32 bits protegido.
  * **L**: 1 bit que indica el descriptor de código de **x86-64**. Está reservado para los segmentos de datos.

La estructura del GDT está organizada de la siguiente manera:




### Acceder a más memoria
El BIOS solo carga los primeros 512 bytes del sector de boot. Si se desea escribir programas de más tamaño se debe cargar más espacio en memoria. Para hacer esto se debe usar la instrucción `mov ah, 0x2` que se encargan de leer los sectores de un disco; en conjunto con la interrupción del BIOS `int 0x13` que provee los servicios de disco.

Hecho esto, ya se puede cargar más memoria del segundo sector de memoria, desde está parte se puede realizar un programa fuera de los 512 bytes booteables.

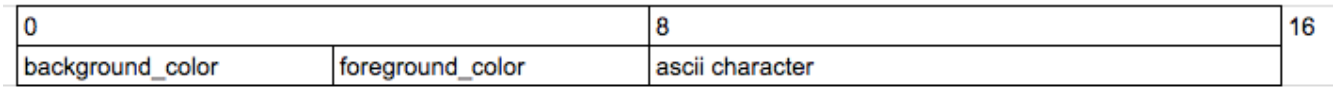
## Diseño de juego
Como se ha mencionado antes se realizará una versión del juego Lead para la Atari 2600, para lo cuál se tendrán ciertas consideraciones para el diseño del mismo.

### Tamaño de la pantalla
La pantalla que será usada para representar el juego, se usará el estándar de gráficos VGA en modo de texto 3, está tiene las siguientes especificaciones:
* Resolución: 720X400 píxeles.
* Caracteres: 80X25 caracteres.
* Resolución de Carácter: 9X16 píxeles.

Ahora es importante conocer la resolución del juego original para poder realizar una conversión fiel al mismo. Se sabe que el hardware original de la [Atari 2600](https://en.wikipedia.org/wiki/Atari_2600_hardware) tenía las siguientes especificaciones:
* Resolución: 40X192 píxeles.
* Resolución sprite: 8X192 píxeles.
* Resolución balas: 1x192 píxeles.

**Notése** que estos valores están en la forma **ancho** X **alto**, y en la parte lógica del programa será tratado como **columnas** X **filas**

### Colores y caracteres
Para crear una experiencia, en este caso se cambia el color del texto en el modo de texto de VGA. Para esto es importante conocer la estructura del [buffer de texto de VGA](https://en.wikipedia.org/wiki/VGA-compatible_text_mode). Estos tienen la siguiente estructura:
* **Color de fondo**: 4 bits que determinan el color del fondo del carácter. Son 16 colores posibles.
* **Color de texto**: 4 bits que determinan el color del carácter. Son 16 colores posibles.
* **Código del carácter**: 8 bits del código ASCII del carácter.




Para construir el bootloader se usan el comando:
```
make
```

## Referencias
Referencia colores VGA: https://www.fountainware.com/EXPL/vga_color_palettes.htm
Referencia de caracteres posibles/Code page 737: https://en.wikipedia.org/wiki/Code_page_737
LEAD: https://atariage.com/store/index.php?l=product_detail&p=932
Instrucción A20: https://wiki.osdev.org/A20_Line
Global Descriptor Table: https://en.wikipedia.org/wiki/Global_Descriptor_Table
VGA: https://en.wikipedia.org/wiki/VGA-compatible_text_mode
Atari Hardware: https://en.wikipedia.org/wiki/Atari_2600_hardware
ReaD Time-Stamp Counter:
https://github.com/cirosantilli/x86-bare-metal-examples/blob/master/rtc.S
https://c9x.me/x86/html/file_module_x86_id_278.html
I/O: https://wiki.osdev.org/Inline_Assembly/Examples
Real-Time-Clock: https://en.wikipedia.org/wiki/Real-time_clock
Key Input: http://stanislavs.org/helppc/scan_codes.html