Universidad Nacional de Córdoba

Facultad de Cs. Exactas, Físicas y Naturales

Sistemas de Computación

Trabajo Práctico N°4

Grupo: **ASMAN**

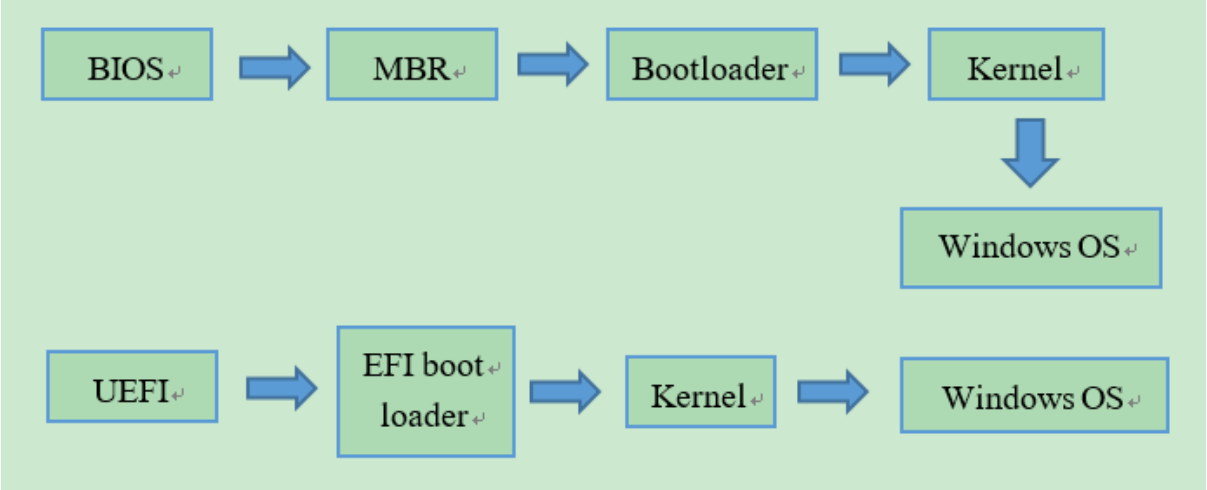
Integrantes:	Matrícula:
• Amallo, Sofía	41279731
• Prieto, Julieta	42304327
• Ojeda, Gastón	40248481

Docentes: Solinas, Miguel - Jorge, Javier

Índice

MARCO TEÓRICO

Para el presente trabajo nos parece necesario arrancar con una pequeña introducción a los temas.
Para empezar podemos generar una idea básica de como arranca una computadora.
En primer lugar apenas se enciende un ordenador podríamos decir que se carga la BIOS o UEFI según corresponda, siguiendo el siguiente esquema:



Dadas las intenciones de este trabajo solo haremos foco en el primer flujo:

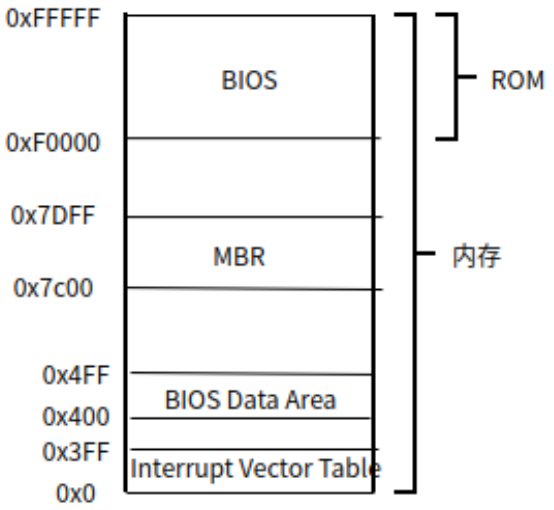
1	BIOS	Basic input / output system executes MBR
2	MBR	Master Boot Record executes GRUB
3	GRUB	Grand Unified Bootloader executes Kernal
4	Kernel	Kernel executes the /sbin/init program
5	Init	Init executes Runlevel programs
6	Runlevel	Runlevel programs are executes from /etc/rc.d/rc*.d/

1- Básicamente al encender la computadora carga la bios (En MODO REAL) y hace un “Power On Self Test” :

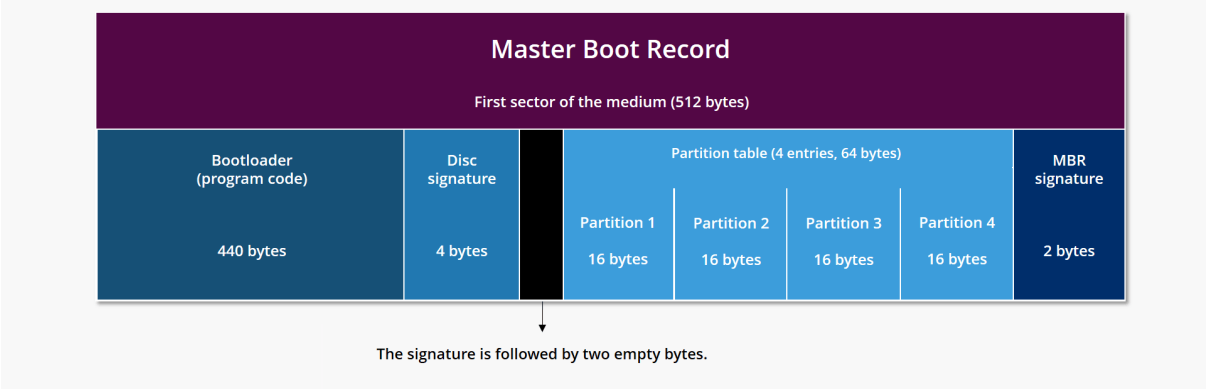
- En el momento del arranque, el registro cs: ip de la CPU se inicializa a la fuerza a 0xF000: 0xFFF0, y el contenido allí es una instrucción (jmp far f000: e05b), que se está ejecutando Después de esta instrucción, cs: ip se convierte en 0xF000: 0xFE05B, y 0xFE05B es donde el código del BIOS realmente comienza a ejecutarse El BIOS necesita verificar varios hardware de computadora e inicializar, y luego debe establecer una estructura de datos (Área de datos del BIOS), una tabla de vectores de interrupción (Tabla de vectores de interrupciones) y completar las rutinas de interrupción[10]

2- Luego la BIOS levanta la MBR :

- El último trabajo del BIOS es verificar **el contenido del disco de arranque en el sector del disco duro (0 disco 0 pista 1 sector) donde está la MBR**, si los dos bytes al final de este sector son el número mágico 0x55 y 0xaa, la BIOS cree que hay un programa ejecutable en este sector, y lo carga en la main memory desde la dirección física 0x7c00, luego salta a esta dirección y continúa la ejecución[10]



3- El MBR ejecuta el GRUB el cual inicializa el disco, levanta todas las particiones se fija cuál de todas es booteable. En caso de tener un dual-boot carga una segunda instancia de Grub donde nos aparece la pantalla para elegir el sistema operativo. Una MBR tiene la siguiente estructura:

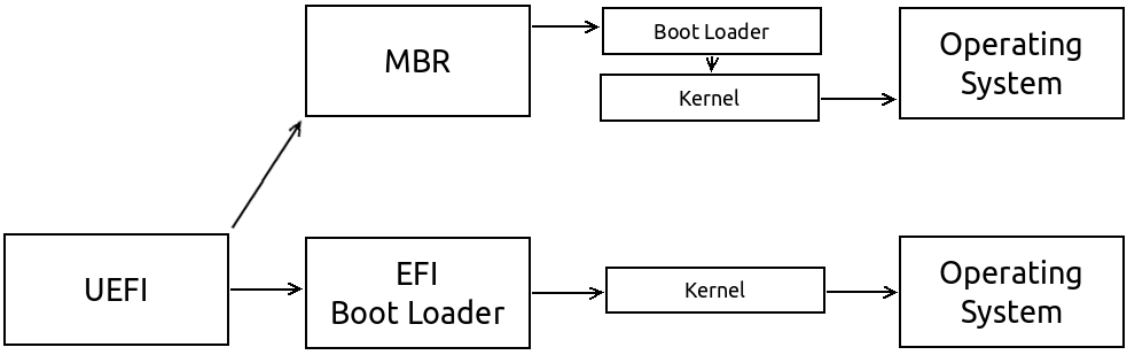


- Programa de inicio (bootloader)
- Soporte de datos, firma de disco (a partir de Windows 2000)
- Tabla de particiones maestra.
- **MBR** o firma de arranque (Magic number)

4-6 Y finalmente por lo general carga el kernel que ejecuta el sbin/init y a su vez el Init ejecuta el Run Level program. **(por supuesto que esto no lo vamos hacer debido a la complejidad que esto implica y nos vamos a limitar a simplemente mostrar un hello world en pantalla en modo protegido).**

Ahora bien, este esquema no necesariamente es así siempre, vamos a explicar un caso particular que se nos presentó cuando desarrollamos el desafío final.
Entonces, nos hicimos la siguiente pregunta **¿Si mi computadora tiene UEFI podemos levantar un MBR?** La respuesta es sí. Y esto es gracias a un módulo de soporte de compatibilidad del UEFI :

- El módulo de soporte de compatibilidad o CSM es de utilidad para dar soporte para cada modelo de placa base a particiones MBR. Básicamente y en resumen, es una manera de mantener la compatibilidad con sistemas de archivos, y más concretamente con sistemas de arranque, antiguos. En otras palabras es una parte de las UEFI modernas para poder iniciar HDD o SSD que tengan particiones MBR como arranque del sistema



Desafío: UEFI y coreboot

- *¿Qué es UEFI? ¿Cómo puedo usarlo? Mencionar además una función a la que podría llamar usando esa dinámica.*

En primer lugar podríamos explicar que es una BIOS para entender cómo vino la UEFI.

La BIOS "Basic Input Output System" o "Sistema Básico de Entrada y Salida" es un sistema que se encarga de gestionar el arranque en los ordenadores introducido a mediados de los años 70. Este es el primero en cargarse y está encargado de iniciar, chequear y configurar los distintos componentes.

La UEFI, "Unified Extensible Firmware Interface" o "Interfaz de Firmware Extensible Unificada" es el sucesor del BIOS creada en el 2005, descrito como más seguro y más eficaz.

Para usar la UEFI[9]: Para utilizar un firmware UEFI, el hardware de tu

disco debe soportar UEFI. Además, el disco del sistema tiene que ser un disco GPT. Si no es así, puedes convertir el disco MBR a GPT con un programa tipo Partition Magic profesional como MiniTool Partition Wizard .

Después de que termine la conversión, tendrás que cambiar el modo de arranque de BIOS a UEFI:

1- al encender la computadora accedemos a la configuración del sistema presionando la tecla “F10, F2, F12, F1 o SUPR” (dependiendo del ordenador), aunque algunas computadoras tienen la posibilidad de presionar un pequeño botón externo.

2- Desde la pantalla principal de la BIOS, elige la sección Boot.

3- Cuando estés en Boot, elige el modo de arranque UEFI/BIOS y pulsa la tecla intro.

4- En el cuadro de diálogo del modo de inicio UEFI/BIOS, elegir modo de inicio UEFI y, a continuación, pulsar intro.

5- Guardar los cambios y salir.

6- repetir el paso 1 para entrar esta vez a la UEFI en vez de la BIOS

- *¿Menciona casos de bugs de UEFI que puedan ser explotados?*

Según algunas investigaciones, se indica que hay más de 23 vulnerabilidades listadas a continuación:

Vulnerabilities	Number of Issues	BINARLY ID	CVE ID
SMM Callout (Privilege Escalation)	10	BRLY-2021-008, BRLY-2021-017, BRLY-2021-018, BRLY-2021-019, BRLY-2021-020, BRLY-2021-022, BRLY-2021-023, BRLY-2021-024, BRLY-2021-025, BRLY-2021-028	CVE-2020-5953, CVE-2021-41839, CVE-2021-41841, CVE-2021-41840, CVE-2020-27339, CVE-2021-42060, CVE-2021-42113, CVE-2021-43522, CVE-2022-24069, CVE-2021-43615,
SMM Memory Corruption	12	BRLY-2021-009, BRLY-2021-010, BRLY-2021-011, BRLY-2021-012, BRLY-2021-013, BRLY-2021-015, BRLY-2021-016, BRLY-2021-026, BRLY-2021-027, BRLY-2021-029, BRLY-2021-030, BRLY-2021-031	CVE-2021-41837,CVE-2021-41838, CVE-2021-33627, CVE-2021-45971, CVE-2021-33626, CVE-2021-45970, CVE-2021-45969, CVE-2022-24030, CVE-2021-42554, CVE-2021-33625, CVE-2022-24031, CVE-2021-43323
DXE Memory Corruption	1	BRLY-2021-021	CVE-2021-42059

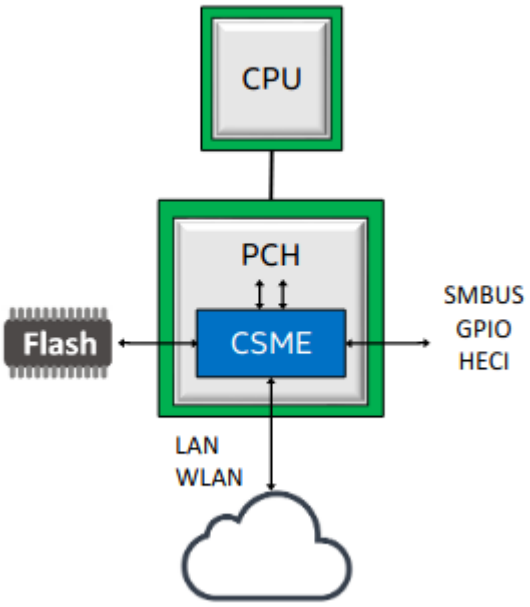
Las cuales podrían ser explotadas para instalar malware persistente, es decir que “sobreviva a las reinstalaciones del sistema operativo y permite eludir las soluciones de seguridad de punto final (EDR/AV), el arranque seguro y el aislamiento de seguridad basado en virtualización” [3].

Además estos problemas no pueden ser detectados por los sistemas de monitoreo de integridad del firmware.

- *¿Qué es Converged Security and Management Engine (CSME), the Intel Management Engine BIOS Extension (Intel MEBx)?*

El CSME significa Módulo de Gestión de Seguridad Convergente. “Es un subsistema dentro de los chips de Intel que viene integrado en las últimas generaciones de sus procesadores. El CSME es una característica de seguridad que se ejecuta al inicio y es responsable de verificar y autenticar todo el firmware posterior.” [1] (Rus)

Este es un procesador de Intel independiente y de bajo consumo, que proporciona un entorno de ejecución aislado y protegido desde el SW del host que se ejecuta en la CPU principal.



Roles:

- Inicialización de Silicio:
Arranque seguro, configuración de clocks y carga de microcódigo en motores PCH/CPU HW

-Seguridad:
Ejecución aislada y confiable de servicios de seguridad (TPM, DRM, DAL)

-Gestión:
Gestión remota independiente del SO (AMT de Intel)

MEBx significa “extensión de BIOS del ME”.
MEBx proporciona “opciones de configuración a nivel de plataforma para poder configurar el Management Engine (ME). Las opciones incluyen la activación y desactivación de funciones individuales y el ajuste de las configuraciones de energía” [2]

- *¿Qué es coreboot? ¿Qué productos lo incorporan? ¿Cuáles son las ventajas de su utilización?*

Inicialmente llamado LinuxBIOS, es un proyecto de código abierto cuyo fin es ofrecer las funciones más elementales al inicio del sistema para luego pasar a inicializar el hardware (es decir, reemplazar a la BIOS propietaria que viene de fábrica). [5]

Todos los productos ChromeOS incorporan coreboot, como también los de Star Labs, System76, NovaCustom, Purism y PC Engines. [6]

Además de ser rápido (fue diseñado con la intención de que realice su tarea con el mínimo de instrucciones posible) y seguro, su característica más importante es la de ser software libre, lo que permite tener un control máximo del producto, además de cumplir con los estándares de la Free Software Foundation.

Desafío: Linker

- *¿Qué es un linker? ¿Qué hace?*

Un linker es un programa que crea un archivo ejecutable o biblioteca a partir de código objeto, típicamente la salida de un compilador.

Un enlazador de scripts (como el archivo `link.ld`) es un conjunto de comandos que definen qué partes de código objeto copiar y en qué orden.

- Los principales entre ellos son las secciones (fundamental, especifica la estructura del archivo binario en la salida) y la memoria (opcional, que describe la memoria disponible).
- Los comentarios, sintácticamente equivalentes a espacios en blanco y delimitados por / * y */, son similares a los de C89.
- La variable especial " . '' (contador de ubicación) le permite especificar la ubicación actual en la memoria en la que insertar el código objeto.
- La sintaxis proporciona varios operadores aritméticos y algunas funciones con las que construir expresiones.
- El comando SECTIONS solo se puede usar una vez en cada script, y le permite definir un número arbitrario de secciones. Cada sección tiene un nombre, seguido de dos puntos y un bloque entre corchetes que define su contenido. [7]

- *¿Cuál es la dirección que aparece en el script del linker? ¿Por qué es necesaria?*

Corresponde a la dirección donde se almacena el registro de arranque principal (0x7C00).

Es necesario especificarla porque dicha estructura (la MBR) posee la tabla de particiones del disco y una pequeña porción de código ejecutable para el arranque, lo que significa una parte clave del sistema de inicio.

- *Compare la salida de objdump con hd, verifique donde fue colocado el programa dentro de la imagen.*

Salida objdump -S main.o

```
gaston@gaston:~/Desktop/Gaston/sist de comp/protected-mode-sdc/01HelloWorld$ objdump -S main.o

main.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <loop-0x5>:
.code16
    mov $msg, %si
   0:   be 00 00 b4 0e          mov    $0xeb40000,%esi

0000000000000005 <loop>:
    mov $0x0e, %ah
loop:
    lodsb
   5:   ac                      lods   %ds:(%rsi),%al
    or %al, %al
   6:   08 c0                   or     %al,%al
    jz halt
   8:   74 04                   je     e <halt>
    int $0x10
   a:   cd 10                   int    $0x10
    jmp loop
   c:   eb f7                   jmp    5 <loop>

000000000000000e <halt>:
halt:
    hlt
   e:   f4                      hlt

000000000000000f <msg>:
   f:   68 65 6c 6c 6f          pushq  $0x6f6c6c65
  14:   20 77 6f                and    %dh,0x6f(%rdi)
  17:   72 6c                   jb     85 <msg+0x76>
  19:   64                      fs
        ...
```

objdump -S muestra el código fuente junto con el disassembly

Salida hd main.img

```
gaston@gaston:~/Desktop/Gaston/sist de comp/protected-mode-sdc/01HelloWorld$ hd main.img
00000000  be 0f 7c b4 0e ac 08 c0  74 04 cd 10 eb f7 f4 68  |..|.....t......h|
00000010  65 6c 6c 6f 20 77 6f 72  6c 64 00 66 2e 0f 1f 84  |ello world.f....|
00000020  00 00 00 00 00 66 2e 0f  1f 84 00 00 00 00 00 66  |.....f.........f|
00000030  2e 0f 1f 84 00 00 00 00  00 66 2e 0f 1f 84 00 00  |.........f......|
00000040  00 00 00 66 2e 0f 1f 84  00 00 00 00 00 66 2e 0f  |...f.........f..|
00000050  1f 84 00 00 00 00 00 66  2e 0f 1f 84 00 00 00 00  |.......f........|
00000060  00 66 2e 0f 1f 84 00 00  00 00 00 66 2e 0f 1f 84  |.f.........f....|
00000070  00 00 00 00 00 66 2e 0f  1f 84 00 00 00 00 00 66  |.....f.........f|
00000080  2e 0f 1f 84 00 00 00 00  00 66 2e 0f 1f 84 00 00  |.........f......|
00000090  00 00 00 66 2e 0f 1f 84  00 00 00 00 00 66 2e 0f  |...f.........f..|
000000a0  1f 84 00 00 00 00 00 66  2e 0f 1f 84 00 00 00 00  |.......f........|
000000b0  00 66 2e 0f 1f 84 00 00  00 00 00 66 2e 0f 1f 84  |.f.........f....|
000000c0  00 00 00 00 00 66 2e 0f  1f 84 00 00 00 00 00 66  |.....f.........f|
000000d0  2e 0f 1f 84 00 00 00 00  00 66 2e 0f 1f 84 00 00  |.........f......|
000000e0  00 00 00 66 2e 0f 1f 84  00 00 00 00 00 66 2e 0f  |...f.........f..|
000000f0  1f 84 00 00 00 00 00 66  2e 0f 1f 84 00 00 00 00  |.......f........|
00000100  00 66 2e 0f 1f 84 00 00  00 00 00 66 2e 0f 1f 84  |.f.........f....|
00000110  00 00 00 00 00 66 2e 0f  1f 84 00 00 00 00 00 66  |.....f.........f|
00000120  2e 0f 1f 84 00 00 00 00  00 66 2e 0f 1f 84 00 00  |.........f......|
00000130  00 00 00 66 2e 0f 1f 84  00 00 00 00 00 66 2e 0f  |...f.........f..|
00000140  1f 84 00 00 00 00 00 66  2e 0f 1f 84 00 00 00 00  |.......f........|
00000150  00 66 2e 0f 1f 84 00 00  00 00 00 66 2e 0f 1f 84  |.f.........f....|
00000160  00 00 00 00 00 66 2e 0f  1f 84 00 00 00 00 00 66  |.....f.........f|
00000170  2e 0f 1f 84 00 00 00 00  00 66 2e 0f 1f 84 00 00  |.........f......|
00000180  00 00 00 66 2e 0f 1f 84  00 00 00 00 00 66 2e 0f  |...f.........f..|
00000190  1f 84 00 00 00 00 00 66  2e 0f 1f 84 00 00 00 00  |.......f........|
000001a0  00 66 2e 0f 1f 84 00 00  00 00 00 66 2e 0f 1f 84  |.f.........f....|
000001b0  00 00 00 00 00 66 2e 0f  1f 84 00 00 00 00 00 66  |.....f.........f|
000001c0  2e 0f 1f 84 00 00 00 00  00 66 2e 0f 1f 84 00 00  |.........f......|
000001d0  00 00 00 66 2e 0f 1f 84  00 00 00 00 00 66 2e 0f  |...f.........f..|
000001e0  1f 84 00 00 00 00 00 66  2e 0f 1f 84 00 00 00 00  |.......f........|
000001f0  00 66 2e 0f 1f 84 00 00  00 00 00 0f 1f 00 55 aa  |.f............U.|
00000200
```

Analizando la salida del hd main.img se puede observar que el programa fue colocado en los bytes del MBR asignados al área de código de arranque (desde la posición 000h), Indicando que este codigo sera lo primero en ejecutar la BIOS, justo después de la instrucción hlt en 0xf4 se coloca la impresión del msg “Hello World”

Grabar la imagen en un pendrive y probarla en una pc y subir una foto

- *¿Para qué se utiliza la opción* `--oformat` *binary en el linker?*

  Permite especificar el formato binario de la salida del archivo objeto. [8]

Desafío Final: Modo Protegido

- *Crear un código assembler que pueda pasar a modo protegido (sin macros).*
- *¿Cómo sería un programa que tenga dos descriptores de memoria diferentes, uno para cada segmento (código y datos) en espacios de memoria diferenciados?*

La GDT deberá tener 2 entradas, una por cada descriptor de segmento correspondiente.

- *Cambiar los bits de acceso del segmento de datos para que sea de solo lectura, intentar escribir, ¿Qué sucede? ¿Qué debería suceder a continuación? Verificarlo con gdb.*

En caso de que la tabla GDT se defina con segmento de datos de sólo lectura, cuando se intente escribir en la memoria el mensaje, se activará la bandera de interrupción IF, del registro EFLAGS, impidiendo la ejecución del código restante, quedando en un bucle infinito.

- *En modo protegido, ¿Con qué valor se cargan los registros de segmento? ¿Por qué?*

En resumidas palabras, en la GDT se carga el offset para la entrada al descriptor de segmento, cada vez que hago un cambio de contexto.

Desarrollamos un poco más a continuación:

Visible Part	Hidden Part	
Segment Selector	Base Address, Limit, Access Information	CS
		SS
		DS
		ES
		FS
		GS

Figure 3-7. Segment Registers

En modo protegido, los registros de segmento CS, DS, SS, ES, FS, GS, contienen una estructura de datos más compleja que una simple dirección como en modo real, que contiene la base, que en conjunto con el offset nos

permite obtener la dirección física.

En cambio en modo protegido, el registro de segmento está compuesto por el índice + el TI (indicador de tabla) + RPL (privilegio).

Con el índice podemos acceder a una tabla de descriptores que nos brinda información sobre la base, el límite y los atributos del segmento al que queremos acceder.

Como ahora tenemos registros como el RPL (registro de segmento) y el DPL (descriptor) nos permite implementar mecanismos de protección e introducir niveles de privilegio para el acceso a los datos.

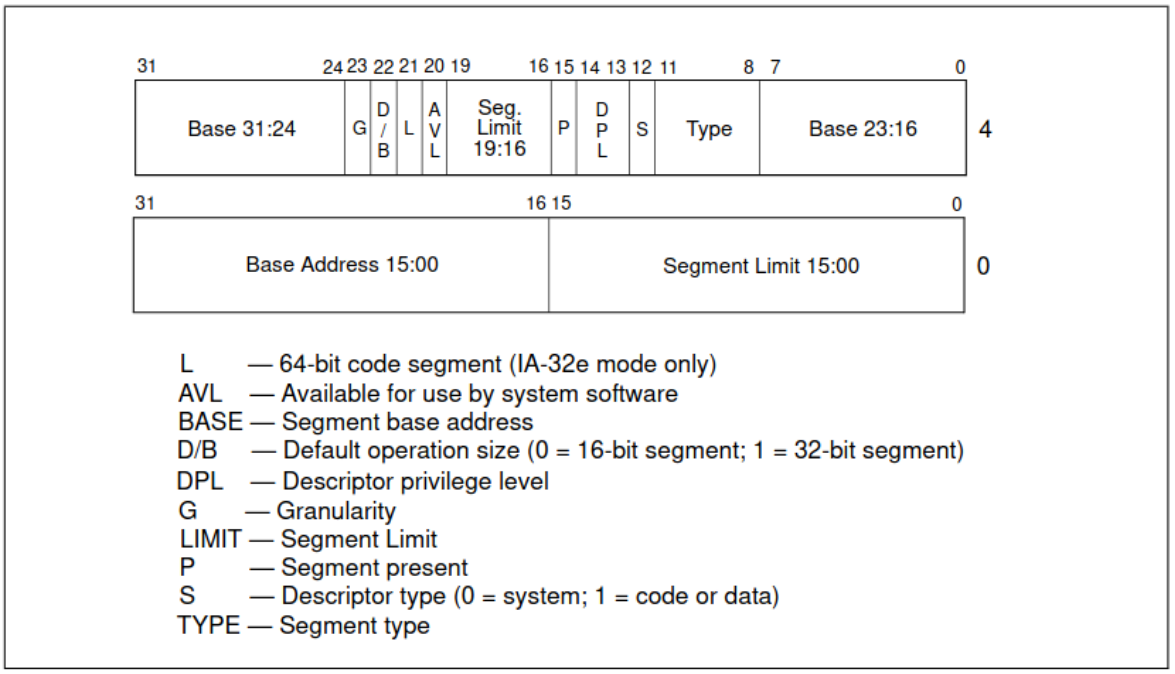
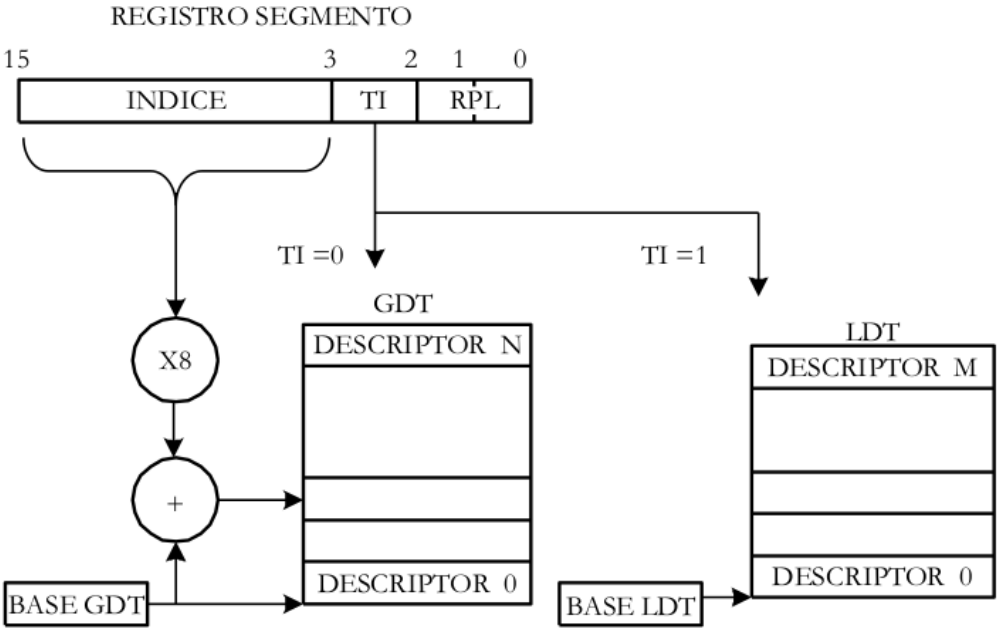


Figure 3-8. Segment Descriptor



Según el manual Intel 325384-053U parte 3 sección 3.4.2 “El procesador no utiliza la primera entrada de la GDT. Un selector de segmento que apunta a esta entrada de la GDT (que es decir, un selector de segmento con un índice de 0 y el indicador TI establecido en 0) se utiliza como un "selector de segmento nulo.”

Referencias

1. https://www.xataka.com/seguridad/error-csme-intel-permite-ataques-imposibles-detectar-para-parchearlo-al-completo-hace-falta-sustituir-chip
2. http://support.dell.com/support/edocs/systems/latd630/sp/AMT/MEBX.htm
3. https://securityaffairs.co/wordpress/127506/breaking-news/uefi-firmware-vulnerabilities.html
4. https://i.blackhat.com/USA-19/Wednesday/us-19-Hasarfaty-Behind-The-Scenes-Of-Intel-Security-And-Manageability-Engine.pdf
5. https://dplinux.net/coreboot-la-bios-libre/
6. https://doc.coreboot.org/distributions.html#hardware-shipping-with-coreboot
7. https://kripkit.com/enlazador-de-gnu/
8. https://ftp.gnu.org/old-gnu/Manuals/ld-2.9.1/html_node/ld_3.html
9. https://www.minitool.com/es/particionar-disco/uefi-vs-bios.html
10. https://programmerclick.com/article/5567978997