

Modo Protegido

Programación de Sistemas Operativos

David Alejandro González Márquez

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Modo Real

Programación en 16bits

Modo Real

Programación en 16bits

- No hay protección de memoria

Modo Real

Programación en 16bits

- No hay protección de memoria
- No se pueden restringir las instrucciones

Modo Real

Programación en 16bits

- No hay protección de memoria
- No se pueden restringir las instrucciones
- AX, CX y DX no son de propósito general, no se pueden usar para acceder a memoria

Modo Real

Programación en 16bits

- No hay protección de memoria
- No se pueden restringir las instrucciones
- AX, CX y DX no son de propósito general, no se pueden usar para acceder a memoria
- Los compiladores modernos no generan código para modo real

Modo Real

Programación en 16bits

- No hay protección de memoria
- No se pueden restringir las instrucciones
- AX, CX y DX no son de propósito general, no se pueden usar para acceder a memoria
- Los compiladores modernos no generan código para modo real
- + Podemos usar las rutinas del BIOS (por ejemplo, para imprimir por pantalla)

Modo Real

Programación en 16bits

- No hay protección de memoria
- No se pueden restringir las instrucciones
- AX, CX y DX no son de propósito general, no se pueden usar para acceder a memoria
- Los compiladores modernos no generan código para modo real
- + Podemos usar las rutinas del BIOS (por ejemplo, para imprimir por pantalla)
- + El BIOS atiende las interrupciones

Modo Real

Programación en 16bits

- No hay protección de memoria
- No se pueden restringir las instrucciones
- AX, CX y DX no son de propósito general, no se pueden usar para acceder a memoria
- Los compiladores modernos no generan código para modo real
- + Podemos usar las rutinas del BIOS (por ejemplo, para imprimir por pantalla)
- + El BIOS atiende las interrupciones
- ~ Tenemos Registros de Segmento (CS, DS, SS)

Modo Real

Programación en 16bits

- No hay protección de memoria
- No se pueden restringir las instrucciones
- AX, CX y DX no son de propósito general, no se pueden usar para acceder a memoria
- Los compiladores modernos no generan código para modo real
- + Podemos usar las rutinas del BIOS (por ejemplo, para imprimir por pantalla)
- + El BIOS atiende las interrupciones
- ~ Tenemos Registros de Segmento (CS, DS, SS)
- ~ Tenemos control total del sistema a nivel de usuario

Modo Real

Programación en 16bits

Pero, estamos solos contra el mundo...

Modo Real

Programación en 16bits

Pero, estamos solos contra el mundo...

No hay librerías no hay printf, ¡no hay nada!

Tenemos un binario plano, y *chau*

¡Ojo con ejecutar los datos!

section .data section .text...jajaja

¡Ojo con modificar el código en tiempo de ejecución!

No hay segmentation fault

Toda la memoria es casi nuestra

Modo Real

Programación en 16bits

Pero, estamos solos contra el mundo...

No hay librerías no hay printf, ¡no hay nada!

Tenemos un binario plano, y *chau*

¡Ojo con ejecutar los datos!

section .data section .text...jajaja

¡Ojo con modificar el código en tiempo de ejecución!

No hay segmentation fault

Toda la memoria es casi nuestra

Hasta que no pasemos a modo protegido,
tenemos el mejor 8086 de la historia

Direcccionamiento en 16bits

Modos de
direcccionamiento

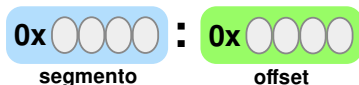
[val]	[BX + val]	[BX + SI + val]
	[SI + val]	[BX + DI + val]
	[DI + val]	[BP + SI + val]
	[BP + val]	[BP + DI + val]

Direccionamiento en 16bits

Modos de
direccionamiento

[val]	[BX + val]	[BX + SI + val]
	[SI + val]	[BX + DI + val]
	[DI + val]	[BP + SI + val]
	[BP + val]	[BP + DI + val]

Cada dirección de memoria esta definida por un **segmento** y un **offset**
(de 16 bits cada uno)

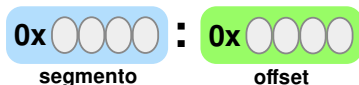


Direcccionamiento en 16bits

Modos de
direcccionamiento

[val]	[BX + val]	[BX + SI + val]
	[SI + val]	[BX + DI + val]
	[DI + val]	[BP + SI + val]
	[BP + val]	[BP + DI + val]

Cada dirección de memoria esta definida por un **segmento** y un **offset**
(de 16 bits cada uno)



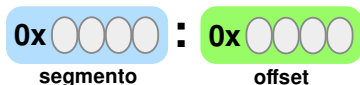
La forma de calcular a que dirección física que corresponde es:
(segmento << 4) + offset

Direccionamiento en 16bits

Modos de
direccionamiento

[val]	[BX + val]	[BX + SI + val]
	[SI + val]	[BX + DI + val]
	[DI + val]	[BP + SI + val]
	[BP + val]	[BP + DI + val]

Cada dirección de memoria esta definida por un **segmento** y un **offset**
(de 16 bits cada uno)



La forma de calcular a que dirección física que corresponde es:
(segmento << 4) + offset

Por ejemplo:

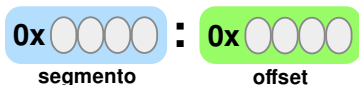
$$(0x07C0 \ll 4) + 0x0120 = 0x7C00 + 0x0120 = 0x7D20$$

Direcccionamiento en 16bits

Modos de
direccionamiento

[val]	[BX + val]	[BX + SI + val]
	[SI + val]	[BX + DI + val]
	[DI + val]	[BP + SI + val]
	[BP + val]	[BP + DI + val]

Cada dirección de memoria esta definida por un **segmento** y un **offset**
(de 16 bits cada uno)



La forma de calcular a que dirección física que corresponde es:
(segmento << 4) + offset

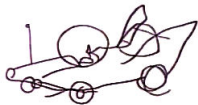
Por ejemplo:

$$(0x07C0 \ll 4) + 0x0120 = 0x7C00 + 0x0120 = 0x7D20$$

segmento offset

Modo Real vs Modo Protegido

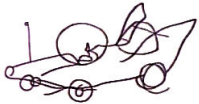

Modo Real



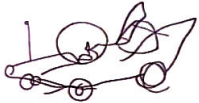

Modo Protegido



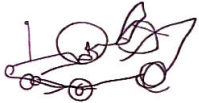

Modo Real vs Modo Protegido

	Modo Real	Modo Protegido
Memoria disponible	 1Mb*	 4Gb*

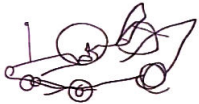

Modo Real vs Modo Protegido

	Modo Real	Modo Protegido
		
Memoria disponible	1Mb*	4Gb*
Privilegios	¿cuac?	4 niveles de protección

Modo Real vs Modo Protegido

	Modo Real	Modo Protegido
		
Memoria disponible	1Mb*	4Gb*
Privilegios	¿cuac?	4 niveles de protección
Manejo de Interrupciones	rutinas de atención	rutinas de atención con privilegios

Modo Real vs Modo Protegido

	Modo Real	Modo Protegido
		
Memoria disponible	1Mb*	4Gb*
Privilegios	¿cuac?	4 niveles de protección
Manejo de Interrupciones	rutinas de atención	rutinas de atención con privilegios
Acceso a instrucciones	todas	depende del nivel de protección

GDT - Global Descriptor Table

Tabla en memoria donde **cada entrada es de 8 bytes**.

Define alguno de los siguientes descriptores:

GDT - Global Descriptor Table

Tabla en memoria donde **cada entrada es de 8 bytes**.
Define alguno de los siguientes descriptores:

- **Descriptor de segmento de memoria** (S=1)

GDT - Global Descriptor Table

Tabla en memoria donde **cada entrada es de 8 bytes**.

Define alguno de los siguientes descriptores:

- **Descriptor de segmento de memoria** ($S=1$)
- **Descriptor de Task State Segment (TSS)** ($S=0$)
Guarda el estado de una tarea, sirve para intercambiar tareas

GDT - Global Descriptor Table

Tabla en memoria donde **cada entrada es de 8 bytes**.
Define alguno de los siguientes descriptores:

- **Descriptor de segmento de memoria** ($S=1$)
- Descriptor de Task State Segment (TSS) ($S=0$)
Guarda el estado de una tarea, sirve para intercambiar tareas
- Descriptor de call gate ($S=0$)
Permite transferir control entre niveles de privilegios
Actualmente no se usan en SO modernos

GDT - Global Descriptor Table

Tabla en memoria donde **cada entrada es de 8 bytes**.
Define alguno de los siguientes descriptores:

- **Descriptor de segmento de memoria** ($S=1$)
- Descriptor de Task State Segment (TSS) ($S=0$)
Guarda el estado de una tarea, sirve para intercambiar tareas
- Descriptor de call gate ($S=0$)
Permite transferir control entre niveles de privilegios
Actualmente no se usan en SO modernos
- Descriptor de LDT ($S=0$)

GDT - Global Descriptor Table

Tabla en memoria donde **cada entrada es de 8 bytes**.
Define alguno de los siguientes descriptores:

- **Descriptor de segmento de memoria** ($S=1$)
- Descriptor de Task State Segment (TSS) ($S=0$)
Guarda el estado de una tarea, sirve para intercambiar tareas
- Descriptor de call gate ($S=0$)
Permite transferir control entre niveles de privilegios
Actualmente no se usan en SO modernos
- Descriptor de LDT ($S=0$)

El primer descriptor de la tabla siempre es NULO

LDT - Local Descriptor Table

Tabla en memoria, igual que la GDT.

Puede conterner las **mismas entradas que la GDT**

LDT - Local Descriptor Table

Tabla en memoria, igual que la GDT.

Puede conterner las **mismas entradas que la GDT**

Se diferencia en:

- La GDT tiene los descriptores globales y es única para todo el sistema.

LDT - Local Descriptor Table

Tabla en memoria, igual que la GDT.

Puede conterner las **mismas entradas que la GDT**

Se diferencia en:

- La GDT tiene los descriptores globales y es única para todo el sistema.
- La LDT tiene los descriptores locales a una tarea y puede existir más de una LDT en el sistema, una por cada tarea.

LDT - Local Descriptor Table

Tabla en memoria, igual que la GDT.

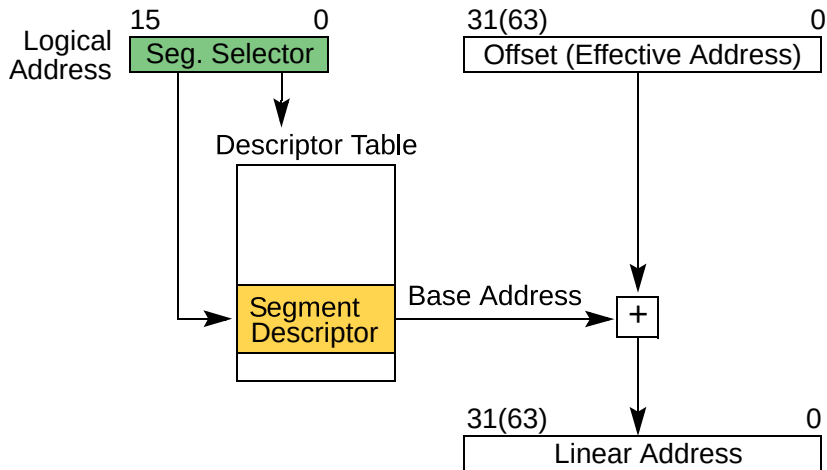
Puede conterner las **mismas entradas que la GDT**

Se diferencia en:

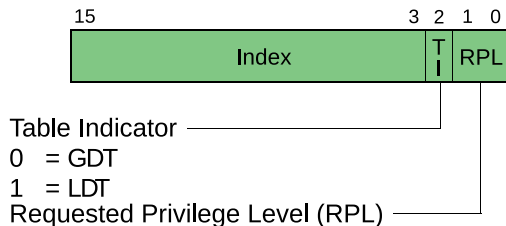
- La GDT tiene los descriptores globales y es única para todo el sistema.
- La LDT tiene los descriptores locales a una tarea y puede existir más de una LDT en el sistema, una por cada tarea.

Tabla obsoleta por el uso del mecanismo de paginación

Unidad de Segmentación



Selector de Segmento



CS: Para acceder a código

SS: Para acceder a pila

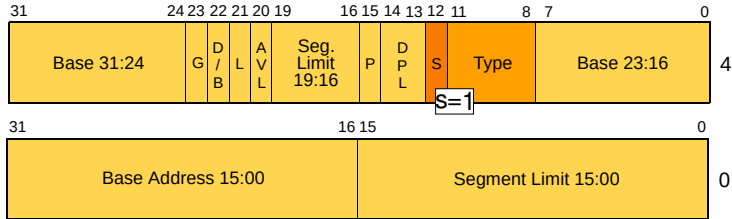
DS: Para acceder a datos (default)

ES: Para acceder a datos

GS: Para acceder a datos

FS: Para acceder a datos

Descriptor de Segmento



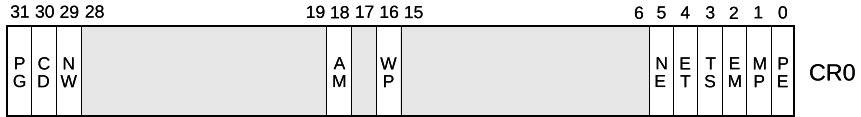
- L — 64-bit code segment (IA-32e mode only)
- AVL — Available for use by system software
- BASE — Segment base address
- D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
- DPL — Descriptor privilege level
- G — Granularity
- LIMIT — Segment Limit
- P — Segment present
- S — Descriptor type (0 = system; 1 = code or data)
- TYPE — Segment type

Tipo de Selector de segmento

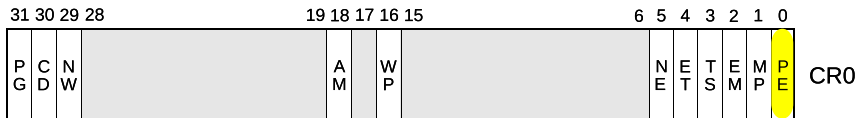
Type

Type Field					Descriptor Type	Description
Decimal	11	10 E	9 W	8 A		
0	0	0	0	0	Data	Read-Only
1	0	0	0	1	Data	Read-Only, accessed
2	0	0	1	0	Data	Read/Write
3	0	0	1	1	Data	Read/Write, accessed
4	0	1	0	0	Data	Read-Only, expand-down
5	0	1	0	1	Data	Read-Only, expand-down, accessed
6	0	1	1	0	Data	Read/Write, expand-down
7	0	1	1	1	Data	Read/Write, expand-down, accessed
		C	R	A		
8	1	0	0	0	Code	Execute-Only
9	1	0	0	1	Code	Execute-Only, accessed
10	1	0	1	0	Code	Execute/Read
11	1	0	1	1	Code	Execute/Read, accessed
12	1	1	0	0	Code	Execute-Only, conforming
13	1	1	0	1	Code	Execute-Only, conforming, accessed
14	1	1	1	0	Code	Execute/Read, conforming
15	1	1	1	1	Code	Execute/Read, conforming, accessed

Pasar a modo protegido



Pasar a modo protegido



Activar **Modo Protegido** es setear en 1 el bit **PE** del registro de control **CR0**

Protected Environment

cataplum.

1. onomat. U. para expresar ruido, explosión o golpe.

rae.org

Pasar a Modo Protegido - ¿Por qué cataplum?

¿Cómo sabemos donde esta la GDT?

Pasar a Modo Protegido - ¿Por qué cataplum?

¿Cómo sabemos donde esta la GDT?

Cargar el registro GDTR utilizando LGDT

¿Qué tiene la GDT?

Pasar a Modo Protegido - ¿Por qué cataplum?

¿Cómo sabemos donde esta la GDT?

Cargar el registro GDTR utilizando LGDT

¿Qué tiene la GDT?

Al menos, un descriptor nulo, un descriptor de código y uno de datos

¿Cuál es la próxima instrucción a ejecutar?

Pasar a Modo Protegido - ¿Por qué cataplum?

¿Cómo sabemos donde esta la GDT?

Cargar el registro GDTR utilizando LGDT

¿Qué tiene la GDT?

Al menos, un descriptor nulo, un descriptor de código y uno de datos

¿Cuál es la próxima instrucción a ejecutar?

La instrucción en la dirección CS:EIP

¿Qué valor tiene que tener CS y cómo lo cambiamos?

Pasar a Modo Protegido - ¿Por qué cataplum?

¿Cómo sabemos donde esta la GDT?

Cargar el registro GDTR utilizando LGDT

¿Qué tiene la GDT?

Al menos, un descriptor nulo, un descriptor de código y uno de datos

¿Cuál es la próxima instrucción a ejecutar?

La instrucción en la dirección CS:EIP

¿Qué valor tiene que tener CS y cómo lo cambiamos?

```
..... ; esto se ejecuta en modo real
```

```
jmp 0x08:modoprotegido
```

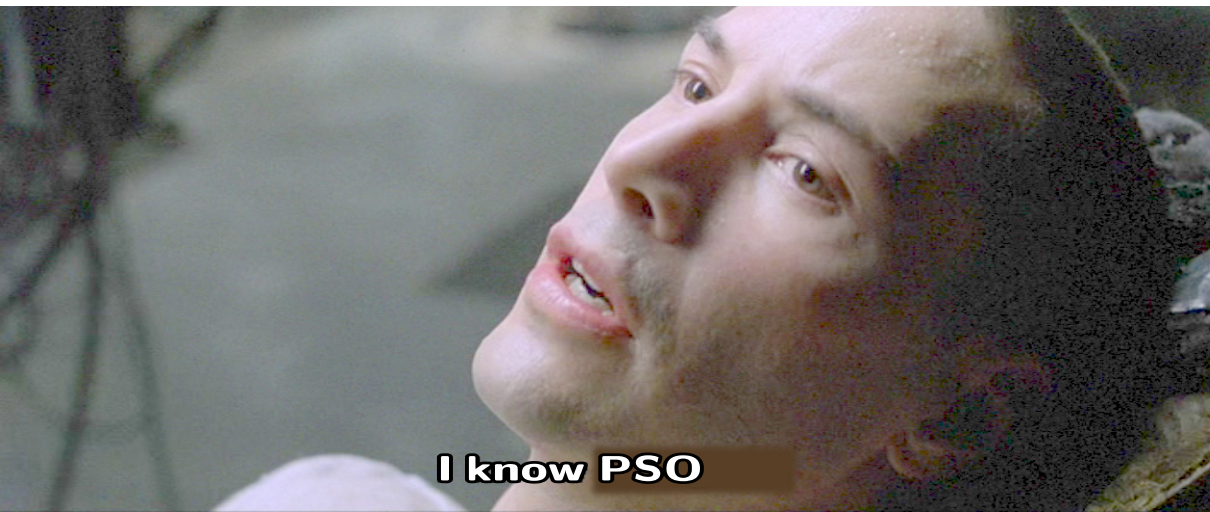
```
; GRAN SALTO!
```

```
modoprotegido:
```

```
.... ; esto se ejecuta en modo protegido
```

Pasar a Modo Protegido - Pasos

- 0- Completar la GDT
- 1- Deshabilitar interrupciones (CLI)
- 2- Cargar el registro GDTR con la dirección base de la GDT
LGDT <offset>
- 3- Setear el bit PE del registro CR0
MOV eax,cr0
OR eax,1
MOV cr0,eax
- 4- FAR JUMP a la siguiente instrucción
JMP <selector>:<offset>
- 5- Cargar los registros de segmento (DS, ES, GS, FS y SS)



I know PSO

Bibliografía: Fuentes y material adicional

- Convenciones de llamados a función en x86:
https://en.wikipedia.org/wiki/X86_calling_conventions
- Notas sobre System V ABI:
https://wiki.osdev.org/System_V_ABI
- Documentación de NASM:
<https://nasm.us/doc/>
- Artículo sobre el flag -pie:
<https://eklitzke.org/position-independent-executables>
- Documentación de System V ABI:
https://uclibc.org/docs/psABI-x86_64.pdf
- Manuales de Intel:
<https://software.intel.com/en-us/articles/intel-sdm>

¡Gracias!

Recuerden leer los comentarios al final de este video por aclaraciones o fe de erratas.