

# Organización del Computador II

## Procesadores IA-32 e Intel®64

Alejandro Furfaro

Departamento de Computación - UBA-FCEyN

22 de marzo de 2019

# Contenido

1

## Inroducción

- Genealogía

- Arquitectura Básica

2

## Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32
- Arquitectura Intel® 64

3

## Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro
- Modos de Direccionamiento a memoria

- Modo Desplazamiento
- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

4

## Tipos de Datos

- Almacenamiento en memoria
- Alineación en memoria

5

## Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

- 1 Introducción
  - Genealogía
  - Arquitectura Básica
- 2 Modelo del Programador de aplicaciones
  - Arquitectura de 16 bits básica
  - IA-32
  - Arquitectura Intel® 64
- 3 Modos de Direccionamiento
  - Modo Implícito
  - Modo Inmediato
  - Modo Registro
  - Modos de Direccionamiento a memoria

- Modo Desplazamiento
- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

- 4 Tipos de Datos
  - Almacenamiento en memoria
  - Alineación en memoria
- 5 Ejemplos de uso de pila
  - ¿Como funciona un llamado Near?
  - ¿Como funciona un llamado Far?
  - Interrupciones

# Orígenes

## La bisagra

Unos meses luego del lanzamiento del 8086, Intel presenta una variante del mismo procesador: el 8088, idéntico en su arquitectura interna de 16 bits, pero con un bus de datos de 8 bits, que a diferencia del bus de datos de 16 bits del 8086, permitía una conectividad más directa con la innumerable cantidad de periféricos de 8 bits que por entonces había disponibles. Esta razón terminó de convencer a los ingenieros de IBM que lanzaría su primer PC basada en un procesador 8088 (nunca usó el 8086). Esta decisión de IBM es la causa del liderazgo de Intel en esta industria desde entonces.

# Orígenes

## la primer prueba

En 1982 Intel presenta el 80286, que mejoraba la CPU 8086 en la ejecución de numerosas instrucciones e incorporó multitasking. Su arquitectura de 16 bits seguía siendo la misma, y era compatible a nivel binario (si.. no hay que recompilar siquiera) con el 8088. A partir de este procesador IBM diseña el modelo de PC conocido como AT.

# Orígenes

## La consolidación

- En 1984 Intel lanza el primer procesador de 32 bits, el 80386, y con él la arquitectura IA-32
- La industria comienza a hablar de los procesadores x86, que en ese momento constituyen un standard
- Aparecen fabricantes que producirán procesadores IA-32, bajo licencia de Intel. Entre ellos se destaca AMD

# Orígenes

Algún bache siempre aparece...

- A fines de los '90 Intel y Hewlett Packard desarrollan una arquitectura de 64 bits.
- Los procesadores Itanium e Itanium2 no tendrán el éxito de los x86.
- AMD agregó las extensiones de 64 bits a los procesadores IA-32. Esta nueva arquitectura x86-64 (o AMD64) fue un éxito.
- Intel debió adoptarla para IA-32. Llama a esta arquitectura Intel® 64 .

# Primeras conclusiones

- Palabra clave: Compatibilidad. Fue la llave el enorme éxito de esta arquitectura, pero también la ha condicionado en determinados períodos de su evolución. En los mas de 40 años pasados ha habido períodos en los que otras alternativas le han sacado ventaja y le ha costado mas esfuerzo poder competir.
- Es una arquitectura que no puede ser analizada sin tener en cuenta estos factores “históricos”. El compromiso de Compatibilidad adoptado en 1978 significa que las decisiones de fondo adoptadas inicialmente en el diseño de la arquitectura, inevitablemente condicionarán lo que puede o no puede hacerse a futuro. La administración de memoria por segmentación como veremos es un clarísimo ejemplo de ello.

1

## Inroducción

- Genealogía

## Arquitectura Básica

2

## Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32
- Arquitectura Intel® 64

3

## Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro
- Modos de Direccionamiento a memoria

- Modo Desplazamiento

- Modo Base Directo

- Base + Desplazamiento

- Base + Desplazamiento

- Índice \* escala + desplazamiento

- Base + Índice + Desplazamiento

- Modo Base + Índice \* Escala + Desplazamiento

4

## Tipos de Datos

- Almacenamiento en memoria
- Alineación en memoria

5

## Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

# Modos de Operación

Los procesadores IA-32 tienen tres modos de operación, de los cuales surgen sus capacidades arquitecturales e instrucciones disponibles, y un modo especial para tareas específicas del sistema:

- ① Modo Real
- ② Modo Protegido
- ③ Modo Mantenimiento del Sistema
- ④ Modo extendido a 64 bits (IA-32e)
  - Submodo Compatibilidad
  - Submodo 64 bits

# Modos de Operación: *Modo Real*

- En este modo el procesador implementa el entorno de operación del 8086, con algunas extensiones:
  - puede utilizar registros de 32 bits
  - puede reconfigurar la ubicación del vector de interrupciones, ya que a pesar de que el 8086 no lo tenía, ahora existe y es accesible desde modo Real el registro IDTR (ver Interrupciones mas adelante)
- Desde este modo se puede pasar por software al Modo Protegido o al Modo Mantenimiento del Sistema.
- Aunque nadie trabaja en este modo, es el modo de arranque de cualquier procesador IA-32 e Intel® 64 actual y futuro... Delicias de la ***compatibilidad***.

# Modos de Operación *Modo Protegido*

- En este modo se implementa multitasking
- Este es el modo por excelencia de los procesadores de esta familia inaugurado por el procesador 80286, primer procesador con capacidad multitarea, a pesar de su arquitectura de 16 bits similar a la de su antecesor 8086.
- A partir del 80386 se pasa a 32 bits (nace IA-32) y se despliega un espacio de direccionamiento de 4 Gbytes, extensible a 64 Gbytes.
- Se introduce un sub-modo al que puede ponerse a una determinada tarea, denominado Virtual-8086 que permite a un programa diseñado para ejecutarse en un procesador 8086, poder ejecutarse como una tarea en Modo Protegido. Esto fue muy útil para implementar en Windows la “Ventana DOS”. Actualmente no es utilizado, ya que la consola que se ejecuta utiliza un código diferente del DOS original, y es en general una tarea mas.

# Modos de Operación **Modo Mantenimiento**

- El procesador ingresa a este modo por dos caminos
  - Activación de la señal de interrupción #SMM.
  - mediante un mensaje SMI desde su APIC local (Ver Interrupciones y SMP mas adelante)
- Este modo fue introducido a partir de los modelos 386SL y 486SL para realizar funciones específicas para la plataforma de hardware en la cual se desempeña el procesador, como lo son ahorro de energía y seguridad. Estos procesadores fueron los primeros diseñados para notebooks.
- Al ingresar a este modo el procesador resguarda en forma automática el contexto completo de la tarea o programa interrumpido, y pasa a ejecutar en un espacio separado. Una vez efectuadas las operaciones necesarias y cuando debe salir de este modo el procesador reasume la tarea o programa interrumpida en el modo de operación en el que se encontraba.

# Modos de Operación: *Extendido a 64 bits*

- Los procesadores Intel® 64 además de los modos de trabajo de los procesadores IA-32 incluyen un modo IA-32e en el que se activa la arquitectura de 64 bits.
- Para pasar a este modo, el procesador debe estar trabajando en modo protegido, con paginación habilitada, y PAE activo (Como estudiaremos en Paginación)
- En IA-32e a su vez existen dos sub-modos (nada es simple en este mundo).
  - ① Sub modo Compatibilidad
  - ② Sub modo 64 bits.

# Modos de Operación IA-32e: *Modo Compatibilidad*

- Pensado para garantizar la transición de 32 a 64 bits, permite a las aplicaciones de 16 y 32 bits legacy ejecutarse sin recompilación bajo un sistema operativo de 64 bits.
- Los aspectos relacionados con el sistema operativo se mantienen en 64 bits. La compatibilidad es solo para las aplicaciones:
  - ① El entorno de ejecución de la tarea de 32 bits es el de la arquitectura IA-32.
  - ② El kernel no soporta el manejo de tareas del modo IA-32,
  - ③ El sub-modo Virtual 8086 desaparece(por esto dijimos que este modo está obsoleto en un slide anterior).
  - ④ Incluye los mecanismos de protección del modo 64 bits.
- El Sistema Operativo de 64 bits puede ejecutar tareas de 16 y 32 bits junto con otras de 64 bits, sobre la base de diferentes segmentos de código. La tarea de 32 bits accede a una arquitectura IA-32 pura, utilizando direcciones de 16 y 32 bits, con 4 Gbytes de espacio de direccionamiento y con la posibilidad de acceder por encima de ese límite habilitando PAE.

# Modos de Operación de 64 bits

- **Modo 64 bits:** Este modo habilita a un Sistema Operativo de 64 bits a ejecutar tareas escritas utilizando direcciones lineales de 64 bits. En este modo se extienden de 8 a 16 los Registros de propósito general cuyo ancho de palabra ahora es de 64 bits (para ello se introduce el prefijo REX para las instrucciones que deseen acceder a las versiones de Registros de propósito general de 64 bits), agregándose los registros R8 a R15, y los registros SIMD también se extienden a 16 manteniendo su ancho de 128 bits, XMM0 a XMM15.

1

## Inroducción

- Genealogía
- Arquitectura Básica

2

## Modelo del Programador de aplicaciones

### ● Arquitectura de 16 bits básica

- IA-32
- Arquitectura Intel® 64

3

## Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro
- Modos de Direccionamiento a memoria

- Modo Desplazamiento

- Modo Base Directo

- Base + Desplazamiento

- Base + Desplazamiento

- Índice \* escala + desplazamiento

- Base + Índice + Desplazamiento

- Modo Base + Índice \* Escala + Desplazamiento

4

## Tipos de Datos

- Almacenamiento en memoria
- Alineación en memoria

5

## Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

# Registros y espacio de direccionamiento

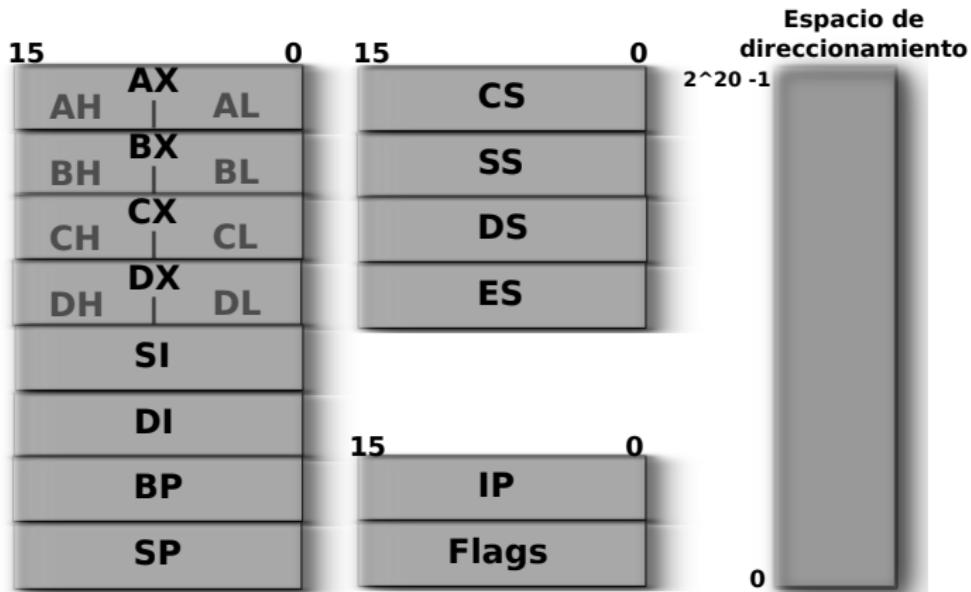


Figura: Entorno Básico de ejecución en 16 bits

1

## Introducción

- Genealogía
- Arquitectura Básica

2

## Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32**
- Arquitectura Intel® 64

3

## Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro
- Modos de Direccionamiento a memoria

- Modo Desplazamiento
- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

4

## Tipos de Datos

- Almacenamiento en memoria
- Alineación en memoria

5

## Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

# Arquitectura de 32 bits compatible

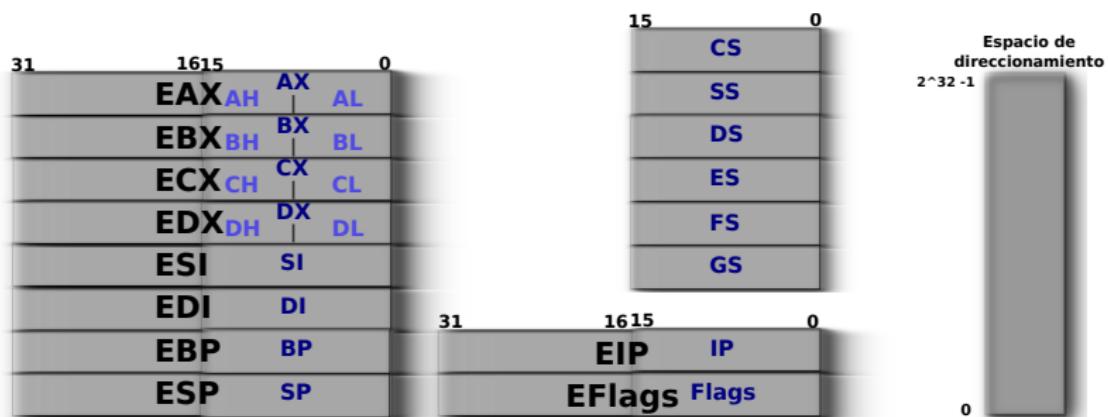


Figura: Entorno Básico de ejecución del 80386

# Floating Point Unit

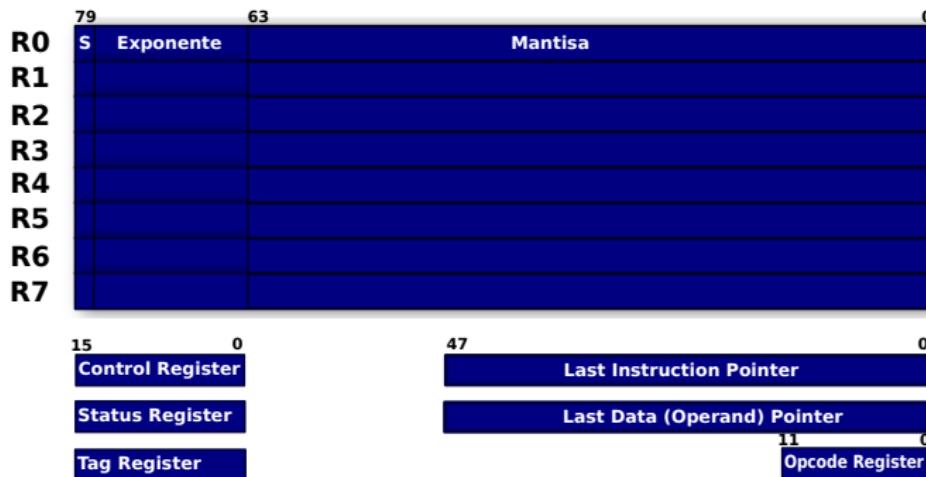


Figura: Entorno Básico de ejecución de la FPU

# Floating Point Unit

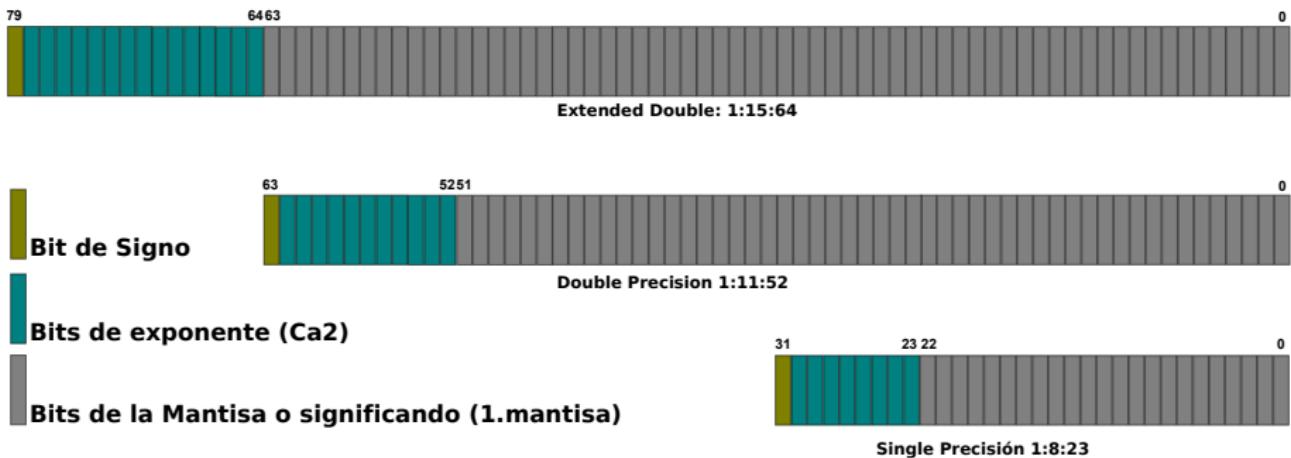


Figura: Formatos de los Datos de Punto Flotante de la FPU

# Floating Point Unit on board

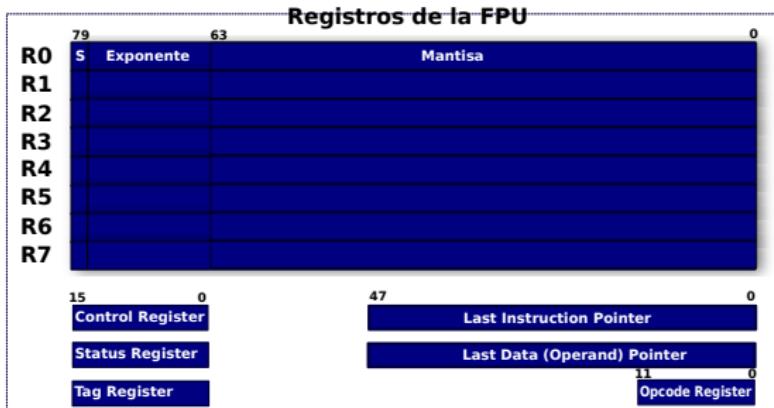


Figura: Entorno Básico de ejecución con la FPU on board

# Extensiones MMX

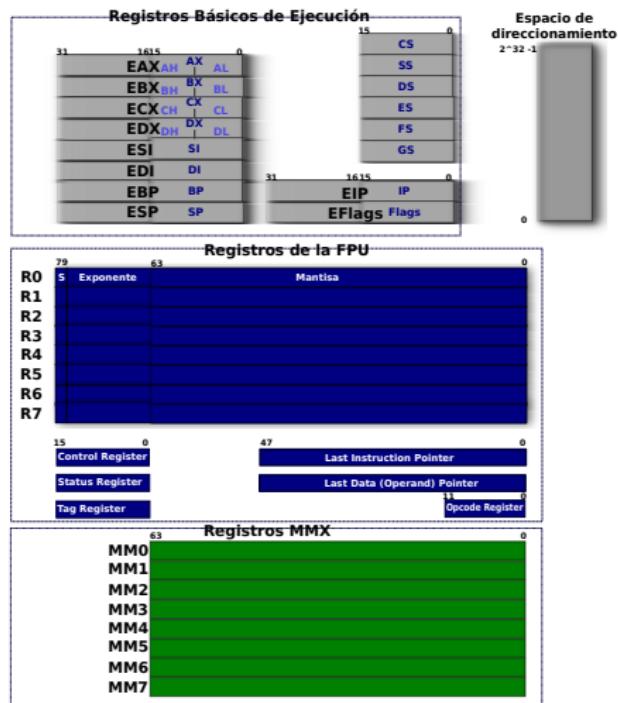


Figura: Entorno Básico de ejecución con tecnología MMX

# Llega el Pentium III

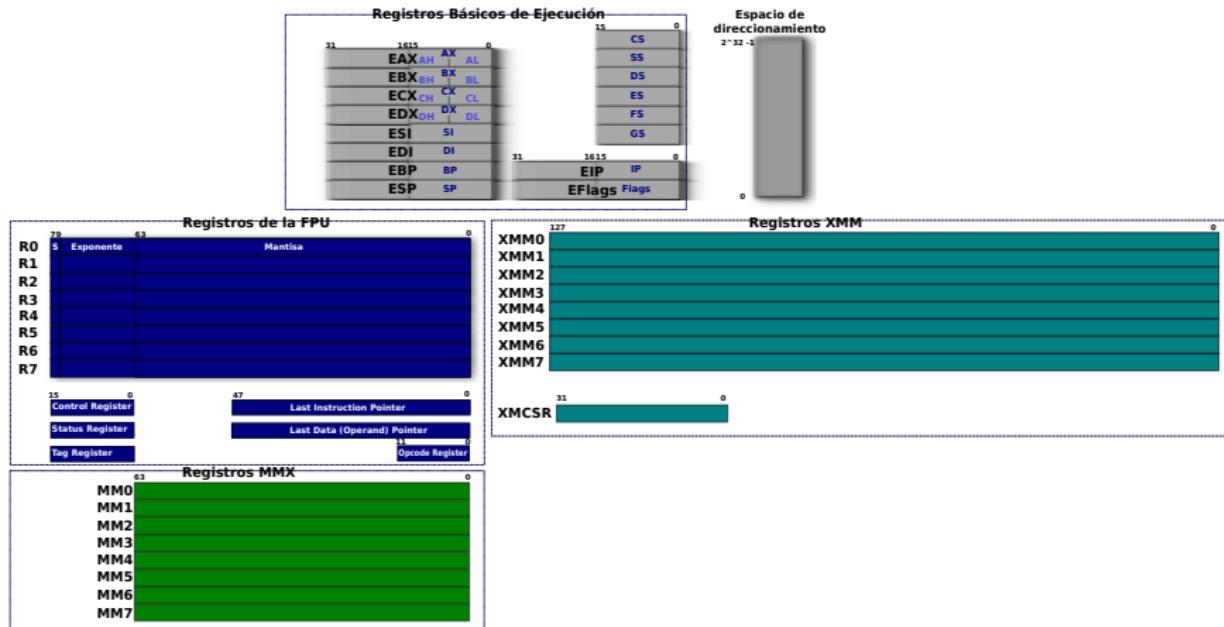


Figura: Entorno Básico de ejecución IA-32 actual

1

## Inroducción

- Genealogía
- Arquitectura Básica

2

## Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32

## ● Arquitectura Intel® 64

3

## Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro
- Modos de Direccionamiento a memoria

- Modo Desplazamiento

- Modo Base Directo

- Base + Desplazamiento

- Base + Desplazamiento

- Índice \* escala + desplazamiento

- Base + Índice + Desplazamiento

- Modo Base + Índice \* Escala + Desplazamiento

4

## Tipos de Datos

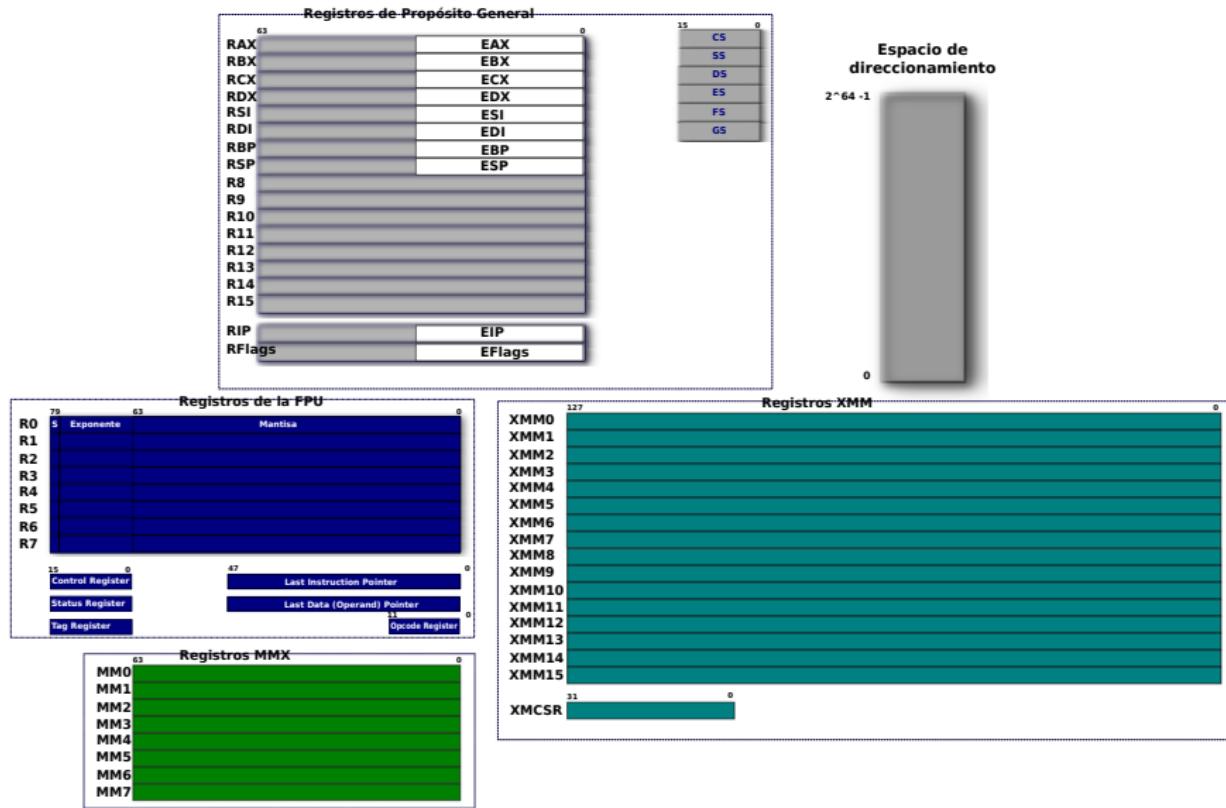
- Almacenamiento en memoria
- Alineación en memoria

5

## Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

# Extensiones de 64 bits



## Extensiones de 64 bits

En este modo los registros de Propósito General se extienden a 64 bits y aparecen 8 registros de 64 bits adicionales. El procesador a pesar de estar en modo 64 bits puede acceder a diferentes tamaños de operador. Para ello es necesario que utilice el prefijo REX, antecediendo a cada instrucción que requiera este tipo de operandos.

Registros de Propósito General

	63	0
RAX		EAX
RBX		EBX
RCX		ECX
RDX		EDX
RSI		ESI
RDI		EDI
RBP		EBP
RSP		ESP
R8		R8D
R9		R9D
R10		R10D
R11		R11D
R12		R12D
R13		R13D
R14		R14D
R15		R15D

# Extensiones de 64 bits

Registros de Propósito General

	63	0
RAX		AX
RBX		BX
RCX		CX
RDX		DX
RSI		SI
RDI		DI
RBP		BP
RSP		SP
R8		R8W
R9		R9W
R10		R10W
R11		R11W
R12		R12W
R13		R13W
R14		R14W
R15		R15W

Figura: Intel® 64 : Registros de Propósito general para operandos word, utilizando prefijo REX

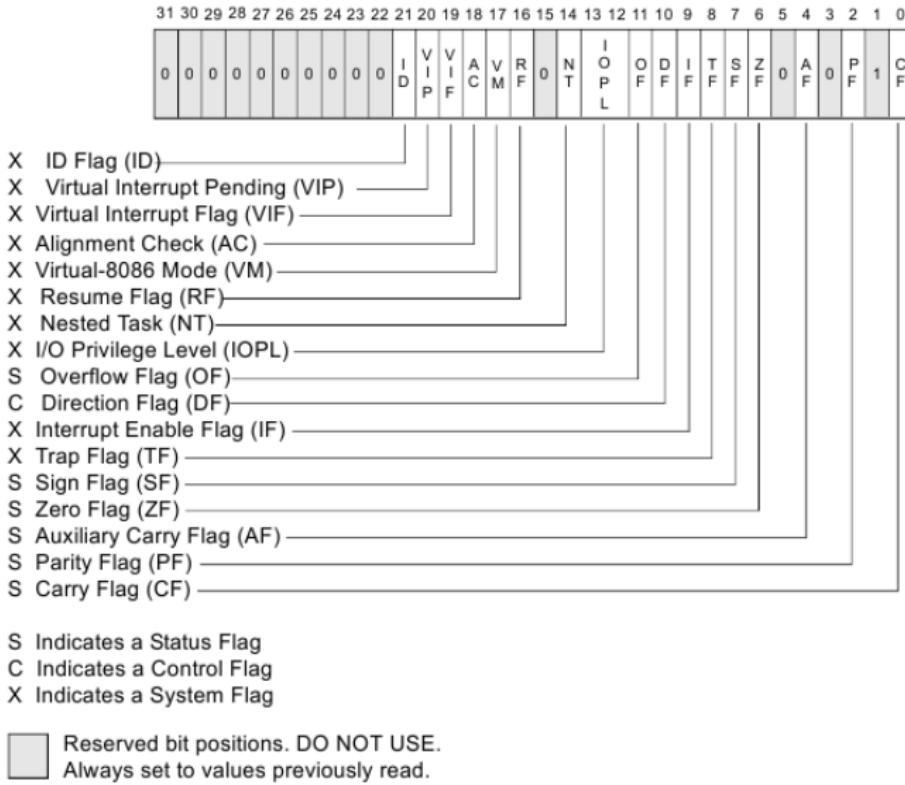
# Extensiones de 64 bits

Registros de Propósito General

	63	0
RAX		AL
RBX		BL
RCX		CL
RDX		DL
RSI		SIL
RDI		DIL
RBP		BPL
RSP		SPL
R8		R8L
R9		R9L
R10		R10L
R11		R11L
R12		R12L
R13		R13L
R14		R14L
R15		R15L

Figura: Intel® 64 : Registros de Propósito general para operandos byte, utilizando prefijo REX

# Flags, EFLAGS, RFLAGS



## Como Obtener los Operandos

Como reglas generales, de acuerdo a la documentación disponible, las instrucciones de estos procesadores pueden obtener los operandos desde:

- La instrucción en si misma, es decir que el operando está implícito en la instrucción)
- Un registro
- Una posición de memoria
- Un port de E/S

# Como Almacenar los resultados

Del mismo modo el resultado de una instrucción puede tener como destino para su almacenamiento:

- Un registro
- Una posición de memoria
- Un port de E/S.

- 1 Introducción
  - Genealogía
  - Arquitectura Básica
- 2 Modelo del Programador de aplicaciones
  - Arquitectura de 16 bits básica
  - IA-32
  - Arquitectura Intel® 64
- 3 Modos de Direccionamiento
  - Modo Implícito
  - Modo Inmediato
  - Modo Registro
  - Modos de Direccionamiento a memoria

- Modo Desplazamiento
- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

- 4 Tipos de Datos
  - Almacenamiento en memoria
  - Alineación en memoria
- 5 Ejemplos de uso de pila
  - ¿Como funciona un llamado Near?
  - ¿Como funciona un llamado Far?
  - Interrupciones

# ¿Que significa Direccionamiento Implícito?

Se trata de instrucciones en las cuales el código de operación es suficiente como para establecer que operación realizar, y cual es el operando.

Son ejemplos de este Modo, las instrucciones que operan sobre los Flags, como por ejemplo:

- CLC: Clear Carry. El operando (el Flag CF), está implícito en la operación. Lo mismo ocurre con STC (Set Carry), CMC (Complement Carry),
- CLD (Clear Direction Flag), STD (Set Direction Flag),
- CLI y STI para limpiar y setear el flag de Interrupciones (IF),

entre otros.

# Mas Instrucciones con Direcciónamiento Implícito

Otras instrucciones con modo de direcciónamiento implícito son algunas que en realidad no requieren operandos, ya que tienen por objeto indicar al procesador alguna acción específica. Por ejemplo:

- HLT, o NOP
- Ajustes para operaciones aritméticas sobre números en formato BCD o ASCII empaquetado, como AAA, AAD, AAM, AAS, DAA, DAS. Todas estas instrucciones operan sobre el Acumulador AX, razón por la cual no es necesario indicar el operando en la instrucción.
- CWD/CDQ/CQO Convert Word to Double Word y Convert Double Word to Quadword, que operan con el contenido de AX EAX y RAX extendiéndolo con signo a DX:AX, EDX:EAX, y RDX:RAX respectivamente.

1

## Introducción

- Genealogía
- Arquitectura Básica

2

## Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32
- Arquitectura Intel® 64

3

## Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato**
- Modo Registro
- Modos de Direccionamiento a memoria

- Modo Desplazamiento

- Modo Base Directo

- Base + Desplazamiento

- Base + Desplazamiento

- Índice \* escala + desplazamiento

- Base + Índice + Desplazamiento

- Modo Base + Índice \* Escala + Desplazamiento

4

## Tipos de Datos

- Almacenamiento en memoria
- Alineación en memoria

5

## Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

# ¿Que significa direccionar en Modo Inmediato?

En este modo, el operando fuente viene dentro del código de la instrucción, con lo cual no es necesario ir a buscarlo a memoria luego de decodificada la instrucción.

```
1 ; /////////////////////////////////
2 ; ascii recibe en al un byte decimal no empaquetado
3 ; y retorna su ascii en el mismo registro
4 ; /////////////////////////////////
5 ascii:
6     add    al , '0'
7     cmp    al , '9'
8     jle    listo
9     add    al , 'A' - '9' - 1
10 listo :   ret
```

- 1 Introducción
  - Genealogía
  - Arquitectura Básica
- 2 Modelo del Programador de aplicaciones
  - Arquitectura de 16 bits básica
  - IA-32
  - Arquitectura Intel® 64
- 3 Modos de Direcccionamiento
  - Modo Implícito
  - Modo Inmediato
  - Modo Registro**
  - Modos de Direcccionamiento a memoria

- Modo Desplazamiento
- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

- 4 Tipos de Datos
  - Almacenamiento en memoria
  - Alineación en memoria
- 5 Ejemplos de uso de pila
  - ¿Como funciona un llamado Near?
  - ¿Como funciona un llamado Far?
  - Interrupciones

# ¿Que significa direccionar a Registro?

Todos los operandos involucrados son Registros del procesador. No importa si hay uno o dos operandos, son todos Registros.

- Registros de Propósito General tanto de 64, 32, 16 u 8 bits de acuerdo con las arquitecturas IA-32 e Intel® 64
- Registros de segmentos
- EFlags (o RFlags)
- Registros de la FPU,
- MM0 a MM7,
- XMM0 a XMM7 o XMM15 según sea IA-32 o Intel® 64 respectivamente,
- Registros de Control
- Registros de Debug
- MSR's (Model Specific Registers)

```
1 inc rdx          ; Incrementa en contenido del registro RD
2 mov eax,ebp      ; Mueve al registro EAX el contenido de EBP
3 mov cr0,eax      ; Mueve al registro cr0 el contenido de EAX
4 fmul st(0),st(3) ; ST(0) = ST(0) * ST(3)
5 sqrtpd xmm2,xmm6 ; Raíz cuadrada de dos double precision FP
6                      ; empaquetados en xmm6. Resultados en xmm6
```

**1** Introducción

- Genealogía
- Arquitectura Básica

**2** Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32
- Arquitectura Intel® 64

**3** Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro

**4** Modos de Direccionamiento a memoria

- Modo Desplazamiento
- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

**4** Tipos de Datos

- Almacenamiento en memoria
- Alineación en memoria

**5** Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

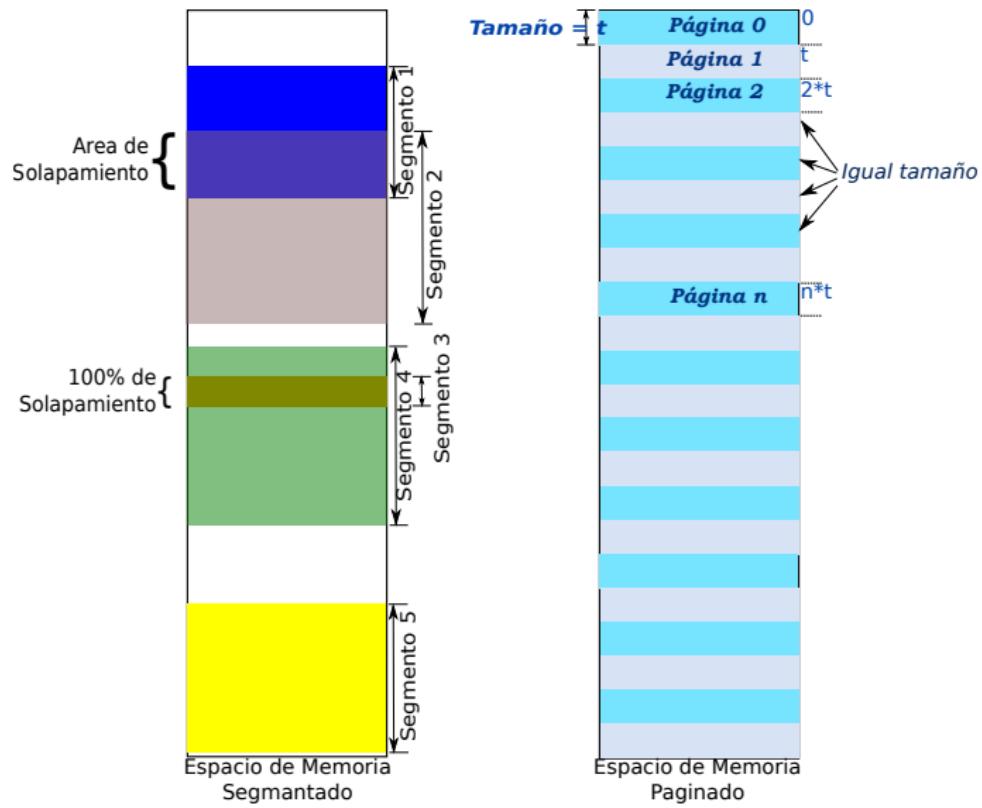
# Espacio Físico

- Los procesadores IA-32 organizan la memoria como una secuencia de bytes, direccionables a través de su Bus de Address.
- La memoria conectada a este bus se denomina **memoria física**.
- El espacio de direcciones que pueden volcarse sobre este bus se denomina **direcciones físicas**.

## Capacidad de direccionamiento de memoria

Los procesadores IA-32 son la continuación del 8086. Este procesador fue el primer de 16 bits de ancho de palabra, y por ende todos sus registros internos tienen ese tamaño. Su espacio de Direcccionamiento es de 1 Mbyte ∵ Address Bus es de 20 líneas. Este espacio de direccionamiento se administra por segmentación.

# Segmentación vs. Paginación



# Espacio Lógico

## Segmentación

Por diversos motivos que en su momento tuvieron sentido, Intel definió organizar el espacio de direccionamiento de la Familia iAPx86 en segmentos. El compromiso de compatibilidad ató a los siguientes procesadores a mantener este esquema

# Espacio Lógico

## Condiciones iniciales de segmentación

- 4 registros de segmento para almacenar hasta 4 selectores de segmento.
- Registros de 16 bits los segmentos tienen a lo sumo 64K de tamaño
- Expresión de las direcciones en el modelo de programación mediante dos valores:
  - ① Identificador del segmento en el que se encuentra la variable o la instrucción que se desea direccionar,
  - ② Desplazamiento, offset, o **dirección efectiva** a partir del inicio de ese segmento en donde se encuentra efectivamente

# Operando en Memoria... CISC

Para identificar un operando (fuente o destino) de una instrucción en memoria, se utiliza lo que Intel denomina dirección lógica.

Este nombre obedece a que es una dirección abstracta expresada en términos de su arquitectura pero que necesita ser procesada para convertirse finalmente en la dirección física que es la que saldrá hacia el bus del sistema por las líneas de address.

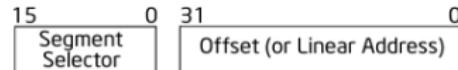


Figura: IA-32: Dirección lógica

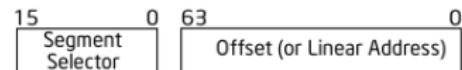


Figura: Intel® 64 : Dirección Lógica

# Segmentos para cada dirección Lógica

- El valor del segmento en una dirección lógica, puede especificarse de manera implícita.
- Por lo general este es el modo en que se hace, aunque es posible explicitar con que segmento se desea direccionar un operando de memoria determinado.
- Si no se especifica explícitamente el segmento como parte de la dirección lógica el procesador lo establecerá automáticamente de acuerdo a la tabla

<b>Referencia a:</b>	<b>Reg.</b>	<b>Segmento</b>	<b>Regla de selección por defecto</b>
Instrucciones	CS	Segmento de Código	Cada opcode fetch
Pila	SS	Segmento de Pila	Todos los push y pop, cualquier referencia a memoria que utilice como registro base ESP o EBP.
Datos Locales	DS	Segmento de datos	Cualquier referencia a un dato, excepto en el stack o un destino de instrucción de string
Strings Destino	ES	Segmento de datos extra direccionado por ES	Destino de Instrucciones de manejo de strings

Reglas de selección de segmento predefinidos

# Desplazamiento para ubicar operandos en Memoria

La riqueza de modos de direccionamiento de operandos en memoria la da el desplazamiento. Aquí es Intel puso todo el esfuerzo dando una gran cantidad de alternativas para calcular el desplazamiento.

Básicamente un desplazamiento tiene al menos uno de los siguientes componentes, o cualquiera de las combinaciones posibles:

- Desplazamiento directo: Se trata de un valor de 8, 16, o 32 bits, explícitamente incluido en la instrucción.
- Base: Se trata de un valor contenido en un registro de propósito general, que indica una dirección a partir de la cual se calcula el desplazamiento. Es un valor de 32 bits en el modo IA-32 y de 64 bits en IA-32e.
- Índice: Se trata de un valor contenido en un registro de propósito general, que se representa la dirección a la cual nos queremos referir. Típicamente es un valor que al incrementarse permite recorrer por ejemplo un buffer de memoria. Es un valor de 32 bits en el modo IA-32 y de 64 bits en IA-32e.
- Escala: Es un valor por el cual se multiplica el valor del Índice: Puede valer 2, 4, u 8.

# Calculando el desplazamiento

A partir de estos cuatro componentes o combinación de algunos de ellos, se obtiene lo que Intel denomina Dirección Efectiva. Esta denominación intenta representar el significado del offset dentro de un segmento: es la dirección que ocupa efectivamente el elemento direccionado respecto del inicio del segmento.

Los cuatro componentes anteriores pueden ser positivos o negativos en representación Ca2 (excepto el valor del factor de escala que es siempre positivo). A continuación se presenta los diferentes registros y valores que pueden integrar cada uno de los cuatro componentes descriptos:

Base	Índice	Escala	Desplazamiento
$\begin{bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ EDI \\ ESI \end{bmatrix}$	$+ \begin{bmatrix} (EAX) \\ (EBX) \\ (ECX) \\ (EDX) \\ (ESP) \\ (EBP) \\ (EDI) \\ (ESI) \end{bmatrix}$	$* \begin{bmatrix} (1) \\ (2) \\ (4) \\ (8) \end{bmatrix}$	$+ \begin{bmatrix} Nada \\ 8bits \\ 16bits \\ 32bits \end{bmatrix}$

En general la expresión general que representa el cálculo interno del procesador es:

$$\text{DireccionEfectiva} = \text{Base} + (\text{Indice} * \text{escala}) + \text{Desplazamiento}$$

# Casos particulares

Cuestiones a destacar, respecto de la matriz anterior:

- El registro ESP (o RSP), no puede utilizarse de Índice. Esto es bastante lógico ya que su uso privilegiado es como puntero de pila. Por lo tanto modificarlo para recorrer otro array de datos que no sea la pila es poco menos que imprudente.
- Cuando se emplean como registros base ESP y EBP (o RSP y RBP), se utilizan asociados al registro de segmento SS. Para el resto de los usos se asocian al DS.
- El factor de escala solo se puede emplear cuando se utiliza un Registro Índice. En otro caso no se emplea.

1

## Introducción

- Genealogía
- Arquitectura Básica

2

## Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32
- Arquitectura Intel® 64

3

## Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro
- Modos de Direccionamiento a memoria

## Modo Desplazamiento

- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

4

## Tipos de Datos

- Almacenamiento en memoria
- Alineación en memoria

5

## Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

# ¿El Offset puede venir directo en la Instrucción?

Se incluye en la instrucción un valor en forma explícita que representa en forma directa el valor del offset.

En el listado siguiente se muestran diversos ejemplos de instrucciones de este Modo.

```
1 or    ecx, dword [0x300040A0] ; Calcula la or lógica entre  
2                                ; ECX y la doble word contenida  
3                                ; a partir de la dirección de  
4                                ; memoria 0x300040A0.  
5 inc   byte [0xAF007600]       ; Incrementa el byte contenido  
6                                ; por la dirección de memoria  
7                                ; 0xAF007600  
8 dec   dword [ i ]           ; El valor de la dirección de  
9                                ; variable i se calcula directa-  
10                               ; mente y el valor se reemplaza  
11                               ; en tiempo de compilación
```

1

## Introducción

- Genealogía
- Arquitectura Básica

2

## Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32
- Arquitectura Intel® 64

3

## Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro
- Modos de Direccionamiento a memoria

## Modo Desplazamiento

### Modo Base Directo

- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

4

## Tipos de Datos

- Almacenamiento en memoria
- Alineación en memoria

5

## Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

# Una de las formas de direccionar en forma indirecta

- En este modo el offset está contenido directamente en un registro Base.
- El procesador simplemente lo toma desde el registro, sin otro cálculo.

```
1 | mov    edx , i      ; edx = desplazamiento de i en el segmento
2 |                   ; de datos
3 | inc    [ edx ]     ; incrementamos i direccionada a través de
4 |                   ; un registro puntero base.
```

El ejemplo anterior es trivial ya que la variable como vimos puede incrementarse de manera directa

1

## Introducción

- Genealogía
- Arquitectura Básica

2

## Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32
- Arquitectura Intel® 64

3

## Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro
- Modos de Direccionamiento a memoria

- Modo Desplazamiento

- Modo Base Directo

## Base + Desplazamiento

- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

4

## Tipos de Datos

- Almacenamiento en memoria
- Alineación en memoria

5

## Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

# Formas mas útiles de direccionar en forma indirecta

- Combina el valor contenido en un registro que apunta a la base de un bloque de datos con un valor explícito puesto en la instrucción, que permite calcular la dirección efectiva del operando.
- También resulta útil para acceder a una estructura mas compleja de datos, apuntando a la base de la estructura con un registro y utilizando el Desplazamiento para acceder al campo deseado de la estructura.

1

## Introducción

- Genealogía
- Arquitectura Básica

2

## Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32
- Arquitectura Intel® 64

3

## Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro
- Modos de Direccionamiento a memoria

- Modo Desplazamiento

- Modo Base Directo

- Base + Desplazamiento

## Base + Desplazamiento

- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

4

## Tipos de Datos

- Almacenamiento en memoria
- Alineación en memoria

5

## Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

# Formas mas útiles de direccionar en forma indirecta

```

1 ORG 8000h
2 use16           ;Estamos en Modo Real=>Código de 16 bits
3 start: jmp main ;Salto al inicio del programa.
4 ALIGN 8
5 gdt:    resb 8      ;NULL Descriptor. Dejamos 8 bytes sin usar.
6 Data_sel equ $-gdt   ;Calcula dinámicamente la posición del selector
7 K_data:   dw 0xffff   ;límite 15.00
8          dw 0x0000   ;base 15.00
9          db 0x00     ;base 23.16
10         db 10010010b ;Presente Segmento Datos Read Write
11         db 0xCF      ;G = 1, D/B = 1, y límite 0Fh
12         db 0x00     ;base 31.24
13 gdt_size equ $-gdt   ;calcula dinámicamente el tamaño de la gdt
14 main:
15 ...
16 ...
17         mov ebx,K_data ;ebx apunta a la base del descriptor
18 ;//////////Lee con direccionamiento base + desplazamiento los atributos del descriptor
19 ;de segmento de datos definido en la tabla anterior
20         mov al, byte [ebx + 5]
21 ;//////////Testea bit de presente
22         test al,0x80    ;Testea bit de presente
23         jz NoPresente ;Si no está presente, salta
24

```

1

## Introducción

- Genealogía
- Arquitectura Básica

2

## Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32
- Arquitectura Intel® 64

3

## Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro
- Modos de Direccionamiento a memoria

- Modo Desplazamiento

- Modo Base Directo

- Base + Desplazamiento

- Base + Desplazamiento

- Índice \* escala + desplazamiento

- Base + Índice + Desplazamiento

- Modo Base + Índice \* Escala + Desplazamiento

4

## Tipos de Datos

- Almacenamiento en memoria
- Alineación en memoria

5

## Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

# Formas mas útiles de direccionar en forma indirecta

- Es una forma eficiente de acceder a elementos de un array cuyo tamaño es 2, 4, u 8 bytes
- El Desplazamiento puede ubicar el inicio del array y el valor índice se guarda en un registro que al incrementar pasa al siguiente elemento permitiendo con la escala ajustar al tamaño del mismo

```
1 %define Dir_Tabla      0x2000F000
2 %define mascara        0xFFFFFFF
3     mov    ecx , size_tabla ;ecx = cantidad de elementos de
4                           ;4 bytes de la tabla.
5     xor    esi , esi       ;esi apunta al inicio de la tabla
6 mas:
7     and    [esi*4 + Dir_Tabla], mascara
8                           ;borra bit menos significativo
9                           ;del elemento de la tabla
10    inc    esi            ;esi apunta al siguiente
11                           ;elemento de 4 bytes
12    loop   mas           ;va por el siguiente elemento
13                           ;hasta que exc sea 0
```

**1** Introducción

- Genealogía
- Arquitectura Básica

**2** Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32
- Arquitectura Intel® 64

**3** Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro
- Modos de Direccionamiento a memoria

**1** Modo Desplazamiento

- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento**
- Modo Base + Índice \* Escala + Desplazamiento

**4** Tipos de Datos

- Almacenamiento en memoria
- Alineación en memoria

**5** Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

# Formas mas útiles de direccionar en forma indirecta

- Este modo es especial para acceder a matrices bidimensionales.
- Un ejemplo obligado es el buffer de video
- Cuando trabaja en modo texto el buffer de video es una matriz de 25 filas por 80 columnas.
- Cada elemento consta de dos bytes: el primero contiene el ASCII del carácter a presentar y el segundo los atributos (intensificado, video inverso, y colores de carácter y fondo).
- Vamos a escribir un código que aprovechando este modo de direccionamiento sirva para limpiar el contenido de la pantalla.
- El registro base apunta a cada línea de la pantalla y el índice apunta a cada elemento de la línea.

# Formas mas útiles de direccionar en forma indirecta

```
1 ; El siguiente código limpia la pantalla dejándola en modo
2 ; blanco sobre negro. El buffer de video comienza en la
3 ; dirección física 0x000B8000. Utilizamos este valor como
4 ; desplazamiento en el cálculo de la dirección efectiva.
5 xor    ebx , ebx      ; resultado ebx = 0.
6                      ; ebx apunta a la 1er. fila de 80 caracteres
7 col:
8 xor    edi , edi      ; resultado edi = 0.
9                      ; edi apunta al primer elemento de la fila
10 mov   ecx , size_row ; ecx = cantidad de filas para loop
11 row:
12 ; carácter nulo no imprime nada en pantalla
13 mov   byte [ebx + edi + 0x000B8000],0x00
14
15 add   edi , 2          ; edi apunta al porx. elemento de 2 bytes
16 loop  row              ; si CX = 0 se completó fila
17 add   ebx , 160         ; Apunta a la siguiente fila
18 cmp   ebx , 0x1000       ; fin del buffer?
19     col
```

**1** Introducción

- Genealogía
- Arquitectura Básica

**2** Modelo del Programador de aplicaciones

- Arquitectura de 16 bits básica
- IA-32
- Arquitectura Intel® 64

**3** Modos de Direccionamiento

- Modo Implícito
- Modo Inmediato
- Modo Registro
- Modos de Direccionamiento a memoria

**● Modo Desplazamiento****● Modo Base Directo****● Base + Desplazamiento****● Base + Desplazamiento****● Índice \* escala + desplazamiento****● Base + Índice + Desplazamiento****● Modo Base + Índice \* Escala + Desplazamiento****4** Tipos de Datos

- Almacenamiento en memoria
- Alineación en memoria

**5** Ejemplos de uso de pila

- ¿Como funciona un llamado Near?
- ¿Como funciona un llamado Far?
- Interrupciones

# Formas mas útiles de direccionar en forma indirecta

```
1 ; El siguiente código limpia la pantalla dejándola en modo
2 ; blanco sobre negro. El buffer de video comienza en la
3 ; dirección física 0x000B8000. Utilizamos este valor como
4 ; desplazamiento en el cálculo de la dirección efectiva.
5 xor    ebx , ebx      ; resultado ebx = 0.
6                      ; ebx apunta a la 1er. fila de 80 caracteres
7 col:
8 xor    edi , edi      ; resultado edi = 0.
9                      ; edi apunta al primer elemento de la fila
10 mov   ecx , size_row ; ecx = cantidad de filas para loop
11 row:
12 ; carácter nulo no imprime nada en pantalla
13 mov   byte [ebx + edi*2 + 0x000B8000],0x00
14
15 inc    edi      ; edi apunta al porx. elemento de 2 bytes
16 loop  row       ; si CX = 0 se completó fila
17 add   ebx ,160   ; Apunta a la siguiente fila
18 cmp   ebx ,0x1000 ; fin del buffer?
19     col
```

# Tipos básicos de datos

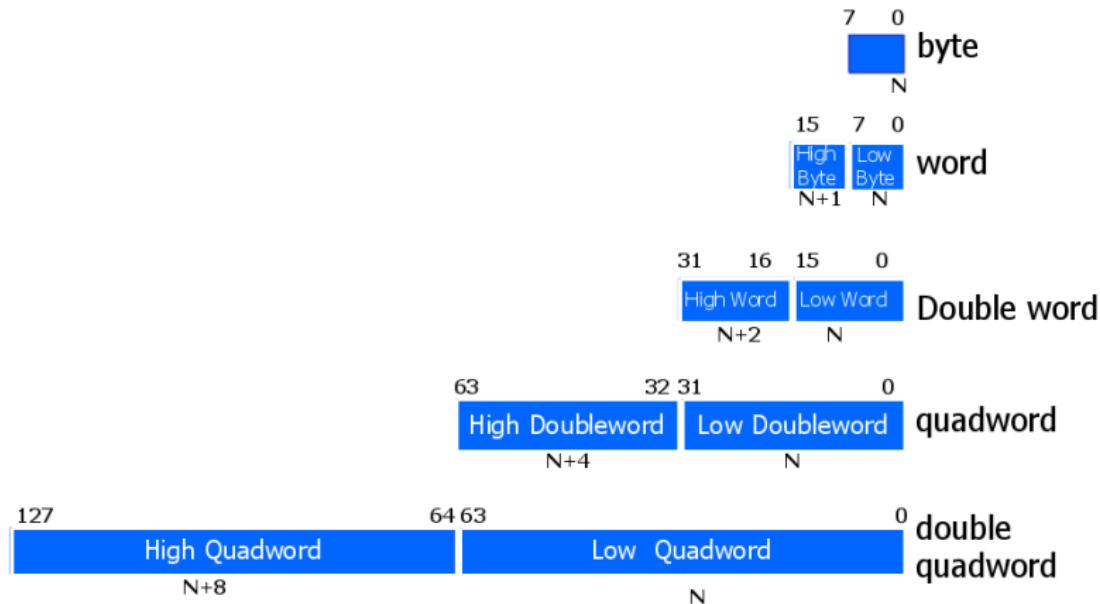


Figura: IA-32 e Intel® 64 : Tipos de datos fundamentales

- 1 Introducción
  - Genealogía
  - Arquitectura Básica
- 2 Modelo del Programador de aplicaciones
  - Arquitectura de 16 bits básica
  - IA-32
  - Arquitectura Intel® 64
- 3 Modos de Direccionamiento
  - Modo Implícito
  - Modo Inmediato
  - Modo Registro
  - Modos de Direccionamiento a memoria

- Modo Desplazamiento
- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

- 4 Tipos de Datos
  - Almacenamiento en memoria
  - Alineación en memoria
- 5 Ejemplos de uso de pila
  - ¿Como funciona un llamado Near?
  - ¿Como funciona un llamado Far?
  - Interrupciones

# Little endian

- Desde el procesador 8086, esta familia maneja el almacenamiento en memoria de las variables en el formato little endian.
- Dicho de otra forma: una variable de varios bytes de tamaño almacena su byte menos significativo en la dirección con que se referencia la variable y a partir de allí coloca el resto de los bytes en orden de significancia, terminando con el almacenamiento del byte mas significativo, en la dirección de memoria mas alta (es decir termina con el menor, de allí little endian).
- Esta situación se representa en próximo slide. A simple vista pareciera que están almacenados al revés, ya que si lo miramos en la memoria está de atrás hacia adelante.
- Otros procesadores utilizan el formato BigEndian, es decir colocando la información en el orden en el que normalmente esperamos encontrarla.

# Little endian

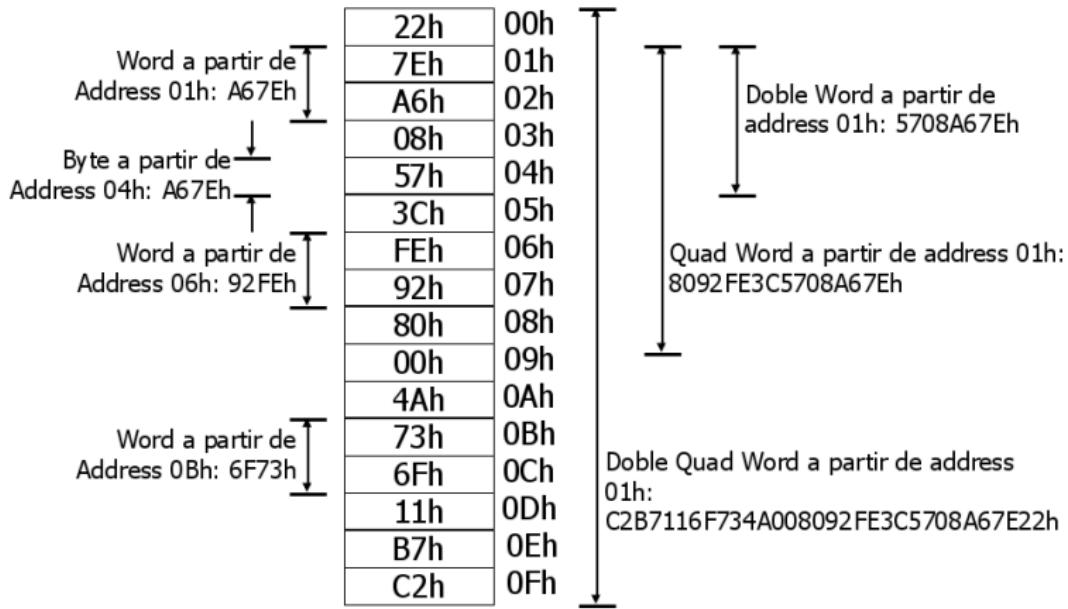
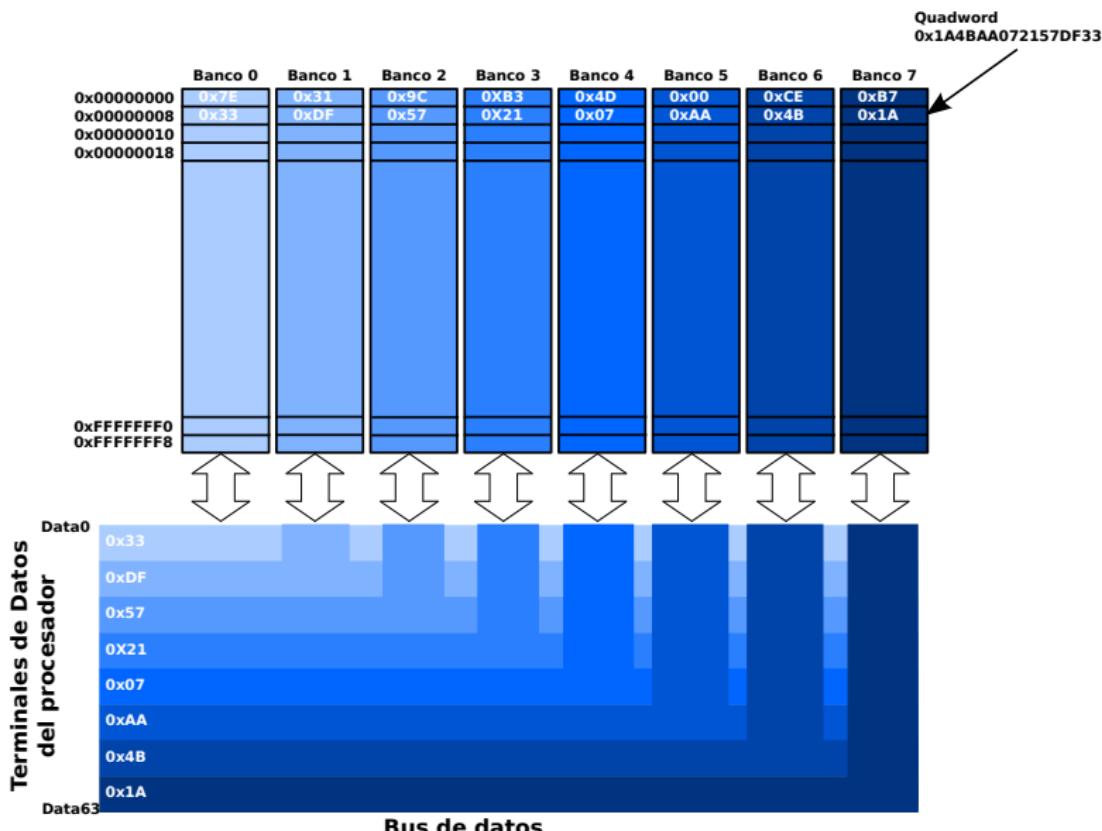


Figura: IA-32 e Intel® 64 : Alineamiento en memoria para los diferentes tipos de datos

# Little endian

- La razón por la que Intel adoptó Little Endian, obedece a que el procesador 8086 (y sus sucesores por cuestión de compatibilidad), administra la memoria de a bytes.
- Esto significa que cada dirección de memoria tiene una capacidad de almacenamiento de 8 bits.
- Por lo tanto, y debido a esta decisión de diseño, es que un bus de datos de 16 bits primero, 32 mas tarde, y 64 actualmente, se conecta a bancos de memoria RAM Dinámica organizados en bytes.
- Por lo tanto, cuando el procesador lee un dato de 64 bits a través del bus de datos, cada byte de las direcciones de memoria que se leen viaja por un byte del bus de datos, de acuerdo al ordenamiento que la información tiene en la memoria, de la manera en que se muestra en el próximo slide.

# Little endian



- 1 Introducción
  - Genealogía
  - Arquitectura Básica
- 2 Modelo del Programador de aplicaciones
  - Arquitectura de 16 bits básica
  - IA-32
  - Arquitectura Intel® 64
- 3 Modos de Direccionamiento
  - Modo Implícito
  - Modo Inmediato
  - Modo Registro
  - Modos de Direccionamiento a memoria

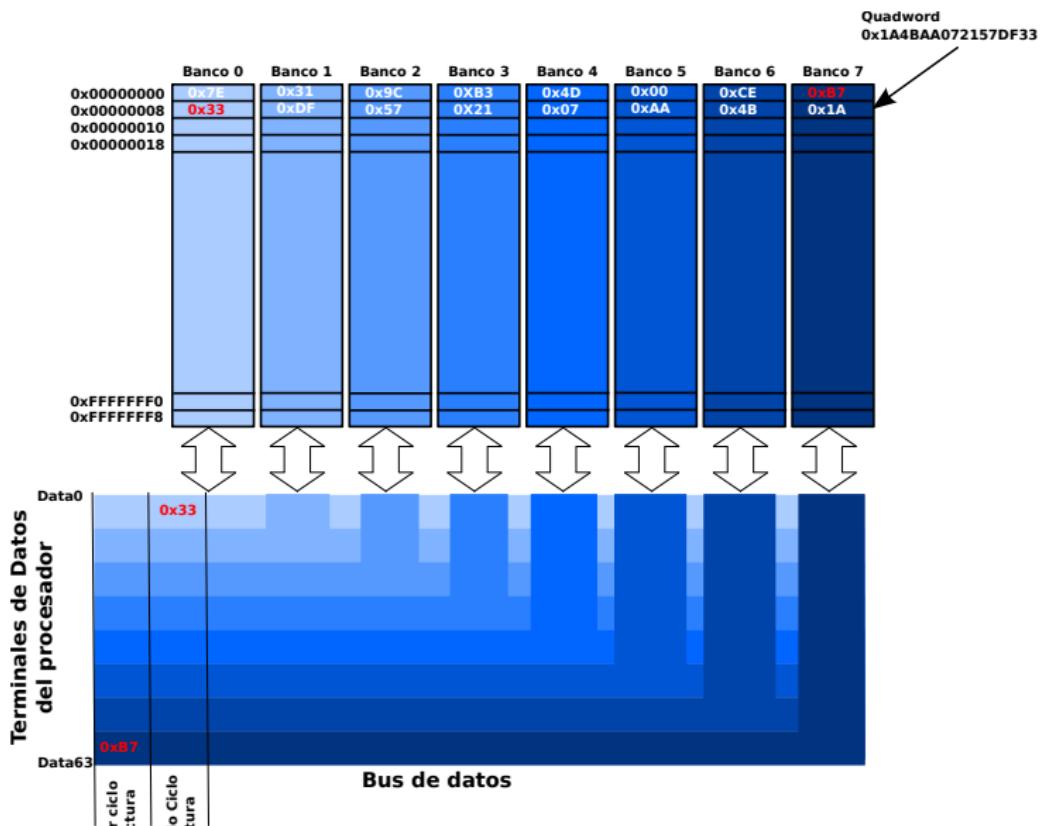
- Modo Desplazamiento
- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

- 4 Tipos de Datos
  - Almacenamiento en memoria
  - Alineación en memoria
- 5 Ejemplos de uso de pila
  - ¿Como funciona un llamado Near?
  - ¿Como funciona un llamado Far?
  - Interrupciones

# ¿Porque conviene alinear los datos?

- Los procesadores IA-32 e Intel® 64 no ponen restricciones respecto de la alineación en memoria para las diferentes variables de los programas (consecuencia favorable de la administración de memoria de a bytes).
- Esto otorga gran flexibilidad a la hora de aprovechar al máximo la memoria.
- Pero si una variable queda repartida en dos filas diferentes, se requerirán dos ciclos de lectura para accederla.
- Esta situación se representa en el siguiente slide.
- La variable se trae al procesador con dos lecturas de memoria
- Estas dos lecturas de memoria son transparentes a nivel de software (la aplicación no debe ser modificada en absoluto), ya que el procesador autónomamente realiza las dos lecturas.
- Entonces ¿cuál es el problema?. La respuesta es: **performance**. Dos ciclos de lectura en lugar de uno solo tornan mas lento el acceso a la variable. Esto puede evitarse usando las directivas de alineación que todos los lenguajes poseen.

# Acceso a datos no alineados



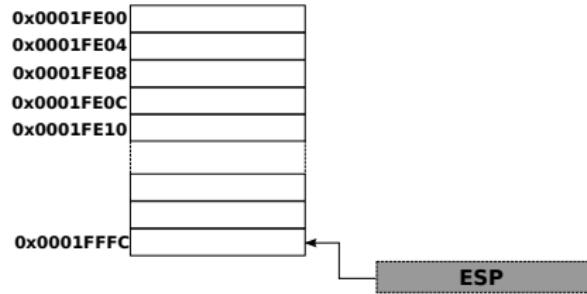
- 1 Introducción
  - Genealogía
  - Arquitectura Básica
- 2 Modelo del Programador de aplicaciones
  - Arquitectura de 16 bits básica
  - IA-32
  - Arquitectura Intel® 64
- 3 Modos de Direccionamiento
  - Modo Implícito
  - Modo Inmediato
  - Modo Registro
  - Modos de Direccionamiento a memoria

- Modo Desplazamiento
- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

- 4 Tipos de Datos
  - Almacenamiento en memoria
  - Alineación en memoria
- 5 Ejemplos de uso de pila
  - ¿Como funciona un llamado Near?
  - ¿Como funciona un llamado Far?
  - Interrupciones

# Como en un debugger :)

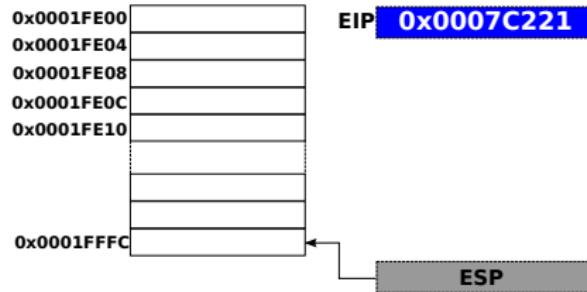
- Ejecutamos la primer instrucción
- Lee el port de E/S
- Y luego.....



```
1 %define mask      0xffff0
2 main:
3 ...
4 ...
5 in ax,0x300 ;lee port
6 call setmask ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 setmask:
10 and ax,mask    ;aplica la mascara
11 ret            ;retorna
```

# Estamos a punto de ejecutar CALL

- ...ejecutamos la instrucción Call.
- La misma está almacenada a partir de la dirección de memoria contenida por **EIP**.
- El ESP apunta a la base de la pila.



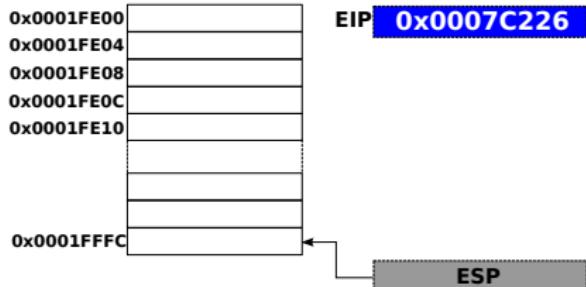
```

1 %define mask      0xffff0
2 main:
3 ...
4 ...
5 in    ax,0x300      ; lee port
6 call setmask ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 setmask:
10 and   ax,mask     ; aplica la mascara
11 ret               ; retorna

```

# CALL por dentro...

- En primer lugar el procesador apunta con EIP a la siguiente instrucción.
- Un CALL near se compone de 1 byte de código de operación y cuatro bytes para la dirección efectiva (offset), ya que estamos en 32 bits.
- Por eso el EIP apunta 5 bytes mas adelante, ya que allí comienza la siguiente instrucción del CALL



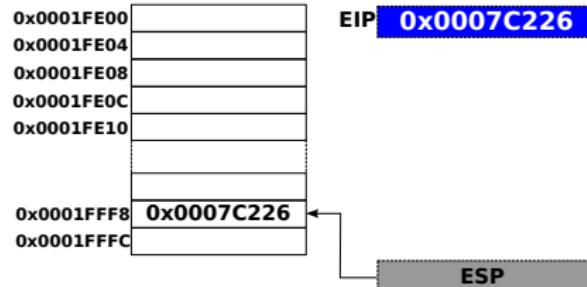
```

1 %define mask      0xffff0
2 main:
3 ...
4 ...
5 in  ax,0x300    ;lee port
6 call setmask   ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 setmask:
10 and ax,mask    ;aplica la mascara
11 ret            ;retorna

```

# CALL por dentro...

- El procesador decrementa ESP y guarda el valor de EIP.
- Así resguarda su dirección de retorno a la instrucción siguiente a CALL.
- Para saber a donde debe saltar saca de la instrucción CALL la dirección efectiva de la subrutina *setmask*.
- En nuestro caso 0x0007C44F.



```

1 %define mask      0xffff0
2 main:
3 ...
4 ...
5 in  ax,0x300    ;lee port
6 call setmask   ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 setmask:
10 and ax,mask    ;aplica la mascara
11 ret            ;retorna

```

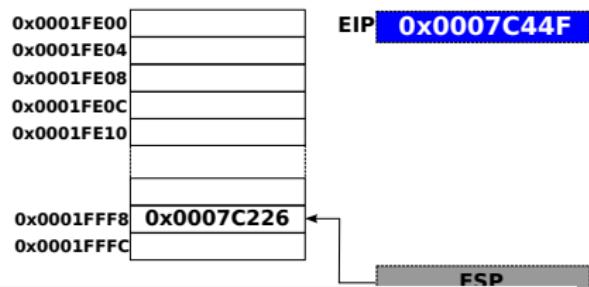
# Resultado del CALL

- Como resultado el valor de EIP, es reemplazado por la dirección efectiva de la subrutina *setmask*.
- Y sin mas.... el procesador está buscando la primer instrucción de la subrutina *setmask*, en este caso, la operación *and*.

```

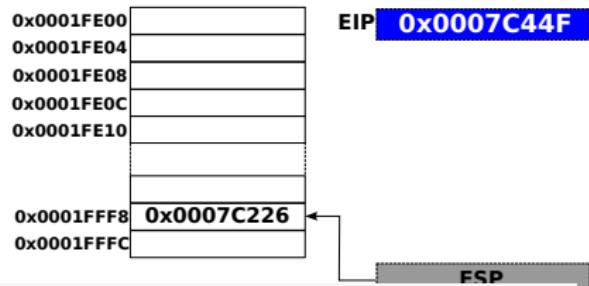
1 %define mask      0xffff0
2 main:
3 ...
4 ...
5 in    ax,0x300    ;lee port
6 call  setmask   ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 setmask:
10 and ax,mask ;aplica la mascara
11 ret           ;retorna

```



# Volver.....

- Esta subrutina es trivial a los efectos del ejemplo.
- Para volver (sin la frente marchita)...
- Es necesario **retornar**



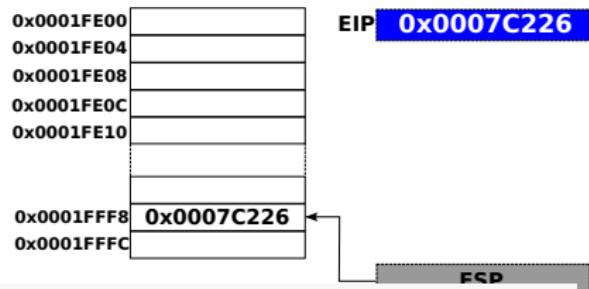
```

1 %define mask      0xffff0
2 main:
3 ...
4 ...
5 in    ax,0x300    ;lee port
6 call  setmask   ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 setmask:
10 and  ax,mask    ;aplica la mascara
11 ret ;retorna

```

# Volviendo.....

- La ejecución de **ret** consiste en recuperar de la pila la dirección de retorno.
- Esa dirección se debe cargar en EIP
- Una vez hecho.....



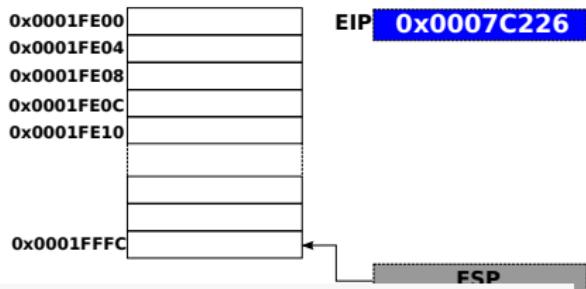
```

1 %define mask      0xffff0
2 main:
3 ...
4 ...
5 in  ax,0x300    ;lee port
6 call setmask   ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 setmask:
10 and ax,mask    ;aplica la mascara
11 ret            ;retorna

```

# Volvimos!

- Finalizada la ejecución de **ret** estamos otra vez en el código llamador.
- Pero en la instrucción siguiente a CALL



```
1 % define mask      0xffff0
2 main:
3 ...
4 ...
5 in  ax,0x300      ;lee port
6 call setmask      ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 setmask:
10 and ax,mask      ;aplica la mascara
11 ret               ;retorna
```

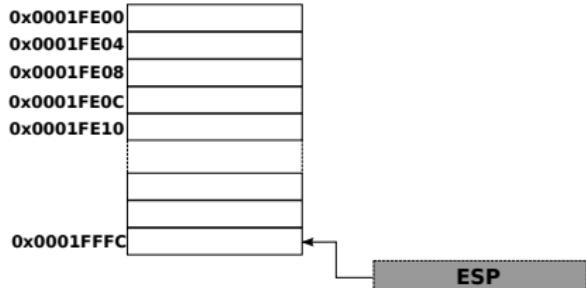
- 1 Introducción
  - Genealogía
  - Arquitectura Básica
- 2 Modelo del Programador de aplicaciones
  - Arquitectura de 16 bits básica
  - IA-32
  - Arquitectura Intel® 64
- 3 Modos de Direccionamiento
  - Modo Implícito
  - Modo Inmediato
  - Modo Registro
  - Modos de Direccionamiento a memoria

- Modo Desplazamiento
- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

- 4 Tipos de Datos
  - Almacenamiento en memoria
  - Alineación en memoria
- 5 Ejemplos de uso de pila
  - ¿Como funciona un llamado Near?
  - **¿Como funciona un llamado Far?**
  - Interrupciones

# Ahora el destino está en un segmento diferente

- Ejecutamos la primer instrucción
- Lee el port de E/S
- Y luego.....



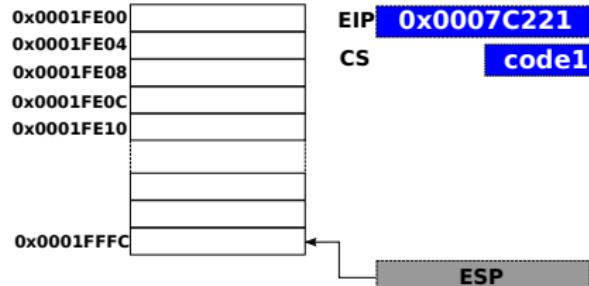
```

1 %define mask      0xffff0
2 section code1
3 main:
4 ...
5 in ax,0x300 ;lee port
6 call      code2:setmask      ;llama a aplicar m scara
7 ...
8 ...
9 section code2
10 setmask:
11 and ax,mask          ;aplica la m scara
12 retf                ;retorna

```

# Por lo tanto importa el valor de CS...

- Nuevamente nos paramos en el CALL
- Pero ahora necesitamos memorizar EIP, y también CS
- Ya que al estar el destino en otro segmento CS se modificará



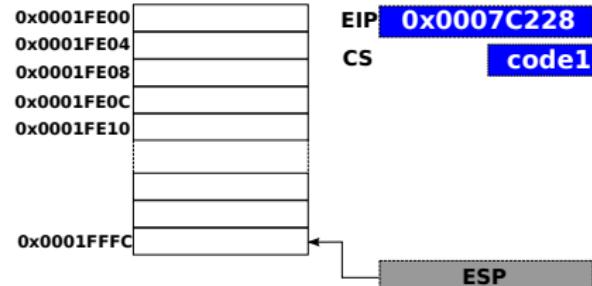
```

1 %define mask      0xffff0
2 section code1
3 main:
4 ...
5 in    ax,0x300      ;lee port
6 call code2:setmask ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 section code2
10 setmask:
11 and   ax,mask      ;aplica la m scara
12 retf               ;retorna

```

# Otra vez adentro del CALL... pero FAR

- Ahora la instrucción mide 7 bytes ya que se agrega el segmento
- Por lo tanto el EIP se incrementa 7 lugares
- Y se memoriza en la pila la dirección FAR.



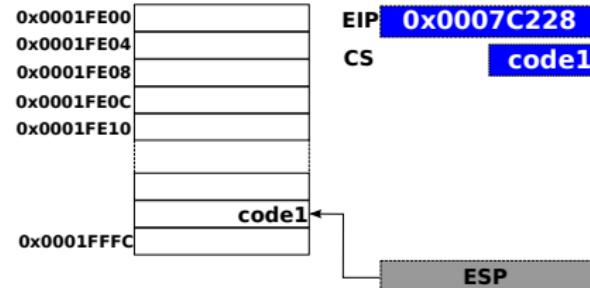
```

1 %define mask      0xffff0
2 section code1
3 main:
4 ...
5     in    ax,0x300    ;lee port
6     call  code2:setmask ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 section code2
10 setmask:
11     and   ax,mask      ;aplica la mascara
12     retf               ;retorna

```

# Otra vez adentro del CALL... pero FAR

- En primer lugar guarda en la pila, el valor del segmento al cual debe retornar.
- Siempre antes de almacenar nada en la pila, debe antes decrementar el valor del ESP.



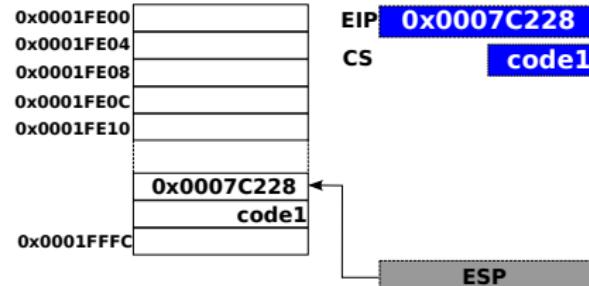
```

1 %define mask      0xffff0
2 section code1
3 main:
4 ...
5     in    ax,0x300    ;lee port
6     call  code2:setmask ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 section code2
10 setmask:
11     and   ax,mask           ;aplica la mascara
12     retf                    ;retorna

```

# Otra vez adentro del CALL... pero FAR

- Luego del valor del segmento guarda en la pila, el valor de EIP al cual debe retornar, y que lo llevará a buscar la siguiente instrucción al CALL.
- Decrementará nuevamente el valor del ESP, antes de almacenar.



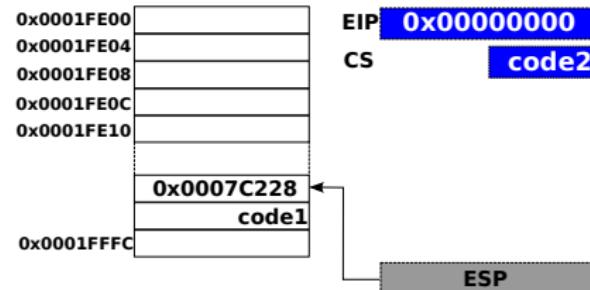
```

1 %define mask      0xffff0
2 section code1
3 main:
4 ...
5     in    ax,0x300    ;lee port
6     call  code2:setmask ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 section code2
10 setmask:
11     and   ax,mask           ;aplica la mascara
12     retf                    ;retorna

```

# Otra vez adentro del CALL... pero FAR

- La dirección de la rutina `setmask`, ahora es `code2:offset`.
- Como comienza justo al inicio del segmento, su offset es `0x00000000`.



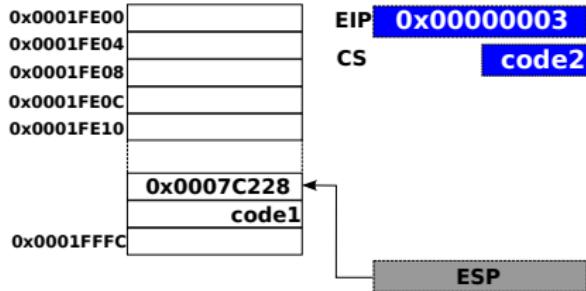
```

1 %define mask      0xffff0
2 section code1
3 main:
4 ...
5     in    ax,0x300    ;lee port
6     call  code2:setmask ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 section code2
10 setmask:
11     and ax,mask ;aplica la mascara
12     retf           ; retorna

```

# Retornando de un Call Far

- Para volver de un call far hay que sacar de la pila no solo el offset sino también el segmento.
- Entonces no sirve la misma instrucción que se usa para volver de una rutina Near.



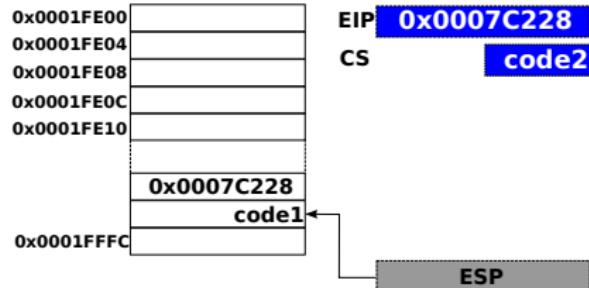
```

1 %define mask      0xffff0
2 section code1
3 main:
4 ...
5     in    ax,0x300    ;lee port
6     call  code2:setmask ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 section code2
10 setmask:
11     and   ax,mask      ;aplica la mascara
12     retf ;retorna

```

# Retornando de un Call Far

- Recupera la dirección efectiva
- Luego decrementa el Stack Pointer



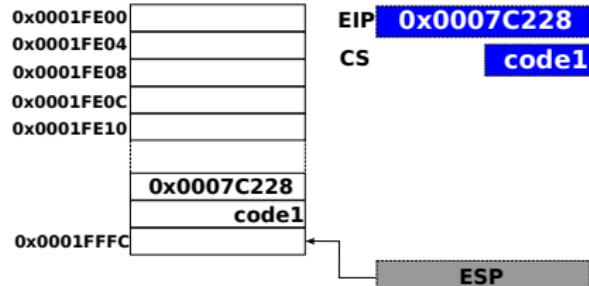
```

1 %define mask      0xffff0
2 section code1
3 main:
4 ...
5     in    ax,0x300    ;lee port
6     call  code2:setmask ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 section code2
10 setmask:
11     and   ax,mask           ;aplica la mascara
12     retf                    ;retorna

```

# Retornando de un Call Far

- Recupera el valor del segmento
- Luego decrementa el Stack Pointer
- ...y volvió...



```

1 %define mask      0xffff0
2 section code1
3 main:
4 ...
5     in    ax,0x300    ;lee port
6     call  code2:setmask ;llama a subrutina para aplicar una mascara
7 ...
8 ...
9 section code2
10 setmask:
11     and   ax,mask           ;aplica la mascara
12     retf                  ;retorna

```

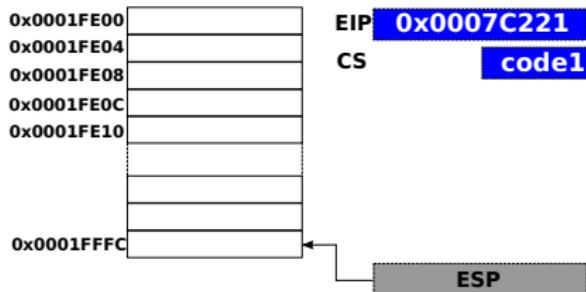
- 1 Introducción
  - Genealogía
  - Arquitectura Básica
- 2 Modelo del Programador de aplicaciones
  - Arquitectura de 16 bits básica
  - IA-32
  - Arquitectura Intel® 64
- 3 Modos de Direccionamiento
  - Modo Implícito
  - Modo Inmediato
  - Modo Registro
  - Modos de Direccionamiento a memoria

- Modo Desplazamiento
- Modo Base Directo
- Base + Desplazamiento
- Base + Desplazamiento
- Índice \* escala + desplazamiento
- Base + Índice + Desplazamiento
- Modo Base + Índice \* Escala + Desplazamiento

- 4 Tipos de Datos
  - Almacenamiento en memoria
  - Alineación en memoria
- 5 Ejemplos de uso de pila
  - ¿Como funciona un llamado Near?
  - ¿Como funciona un llamado Far?
  - Interrupciones

# ¿Qué pasa cuando se produce una Interrupción?

- Ejecutamos una instrucción cualquiera
- y en el medio de esa instrucción se produce una interrupción
- ...



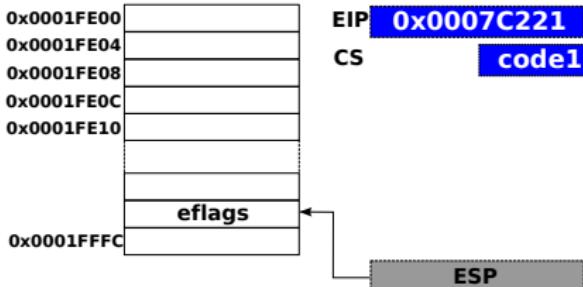
```

1 section code
2 main:
3 ...
4 next:
5 test [var],1 ;chequea bit 0 de variable
6 jnz next
7 ...
8 section kernel
9 handler_int:
10    in al, port      ;lee port de E/S
11    iret             ;retorna

```

# ¿Que pasa cuando se produce una Interrupción?

- Es necesario guardar además de la dirección de retorno, el estado del procesador.
- De otro modo si al final de la interrupción alguna instrucción modifica un flag, el estado de la máquina se altera y le vuelve al programa modificado.
- Esto puede tener resultados impredecibles si al retorno hay que usar el flag que cambió.



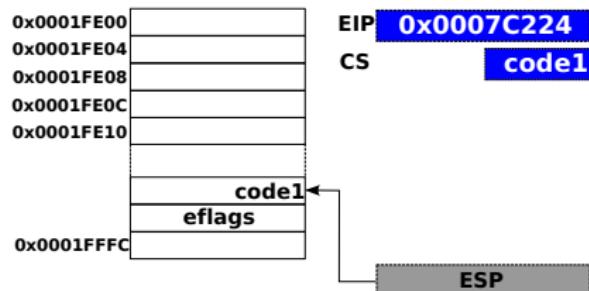
```

1 section code
2 main:
3 ...
4 next:
5 test      [var],1      ; chequea bit 0 de variable
6 jnz next
7 ...
8 section kernel
9 handler_int:
10 in al, port      ; lee port de E/S
11 iret            ; retorna

```

# ¿Que pasa cuando se produce una Interrupción?

- La dirección de retorno es far.
- Especialmente en sistemas multitasking donde cada proceso tiene una pila de kernel diferente.
- Así que luego de los flags se guarda el segmento de código.



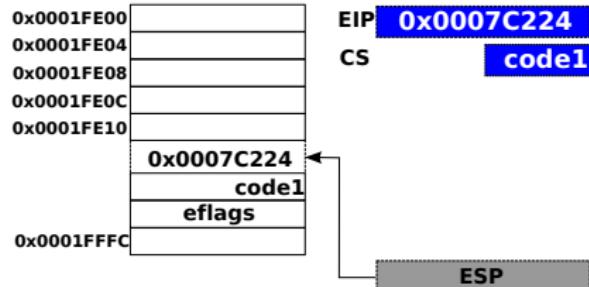
```

1 section code
2 main:
3 ...
4 next:
5 test      [var],1      ; chequea bit 0 de variable
6 jnz next
7 ...
8 section kernel
9 handler_int:
10 in al, port      ; lee port de E/S
11 iret            ; retorna

```

# ¿Que pasa cuando se produce una Interrupción?

- Se resguarda finalmente la dirección efectiva
- Notar que es la de la instrucción siguiente a la de la interrupción



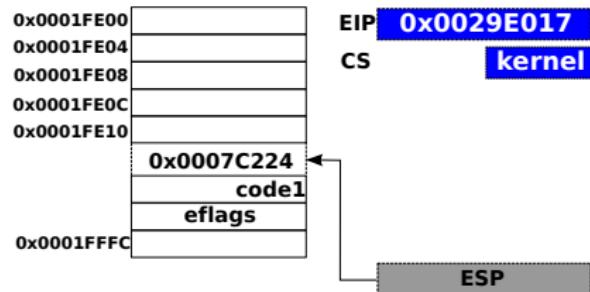
```

1 section code
2 main:
3 ...
4 next:
5 test      [var],1      ;chequea bit 0 de variable
6 jnz next
7 ...
8 section kernel
9 handler_int:
10 in al, port      ;lee port de E/S
11 iret            ;retorna

```

# ¿Que pasa cuando se produce una Interrupción?

- Los nuevos valores de segmento y desplazamiento que debe cargar en CS:EIP, los obtiene del vector de interrupciones en modo real, o de la Tabla de descriptores de interrupción en modo protegido, o en el modo 64 bits.



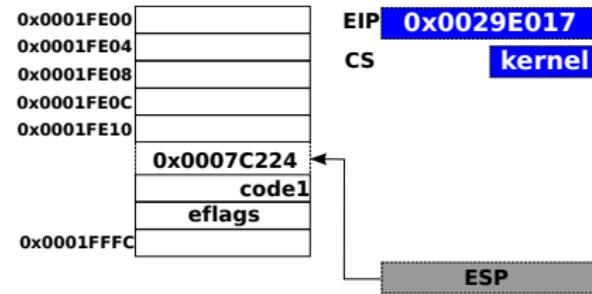
```

1 section code
2 main:
3 ...
4 next:
5 test      [var],1      ;chequea bit 0 de variable
6 jnz next
7 ...
8 section kernel
9 handler_int:
10 in al,port ;lee port de E/S
11 iret        ;retorna

```

# ¿Como se vuelve de una Interrupción?

- Recuperando además de la dirección de retorno, los flags
- Por lo tanto necesitamos otra instrucción particular de retorno...
- ... `iret`...

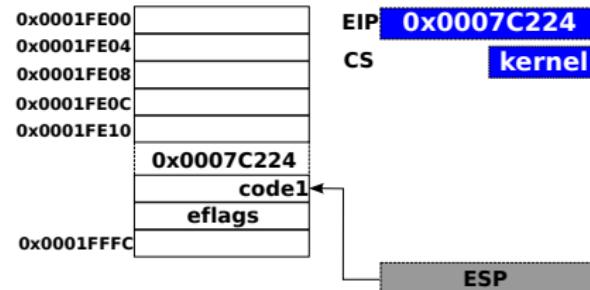


```

1 section code
2 main:
3 ...
4 next:
5 test      [var],1      ;chequea bit 0 de variable
6 jnz next
7 ...
8 section kernel
9 handler_int:
10 in al, port      ;lee port de E/S
11 iret ;retorna

```

# Volviendo...

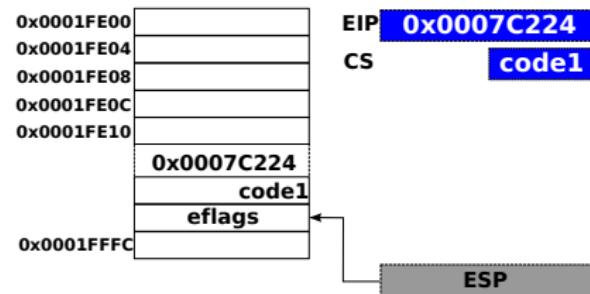


```

1 section code
2 main:
3 ...
4 next:
5 test      [var],1      ; chequea bit 0 de variable
6 jnz next
7 ...
8 section kernel
9 handler_int:
10 in al, port      ; lee port de E/S
11 iret            ; retorna

```

# Volviendo...

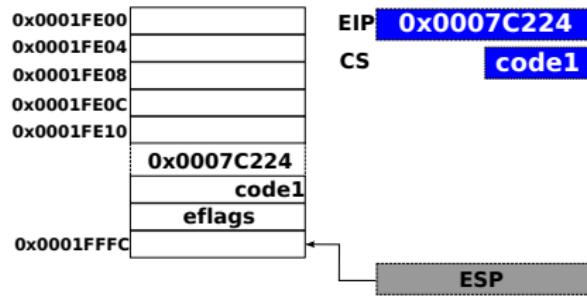


```

1 section code
2 main:
3 ...
4 next:
5 test      [var],1      ; chequea bit 0 de variable
6 jnz next
7 ...
8 section kernel
9 handler_int:
10 in al, port      ; lee port de E/S
11 iret             ; retorna

```

# Volviendo...



```

1 section code
2 main:
3 ...
4 next:
5 test      [var],1      ;chequea bit 0 de variable
6 jnz next
7 ...
8 section kernel
9 handler_int:
10    in al, port        ;lee port de E/S
11    iret                ;retorna

```