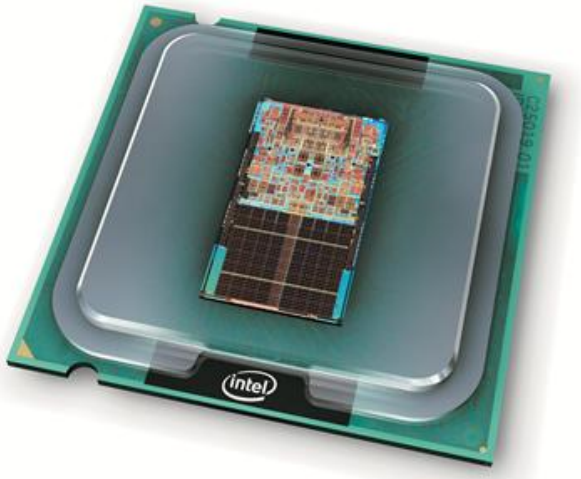


Procesadores IA-32 e Intel® 64
Gestión de Interrupciones

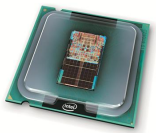
Alejandro Furfaro

Abril 2012

Agenda

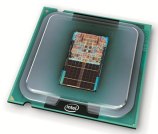
Fuentes de interrupción

- *Hardware*.
  Señales eléctricas enviadas desde los dispositivos de hardware, generalmente concentradas en un controlador externo que las prioriza y envía a la CPU secuencialmente de acuerdo a este criterio.
  Características: Asincrónicas. No determinísticas.
- *Software*.
  Se producen cuando el software ejecuta *INT n*.
  Característica: Determinísticas. La instrucción *INT* se intercala donde se desea generar la interrupción.
- *Internas*.
  Se conocen como excepciones y son generadas por la propia CPU a consecuencia de una situación que le impide completar la ejecución de la instrucción en curso. Por ejemplo, una división por cero, un page fault, una violación de protección, entre otras.



Caracterísiticas principales

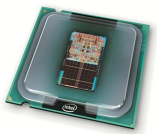
- El sistema de interrupciones fue diseñado para el primer procesador de la familia x86, el 8086, en 1978, y se mantiene idéntico hasta los procesadores actuales de microarquitectura Core.
- Una interrupción se identifica unívocamente mediante un valor numérico de 8 bits que se denomina *Tipo*.
- Por lo tanto, un procesador x86 puede manejar 256 tipos diferentes de interrupción. La cuestión que necesitamos entender es como se obtiene el valor del *Tipo*.



Identificando una fuente de interrupción

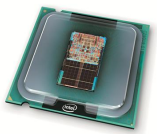
¿Como se obtiene el tipo?

- Es provisto por el hardware en interrupciones que ingresan por el pin *INTR* del procesador.
- Está asociado a un pin del procesador, tal el caso de la Interrupción No enmascarable *NMI* que lleva asociado el tipo 2.
- Acompaña al código de operación en la instrucción *INT type*, para el caso de las interrupciones por software, por ejemplo INT 80h.
- Está asociado a una instrucción (*INTO*,por ejemplo, genera una interrupción de tipo 4).
- Las excepciones tienen asignado un tipo específico cada una.



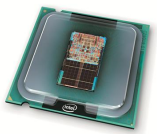
Tipos predefinidos

Tipo	Mnemónico	Descripción	Clase	Código de Error	Fuente
0	#DE	Error de División	Fault	NO	Instrucciones DIV e IDIV
1	#DB	Reservada	Fault/Trap	NO	Solo para uso de Intel
2	-	NMI	Interrupción	NO	Interrupción No enmascarable. Pin NMI
3	#BP	BreackPoint	Trap	NO	Opcode 0xCC
4	#OF	Overflow	Trap	NO	Instrucción INTO
5	#BR	BOUND Range Exceeded	Fault	NO	Instrucción BOUND
6	#UD	Invalid Opcode	Fault	NO	Instrucción UD2 u Opcode Reservado
7	#NM	Coprocesador No disponible	Fault	NO	Instrucciones de Punto Flotante o WAIT / FWAIT
8	#DF	Doble Fault	Abort	SI (Cero)	Cualquier instrucción capaz de generar una excepción, una señal en NMI o en INTR
9		Coprocessor Segment Overrun (reservada)	Fault	SI	Instrucciones de Punto Flotante
10	#TS	TSS Inválido	Fault	SI	Task switch o acceso a un TSS



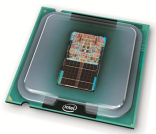
Tipos predefinidos

Tipo	Mnemónico	Descripción	Clase	Código de Error	Fuente
11	#NP	Segmento No Presente	Fault	SI	Carga o acceso a un registro de segmento
12	#SS	Falta en el Stack Segment	Fault	SI	Operacioes de Pila y Carga del registro SS
13	#GP	General Protection	Fault	SI	Cualquier referencia a memoria y otros chequeos de protección
14	#PF	Page Fault	Fault	SI	Cualquier referencia a memoria
15	-	Reservada por Intel (no usar)	NO		
16	#MF	X-87 FPU Error de Punto Flotante	Fault	NO	Instrucción de la FPU o WAI/FWAIT
17	#AC	Alignment Check	Fault	SI (Cero)	Cualquier referencia de datos a memoria
18	#MC	Machine Check	abort	NO	Los Códigos de error si hubiese, así como su fuente, depende del modelo de procesador
19	#XF	Excepción SIMD Floating Point	Fault	NO	Cualquier instrucción SSEx
20-31	-	Reservada por Intel (no usar)			
32-255	-	A definir por el usuario	Interrupción		Interrupciones externas o Instrucción INTn



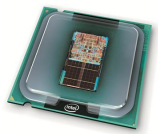
Interrupt Descriptor Table

Cuando el procesador trabaja en Modo Protegido debe definirse una tabla en memoria, llamada *IDT* (por Interrupt Descriptor Table), que a diferencia del funcionamiento en Modo Real, no almacena vectores de interrupción, sino que como su nombre lo indica almacena Descriptores.
Esta tabla tiene únicamente 256 entradas, ya que esa es la cantidad de tipos de interrupciones diferentes que maneja el microprocesador.

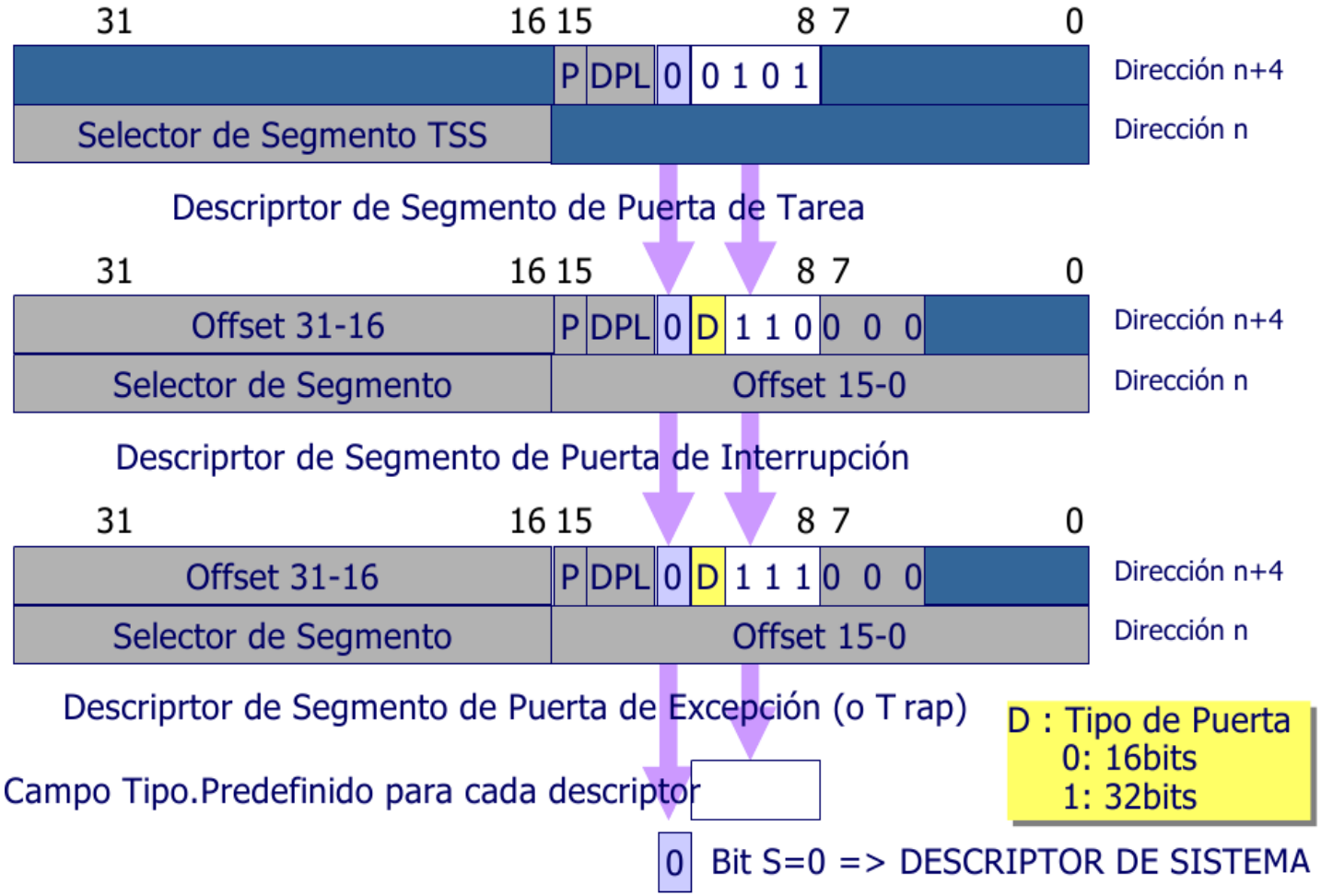


Descriptores de Sistema en la IDT

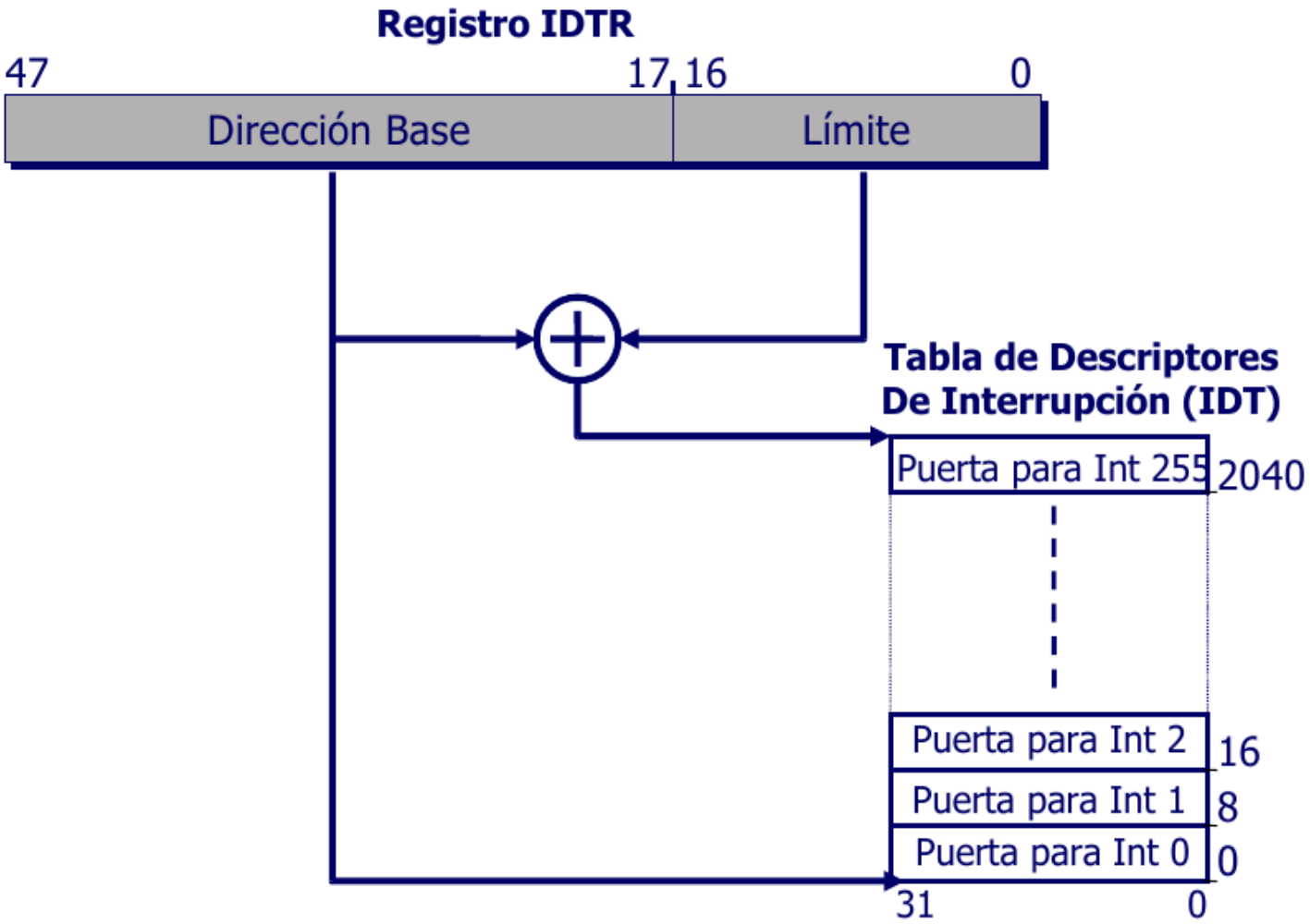
- La IDT puede contener 3 tipos de descriptores solamente:
  1. *Interrupt Gate* (de 16, 32, o 64 bits)
  2. *Trap Gate* (de 16, 32, o 64 bits)
  3. *Task Gate* (solo en el modo protegido de 32 bits!!, en 64 bits no existen)
- En los tres casos se trata de descriptores de Sistema (Bit S=0 en el descriptor)
- No se debe definir en la IDT un descriptor de segmento de datos ni de código, ni otro tipo de Descriptor de Sistema (S=0) diferente de los tres enunciados anteriormente en modo protegido de 32 bits, o de los dos pirmeros en el modo 64 bits. El procesador generará invariablemente una excepción de protección general al intentar vectorizar esa interrupción.

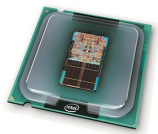


Formato de los Descriptores de 32 bits

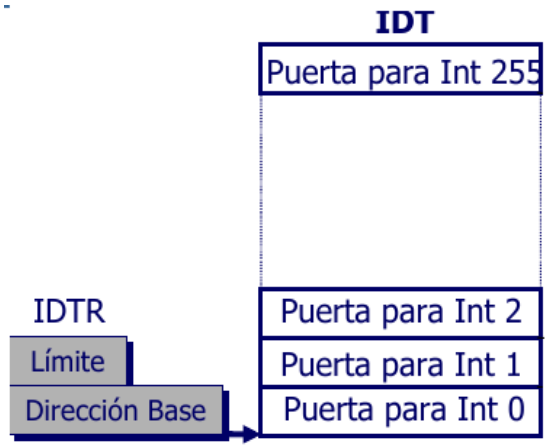


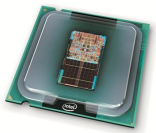
¿Como se llega a la IDT?



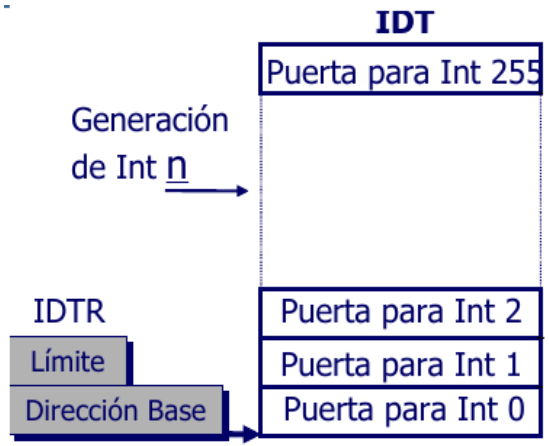


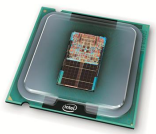
Una vez conocida la ubicación de la IDT, vectoriza la interrupción



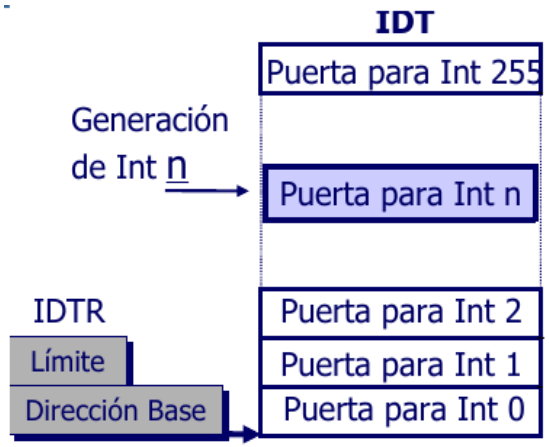


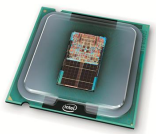
Una vez conocida la ubicación de la IDT, vectoriza la interrupción



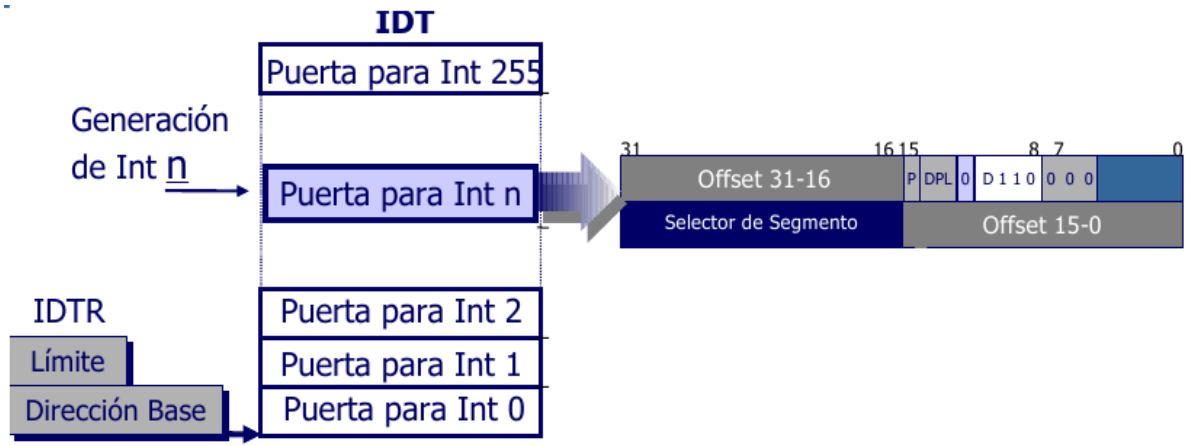


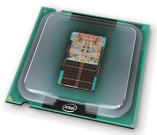
Una vez conocida la ubicación de la IDT, vectoriza la interrupción



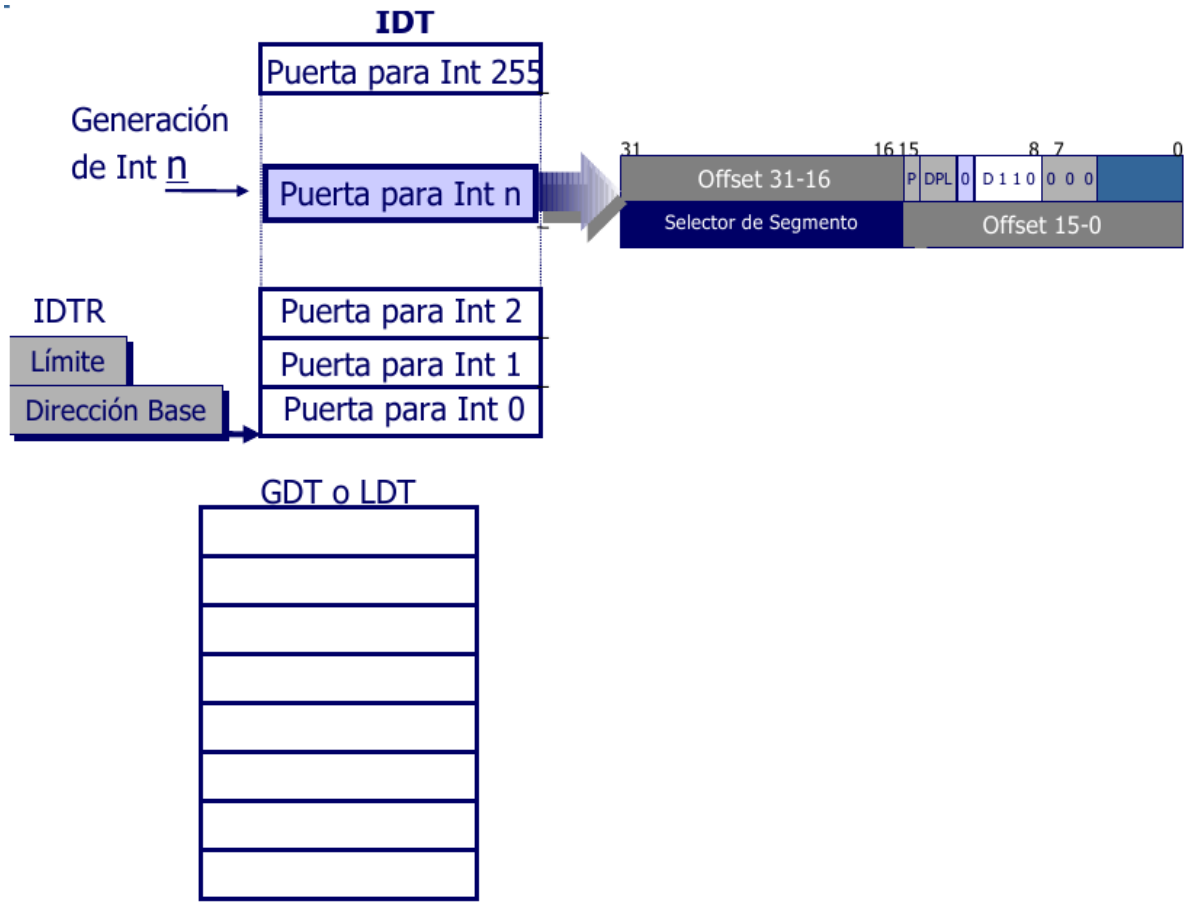


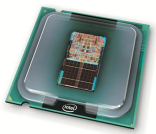
Una vez conocida la ubicación de la IDT, vectoriza la interrupción



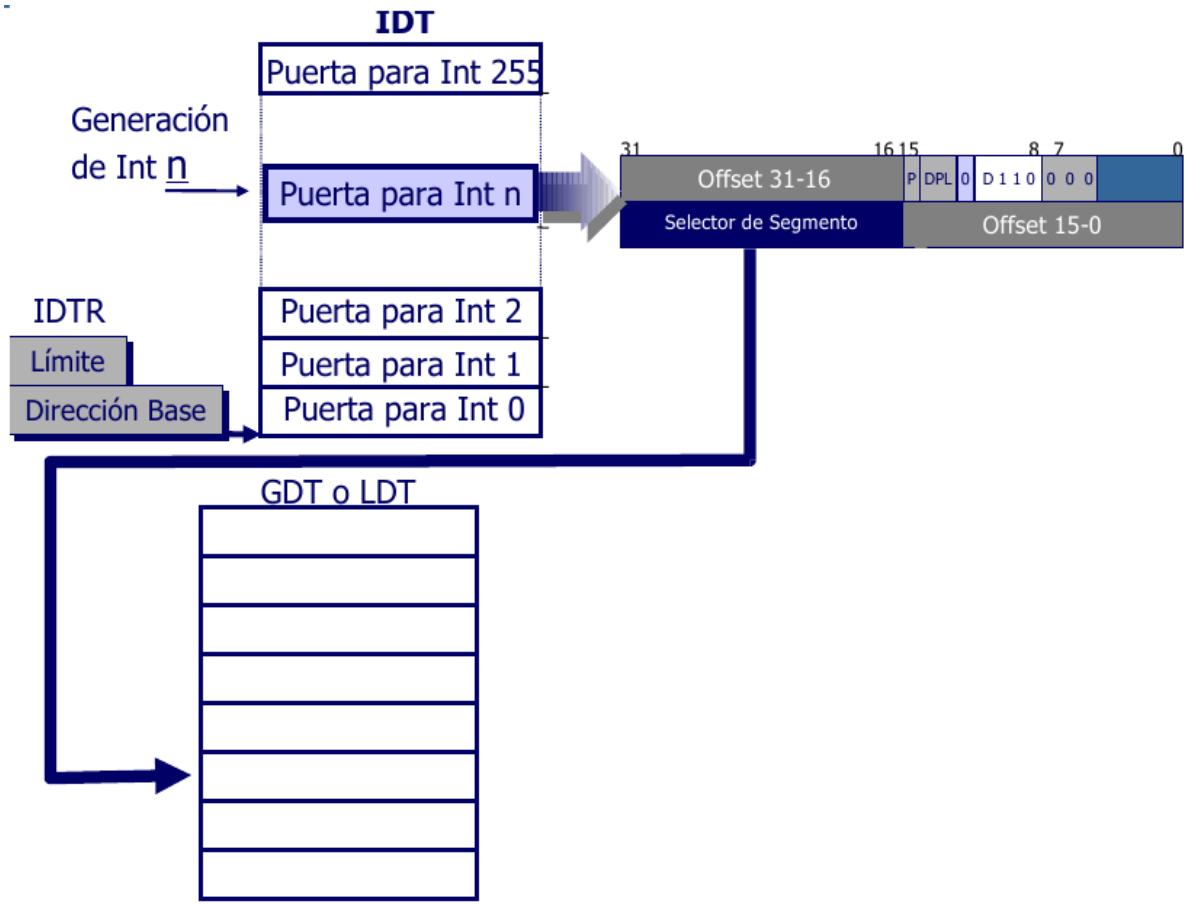


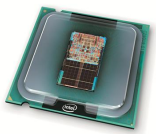
Una vez conocida la ubicación de la IDT, vectoriza la interrupción



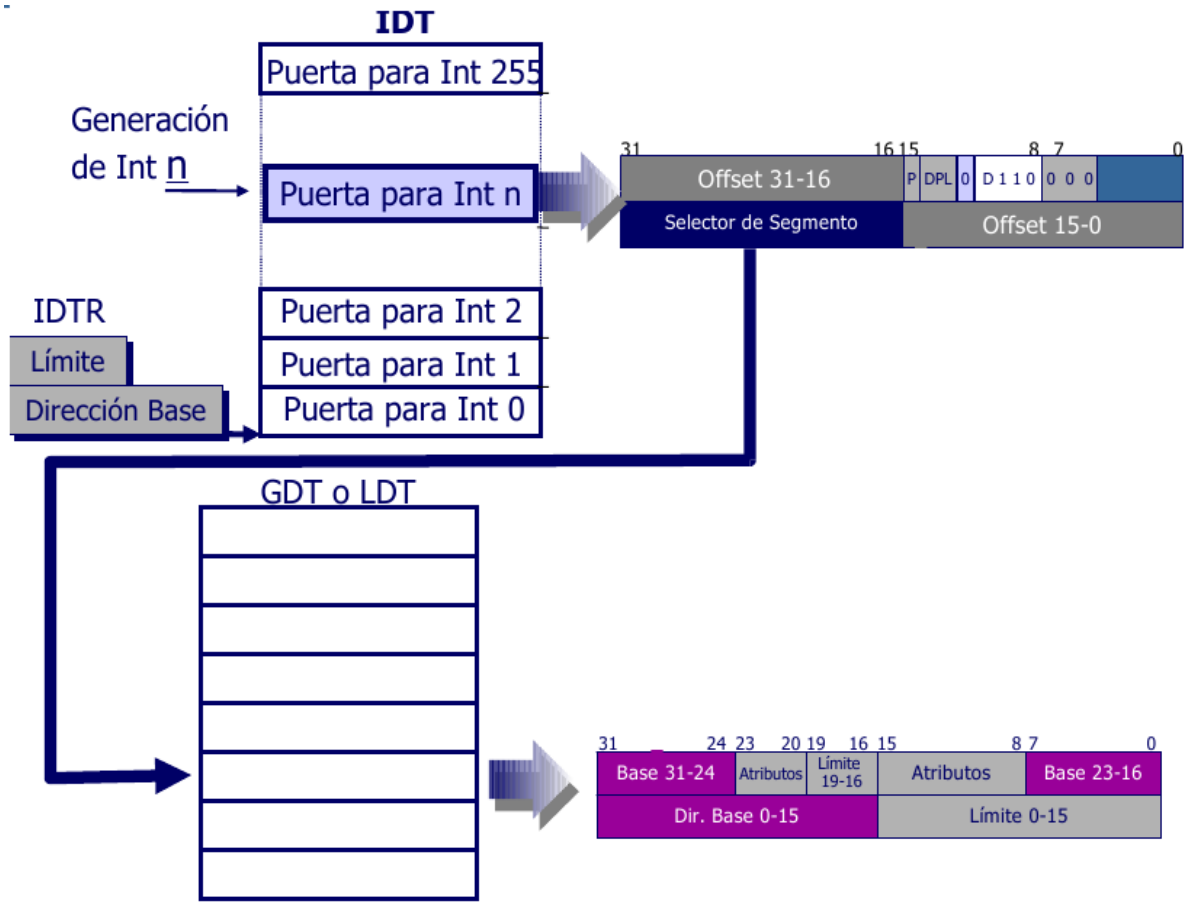


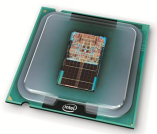
Una vez conocida la ubicación de la IDT, vectoriza la interrupción



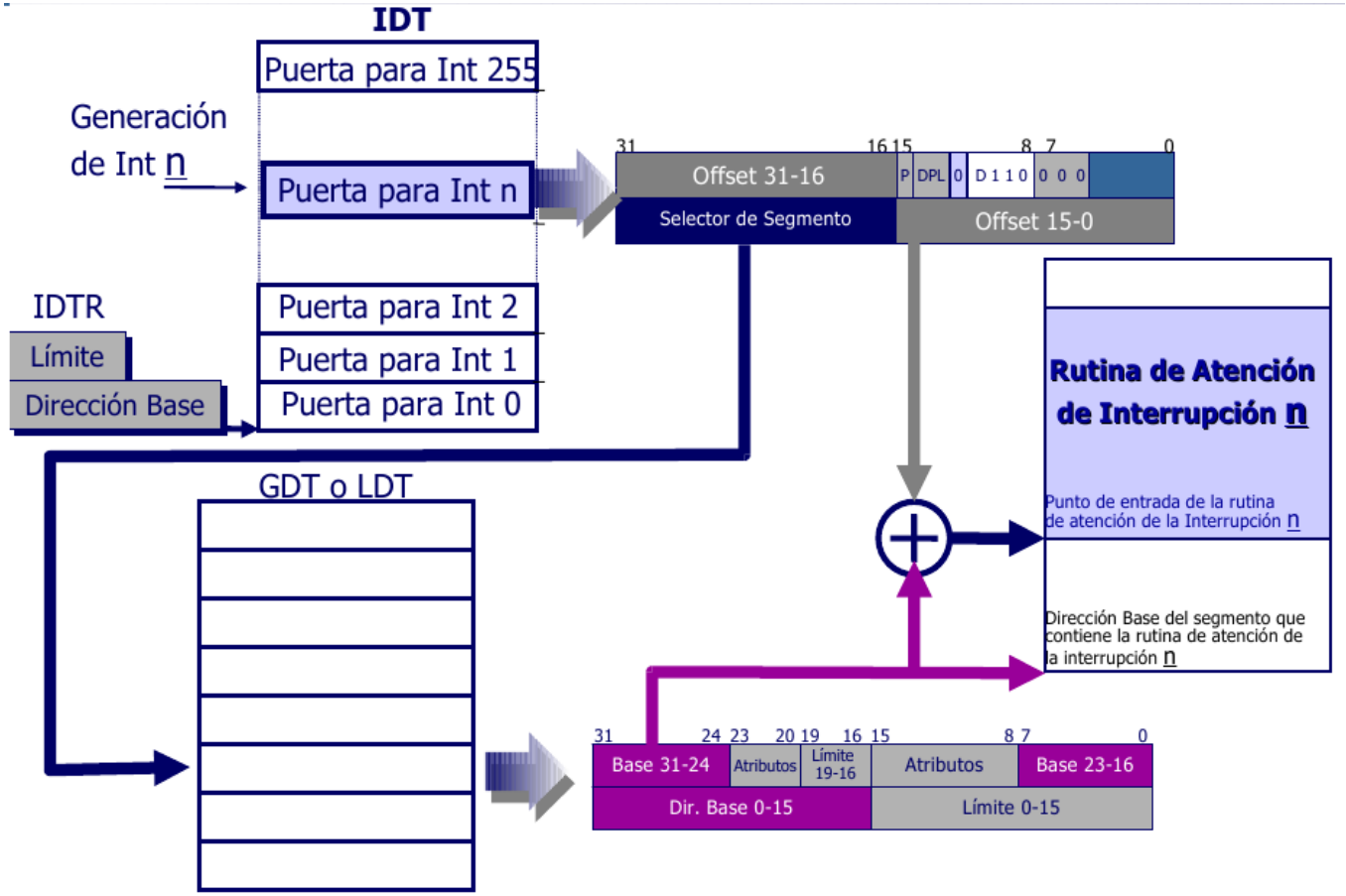


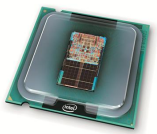
Una vez conocida la ubicación de la IDT, vectoriza la interrupción



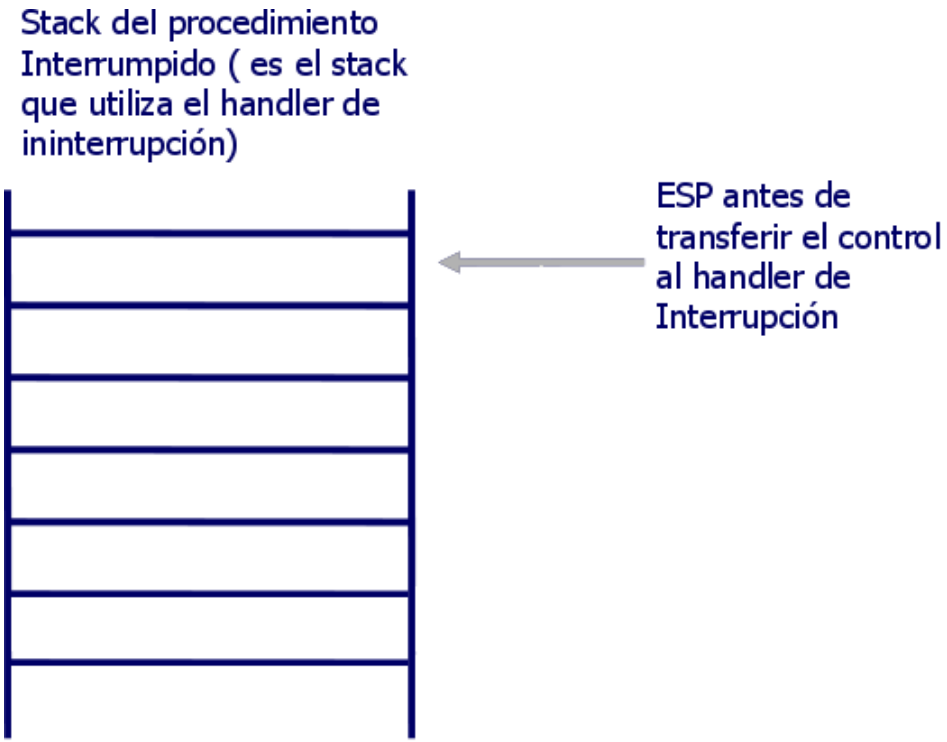


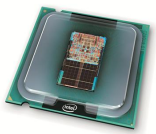
Una vez conocida la ubicación de la IDT, vectoriza la interrupción



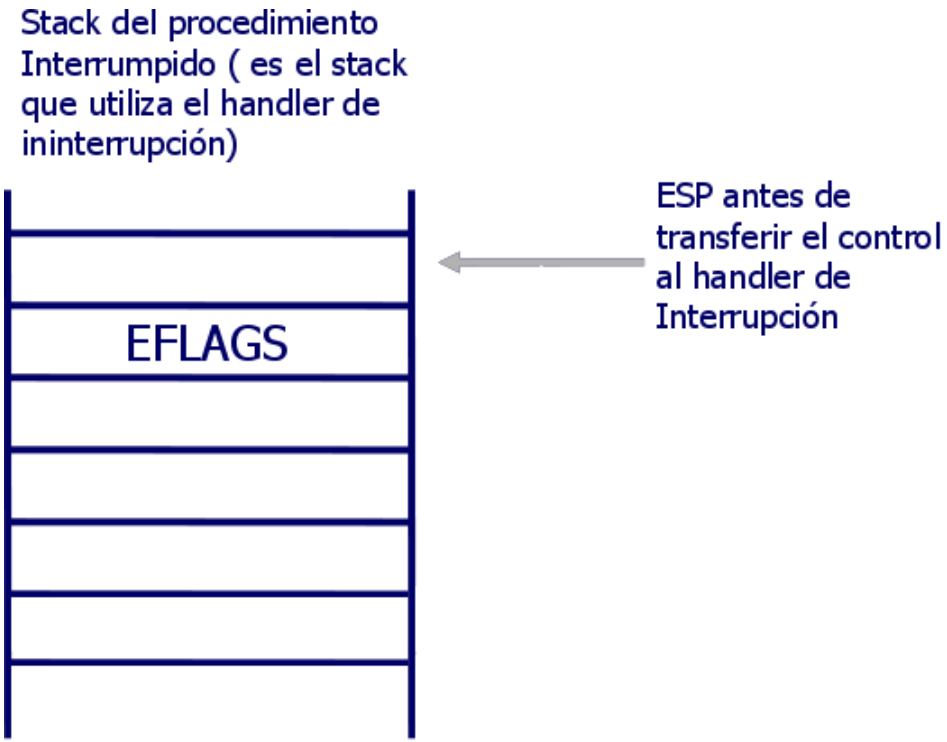


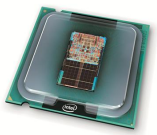
Pila de Modo USER



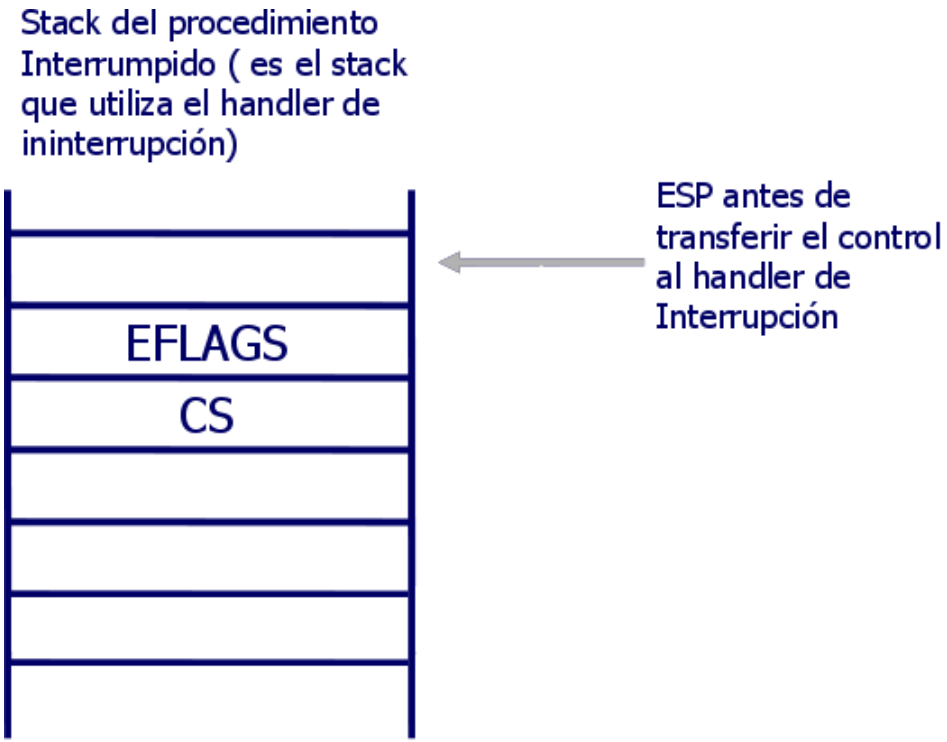


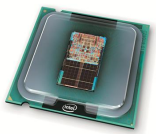
Primero: Almacena los Flags



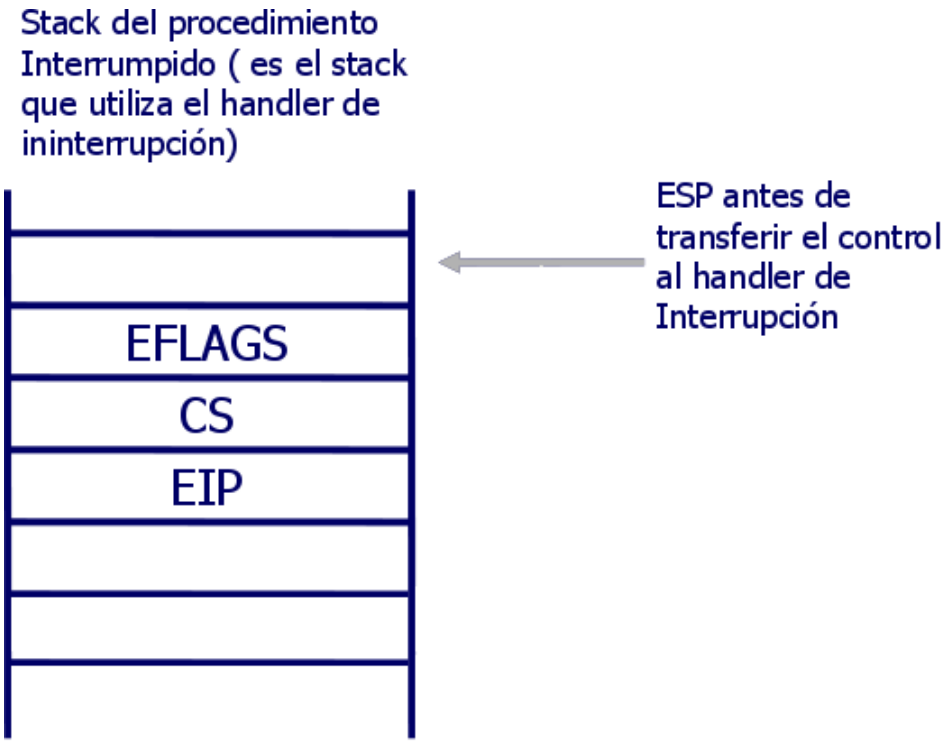


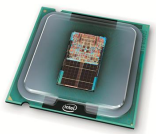
Segundo: Segmento de código de la Aplicación





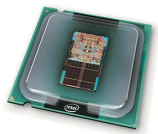
Tercero: Valor del Puntero de Instrucciones



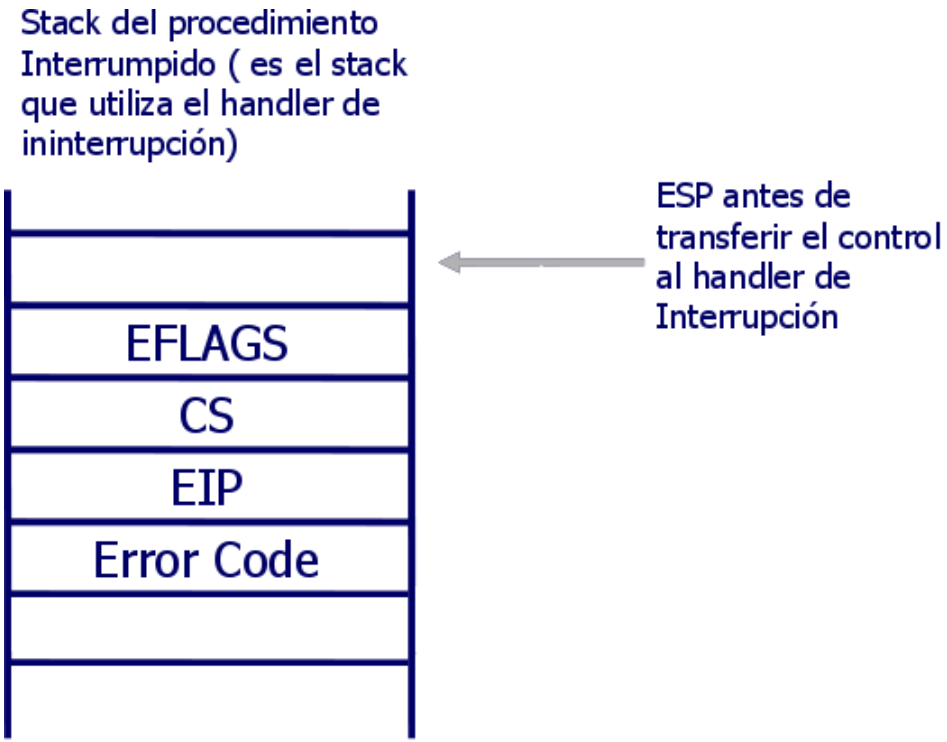


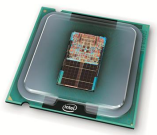
La excepciones se dividen en tres tipos

1. *Fault:* Excepción que puede corregirse permitiendo al programa retomar la ejecución de esa instrucción sin perder continuidad. El procesador guarda en la pila la dirección de la instrucción que produjo la falla.
2. *Traps:* Excepción producida inmediatamente a continuación de una instrucción de trap. Algunas permiten al procesador retomar la ejecución sin perder continuidad. Otras no. El procesador guarda en la pila la dirección de la instrucción a ejecutarse luego de la instrucción trapeada.
3. *Aborts:* Excepción que no siempre puede determinar la instrucción que la causó, ni permite recuperar la ejecución de la tarea que la causó. Reporta errores severos de hardware o inconsistencias en tablas del sistema.



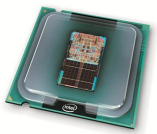
Si es una Excepción, puede haber un Código de error



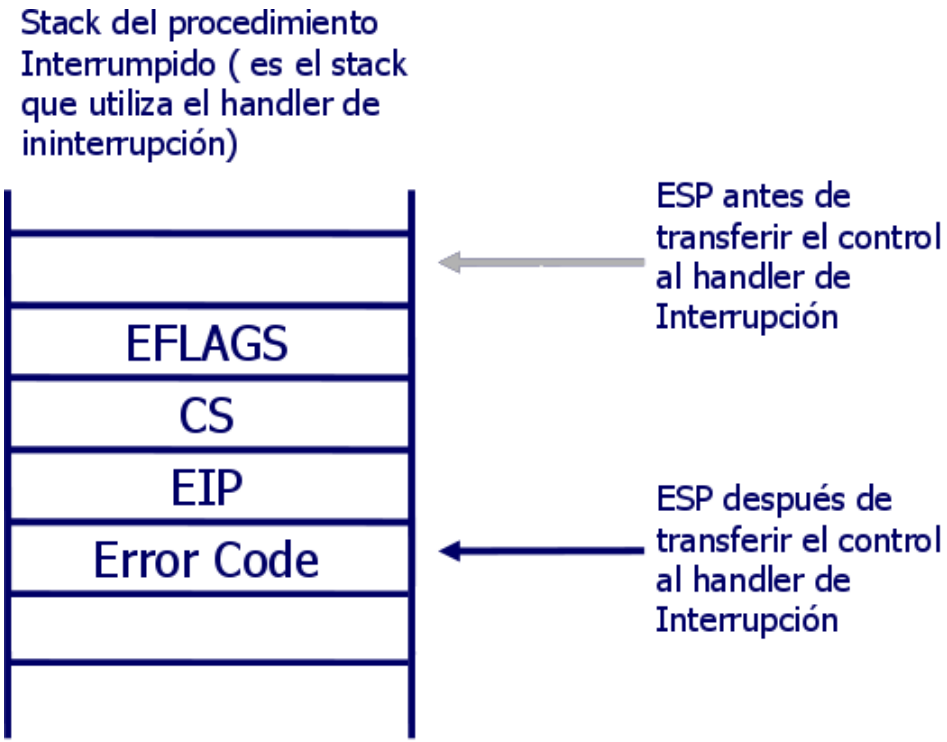


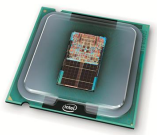
¿Para que sirve el código de error?

- Una excepción identificada con un determinado tipo (por ejemplo # GP), puede obedecer a numerosas causas, todas ellas de diferente origen.
- Una vez vectorizado el handler de la interrupción, necesitamos determinar cual ha sido la causa concreta de la excepción para poder resolverla.
- Por ello, en excepciones que pueden tener causas múltiples, se provee en la pila una palabra de 4 bytes denominda Código de Error.
- El Capítulo 6 del Vol 3A de Intel es muy detallado en este aspecto ya que allí se describen cada interrupción con sus características, de modo que es necesario prestar mucha atención a esta documentación cuando se diseña un handler de Interrupción.

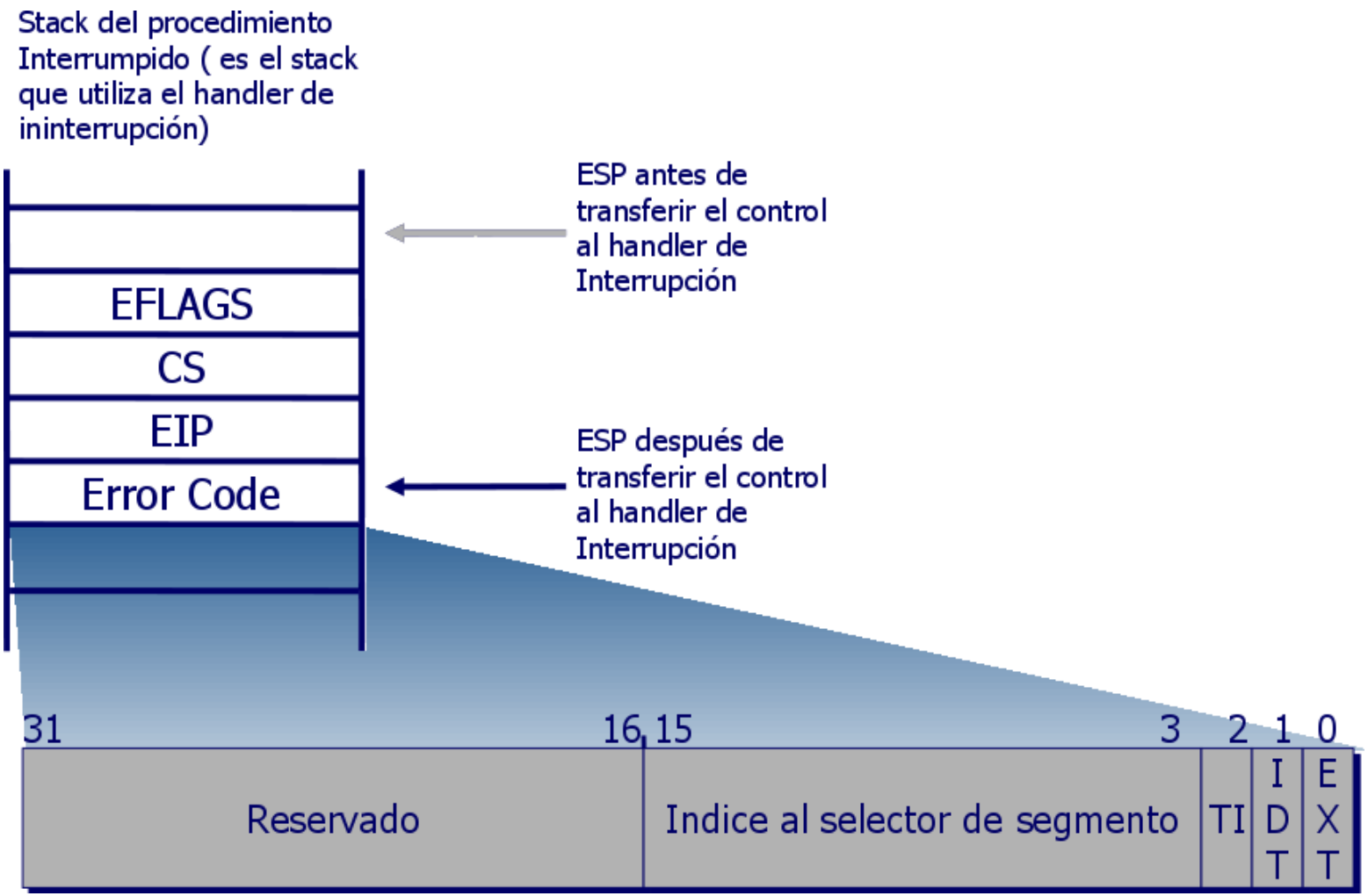


Excepciones: Código de error



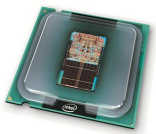


Excepciones: Código de error



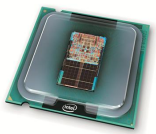
Significado del Error Code

1. *EXT:* External Event (bit 0): Se setea para indicar que la excepción ha sido causada por un evento externo al procesador.
2. *IDT:* Descriptor Location (bit 1): Cuando está seteado indica que el campo Segment Selector Index se refiere a un descriptor de puerta en la IDT: Cuando está en cero indica que dicho campo se refiere a un descriptor en la GDT o en la LDT de la tarea actual.
3. *TI:* GDT-LDT (bit 2): Tiene significado cuando el bit anterior está en cero. Indica a que tabla de descriptores corresponde el selector del campo Indice: 0 GDT, 1 LDT.



Cambio de Nivel de Privilegio de ejecución

- En un sistema Multitarea, el kernel es quien maneja las Interrupciones, y Excepciones. *¡SIEMPRE!*.
- Si al generase la excepción o la Interrupción se estaba ejecutando código del Kernel (KERNEL Mode), el resto de esta sección no aplica.
- Si la interrupción o excepción se ha producido en medio del código de una aplicación (Modo USER), será el Kernel quien tome el control.
- Conclusión: No hay cambio de contexto, pero si hay un cambio de Nivel de privilegio de ejecución.
- Entonces debemos tener presentes las reglas de protección del procesador.



Cambio de Nivel de Privilegio de ejecución

- En particular:

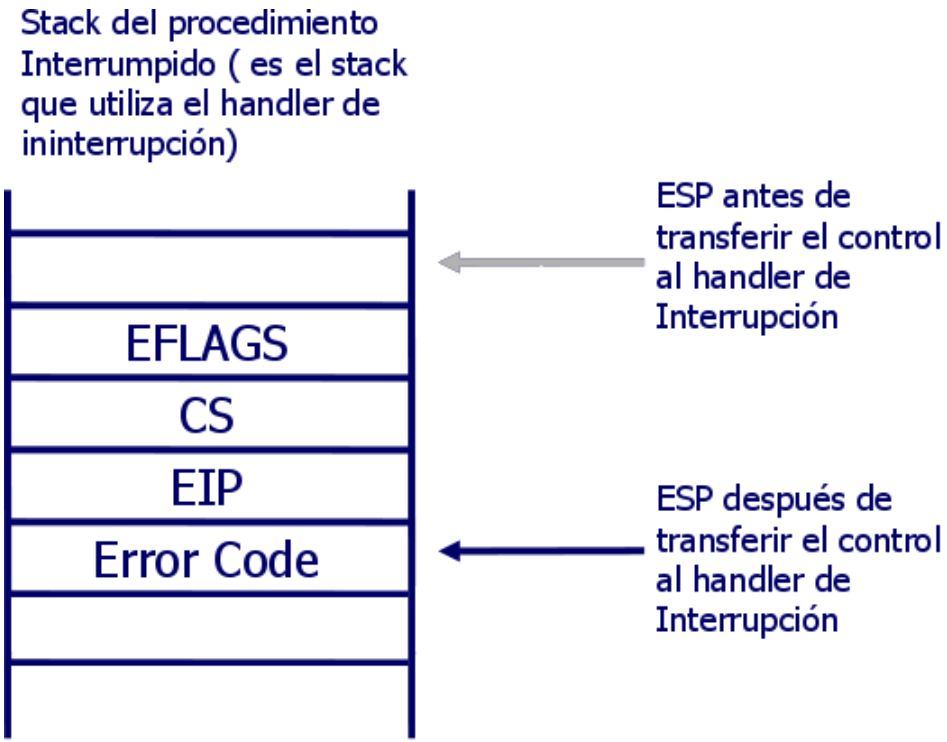
Un segmento de pila puede ser accedido por segmentos de código de mismo nivel de privilegio (Atributo DPL de sus descriptores respectivos).

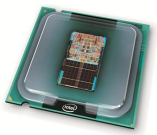
- Los datos que se terminan de guardar, se han almacenado en la pila de la aplicación, ya que la interrupción no cambia el contexto de ejecución actual. Es decir *la interrupción no produce un task switch*.

¡Importante!

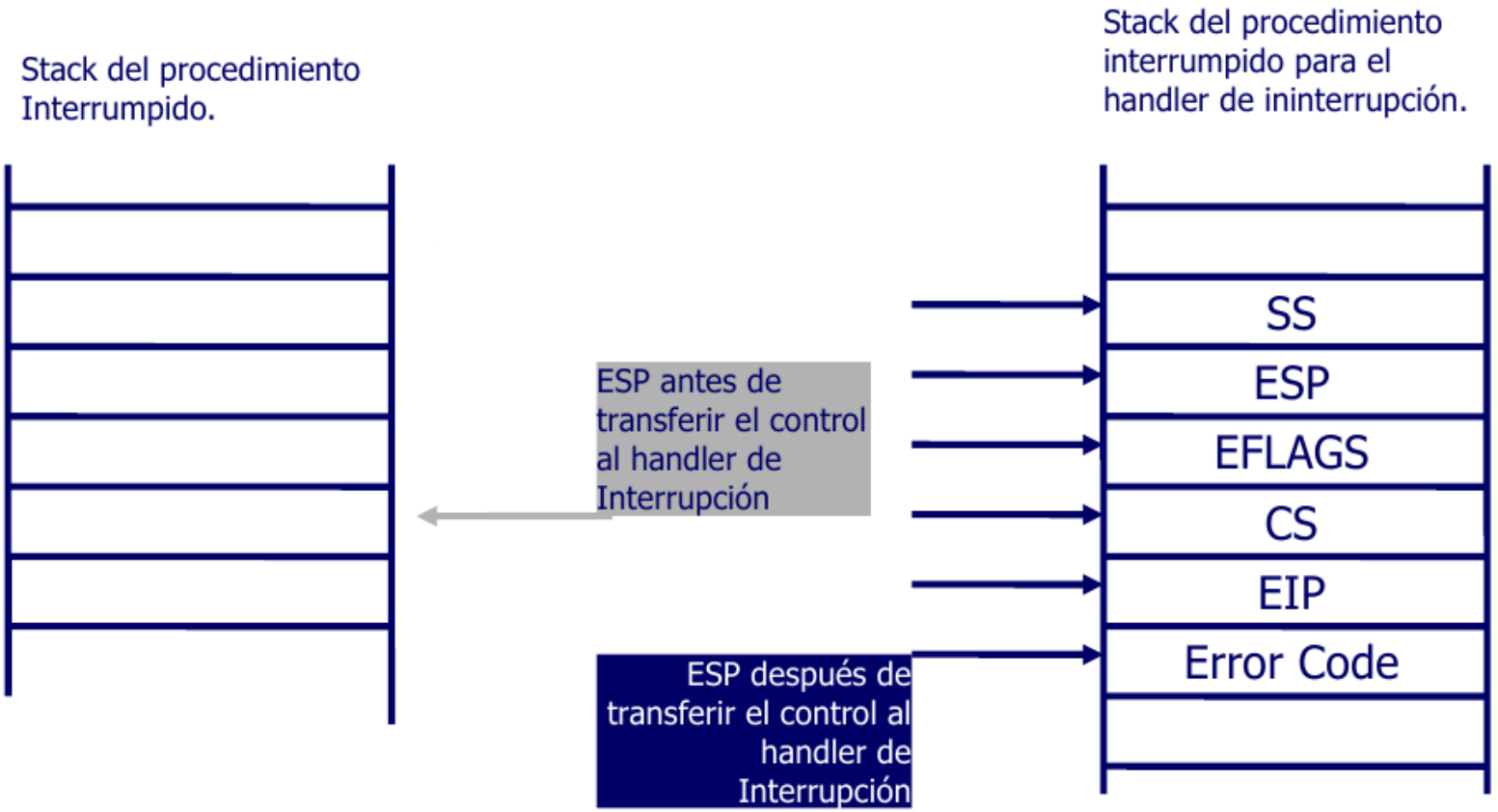
Por este motivo, necesariamente el procesador debe transferir los datos resguardados en la pila de la aplicación de USER Mode, a la pila de la aplicación de KERNEL Mode.

El procesador resguarda lo necesario en el stack de Modo USER





Y luego lo copia automática al stack Modo Kernel

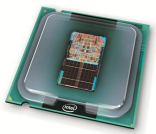


Observación

A los datos resguardados en el stack de Modo USER se suman los registros del stack de Modo USER ya que éstos van a ser escritos con los nuevos valores correspondientes al stack de Modo KERNEL. De este modo, se recuperarán al momento de que se ejecute *iret* desde el handler de la interrupción.

Si hay simultaneidad de interrupciones, ¿cual se atiende primero?

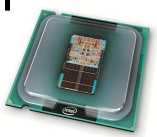
Prioridad	Descripción
1 (Máxima)	Reset y Machine check
2	Trap en Conmutacón de tarea. Flag T de TSS = 1
3	Eventos de Hardware Externos -FLUSH -STOPCLK -SMI -INIT
4	Trap en la instrucción previa -Breackpoint, o Debug Trap Exception. -TF=1 o breackpoint de dato o E/S
5	NMI, Non Maskable Interrupt
6	INTR, Interrupciones enmascarables
7	Code Breakpoint Fault
8	Fault en el ciclo de Fectch de la siguiente instrucción. -Violación del límite del segmento de código -Code-Page Fault)
9	Faults en la decodificación de la siguiente instrucción. -Instruction length mayor a 15 bytes -Invalid Opcode -Coprocessor Not Available
10 (Mínima)	Faults en la ejecución de una instrucción. -Overflow -Bound error -Invalid TSS -Segment Not Present -Stack fault -General Protection -Data Page Fault -Alignment Check -x87 FPU Floating-point except -SIMD floating-point exception



¿Que cambia en 64 bits?

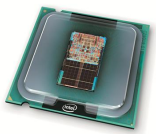
En este modo de trabajo el procesador conserva todo el mecanismo central de manejo de interrupciones. Los cambios se enumeran a continuación (no afectan la sustancia del sistema de interrupciones).

1. Excepto el handler de SMI (System Management Mode Interruption), **todos** los handlers apuntados en la IDT en el modo 64-bit **deben** apuntar a código de 64 bits.
2. Cada dato que se pushea en la pila consume 64 bits (aún un simple byte como podría ser un ASCII). Al ser la pila de uso automático en las interrupciones este punto debe tenerse presente. Si el dato guardado en la pila tiene menos de 64 bits, el procesador utiliza 8 bytes para su apilado de todos modos extendiendo a cero su parte mas significativa.
3. La pila (ss:RSP) se pushea **siempre incondicionalmente** en cada interrupción.



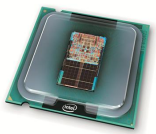
¿Que cambia en 64 bits?

4. Si a causa de la vectorización de la interrupción o excepción se eleva el nivel de privilegio, el registro SS se pone automáticamente en 0 (NULL).
5. Cambia el comportamiento de la instrucción IRET
6. Cambia el mecanismo de cambio de stack
7. Cambia el contenido del stack frame.

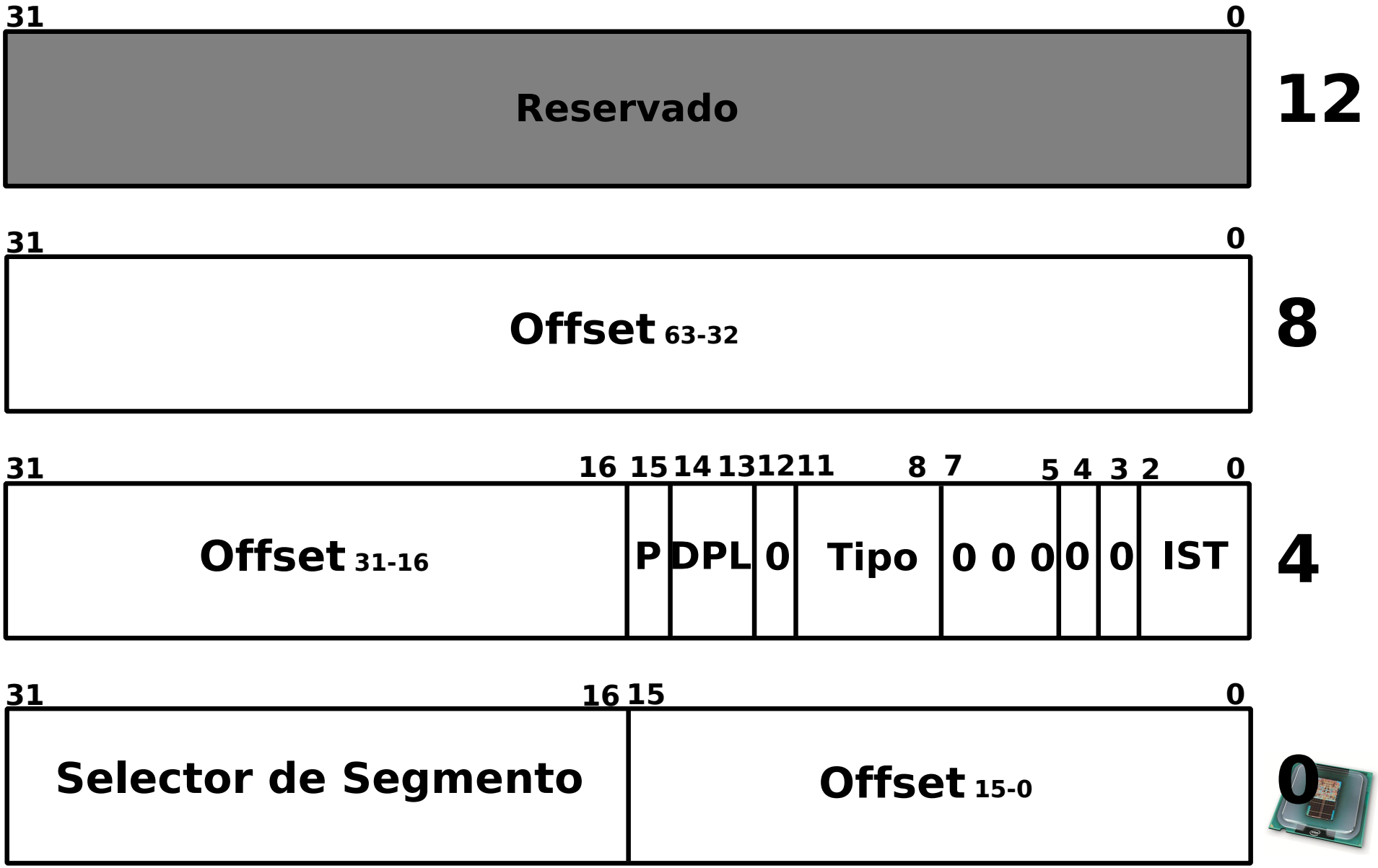


IDT en Modo 64-bit

- Los descriptores de la IDT en el Modo 64-bit ocupan invariablemente 16 bytes
- La dirección lógica que debemos almacenar requiere un offset de 64 bits (van al registro RIP).
- Tamaño máximo de una IDT es de 4 Kbytes (utilizando los 256 vectores de Interrupción)
- Hay además algún cambio menor en el descriptor.

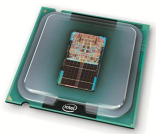


Formato de Descriptores de Interrupción/Excepción de 64 bits



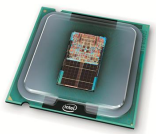
Las sutiles diferencias

- Para vectorizar una interrupción en el modo IA-32e, el procesador escala el Tipo de Interrupción por 16, ya que el tamaño del descriptor es de 16 bytes en el modo 64-bit.
- Los 8 primeros bytes de la entrada de la IDT son casi idénticos al conocido descriptor de Interrupción o Excepción de 32 bits, excepto el campo IST (Interrupt Stack Table) que se emplea para efectuar el stack switching cuando hay un cambio en el Nivel de Privilegio en el modo 64-bit.
- El uso del campo IST, se describirá en detalle en la sección Tareas.
- Los bytes 12 a 15 de estos descriptores permanecen reservados y no deben ser utilizados, para evitar conflictos con futuras versiones de estos procesadores.



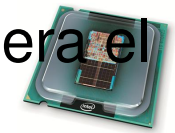
Cuestiones de protección

- El segmento de código referenciado por el selector de segmento de estas puertas (bytes 2 y 3 del descriptor de la Figura anterior, debe ser un segmento de 64 bits. De otro modo se genera una Excepción #GP guardándose en el stack un código de error en el que se almacena el selector de IDT (Tipo de interrupción)
- En el modo IA-32e no se puede incluir en la IDT descriptores de 32 bits. Solo se pueden incluir en la IDT de 64 bits Descriptores de Interrupción o de Excepción de 64 bits. Cualquier otro descriptor producirá al invocarse esa interrupción una excepción #GP.



Stack Frame en Modo 64-bit

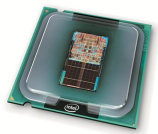
1. En el modo 64-bit, el tamaño del cada elemento que se apila en el stack-frame como resultado de una interrupción o excepción es fijo: 8 bytes siempre.
2. Cuando se vectoriza una Interrupción o Excepción en el Modo 64-bit, el par de registros SS:RSP se guarda en la pila incondicionalmente, independientemente de que se haya producido o no un cambio en el Nivel de Privilegio de ejecución (cambio en CPL).
3. En el modo IA-32e el contenido del RSP se alinea a 16 bytes antes de generar el stack frame para asegurar su alineación a 16 bytes cuando se ingrese al handler de la interrupción (o excepción). El procesador puede alinear el nuevo RSP en forma automática en respuesta a una Interrupción o Excepción, aún si el viejo RSP (que ahora se guardará incondicionalmente en el stack) no está alineado a 16 bytes. Cuando se ejecuta *iret* se recupera el viejo RSP en su valor original (alineado o no).



Stack Frame en Modo 64-bit

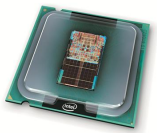
4. Las ventajas de este requerimiento de alineación del stack a 16 bytes, son claras a la hora de apilar por ejemplo registros XMM, cuyo tamaño es 16 bytes. De otro modo las instrucciones de almacenamiento de estos registros (MOVAPS y MOVUPS) se dificultan por utilizar mas ciclos de bus para acceder al stack resintiendo la performance del sistema.
5. Aunque la alineación a 16 bytes del RSP se realizará siempre si el bit LMA del MSR IA-32_EFER [1] esté en '1', esto es solo de interés para el Modo Kernel en cuyo caso no hay cambio de stack, ni se utiliza el contenido del campo IST del descriptor de la Interrupción o Excepción. Cuando se deba realizar cambio de stack el procesador asume que el Sistema Operativo ha colocado los valores de RSP en la TSS alineados a 16 bytes.

[1] MSR: Model Specific Register. Es un set de registros que se comenzó a incluir a partir del procesador Pentium, y que pueden no estar o cambiar en versiones futuras de estos procesadores. IA-32_EFER es un MSR cuyo bit LMA indica que el procesador está en Modo IA-32e y se ha configurado el



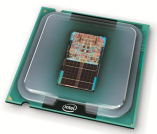
IRET en Modo 64-bit

1. En el modo de funcionamiento IA-32e *iret* trabaja con operandos de 8 bytes de tamaño. No hay motivo alguno para forzar este requerimiento, pero dada la alineación que posee el registro RSP, trabajar con un operando de tamaño submúltiplo exacto de la alineación del stack no trae ningún problema.
2. Los push en el stack frame son siempre de 8 bytes en el Modo IA-32e, por lo tanto *iret* ejecutará pop de a 8 bytes. Para ello se debe preceder a *iret* del prefijo de tamaño de operando de 8 bytes. De todos modos el tamaño del pop se determina con el tamaño de la dirección de memoria de la instrucción. El ajuste del tamaño a SS/ESP/RSP está determinado por el tamaño del stack.
3. *iret* extrae SS:RSP incondicionalmente del stack solo en el modo 64-bit. En el modo compatibilidad extrae SS:RSP del stack solo si cambia el CPL a causa de la Interrupción o Excepción. Esto es razonable tratándose del modo Compatibilidad en donde se espera que las aplicaciones de 32 bits puedan funcionar.



IRET en Modo 64-bit

4. En el Modo IA-32e, *iret* puede cargar NULL en el registro SS bajo ciertas condiciones.
   - Si el Modo de trabajo del código destino es 64-bit y su CPL es mayor de 11, *iret* puede cargar un NULL selector en SS.
   - Como parte del mecanismo de cambio de stack una Interrupción o Excepción puede poner el nuevo SS en NULL, a pesar de buscar un nuevo selector de SS en el TSS y cargar su descriptor de la LDT o GDT. El nuevo selector de SS se pone en NULL a fin de resolver adecuadamente los retornos de subsiguientes transferencias far anidadas.
   - Si el procedimiento llamado es interrumpido por si mismo (una excepción típicamente) se apila en el stack frame un SS NULL. En los *iret* subsiguientes, el SS NULL en el stack actúa como una suerte de flag para avisarle al procesador que cargue el nuevo descriptor de SS.



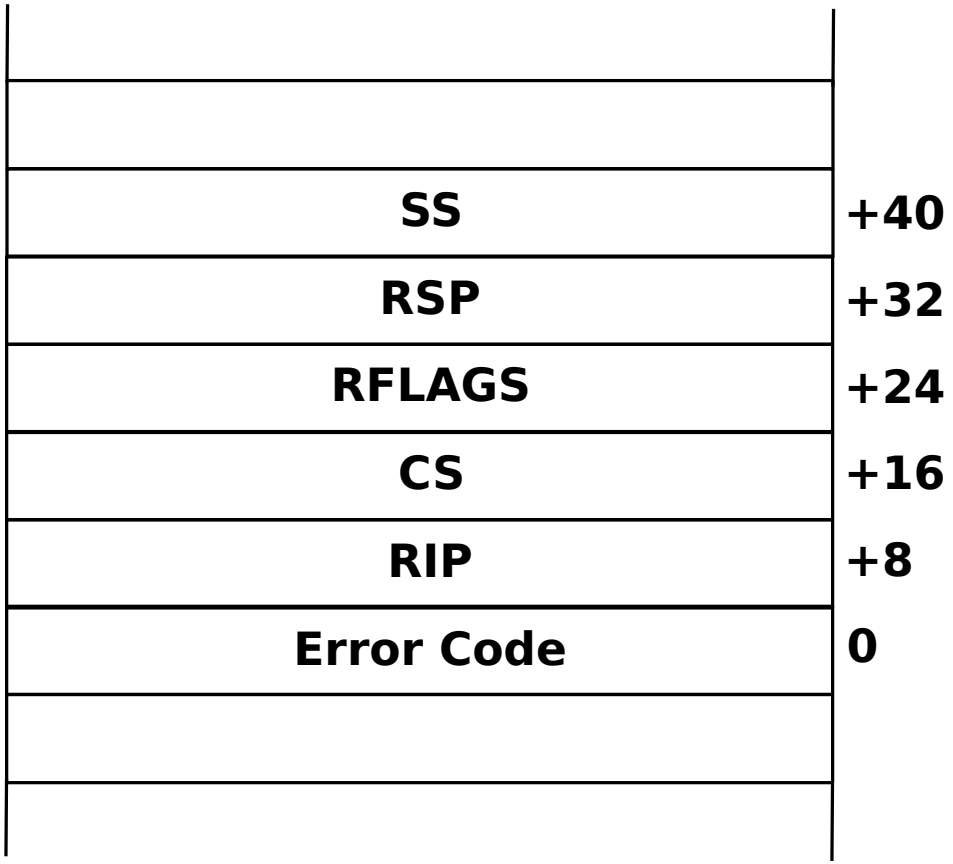
Stack Switching en Modo 64-bit

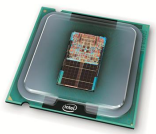
1. Cuando se produce una interrupción en el Modo 64-bit, se produce un cambio de stack, aun si no cambia el CPL.
2. No se carga un nuevo descriptor en SS, y RSP se carga con el siguiente valor interno de la IST.
3. El nuevo SS se fuerza a NULL, (excepto su campo RPL que se setea al valor del nuevo CPL), en toda transferencias far anidadas, con CALLF, INT n, Interrupciones de hardware, y Excepciones.
4. Siempre el par SS:RSP viejo se salva en el nuevo stack.
5. Todos estos métodos retornan vía *iret*, que recupera del stack que está a punto de abandonar los valores almacenados previamente de SS:RSP, para volver a ese procedimiento.

Resumen

El stack switch en modo IA-32e trabaja como en el modo legacy excepto que el registro SS no se carga con el nuevo selector de la TSS, sino que se pone a NULL.

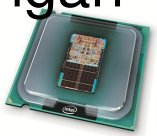
Stack Frame en Modo 64-bit





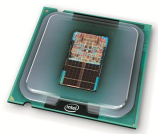
Interrupt Stack Table

- A diferencia de la arquitectura legacy de 32 y 16 bits, en el Modo 64 bits, se genera una conmutación de stack cada vez que el procesador recibe una Interrupción de Hardware, o genera una Excepción, o ejecuta una instrucción INT n o CALLF.
- La Interrupt Stack Table (IST) es un mecanismo exclusivo del modo IA-32e, que permite al procesador conseguir el nuevo stack cada vez que se produce una transferencia far.
- El mecanismo IST se habilita a nivel de cada Interrupción individual, completando el campo homónimo (IST) en aquellos descriptores de IDT que corresponden a Interrupciones o Excepciones para las que queremos que el procesador active este mecanismo.
- Aquellos vectores de interrupción cargados en IDT que no tengan este mecanismo habilitado, trabajan en el modo legacy.



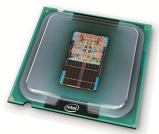
Interrupt Stack Table

- El mecanismo IST es parte de la TSS de Modo 64-bit.
- Intel establece en su documentación que este mecanismo fue pensado para interrupciones que requieran ejecutar siempre sobre un stack conocido y seguro. Tal sería el caso de las NMI, Doble Falta, Machine Check, entre otras.
- En el modo legacy de 32 y 16 bits, esta necesidad se satisfacía colocando una *Task Gate* en el descriptor correspondiente de la IDT. De este modo al producirse esas interrupciones específicas el procesador genera una conmutación de tarea que lo lleva a un entorno seguro, pero a costa de una demora adicional que implica conmutar los contextos de ejecución en forma automática. *Este mecanismo de conmutación de tareas no está disponible en el Modo IA-32e*.



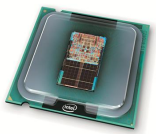
Interrupt Stack Table

- El Mecanismo IST proporciona hasta siete punteros a pilas en el TSS.
- Estos punteros son referenciados por un Descriptor de Puerta de Interrupción o Excepción en la IDT.
- Este descriptor contiene un campo de 3 bits denominado IST que hace las veces de índice dentro de la sección IST del TSS.
- Mediante este mecanismo de indexación el procesador puede cargar el valor apuntado por el índice IST en el registro RSP cuando vectoriza la interrupción correspondiente al Tipo que lleva a esa entrada de la IDT.
- Por lo demás el procesamiento de la interrupción es el habitual del modo de 32 bits.
- Cuando el índice IST está en 000, se utiliza el modo de conmutación de stack legacy (32 bits).



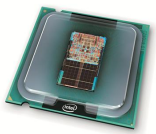
Interrupción Tipo 0: Divide Error Exception #DE

- Esta excepción es de Tipo Fault, y se produce cuando el tamaño del resultado de una instrucción DIV o IDIV supera la cantidad de bits del operando destino.
- Esta excepción deja CS:EIP apuntando a la instrucción que generó la excepción o a la siguiente de acuerdo a si fue un Fault o un Trap respectivamente.
- No se almacena código de error en la pila. La información relevante está en los Debug Registers.
- No altera el estado de la tarea ya que la interrupción se produce por no poder completarse la instrucción si es un Falut..



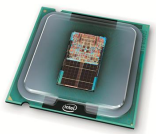
Interrupción Tipo 1: Debug Exception #DB

- Esta excepción es de Tipo Trap o Fault. Queda a responsabilidad del handler de la excepción determinar mediante los Registros de Debug si se trata de una u otra. La tarea de debug excede los límites del presente texto. Si el lector está interesado en conocer sus detalles, Intel dedica un capítulo completo del Volumen 3 de los Manuales del procesador para tratarla.
- Esta excepción deja CS:EIP apuntando a la instrucción que generó la falta.
- No se almacena código de error en la pila.
- No altera el estado de la tarea cuando es un Fault ya que la interrupción se produce por no poder completarse la instrucción. Pero en el caso de un Trap, si altera el estado de la tarea.



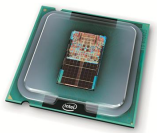
Interrupción Tipo 2: Non Maskable Interrupt

- Es una interrupción de Hardware que se activa el pin *NMI* del procesador.
- Esta interrupción deja CS:EIP apuntando a la instrucción siguiente a la que recibió la Interrupción.
- No se almacena código de error en la pila.
- No altera el estado de la tarea cuando es un Fault ya que la interrupción una vez completada la instrucción en curso.



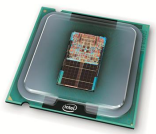
Interrupción Tipo 3: Breakpoint Exception #BE

- Esta excepción es de Tipo Trap. Utilizada ampliamente por los debugers, consiste en reemplazar el código de operación de la instrucción en la que se desea implantar el breackpoint por el código de operación de INT 3 que es el número 0xCC. Si bien, a partir de la microarquitectura P6 se introdujo la capacidad de poner breackpoints a través de los Debug Registers, sin embargo cuando se supera la cantidad soportada por estos registros el método original de reemplazar el opcode por el de INT 3, es aun sumamente práctico.
- Esta excepción deja CS:EIP apuntando a la instrucción siguiente de aquella en donde se detuvo la ejecución para generar esta excepción.
- No se almacena código de error en la pila.
- No altera el estado de la tarea ni valores de registros ni áreas de memoria, ya que el programa debe poder continuar con su ejecución normal luego de esta excepción.



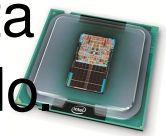
Interrupción Tipo 4: Overflow Exception #OF

- Cuando se ejecuta la Instrucción INTO, esta excepción se produce si el flag EFLAGS.OF (Overflow Flag) está seteado. Esta instrucción es un Trap. Si bien el estado de este flag puede chequearse mediante instrucciones de salto condicional. La ventaja de utilizar INTO es que si el flag está seteado se pasa directamente a ejecutar el handler de la excepción.
- Esta excepción deja CS:EIP apuntando a la instrucción siguiente a la instrucción INTO.
- No se almacena código de error en la pila.
- No altera el estado de la tarea ni valores de registros ni áreas de memoria, ya que el programa debe poder continuar con su ejecución normal luego de esta excepción.



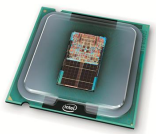
Interrupción Tipo 5: BOUND Range Exceeded Exception #BR

- Esta excepción es un Fault, que ocurre como consecuencia de la ejecución de la instrucción BOUND que se ocupa de verificar si el puntero de un array de memoria está excedido del límite superior o inferior de un array. Si lo está genera esta excepción. La idea de este mecanismo es evitar en un loop chequear una vez por ciclo el valor del puntero. La instrucción BOUND tiene dos operandos: el primero es un registro que es el que se está utilizando como puntero dentro del array, y el segundo operando es una dirección de memoria en la que se almacenan dos valores de direcciones de memoria que corresponden al límite inferior y superior respectivamente. La instrucción chequea de este modo si el contenido del registro está comprendido entre los valores de los límites almacenados en la variable de memoria, y genera esta excepción si es menor que el primero o mayor que el segundo.



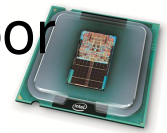
Interrupción Tipo 5: BOUND Range Exceeded Exception #BR

- Esta excepción deja CS:EIP apuntando a la instrucción BOUND, ya que debe volver a ejecutarse para verificar que el puntero ha sido reubicado dentro de los límites del array.
- No se almacena código de error en la pila.
- No altera el estado de la tarea ni valores de registros ni áreas de memoria, ya que el programa debe poder continuar con su ejecución normal luego de esta excepción.



Interrupción Tipo 6: Invalid Opcode Exception #UD

- Esta excepción es un Fault que responde a un sin número de causas, desde ejecutar, acorde a su nombre, una instrucción cuyo código de operación es reservado o inexistente, pero también si se intenta ejecutar instrucciones existentes en condiciones incorrectas, como por ejemplo un operando inconsistente con el código de operación, ejecutar instrucciones SIMD con el bit CR0.EM = 1, o con CR4.OSFXSR = 0. No nos proponemos repetir los textos del manual de Intel, por lo tanto no abundaremos en los detalles que están exhaustivamente detallados en el manual.
- Esta excepción deja CS:EIP apuntando a la instrucción que generó la excepción.
- No se almacena código de error en la pila.
- No altera el estado de la tarea ni valores de registros ni áreas de memoria, ya que la instrucción que la origina no se ejecutó por inválida.

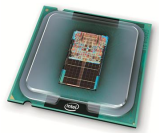


Interrupción Tipo 7: Device Not Available Exception #NM

- Esta excepción es un Fault que responde dos situaciones:
  - Ejecutar una instrucción que involucre registros SIMD o de la FPU, cuando el Flag CR0.TS = 1, y cuyo tratamiento veremos en detalle en clases posteriores.
  - Ejecutar instrucciones SIMD con el bit CR0.EM = 1, ya que implicaría que el procesador no tiene FPU (situación heredada de los primeros procesadores x86 cuya FPU era un chip x87 externo), y en tales circunstancias el handler de esta excepción podría obrar como emulador por software de la FPU. Esta no es actualmente la razón que un sistema operativo considere para esta excepción. Sin embargo debido al compromiso de compatibilidad de Intel sigue estando soportada.
- Esta excepción deja CS:EIP apuntando a la instrucción SIMD que generó la excepción.
- No se almacena código de error en la pila.
- No altera el estado de la tarea ni valores de registros ni áreas de memoria.

Interrupción Tipo 8: Double Fault Exception #DF

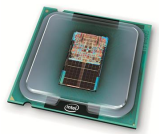
Esta excepción es un Abort que ocurre cuando el procesador detecta una segunda excepción durante la vectorización de una excepción previamente detectada. Generalmente cuando se tiene esta situación, el procesador trata de manejar ambas excepciones en serie. Esto implica completar la vectorización de la primer excepción, y una vez que procesa la primer instrucción del handler de esa excepción, vectoriza la siguiente excepción que se produjo mientras vectorizaba la primera. Esta situación solo se logra cuando la segunda excepción no le impide llegar a la primer instrucción del handler del la primer excepción. Sin embargo en ocasiones no es posible procesarlas en serie. En tal circunstancia se genera esta excepción Double Fault. Para determinar si puede o no procesar serialmente una segunda excepción que se produzca durante la vectorización de la excepción actual, las excepciones de estos procesadores se clasifican en tres categorías: Benign, Contributory, y Page Faults (o en castellano Benigna, Contributiva, y Fallo de Página).



Interrupción Tipo 8: Double Fault Exception #DF

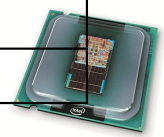
Para determinar si se genera o no #DF ante una segunda excepción durante la vectorización en curso de la primer excepción el procesador aplica los siguientes criterios

- Si la primer excepción es Benigna, la segunda se procesa en serie sin generar #DF, independientemente de su clasificación.
- Si la primer excepción es Contributiva, la segunda generará #DF solo si se trata de una Contributiva. Las demás se procesan en serie con la primera una vez que se haya vectorizado su handler y ejecutado la primer instrucción del mismo.
- Si se encuentra procesando la vectorización de un Fallo de Página, y se produce una segunda excepción, se tratará en serie, solo si la segunda excepción es alguna de las definidas como Benignas en la Tabla del slide siguiente. Si la segunda es Contributiva o un segundo Page Fault, se generará #DF.



Interrupción Tipo 8: Double Fault Exception #DF

Clase	Tipo	Descripción
Benign e Interrupciones	1	Debug
	2	*NMI*
	3	BreackPoint
	4	Overflow
	5	BOUND Range Exceeded
	6	Opcode Inválido
	7	Device Not Available
	9	Coprocessor Segment Overrun
	16	Floating Point Error
	17	Alignment Check
	18	Machine Check
	19	SIMD Floating Point
	Cualquier INT n	NMI
	Cualquier señal por *INTR*	NMI
Contributory	0	Divide Error
	10	*TSS* Inválido
	11	Segmento No presente
	12	Stack Fault
	13	General Protection Fault
Page Faults	14	Page Fault



Interrupción Tipo 8: Double Fault Exception #DF

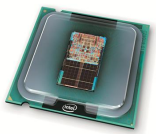
- Los criterios anteriormente detallados se ponen en la tabla siguiente para mayor claridad. En ocasiones resulta mas efectivo visualizarlo.

Primer Excepción	Segunda Excepción		
	Benign	Contributory	Page Fault
Benign	Las procesa en Serie	Las procesa en Serie	Las procesa en Serie
Contributory	Las procesa en Serie	Genera #DF	Las procesa en Serie
Page Fault	Las procesa en Serie	Genera #DF	Genera #DF

- Si se produce una excepción (cualquiera sea) mientras el procesador intenta vectorizar la excepción #DF. . . bueno . . . realmente estamos en problemas. El procesador entra en el modo shutdown. Queda en un estado similar al que lo pone la instrucción HLT. Solo se recupera si se produce *NMI* , Hardware reset, una interrupción SMI, o se activa la señal #INIT. (Observar que las interrupciones comunes de hardware no lo recuperan).

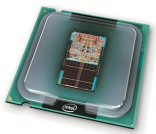
Interrupción Tipo 8: Double Fault Exception #DF

- Les propongo el siguiente ejercicio de razonamiento mirando la primer tabla y analizar para cada contributory que ocurriría durante su vectorización son alguna otra de cada clase para comprender este criterio adoptado para el procesador. Por ejemplo: Tomamos una excepción Segmento no presente. Si durante la vectorización de una #GP (contributory), se produce una excepción *TSS* inválido, la única explicación es que la excepción #GP es manejada por una tarea, de modo que en la *IDT* debe haber una puerta de tarea en la posición correspondiente al descriptor. Si al intentar cargar el descriptor de TSS de la tarea se detecta algún problema, ya sea con el valor del campo límite o algún otro inconveniente, la excepción que se debería generar es *TSS* inválido. Sin embargo esto impediría resolver la #GP.



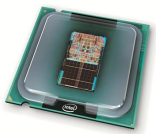
Interrupción Tipo 8: Double Fault Exception #DF

- Otro escenario es que al intentar vectorizar la excepción #GP, se cargue en el par de registros SS:EBP una pila inválida o cuyo segmento tenga un descriptor mal conformado. Esto daría la pauta de un contexto de tarea corrupto.
- Muchas de estas situaciones implican problemas internos en el kernel, pero eso no quiere decir que el hardware no deba contemplar su ocurrencia. En la fase inicial de desarrollo de un kernel estos escenarios de falla son mas cotidianos de lo que podemos imaginar.



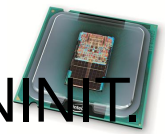
Interrupción Tipo 8: Double Fault Exception #DF

- Esta excepción deja CS:EIP en un valor indefinido.
- Almacena código de error en 0 en la pila, indicando así que es una #DF.
- El estado de la tarea luego de esta excepción es indefinido.



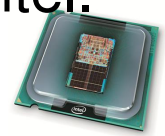
Interrupción Tipo 9: Coprocessor Segment Overrun

- Esta excepción no se genera mas a partir de la Microarquitectura P6. Su uso se reserva solo en procesadores 80386.
  Se produce cuando el procesador genera una violación de página o segmento al transferirle un operando a la FPU. En el 80386 la FPU iba en un chip separado (el 80387). Pero a partir del 80486 al integrar todo en el mismo chip muchas cosas cambiaron en la interacción Procesador x86 - FPU. Una de ellas fue que esta excepción se incluyó en la excepción de Protección General (#GP)
- Esta excepción deja CS:EIP apuntando a la instrucción que generó la excepción.
- No almacena código de error en la pila.
- El estado de la tarea luego de esta excepción es indefinido. Lo único que puede hacerse en el handler de esta excepción es salvar el EIP, y reinicializar la FPU mediante la instrucción FNINIT.



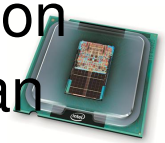
Interrupción Tipo 10: TSS Inválido #TS

- Esta excepción de tipo Fault, se produce cuando una instrucción invoca una conmutación de tareas y se detecta alguna situación anómala durante el proceso de conmutación.
- En general se pueden agrupar en problemas con el *TSS* actual o con su selector,
- Problemas con los selectores de segmento de la tarea en proceso de carga,
- Problemas con el selector de LDT de la tarea que se esta carando,
- o problemas con el *TSS* destino o su selector.
- La lista es bastante larga y está detalladamente enumerada en el Capítulo Interrupciones y Excepciones de los manuales de Intel.



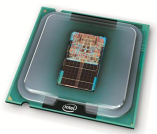
Interrupción Tipo 10: TSS Inválido #TS

- Si la situación que produjo la excepción fue anterior al cambio de tarea y fue resuelta, esta excepción deja CS:EIP apuntando a la instrucción que invoca el cambio de tarea.
  En cambio si la situación que produjo la excepción es posterior al cambio de tarea, una vez resuelta la situación, la excepción deja CS:EIP apuntando a los valores de estos registros correspondientes a la primer instrucción a ejecutar cuando luego de resuelta la excepción se reasuma la ejecución de la tarea..
- No almacena código de error en la pila.
- El estado de la tarea luego de esta excepción es fuertemente dependiente del punto en el que se produjo durante la conmutación de tarea. Si ocurre la excepción antes de la conmutación no hay cambio en el contexto de la tarea. Si ocurre en medio la situación puede quedar indefinida ya que al cargar los selectores de segmento, si alguno de ellos genera la excepción por alguno de los problemas descriptos, los siguientes quedan cargados pero sin ser verificados.



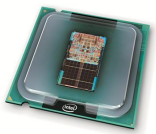
Interrupción Tipo 11: Segmento No Presente #NP

- Esta excepción de tipo Fault, se produce cuando se intenta cargar el descriptor seleccionado por CS, DS, ES, FS, o GS, y su atributo **P** es 0.
- no incluimos al registro SS, ya que si al cargar su descriptor asociado, el atributo **P** es 0, genera un Stack Fault #SS, excepción Tipo 12 que viene a continuación de ésta.
- También genera esta excepción si la instrucción LLDT carga un selector de *LDT* cuyo descriptor tiene **P** en 0. Notar que si el LDTR se carga a partir de una conmutación de tarea y ocurre esta situación la excepción que se genera es la anterior: *TSS* Inválido. Lo mismo ocurre si LTR carga un selector que apunta a un descriptor con **P** en 0.



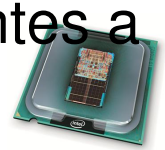
Interrupción Tipo 11: Segmento No Presente #NP

- Deja CS:EIP apuntando a la instrucción que produjo la excepción.
- Almacena código de error en la pila. La interpretación del mismo responde a lo explicado en el slide de Código de error.
- El estado de la tarea luego de esta excepción si fue resultado de la carga de un selector no afecta a la tarea. Si en cambio ocurre como consecuencia de una conmutación de tarea puede quedar indefinida de acuerdo con lo explicado en esa otra excepción.



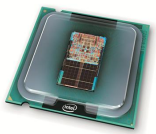
Interrupción Tipo 12: Stack Fault #SS

- Esta excepción de tipo Fault se genera cada vez que una instrucción que utiliza el registro SS para direccionar memoria, ya sea implícitamente (PUSH, POP, ENTER, LEAVE, POPF, CALL, RET, e IRET), como en forma explícita (por ejemplo MOV [SS:EBP+16], 0xF600), viola el límite efectivo del segmento.
- También se genera esta excepción si se escribe en el registro SS un selector cuyo descriptor tiene el bit **P** en 0.
- Finalmente si el procesador trabaja en modo 64 bits y a utilizando como segmento la pila apuntada por el registro SS se obtiene una dirección resultante cuyo formato no es canónico también se produce esta excepción.
- Deja CS:EIP apuntando a la instrucción que produjo la excepción. Si #SS se produce como resultado de un stack switch en el que se carga un valor de SS que corresponde a un descriptor de segmento No Presente, los valores de CS:EIP correspondientes a la primer instrucción a ejecutar cuando luego de resuelta la excepción se reasuma la ejecución de la tarea.



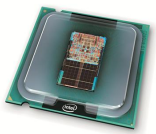
Interrupción Tipo 12: Stack Fault #SS

- Almacena código de error en la pila, dando datos del stack responsable del Fallo, siempre que la excepción se deba a un descriptor de segmento No Presente cuyo selector fue cargado en el registro SS, o que se haya producido un stack overflow por parte del ESP, o RSP en medio de un cambio de nivel de privilegio. La interpretación del mismo responde a lo explicado en Código de Error. Si la excepción se debe a una violación del límite sin cambio de Nivel de privilegio, el código de error es 0.
- Por lo general la instrucción responsable del Stack Fault no se completa de modo que no se afecta el estado de la tarea en ejecución.



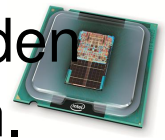
Interrupción Tipo 13: General Protection Fault #GP

- Esta excepción es de clase Fault, y engloba todas las violaciones generales al sistema de protección que no estén comprendidas, en Stack Fault, *TSS* inválida o Page Fault.
  Tal vez se trate de la lista mas extensa de motivos. Al igual que en casos anteriores no tiene sentido copiarla de los manuales de Intel a este texto. Si recomendamos al lector una lectura en profundidad para analizar los diferentes casos. De otro modo va tener que recurrir de todos modos al comenzar a construir un kernel, ya que al principio cualquier error cometido aterriza en esta excepción.
- Deja CS:EIP apuntando a la instrucción que produjo la excepción.



Interrupción Tipo 13: General Protection Fault #GP

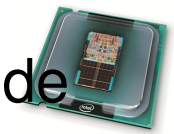
- Almacena código de error en la pila. Si la excepción se generó al cargar un descriptor, el código de error cuyo formato se explicó anteriormente, contiene el el selector del segmento o de la entrada de la IDT correspondiente. De otro modo el código de error almacenado es 0. El selector puede provenir de un operando de una instrucción, de una puerta que es operando de la instrucción, un selector de *TSS* involucrado en una conmutación de tarea, o de un *Tipo* de Interrupción en la *IDT* .
- Por lo general la instrucción responsable de la General Protection Fault no se completa, de modo que no se afecta el estado de la tarea en ejecución. En el caso en que la #GP se genere durante un cambio de tarea, dependerá si es antes o después de aplicar el estado de la tarea a los registros. En el caso en que aun no se hubiese aplicado, el estado de la tarea no ha cambiado, pero si se ha aplicado los valores de los registros de segmento no pueden ser asumidos como confiables por el handler de la excepción.



Interrupción Tipo 14: Page Fault #PF

Tipo Fault. Engloba todas las violaciones al sistema de protección relacionadas con l Sistema de Paginación, que pueden producirse en el momento de traducción de la dirección Lineal. Causas:

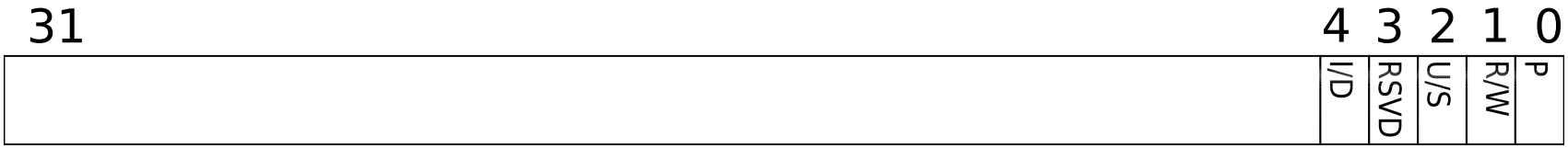
1. El bit **P** de al menos uno de los descriptores de página en cualquiera de las tablas de la estructura de paginación que intervienen en la traducción de la dirección lineal está(n) en '0'.
2. Código con privilegio de usuario (ejecuta en un segmento con ***CPL*** = 11), intenta acceder a una página, en cuya estructura de tablas **al menos una entrada tiene privilegios de Supervisor**.
3. Código con privilegio de usuario intenta escribir una página Read Only. A partir de los procesadores 80486, se suma a esta condición que código en modo supervisor intenta escribir una página Read Only si el bit **CR0.WP** = 1.
4. Una operación de opcode fetch se traduce a una página con el bit ED en su descriptor en '1', si el procesador lo soporta.
5. Uno o mas bits que beden estar reservados en el descriptor de directorio de Tablas de Página está seteado.



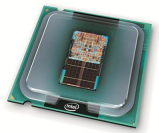
Interrupción Tipo 14: Page Fault #PF

- Deja CS:EIP apuntando a la instrucción que produjo la excepción. En el caso en que #PF provenga de unan cambio de tarea, el valor de CS:EIP, es el de la instrucción que invoca la conmutación de la tarea siempre que la excepción se haya producido antes del cambio de contexto. Si el cambio de contexto ya se aplicó, CS:EIP apuntarán a la primer instrucción de la tarea.
- Almacena código de error en la pila. Sin embargo el formato de este código es diferente al que se explicó anteriormente. Aquel contiene el el selector del segmento o de la entrada de la *IDT* correspondiente. Pero en esta excepción la segmentación no cuenta. Se consideran en ella todos los fallos relacionados con la paginación. Por lo tanto el código de error debe contener la información precisa que permita al handler identificar cual de las causas explicadas en el párrafo anterior es la que debe resolverse. Su estructura se muestra en la Figura en el siguiente slide, en donde debe asumirse que cada uno al ser '1' indica que la excepción corresponde a esa causa.

Interrupción Tipo 14: Page Fault #PF



- Para el análisis de la causa de la excepción el código de error se complementa con el registro de control CR2 que contiene la dirección lineal que produjo el fallo. Esta información es vital para por ejemplo determinar a partir de la dirección lineal que descriptores de página están involucrados a lo largo de la estructura de paginación.
- Una de las primeras acciones que debe hacer un handler de #PF es resguardar este registro en memoria, por si se produce una segunda #PF ya que de otro modo este valor será sobrescrito perdiendo el rastro del CR2 de la primer #PF. La sobre escritura de CR2 se produce aún cuando el segunda Fallo de Página genere en realidad una Excepción Double Fault (#DF).



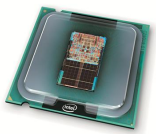
Interrupción Tipo 14: Page Fault #PF

En general durante un cambio de tareas se puede producir un #PF por los siguientes motivos:

- Por lo general la instrucción responsable de la General Protection Fault no se completa, de modo que no se afecta el estado de la tarea en ejecución. En el caso en que la #GP se genere durante un cambio de tarea, dependerá si es antes o después de aplicar el estado de la tarea a los registros. En el caso en que aun no se hubiese aplicado, el estado de la tarea no ha cambiado, pero si se ha aplicado los valores de los registros no pueden ser asumidos en su totalidad como confiables por el handler de la excepción.
- El procesador intenta escribir el estado de la tarea a suspender en el *TSS* , y la página que lo contiene no está en memoria, o ha sido marcada Read-Only, o se modificó alguno de los descriptores de página involucrados en la traducción de la dirección lineal del *TSS* quedando algún(os) bit(s) reservados en ”1’, generándose de este modo un fallo de página.

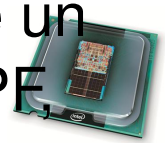
Interrupción Tipo 14: Page Fault #PF

- Cuando el procesador intenta leer la GDT para buscar el *TSS* .
- Cuando el procesador intenta leer el *TSS* de la nueva tarea, y la página está no presente, o tiene modificados los bits reservados de modo que alguno pueda haber quedado en ’1’.
- Cuando el procesador comienza a intentar cargar los descriptores de segmento a partir de los selectores de la nueva tarea.
- Mientras lee el *LDT* de la nueva tarea para verificar los selectores de segmento que se encuentran almacenados en dicha tabla.



Interrupción Tipo 14: Page Fault #PF

- En los dos últimos casos del slide anterior el procesador ya se encuentra en la nueva tarea debido a que ha logrado cargar el contexto de la misma.
- En modo 32 bits para manejar estas excepciones en medio de la conmutación de tareas es recomendable utilizar una Puerta de tarea para manejar la excepción Page Fault. En el modo 64 bits, no son válidas las puertas de tarea. Sin embargo tampoco hay cambio de tareas en forma automática. Por lo tanto, el Page fault se produciría en medio del código que realiza el task switch de modo que si se produjese un Page Fault la instrucción responsable no se habrá completado y el estado del proceso no se verá alterado. Tal vez esto ayude a comprender la preferencia en el modo 64 bits por eliminar la automatización de la conmutación de tareas.
- En Modo 32 bits, en el caso de cambiar el stack en medio de un bloque de código, desde el punto de vista de la excepción #PF, vuelve a ser recomendable utilizar la instrucción LSS.

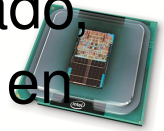


Interrupción Tipo 14: Page Fault #PF

Si en Modo 32 bits ocurre un Page Fault en medio de la secuencia de cambio de stack como la del listado siguiente, se da la siguiente situación:

```
1  new_stack: dd 0xfff ;valor para esp
2   dw 0x20 ;valor para ss
3   ............
4   mov ax,[new_stack + 4]
5   mov  ss,ax ;En esta instrucción se genera un Page Fault
6   mov  esp,dword [new_stack]
```

Tal como indica el comentario de la línea 5 del listado, se produce hipotéticamente una excepción #PF cuando se está cargando el valor en el registro SS, en esta operación el procesador solo inhibe las interrupciones de hardware que ingresan por *INTR* , y las excepciones de debug y single step.
Si el código que realiza el cambio de stack no es código privilegiado, no es un problema, pues el cambio de nivel de privilegio derivará en un stack switch automático.



Interrupción Tipo 14: Page Fault #PF

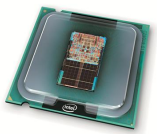
Al retornar al código menos privilegiado desde la Page Fault, el procesador vuelve a ejecutar la instrucción que generó la excepción, es decir, la que modifica el registro SS. En este caso no hay problema. En el caso en que el código que efectúe el cambio de stack sea parte del kernel y produzca el Fallo de Página, no habrá conmutación de stack, ya que no habrá cambio en el nivel de privilegio. Por lo tanto la #PF será atendida con el registro SS apuntando a un stack y el registro ESP apuntando a otro. Una maravilla ¿verdad?. Máxime tratándose del kernel...Para evitar esta situación se recomienda implementarlo mediante el siguiente código:

```
1  new_stack: dd 0xfff ;valor para esp
2   dw 0x20 ;valor para ss
3   ............
4   lss  esp,[new_stack] ;lee ss:esp desde new_stack en una sola
        instrucción
```

Tal vez este sea el motivo que originó que en 64 bits se genere siempre cambio de stack cuando se vectoriza una interrupción o una excepción, así no haya habido cambios en el Nivel de privilegio.

Interrupción Tipo 16: x87 FPU Floating Point Error #MF

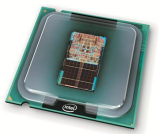
- Esta excepción tipo Fault, indica que la FPU generó un error de punto flotante. Para que se genere esta excepción es necesario que el bit ***CR0.NE*** (Numeric Error) esté seteado y el pin #IGNNE debe estar inactivo. De otro modo ingresa a un modo de emulación de un coprocesador x87 típico de los 80286 y 80386, que tenías la FPU en un chip separado. Esta excepción obedece a seis tipos de errores diferentes de la FPU:
  1. Operación Inválida (#I). Esta a su vez puede ser un Stack Overflow o Underflow, o una operación aritmética inválida
  2. División por cero (#Z)
  3. Operando Denormalizado (#D)
  4. Overflow Numérico (#O)
  5. Underflow Numérico (#U)
  6. Resultado Inexacto (Precisión) (#P)



Interrupción Tipo 16: x87 FPU Floating Point Error #MF

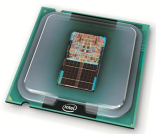
- Por cada una de estas causas hay seis bits en el registro de estados de la FPU, destinados a que el handler de esta excepción pueda identificar uno a uno, y seis registros de máscara en el control register de la FPU que permiten al programador de Sistemas deshabilitar temporariamente o en forma permanente una o mas de las seis causas de esta excepción. La secuencia de esta excepción es la siguiente:
  - Si el bit ***CR0.NE*** = 1, luego de una operación de Punto flotante la FPU setea si corresponde el o los bits del status register.
  - Espera hasta que la FPU ejecute la siguiente instrucción de tipo "waiting"[2], o que ejecute la instrucción WAIT o FWAIT.
  - Si al ejecutar la instrucción encuentra un flag seteado en el status register, envía la señal de error para que la CPU genere la excepción.

[2]Se comportan como la Instrucción WAIT del procesador (esperan que se ejecute la próxima instrucción de la FPU para avisar que existe un error en la anterior). Las instrucciones No Waiting generan la excepción si detectan alguno de los errores señalados en el status register



Interrupción Tipo 16: x87 FPU Floating Point Error #MF

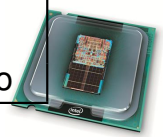
- Deja CS:EIP apuntando a la instrucción que produjo la excepción solo si es una instrucción No Waiting. En la mayoría de los casos, estos registros apuntan a la instrucción siguiente de la FPU que se encuentra en el flujo de instrucciones. La dirección de la instrucción responsable de la excepción se encuentra en el registro Instruction Pointer Register de la FPU.
- No almacena código de error en la pila. Para el handler la información son los bits descriptos en el ítem anterior.
- Por lo general si bien la instrucción responsable de la Fault ya se ejecutó, la FPU guarda suficiente información como para recuperar en el handler de la excepción la condición del programa.



Interrupción Tipo 17: Alignment Check Exception #AC

- Esta excepción tipo Fault, indica que el procesador detectó un operando no alineado en memoria si el chequeo de alineación está activado. Los chequeos de alineación se controlan durante operaciones de acceso a stacks y datos, nunca durante operaciones de opcode fetch. La tabla siguiente, muestra que condición debe reunir la dirección de la variable para cada tipo de variable reconocida.

Tipo de dato	Dirección alineada a:
Word	2
Doble Word	4
Punto Flotante Simple Precisión (32 bits)	4
Punto Flotante Doble Precisión (64 bits)	8
Punto Flotante Doble Precisión Extendida (80 bits)	8
Quad Word	8
Double Quad Word	16
Selector de Segmento	2
Puntero Far de 32 bits	2
Puntero Far de 48 bits	4
Puntero de 32 bits	4
Contenido de GDTR, IDTR, LDTR, TR	4
Área de resguardo para FSTENV/FLDENV	2 o 4 depende del tamaño del operando
Área de resguardo para FSAVE/FRSTOR	2 o 4 depende del tamaño del operando
Bit Strings	2 o 4 depende del atributo tamaño del operando

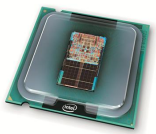


Interrupción Tipo 17: Alignment Check Exception #AC

- Cuando se programa el handler de esta excepción debe tenerse presente que #AC se produce para datos cuyos requerimientos de alineación sean 2, 4, u 8 bytes. Para los de 128 bits se genera una excepción #GP cuando dirección base no es múltiplo de 16 bytes.
- También debe tenerse en cuenta que esta excepción se generará siempre que se la habilite activando (seteando) el bit ***CR0.AM***, y ***EFLAGS.AC***, y además se generará solamente cuando el código que viole los requerimientos de alineación tenga ***CPL*** = 11.
- Las instrucciones FXSAVE/XSAVE y FXRSTOR/XRSTOR resguardan los contextos de la FPU y de los registros XMM en bloques de memoria de 512 bytes. EL requerimiento de alineación para estas estructuras de datos es de 16 bytes. Por lo tanto si se encuentra habilitado el chequeo de alineación del procesador, y este requerimiento no se cumple, se generará una excepción #GP, de acuerdo con lo explicado anteriormente.

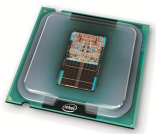
Interrupción Tipo 17: Alignment Check Exception #AC

- Las instrucciones SIMD MOVDQU, MOVUPS, y MOVUPD realizan cargas y almacenamientos no alineados de 128 bits. LDDQU carga 128 bis no alineados en el registro XMM destino. Como fueron pensadas para acceder a datos no alineados, no generarán #GP cuando accedan a variables de 128 bits cuya dirección no sea múltiplo de 16 bytes.
- Deja CS:EIP apuntando a la instrucción que produjo la excepción.
- Almacena código de error en la pila. Para el handler la información son los bits descriptos en el ítem anterior.
- La instrucción responsable del Fault no se termina de ejecutar de modo que no se afecta el estado del programa en curso.



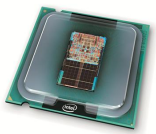
Interrupción Tipo 18: Machine Check Exception #MC

- Esta excepción tipo Abort, indica que el procesador detectó un error interno de hardware, o un error en el bus, o un controlador externo detectó un error en el bus. Se implementó a partir del procesador Pentium, y ha ido evolucionando a partir de este procesador, convirtiéndose en una excepción modelo dependiente. Se recomienda chequear su disponibilidad con la instrucción CPUID.
- En el caso de controladores externos que reportan error en el bus lo hacen activando señales en terminales dedicados del procesador. BUSCHK# era el terminal del pentium. Desde la familia P6 hasta el presente se dispone de BINIT# MCERR#.
- El contenido de CS:EIP no siempre está asociado a la excepción ni a sus causas. Depende de cada Microarquitectura su implementación.



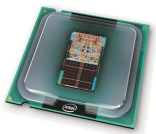
Interrupción Tipo 18: Machine Check Exception #MC

- No almacena código de error en la pila. La información se almacena en los Machine-Check Model Specific Registers (MSRs) del procesador.
- Esta excepción se habilita seteando el bit ***CR4.MCE***. A pesar de que se almacena información asociada a la excepción en los MSRs, por lo general esta excepción no puede recuperar el estado de ejecución ya que se origina ante errores de hardware. Si no se habilita el bit ***CR4.MCE***, ante las miams causas que se explicaron, el procesador en lugar de generar una excepción entra en estado Shutdown.



Interrupción Tipo 19: SIMD Floating Point Exception #XM

- Esta excepción tipo Fault, indica que el procesador detectó un error de punto flotante en alguna instrucción SIMD SSE/SSE2/SSE3. Como consecuencia de la misma se setean en el registro MXCSR, el o los flags que correspondan. Esta excepción obedece a los mismos seis tipos de errores diferentes vistos en la excepción 16 de la FPU:
  1. Operación Inválida (#I).
  2. División por cero (#Z)
  3. Operando Denormalizado (#D)
  4. Overflow Numérico (#O)
  5. Underflow Numérico (#U)
  6. Resultado Inexacto (Precisión) (#P)

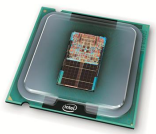


Interrupción Tipo 19: SIMD Floating Point Exception #XM

- Las tres primeras se detectan antes de completarse la instrucción. Las tres últimas se detectan una vez obtenido el resultado.
- Cada causa posee un flag y una máscara en el registro MXCSR. Si una causa de excepción tiene activo el bit de máscara apropiado, el procesador resuelve la excepción de acuerdo a su esquema predefinido. Si no está enmascarada en el MXCSR, genera la excepción y delega en el handler su resolución.
- El flag ***CR4.OSXMMEXCPT***, permite al sistema operativo definir si va a soportar esta excepción o no.
- Para cada causa de excepción sin máscara activa, se generará ésta excepción. Si el ***CR4.OSXMMEXCPT*** está inactivo, significa que el Sistema Operativo no soporta esta excepción. En esta situación si desactivamos una o mas máscaras, al darse alguna de los errores no enmascarados se generará la excepción Invalid Opcode #UD en lugar de esta excepción.

Interrupción Tipo 19: SIMD Floating Point Exception #XM

- Cuando las instrucciones SSE/SSE2/SSE3, trabajan con datos de punto flotante empaquetados, puede producirse mas de un error en la instrucción, y además estos errores no tienen porque ser del mismo tipo. Es decir, un sub-operando puede dar Operación Inválida, mientras que otro puede generar una División por Cero. En rigor si trabajamos con cuatro números de simple precisión empaquetados puede existir un error por cada uno y de características diferentes con lo cual se podrían activar en teoría hasta 4 flags de error diferentes en el registro MXCSR. Sin embargo cuando se producen dos o mas errores el procesador reporta el mas significativo de acuerdo con la tabla del slide siguiente, e ignora al resto.

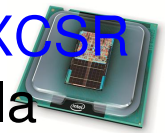


Interrupción Tipo 19: SIMD Floating Point Exception #XM

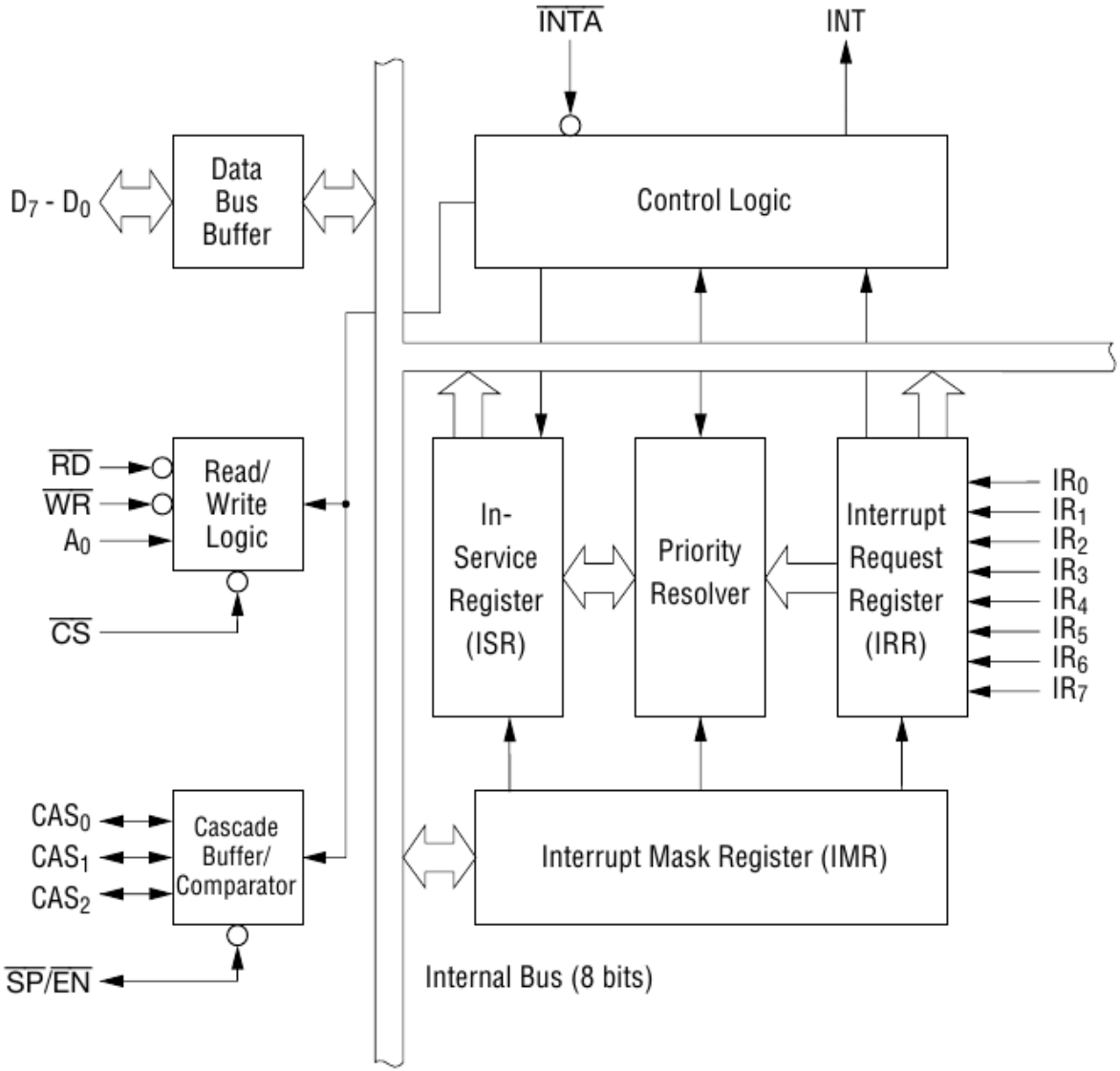
Prioridad	Descripción
1 (Mas alta)	Operación inválida debida al uso de un operando Signaled Not a Number (SNaN), o de un operando Not a Number (NaN) en caso de operaciones Máximo, Mínimo y algunas de Comparación y Conversión.
2	Operando QNaN (Quaetly Not a Number). A pesar que QNaN no generan excepción, su precedencia es mayor que el resto. Un QNaN dividido por cero no genera #Z sino otro QNaN.
3	Cualquier Operación Inválida excepto las anteriores y División por cero.
4	Excepción de Operando Denormalizado
5	Overflow y Underflow, posiblemente con Resultado Inexacto.
6 (Mas baja)	Resultado Inexacto.

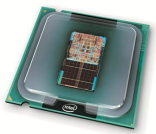
Las prioridades 3 a 5, cuando están enmascaradas dejan paso a las de menor prioridad en caso que se detecten simultáneamente.

- Deja CS:EIP apuntando a la instrucción que produjo la excepción.
- No almacena código de error en la pila.Usa el registro MXCSR.
- La instrucción responsable de la Fault no se terminó de ejecutar, de modo que a menos que esté enmascarada la causa, el registro MXCSR tiene suficiente información como para recuperar en el handler de la excepción la condición del programa.

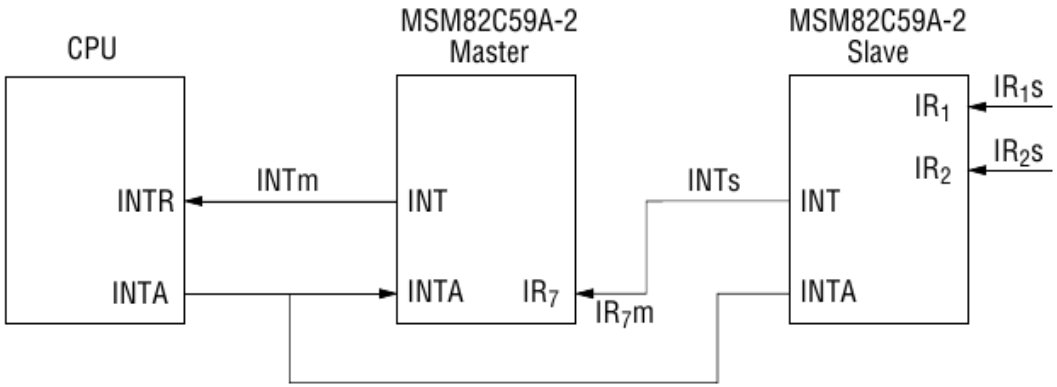


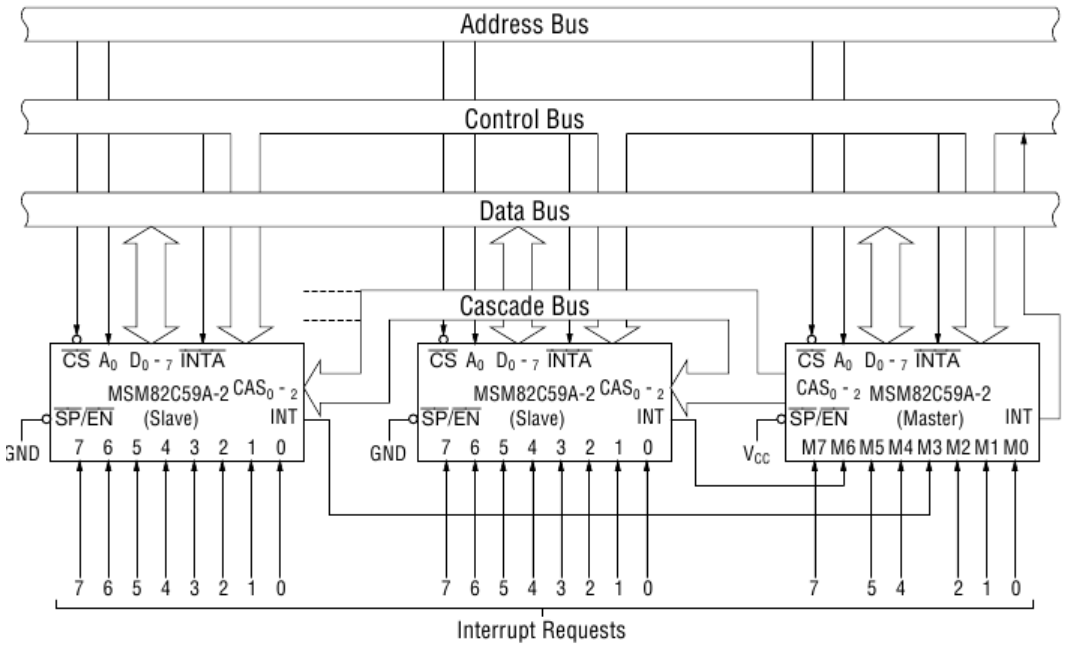
El PIC: Programable Interrupt Controller

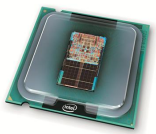




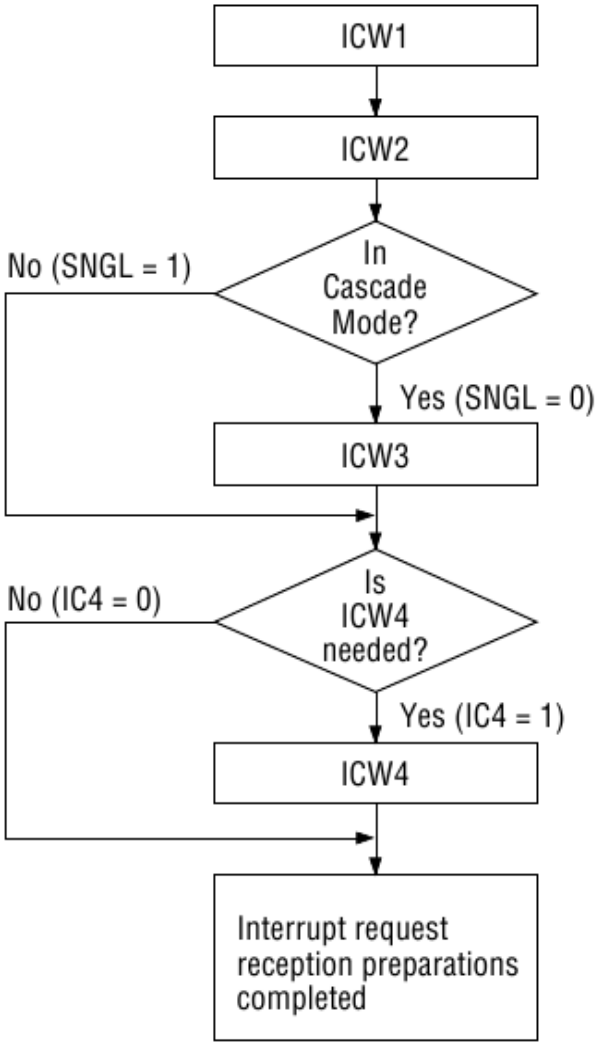
Conexionado

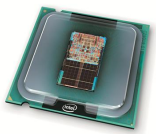






Secuencia de Inicialización





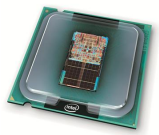
Inicialization Control Words

	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
ICW1	0	A_7	A_6	A_5	1	LTIM	ADI	SNGL	IC4

- IC4: 1: ICW4 required; 0: ICW4 not required
- SNGL: 1: Single; 0: Cascade
- ADI: CALL address interval; 1: Interval = 4; 0: Interval = 8
- LTIM: 1: Level triggered mode; 0: Edge triggered mode
- A_5 thru A_7: Interrupt vector address (Valid only in 85 mode)

	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
ICW2	1	A_{15}/T_7	A_{14}/T_6	A_{13}/T_5	A_{12}/T_4	A_{11}/T_3	A_{10}	A_9	A_8

- A_8 thru A_{15}: Interrupt vector address (85 mode)
- T_3 thru T_7: Interrupt vector address (86 mode)



Inicialization Control Words

ICW3 (Master)

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

1: IR input holds slave
0: IR input does not hold slave

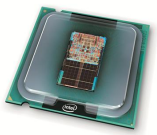
ICW3 (Slave)

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	ID_2	ID_1	ID_0

Slave ID

0	1	2	3	4	5	6	7
0	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1
0	0	0	0	1	1	1	1

NOTE: Slave ID indicates the IR input of the corresponding master.



Inicialization Control Words

	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
ICW4	1	0	0	0	SFNM	BUF	M/S	AEOI	µPM

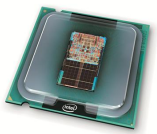
µPM: 1: 86 mode, 0: 85 mode

AEOI: 1: Automatic EOI mode, 0: Normal EOI mode

0	×	Non-buffered mode
1	0	Buffered mode (slave)
1	1	Buffered mode (master)

NOTE: × den 0 tes not specified

SFNM: 1: Special fully nested mode, 0: Not special fully nested mode



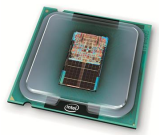
Inicialization Control Words

	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
ICW1	0	A_7	A_6	A_5	1	LTIM	ADI	SNGL	IC4

- IC4: 1: ICW4 required; 0: ICW4 not required
- SNGL: 1: Single; 0: Cascade
- ADI: CALL address interval; 1: Interval = 4; 0: Interval = 8
- LTIM: 1: Level triggered mode; 0: Edge triggered mode
- A_5 thru A_7: Interrupt vector address (Valid only in 85 mode)

	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
ICW2	1	A_{15}/T_7	A_{14}/T_6	A_{13}/T_5	A_{12}/T_4	A_{11}/T_3	A_{10}	A_9	A_8

- A_8 thru A_{15}: Interrupt vector address (85 mode)
- T_3 thru T_7: Interrupt vector address (86 mode)



Operational Control Words

	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
OCW1	1	M7	M6	M5	M4	M3	M2	M1	M0

Intterupt Mask
1: Mask set
0: Mask reset

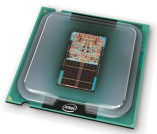
	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
OCW2	0	R	SL	EOI	0	0	L2	L1	L0

Active IR Level

0	1	2	3	4	5	6	7
0	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1
0	0	0	0	1	1	1	1

0	0	1	Non-specific EOI command	End of interrupt
0	1	1	Specific EOI command (NOTE)	
1	0	1	Rotate on non-specific EOI command	Automatic rotation
1	0	0	Rotate in automatic EOI mode (SET)	
0	0	0	Rotate in automatic EOI mode (Clear)	
1	1	1	Rotate on specific EOI command (NOTE)	Specific rotation
1	1	0	Set priority comand (NOTE)	
0	1	0	No operation	

NOTE: L0 thru L2 used



Operational Control Words

	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
OCW3	0	0	ESMM	SMM	0	1	P	RR	RIS

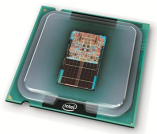
Read Register Command

0	1	0	1
0	0	1	1
No operation		Read IRR by next $\overline{RD}$	Read ISR by next $\overline{RD}$

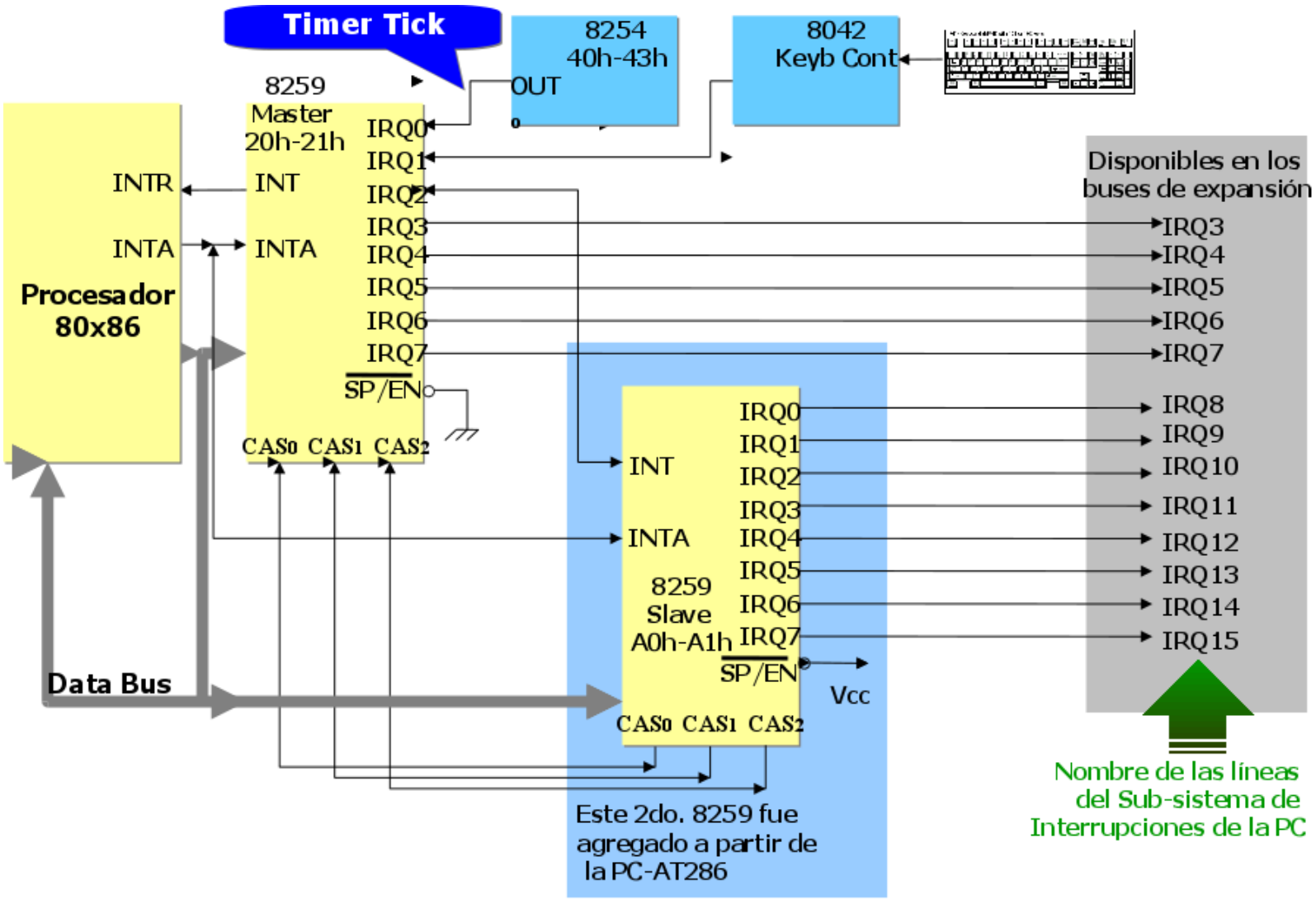
Polling
1: Poll command
0: No-poll command

Special Mask Mode

0	1	0	1
0	0	1	1
No operation		Reset special mask mode	Set special mask mode

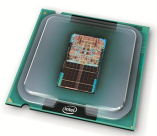


Conexionado de los dos PIC en una PC



Asignación de las líneas IRQ en una PC

IRQ	Tipo	Descripción
IRQ0	08h	Timer tick (T=55 mseg.)
IRQ1	09h	Teclado
IRQ2	0Ah	INT desde 8259A esclavo
IRQ8	70h	Servicio de reloj en tiempo real.
IRQ9	71h	Redireccionamiento por soft. a IRQ2
IRQ10	72h	Reservada
IRQ11	73h	Reservada
IRQ12	74h	Reservada.
IRQ13	75h	Coprocesador numérico.
IRQ14	76h	Controlador de disco rígido.
IRQ15	77h	Reservada.
IRQ3	0Bh	COM2
IRQ4	0Ch	COM1
IRQ5	0Dh	LPT2
IRQ6	0Eh	Controlador de disco flexible (Floppy)
IRQ7	0Fh	LPT1



Mucho mas que un controlador de Interrupciones...

- En 1990 Intel lanza el procesador Pentium, que introduce entre varias novedades el Advanced Programmable Interrupt Controller (APIC) cuya principal función, consiste en dar soporte a Multi Procesamiento Simétrico (SMP).
- El APIC tiene dos componentes de Hardware:
  1. El APIC Local que está incluido en el chip que contiene el Core del procesador (actualmente hay que hablar de un APIC por cada Core integrado en el Chip),
  2. El IO/APIC, que es el concentrador externo de Interrupciones de Hardware que reside en el chipset, y que interactúa con cada APIC local, para entre otras cosas enviar a cada core una interrupción proveniente de un dispositivo de hardware, de acuerdo a diferentes criterios.
- El APIC requiere ser habilitado explícitamente. De otro modo el procesador no lo usa y requiere una lógica externa del tipo 8259A.

...Tanto que lo dejamos para tratar específicamente

- El APIC local permite manejar lo que Intel denomina Interrupciones locales.
- Son las que se generan en los terminales del chip por hardware, y en algunas fuentes adicionales que provee el APIC.
- Interactuando con el IO/APIC, conecta los terminales externos (*LINT0* o *LINT1* según se lo configure) con la señal ***INTR*** del core (que ahora no está disponible hacia el exterior).
- Si el APIC no está habilitado, *LINT0* corresponde a ***INTR***, y *LINT1* a ***NMI***.
- Se trata de un dispositivo cuya funcionalidad excede el manejo de interrupciones, por lo tanto se lo tratará con mas detalle al abordar el tema SMP.