# Unidad 12 - Interrupciones y Excepciones

## 1. Interrupciones

Las interrupciones son mecanismos que rompen la secuencia de ejecución del programa, para atender eventos no previstoss, asíncronos y externos al mismo. Son generadas por los dispositivos de E/S. A nivel microarquitectura, se implementa agregando una etapa de detección de interrupciones al final del ciclo de instrucción.

Las interrupciones se clasifican según el **mecanismo de activación**:

- **Excepciones**: son síncronas, y se producen por **errores durante la ejecución de una instrucción** o **fallas de hardware**. Se las denomina *precisas* si se conoce su cause o *imprecisas* en caso contrario, la cual puede determinarse analizando la traza del programa en curso. Estas excepciones suelen ser causadas por operaciones no permitidas (división por cero, stack overflow, etc.) o falta de datos para ejecutar instrucciones (violación de propiedades al acceder a memoria, etc.). ver más
- **Interrupciones por Hardware** (Iterrupt Request - IRQ): asíncronas a la ejecución del programa, se proden por eventos externos al mismo. Son externas al procesador y están relacionadas con los dispositivos de E/S. Estas
- **Interrupciones por Software** (TRAP): son producidas por el programa en ejecución (síncronas), y para generarlas existen distintas instrucciones que permiten al programador producirlas (`ecall` por ejemplo).

Las interrupciones también pueden clasificarse según su **mecanismo de gestión**:

- Interrupciones **Enmascarables**: interrupción de hardware que puede ser ignorada o bloqueada por medio de *señales de enmascaramiento* en registros especiales.
- Interrupciones **No-Enmascarables** (NMI): no tienen asociado mecanismos de enmascaramiento. Son utilizadas por tareas de alta prioridad como temporizadores o wathdog timers.
- Interrupciones **Interprocesador** (IPI): son un tipo especial generadas por el procesador para interrumpir otros procesadores, utilizadas (obviamente) en sistema multiprocesador.

### 1.1. Identificación de la Fuente

Existen distintas formas de identificar la **fuente de la interrupción**:

- **Consulta**: el procesador comprueba todos los dispositivos para buscar el que solicitó la interrupción. El orden en que se consulta define la prioridad de las interrupciones. Para implementarse, se incroporan registros que guardan banderas de petición (Interrupt Flag).
  - Ventajas: poco hardware, flexible y establece una prioridad en los dispositivos.
  - Desventajas: lento ya que tiene que comprobar todos los dispositivos.
- **Vectorizada**: se asocia una lista de rutinas de interrupción con una lista de solicitudes de interrupción a través de vectores de interrupción almacenados de una tabla (llamada tabla de vectores de interrupción). Al ser afectada por una interrupción, la CPU se fija en la tabla de vectores de interrupción para encontrar dirección de inicio de la rutina de interrupción asociada y transferir el control a esta.
  - Ventajas: procesa múltiples interrupciones con poco hardware.
  - Desvetajas: es lenta y hace mucho uso de los buses.
- **Hardware**: se utiliza un circuito dedicado a gestionar las interrupciones, controlando las señales de forma independiente para cada periférico. Para esto el controlador dispone de registros especiales para definir prioridades y enmascaramiento para cada solicitud.

- Ventajas: eficiente, veloz y flexible.
- Desventajas: caro.

## 1.2. Priorización de Interrupciones

Es necesario un mecanismo para tratar primero las interrupciones más urgentes. Las opciones son:

- Interrupciones **Simultáneas**: se gestionan varias interrupciones a la vez:
  - Un controlador dedicado combina varias fuentes de interrupción a la vez que asigna prioridad a las *salidas* de interrupción.
  - Un método distribuido: se puede implementar **por consulta**, verificando todas las posibles fuentes hasta encontrar la correcta, o por **Daisy Chain**, donde la prioridad se determina por la cercania del dispositivo al procesador.
- Interrupciones **Anidadas**: permite una interrupción sobre la misma rutina de atención de otra interrupción, excepto sobre la rutina de mayor prioridad.
- **Inhibición**: se pueden desactivar las interrupciones al momento de ejecutar rutinas críticas, haciendo uso de la enmascaración.

## 1.3. Direccionamiento de Rutinas

Para determinar la dirección de la rutina de atención de una interrupción existen dos opciones:

- Direcciones **Fíjas**: se hallan físicamente cableadas, por lo que no pueden ser cambiadas.
- Direcciones **Variables**: el dispositivo debe dar información acerca de la localización de su rutina de servicio o de la localización de la dirección de esta
  - Direccionamiento **Absoluto**: el dispositivo (o interfaz) se encarga de conocer y enviar la dirección de la rutina.
  - Direccionamiento **Indirecto**: utiliza la tabla de vectores de interrupción.

## 1.4. Gestión de Periféricos por Interrupciones

El procesador suspende sus actividades actuales, guardando su estado, y ejecuta una **rutina** de servicio de la interrupción. Una vez finalizada, reanuda su ejecución normal.
Esto permite responder a eventos síncronos y asíncronos, especificando qué rutinas de atención requiere cada evento.

- **Ventajas**: el procesador puede ejecutar otro código mientras espera un evento.
- **Desventajas**: tiempo de ejecución y latencia no deterministas y dificultad al gestionar interrupciones anidadas.

Al darse una interrupción, el procesador:

1. Completa, si puede, la instrucción en curso.
2. Determina la fuente de la interrupción.
3. Determina si atender, o no, la interrupción.
4. Si se atiende, guarda el estado del programa en ejecución.
5. Determina la dirección de la rutina de atención y la ejecuta.
6. Una vez finalizada la ejecución de la rutina, reasume la ejecución del programa.

# 2. Excepciones

Las **excepciones** son eventos **síncronos** que rompen la scuencia de ejecución del programa para **atender errores** que ocurren durante la ejecución de una instrucción o fallas de hardware.
Se clasifican en *precisos* (cuando se conoce su causa) e *imprecisos* (cuando no), en cuyo caso la cause se determina analizando la traza del programa en curso.
Las causas de una excepción pueden ser

- Tareas de **depuración** del programa: breakpoint, trace, etc.
- Realizar **operaciones no permitidas**: división por cero, códigos desconocidos.
- **Accesos ilegales** a memoria: stack overflow, acceso a posiciones no permitidas, violación de privilegios.
- **Falta de Datos** para ejecutar instrucciones: tabla de segmentos incompleta, segmento no presente en memoria.

Para detectar la causa de las excepciones, se agregan a nivel microarquitectura etapas de detección de causa de excepción a lo largo del ciclo de instrucción, que chequean al si se produce alguna de las condiciones de excepción. En caso de detectarse una ocurrencia de excepción, el procesador suspende la ejecución de las instrucciones que se están ejecutando y carga el contador de programa con la dirección de la rutina de atención de la excepción.
La gestión de excepciones **es muy similar a la gestión de interrupciones**.