Universidad Nacional Autónoma de México
Facultad de Ingeniería

Microprocesadores y Microcontroladores

Tema 7

Interrupciones y resets

M. I. Christo Aldair Lara Tenorio

2025-1

Tabla de contenidos

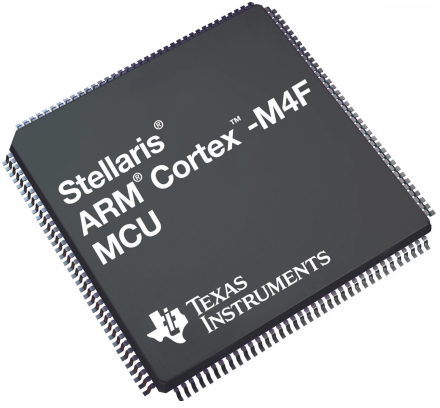
Objetivo del tema

Objetivo general:
El alumno programará las interrupciones y reinicios.

Contenido:

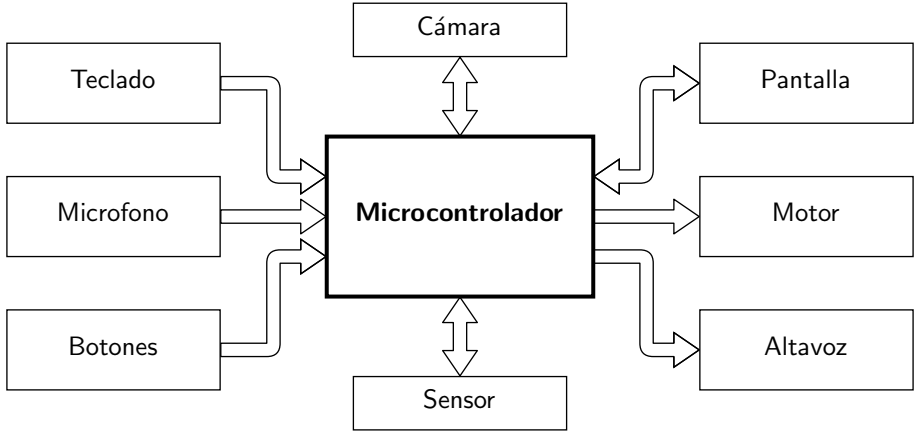
7.1. Conceptos fundamentales de las interrupciones.

7.2. Reinicios.



Interfaz

Conjunto de hardware y software que permite la comunicación (intercambio de información) entre un microcontrolador y otros dispositivos o sistemas.



Sincronización con señales externas

Se consigue a través de métodos que coordinan las operaciones internas del microcontrolador con eventos o señales provenientes del entorno exterior, permitiendo que estos respondan o trabajen en conjunto con otros sistemas.

Métodos de sincronización:

- Ciclo ciego.
  Retardo por software estimado para que se completen las señales de entrada/salida.
- Sondeo en espera (polling).
  Monitoreo continuo de un periférico hasta que ocurra el evento esperado.
- Interrupción.
  Evento que genera la suspensión del código normal para ejecutar un evento que requiere de atención inmediata.

Ciclo ciego

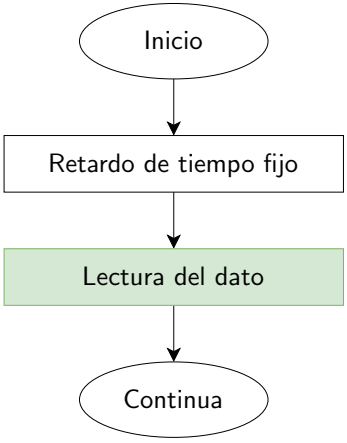
Método en donde se genera un retardo por software de duración fija, con el cual se asume que las señales de entrada/salida estarán listas antes de que el periodo de espera se termine.

Ventajas

- Simplicidad.

Limitaciones

- Baja eficiencia.
- Errores de sincronía.



Sondeo en espera (polling)

Método de monitoreo continuo a través de un software cíclico para verificar el estado de las señales de entrada/salida hasta detectar la condición esperada.

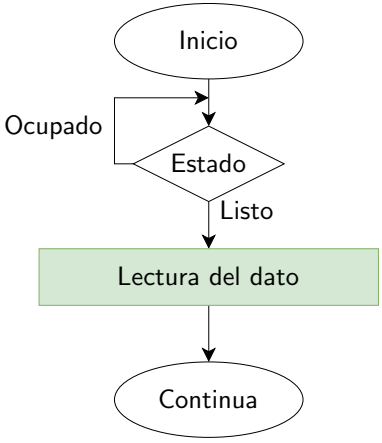
```
while((NVIC_ST_CTRL_R & NVIC_ST_CTRL_COUNT) == 0){};
```

Ventajas

- Simplicidad.
- Control directo de lo que realiza el software.

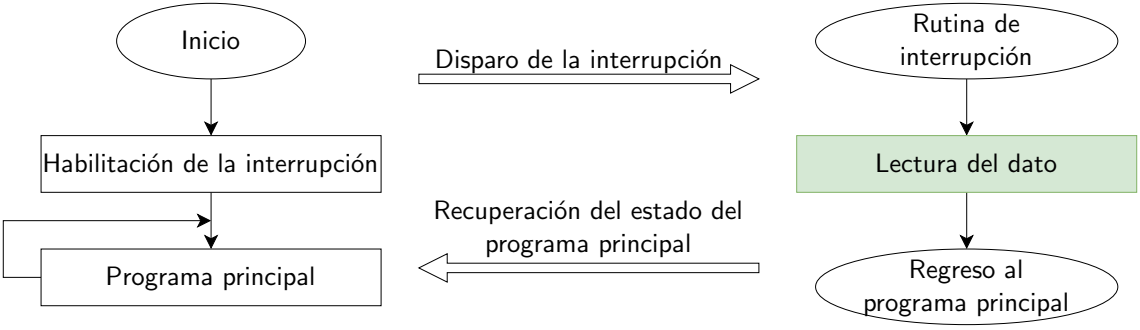
Limitaciones

- Baja eficiencia.
- Uso ineficiente de recursos.
- No permite realizar procesos en multitarea.



Interrupciones

Suspensión temporal de la ejecución normal de un programa, derivada de un evento externo de hardware o software (disparo), para atender inmediatamente la condición esperada.



Interrupciones

Suspensión temporal de la ejecución normal de un programa, derivada de un evento externo de hardware o software (disparo), para atender inmediatamente la condición esperada.

Conceptos generales:

- Evento/disparo de interrupción (trigger).
  Cambio de estado de una señal (evento de hardware interno/externo) que genera el disparo de la interrupción.
- Rutina de servicio de interrupción (ISR).
  Sección de código predefinida en donde se ejecutan las instrucciones relacionadas al evento de interrupción.

Ventajas

- Alta eficiencia.
- Asegura la respuesta rápida a eventos.
- Ahorro de energía (en combinación con modos de bajo consumo).

Desventajas

- Complejidad.
- Gestión de prioridades.
- Latencia.

Interrupciones

Vector de interrupciones

- Índice de las direcciones de memoria que contienen los puntos de acceso a las rutinas de servicio de interrupción (ISR).
- Cuando ocurre un evento de interrupción, el microcontrolador utiliza al vector de interrupciones para determinar qué ISR se debe de ejecutar y a qué dirección de memoria se debe de saltar.

Prioridad

- Indicativo del orden en que se deben de ejecutar las ISR cuando suceden múltiples eventos de interrupción al mismo tiempo.
- La prioridad de una interrupción se establece con el orden en el que esta ocupa en el vector de interrupciones.
- Si ocurre un evento que dispare a una interrupción con mayor prioridad a la que se está operando, esta se suspenderá para atender a la de mayor prioridad (interrupciones concatenadas).

El vector de interrupciones del `TIVA TM4C1294NCPDT` está documentado en la *Tabla 2* del manual de usuario (pp. 116).

Proceso de ejecución de una interrupción

1. Evento de interrupción.
   → Evento externo o interno de hardware (cambio de estado de una señal) que dispara la interrupción.

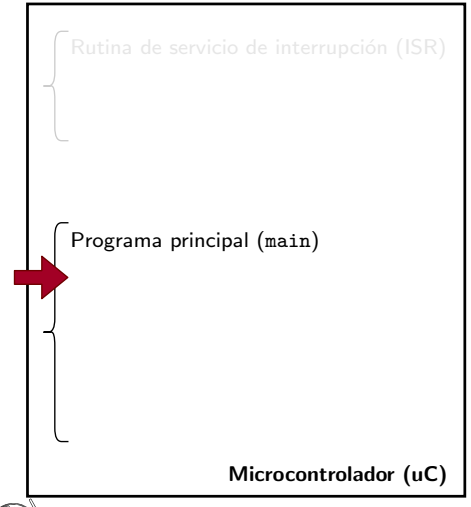
2. Suspensión del programa principal y almacenamiento del estado presente en el microcontrolador.
   → El microcontrolador "detiene" el código que estaba realizando y almacena el estado actual de ejecución (`PC` y registros), de modo que tenga la información suficiente para reanudar sus operaciones al finalizar la ejecución de la interrupción.

3. Ejecución de la rutina de servicio de interrupción (ISR).
   → Salto del microcontrolador a la dirección de memoria en donde inicia la ISR y comienza su ejecución.

4. Restauración del estado en el que se encontraba el microcontrolador, previo a la ejecución de la ISR.
   → Al finalizar la ejecución de la ISR, se restaura el estado del microcontrolador almacenado previamente.

5. Reanudación del programa principal.
   → El microcontrolador reanuda el programa principal en el estado en el que se encontraba antes del evento de interrupción.

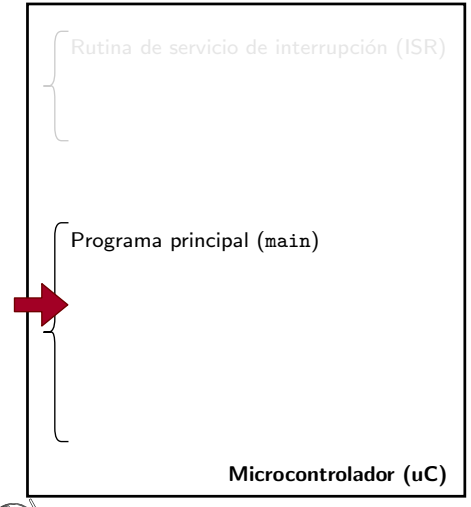
Proceso de ejecución de una interrupción



Evento de hardware (externo o interno)

1. Evento de interrupción.
2. Suspensión del programa principal y almacenamiento del estado presente del uC.
3. Ejecución de la rutina de servicio de interrupción (ISR).
4. Restauración del estado en el que se encontraba el uC, previo a la ejecución de la ISR.
5. Reanudación del programa principal.

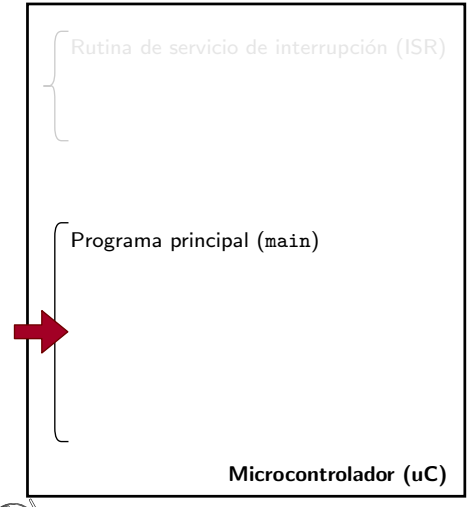
Proceso de ejecución de una interrupción



Evento de hardware (externo o interno)

1. Evento de interrupción.
2. Suspensión del programa principal y almacenamiento del estado presente del uC.
3. Ejecución de la rutina de servicio de interrupción (ISR).
4. Restauración del estado en el que se encontraba el uC, previo a la ejecución de la ISR.
5. Reanudación del programa principal.

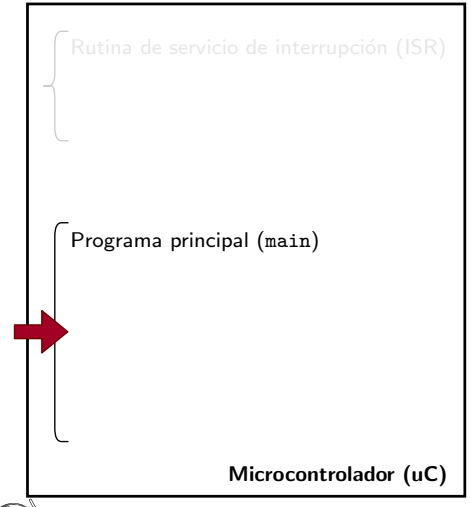
Proceso de ejecución de una interrupción



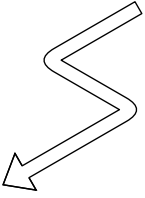
Evento de hardware (externo o interno)

1. Evento de interrupción.
2. Suspensión del programa principal y almacenamiento del estado presente del uC.
3. Ejecución de la rutina de servicio de interrupción (ISR).
4. Restauración del estado en el que se encontraba el uC, previo a la ejecución de la ISR.
5. Reanudación del programa principal.

Proceso de ejecución de una interrupción

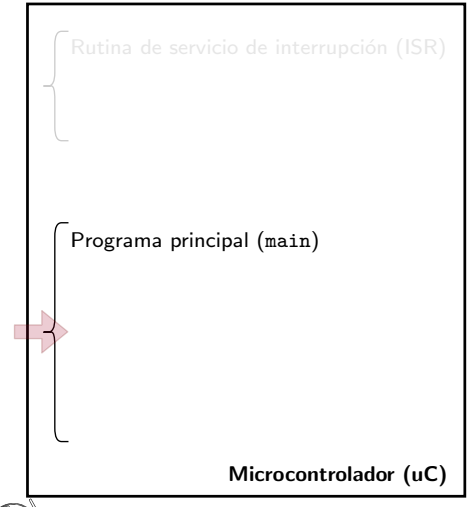


Evento de hardware
(externo o interno)



1. Evento de interrupción.
2. Suspensión del programa principal y almacenamiento del estado presente del uC.
3. Ejecución de la rutina de servicio de interrupción (ISR).
4. Restauración del estado en el que se encontraba el uC, previo a la ejecución de la ISR.
5. Reanudación del programa principal.

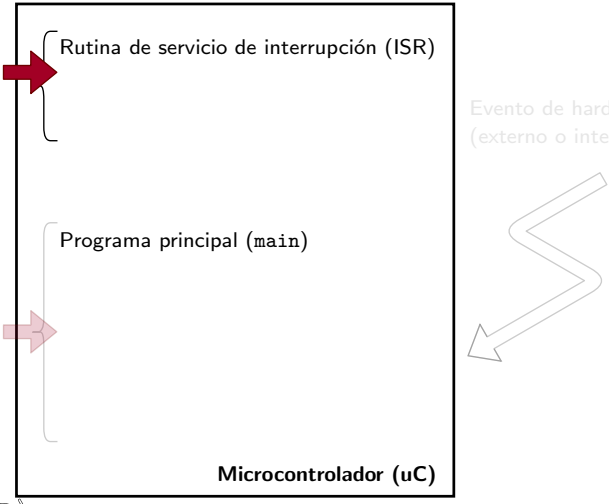
Proceso de ejecución de una interrupción



Evento de hardware (externo o interno)

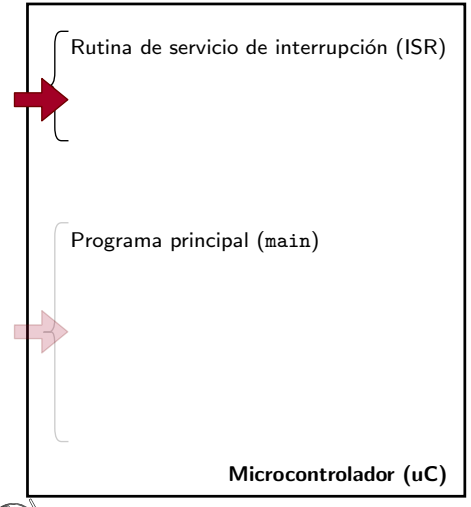
1. Evento de interrupción.
2. Suspensión del programa principal y almacenamiento del estado presente del uC.
3. Ejecución de la rutina de servicio de interrupción (ISR).
4. Restauración del estado en el que se encontraba el uC, previo a la ejecución de la ISR.
5. Reanudación del programa principal.

Proceso de ejecución de una interrupción



1. Evento de interrupción.
2. Suspensión del programa principal y almacenamiento del estado presente del uC.
3. Ejecución de la rutina de servicio de interrupción (ISR).
4. Restauración del estado en el que se encontraba el uC, previo a la ejecución de la ISR.
5. Reanudación del programa principal.

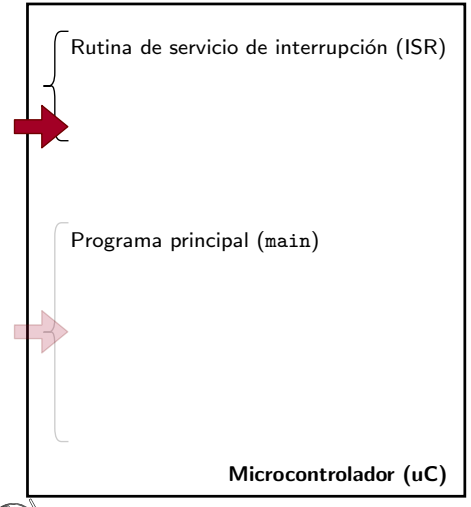
Proceso de ejecución de una interrupción



Evento de hardware (externo o interno)

1. Evento de interrupción.
2. Suspensión del programa principal y almacenamiento del estado presente del uC.
3. Ejecución de la rutina de servicio de interrupción (ISR).
4. Restauración del estado en el que se encontraba el uC, previo a la ejecución de la ISR.
5. Reanudación del programa principal.

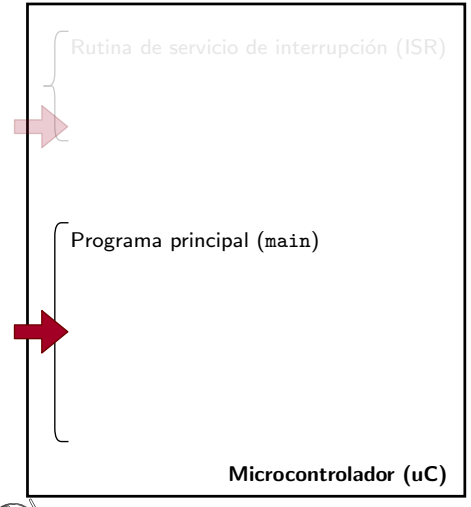
Proceso de ejecución de una interrupción



Evento de hardware (externo o interno)

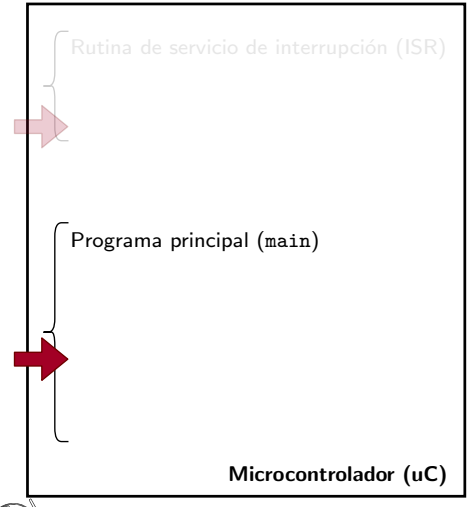
1. Evento de interrupción.
2. Suspensión del programa principal y almacenamiento del estado presente del uC.
3. Ejecución de la rutina de servicio de interrupción (ISR).
4. Restauración del estado en el que se encontraba el uC, previo a la ejecución de la ISR.
5. Reanudación del programa principal.

Proceso de ejecución de una interrupción



1. Evento de interrupción.
2. Suspensión del programa principal y almacenamiento del estado presente del uC.
3. Ejecución de la rutina de servicio de interrupción (ISR).
4. Restauración del estado en el que se encontraba el uC, previo a la ejecución de la ISR.
5. Reanudación del programa principal.

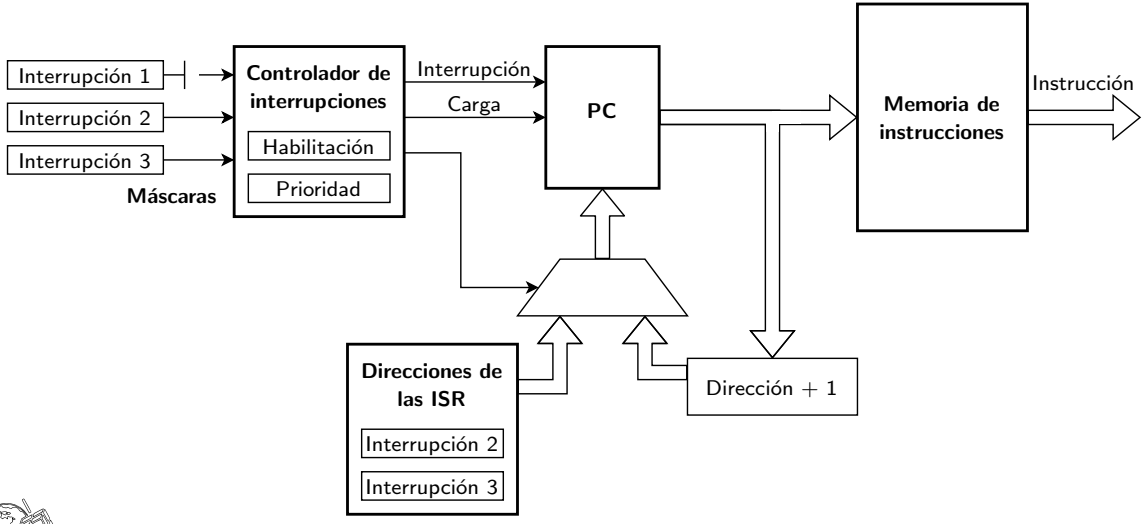
Proceso de ejecución de una interrupción



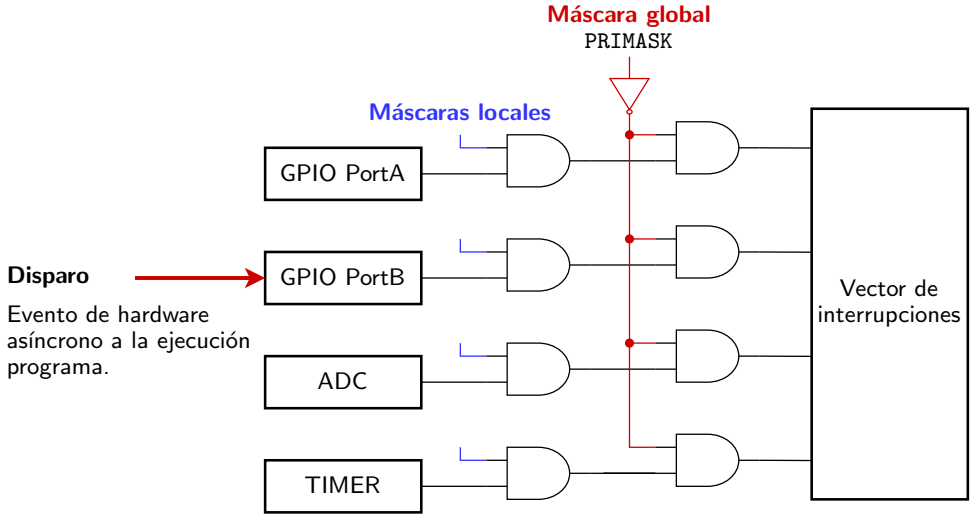
Evento de hardware (externo o interno)

1. Evento de interrupción.
2. Suspensión del programa principal y almacenamiento del estado presente del uC.
3. Ejecución de la rutina de servicio de interrupción (ISR).
4. Restauración del estado en el que se encontraba el uC, previo a la ejecución de la ISR.
5. Reanudación del programa principal.

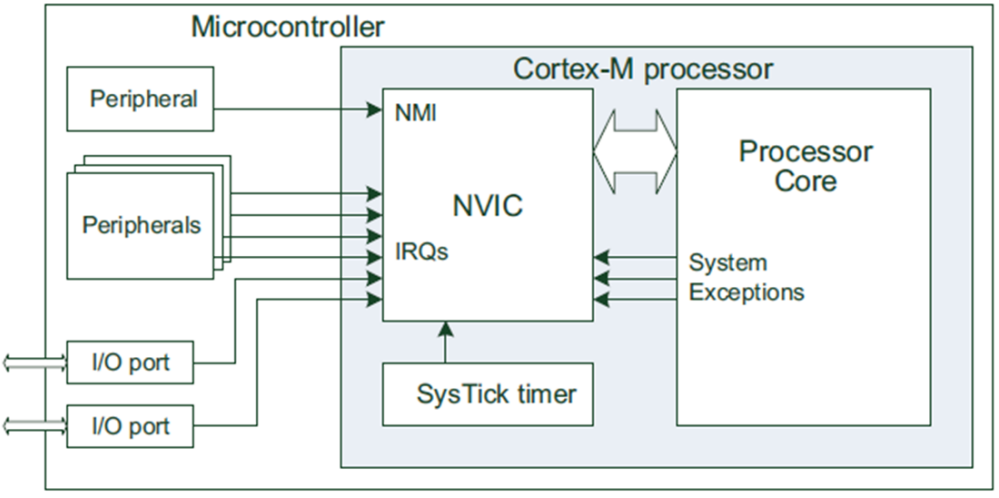
Interrupciones en los procesadores



Máscaras


Máscara global
PRIMASK
Máscaras locales
GPIO PortA
GPIO PortB
ADC
TIMER
Vector de interrupciones
Disparo
Evento de hardware asíncrono a la ejecución programa.

Control de interrupciones en Cortex-M4



Excepciones

En cuanto a concepto, son exactamente lo mismo que una interrupción.

Sin embargo, su función y forma de disparo tienen algunas diferencias:

- Manejan errores del sistema o condiciones inesperadas durante la ejecución de un programa.
- Son provocadas por eventos internos del microcontrolador (instrucciones invalidas, acceso a memoria reservada, desbordamientos del stack, etc.).
- No pueden ser enmascaradas.
- Tienen un tratamiento especial y la máxima prioridad, incluso pueden generar el reinicio del sistema.

Algunos tipos de excepciones en el Cortex-M4 son:

- Reset $\rightarrow$ Excepción asíncrona con el máximo nivel de prioridad.
- Interrupción no enmascarables (NMI) $\rightarrow$ Excepción asíncrona con el siguiente nivel de prioridad al reset, utilizada en situaciones que requieren atención inmediata del procesador, como errores de hardware.

Suspensión del estado del microcontrolador

Cuando sucede un evento de interrupción, el microcontrolador realizará lo siguiente:

1. Evento de interrupción.
2. Suspensión del programa principal y almacenamiento del estado presente en el microcontrolador.
   - 2.1 Ejecución de la instrucción actual.
   - 2.2 Almacenamiento de 8 registros en el Stack → `R0`, `R1`, `R2`, `R3`, `R12`, `LR`, `PC`, `PSR`.
     - Si la unidad de punto flotante está activa, se almacenan 18 palabras adicionales (26 en total).
   - 2.3 El `LR` almacena un valor para indicar que se está ejecutando una **ISR**.
     - Bits `[31:8]` → `0xFFFFFF`.
     - Bits `[7:1]` → Especifican el tipo de retorno de interrupción a ejecutar.
     - Bit `[0]` → `1`.
   - 2.4 El registro de estado del programa de interrupción (`IPSR`) almacena el número de interrupción que se está ejecutando.
   - 2.5 El `PC` se carga con la dirección de la **ISR**.
3. Ejecución de la rutina de servicio de interrupción (ISR).
4. Restauración del estado en el que se encontraba el microcontrolador, previo a la ejecución de la ISR.
5. Reanudación del programa principal.

Restauración del estado del microcontrolador

Cuando sucede un evento de interrupción, el microcontrolador realizará lo siguiente:

1. Evento de interrupción.
2. Suspensión del programa principal y almacenamiento del estado presente en el microcontrolador.
3. Ejecución de la rutina de servicio de interrupción (ISR).
   - Durante la ejecución de la **ISR** es necesario limpiar la bandera de **disparo** (trigger) que generó la interrupción.
4. Restauración del estado en el que se encontraba el microcontrolador, previo a la ejecución de la ISR.
   4.1 Al finalizar la **ISR**, se ejecutará la instrucción `BX LR`.
   4.2 El valor almacenado en `LR` indica la instrucción que extrae el valor de los 8 registros almacenados en el stack para reanudar la ejecución del programa principal.
5. Reanudación del programa principal.

Nested Vectored Interrupt Controller (NVIC)

Las interrupciones en el Cortex-M se controlan por el **NVIC**.

Habilitación de una interrupción

- Las interrupciones están listadas en el vector de interrupciones (pp. 116 del manual de usuario).
- En total, se tienen definidas 114 fuentes de interrupción en el **NVIC** (0 – 113).
- Cada interrupción tiene un bit de habilitación, agrupados en 4 registros (`EN0`, `EN1`, `EN2` y `EN3`).

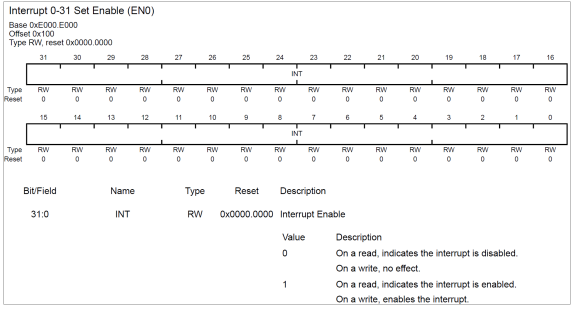
Nested Vectored Interrupt Controller (NVIC) Registers					
0x100	EN0	RW	0x0000.0000	Interrupt 0-31 Set Enable	154
0x104	EN1	RW	0x0000.0000	Interrupt 32-63 Set Enable	154
0x108	EN2	RW	0x0000.0000	Interrupt 64-95 Set Enable	154
0x10C	EN3	RW	0x0000.0000	Interrupt 96-113 Set Enable	154

Nested Vectored Interrupt Controller (NVIC)

Las interrupciones en el Cortex-M se controlan por el **NVIC**.

Habilitación de una interrupción

- Las interrupciones están listadas en el vector de interrupciones (pp. 116 del manual de usuario).
- En total, se tienen definidas 114 fuentes de interrupción en el **NVIC** (0 – 113).
- Cada interrupción tiene un bit de habilitación, agrupados en 4 registros (`EN0`, `EN1`, `EN2` y `EN3`).



Interrupt 0-31 Set Enable (EN0)

Base 0xE000.E000
Offset 0x100
Type RW, reset 0x0000.0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
INT															
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INT															
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31:0	INT	RW	0x0000.0000	Interrupt Enable

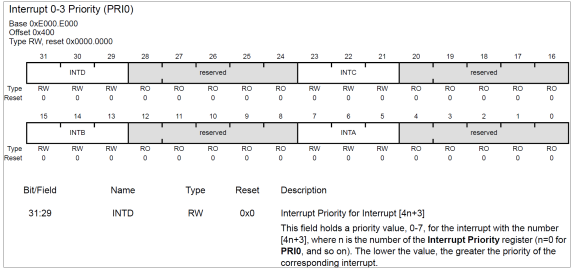
Value	Description
0	On a read, indicates the interrupt is disabled. On a write, no effect.
1	On a read, indicates the interrupt is enabled. On a write, enables the interrupt.

Nested Vectored Interrupt Controller (NVIC)

Las interrupciones en el Cortex-M se controlan por el **NVIC**.

Prioridad de una interrupción

- Después de las excepciones, todas las interrupciones tienen un nivel de prioridad de **0**.
- En el `TIVA TM4C1294NCPDT` se tienen 8 niveles de prioridad (0 al 7), en donde el 0 es el valor de mayor prioridad.
- El nivel de prioridad se escribe en el registro `PRIn` correspondiente (`PRI0` – `PRI28`).



Interrupt 0-3 Priority (PRI0)

Base 0xE000.E000
Offset 0x400
Type RW, reset 0x0000.0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
INTD			reserved					INTC			reserved				
Type	RW	RW	RW	RO	RO	RO	RO	RO	RW	RW	RW	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTB			reserved					INTA			reserved				
Type	RW	RW	RW	RO	RO	RO	RO	RO	RW	RW	RW	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31:29	INTD	RW	0x0	Interrupt Priority for Interrupt [4n+3] This field holds a priority value, 0-7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt.

Proceso de configuración de una interrupción

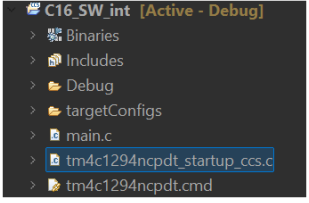
De manera general, se deben de cumplir 5 condiciones para que se ejecute una interrupción:

1. Habilitación de la interrupción en el dispositivo (en el periférico correspondiente).
2. Habilitación del NVIC.
3. Desenmascarar la interrupción global en el registro `PRIMASK`.
4. Establecer el nivel de prioridad de la interrupción en los registros `PRIn`.
5. Habilitar la interrupción en los registros `ENn`.

Configuración de las interrupciones en CCS

En un proyecto de CCS se tiene el archivo `startup_ccs.c`, cuyas funciones principales son:

1. Inicialización del hardware.
2. Inicialización del sistema de memoria.
3. Definición del vector de interrupciones (índice de las direcciones de las ISR).
4. ISR de las excepciones.




```
56 //*****************************************************************************
57 // To be added by user
58 extern void GPIOPortJ_Handler();
59
60 //*****************************************************************************
61 //
62 // The vector table.  Note that the proper constructs must be placed on this to
63 // ensure that it ends up at physical address 0x0000.0000 or at the start of
64 // the program if located at a start address other than 0.
65 //
66 //*****************************************************************************
67 #pragma DATA_SECTION(g_pfnVectors, ".intvecs")
68 void (* const g_pfnVectors[])(void) =
69 {
70     (void (*)(void))((uint32_t)&__STACK_TOP),
71                                             // The initial stack pointer
72     ResetISR,                               // The reset handler
73     NmiSR,                                  // The NMI handler
74     FaultISR,                               // The hard fault handler
75     IntDefaultHandler,                      // The MPU fault handler
76     IntDefaultHandler,                      // The bus fault handler
77     IntDefaultHandler,                      // The usage fault handler
78     0,                                      // Reserved
79     0,                                      // Reserved
80     0,                                      // Reserved
81     0,                                      // Reserved
82     IntDefaultHandler,                      // SVCall handler
83     IntDefaultHandler,                      // Debug monitor handler
84     0,                                      // Reserved
85     IntDefaultHandler,                      // The PendSV handler
86     IntDefaultHandler,                      // The SysTick handler
87     IntDefaultHandler,                      // GPIO Port A
88     IntDefaultHandler,                      // GPIO Port B
89     IntDefaultHandler,                      // GPIO Port C
```

Inicialización y configuración de GPIO

1. Habilitar el reloj del puerto en el registro `RCGCGPIO` (pp. 382), esperando dos ciclos de instrucción para que se estabilice el reloj.
2. Configurar la dirección del GPIO en el registro `GPIODIR` (pp. 760).
3. (OPCIONAL) Configurar la función alterna del GPIO en el registro `GPIOAFSEL` (pp. 770).
4. (OPCIONAL) Configurar los modos de controlador extendidos en el registro `GPIOPC` (pp. 800).
5. (OPCIONAL) Configurar la corriente de entrada/salida del GPIO en los registros `GPIODR4R`, `GPIODR8R` y `GPIODR12R` (pp. 773, 774 y 792) (la corriente máxima con la que operan los GPIO del TIVA TM4C12N4NCPDT es de 12mA).
6. (OPCIONAL) Habilitar las resistencias de pull-up (`GPIOPUR`) o de pull-down (`GPIOPDR`), o configurar como drenaje abierto (`GPIOODR`) (pp. 776, 778 y 775).
7. Configurar las funciones digitales en el registro `GPIODEN` (pp. 781).
8. (OPCIONAL) Configurar los campos de interrupción en los registros `GPIOIS`, `GPIOIBE`, `GPIOIEV` y `GPIOIM` (pp. 761, 762, 763 y 764).
9. (OPCIONAL) Configurar el bloqueo de puertos en el registro `GPIOLOCK` (pp. 783).

Inicialización y configuración de GPIO con interrupción

1. Habilitar el reloj del puerto en el registro `RCGCGPIO` (pp. 382), esperando dos ciclos de instrucción para que se estabilice el reloj.
2. Configurar la dirección del GPIO en el registro `GPIODIR` (pp. 760).
3. (OPCIONAL) Configurar la función alterna del GPIO en el registro `GPIOAFSEL` (pp. 770).
4. (OPCIONAL) Configurar los modos de controlador extendidos en el registro `GPIOPC` (pp. 800).
5. (OPCIONAL) Configurar la corriente de entrada/salida del GPIO en los registros `GPIODR4R`, `GPIODR8R` y `GPIODR12R` (pp. 773, 774 y 792) (la corriente máxima con la que operan los GPIO del TIVA TM4C12N4NCPDT es de 12mA).
6. Habilitar las resistencias de pull-up (`GPIOPUR`) o de pull-down (`GPIOPDR`), o configurar como drenaje abierto (`GPIOODR`) (pp. 776, 778 y 775).
7. Configurar las funciones digitales en el registro `GPIODEN` (pp. 781).
8. Configurar los campos de interrupción en los registros `GPIOIS`, `GPIOIBE`, `GPIOIEV` y `GPIOIM` (pp. 761, 762, 763 y 764).
9. (OPCIONAL) Configurar el bloqueo de puertos en el registro `GPIOLOCK` (pp. 783).

Códigos de clase

Revisión de los códigos de clase que están en Classroom.

Código 15 – Uso de los SW de la tarjeta de desarrollo en C:
El código espera (por polling) la activación de la señal del `SW1` para conmutar el LED `D1`, esperando (también por polling) que se suelte el botón para regresar al inicio.

Código 16 – Uso de los SW de la tarjeta de desarrollo con interrupciones:
El código se mantiene en un loop infinito en donde enciende y apaga los LED de usuario en el orden `D1`, `D2`, `D3` y `D4`, con una frecuencia de $500ms$. Además, se tiene configurada la interrupción del puerto `PortJ` para actualizar el valor de un contador (definido como `cuenta`), de tal modo que se incrementa en uno cuando se presiona el `SW1` y disminuye en uno cuando se presiona el `SW2`, realizando estas dos acciones dentro de la rutina de servicio de interrupción (ISR) llamada `GPIOPortJ_Handler`.

Proyecto 4 – GPIO con interrupciones

Realizar un código en lenguaje C que encienda y apague los 4 LED de la tarjeta de desarrollo con una frecuencia de 1s, esperando 3 eventos de interrupción, con los que realizará lo siguiente:

- **Evento 1:** Con una frecuencia de 1s entre cada acción, conmutar los LED en el siguiente orden, durante 2 ciclos:
  1. `LED1` $\rightarrow$ *ON*
  2. `LED2` $\rightarrow$ *ON*
  3. `LED3` $\rightarrow$ *ON*
  4. `LED4` $\rightarrow$ *ON*
  5. `LED1` $\rightarrow$ *OFF*
  6. `LED2` $\rightarrow$ *OFF*
  7. `LED3` $\rightarrow$ *OFF*
  8. `LED4` $\rightarrow$ *OFF*
- **Evento 2:** Con una frecuencia de 500ms entre cada acción, conmutar los LED en el siguiente orden, durante 4 ciclos:
  1. `LED4` $\rightarrow$ *ON*
  2. `LED3` $\rightarrow$ *ON*
  3. `LED2` $\rightarrow$ *ON*
  4. `LED1` $\rightarrow$ *ON*
  5. `LED4` $\rightarrow$ *OFF*
  6. `LED3` $\rightarrow$ *OFF*
  7. `LED2` $\rightarrow$ *OFF*
  8. `LED1` $\rightarrow$ *OFF*
- **Evento 3:** Con una frecuencia de 150ms entre cada acción, conmutar los LED en el siguiente orden, durante 8 ciclos:
  1. `LED1` $\rightarrow$ *ON*
  2. `LED2` $\rightarrow$ *ON*
  3. `LED3` $\rightarrow$ *ON*
  4. `LED4` $\rightarrow$ *ON*
  5. `LED1` $\rightarrow$ *OFF*
  6. `LED2` $\rightarrow$ *OFF*
  7. `LED3` $\rightarrow$ *OFF*
  8. `LED4` $\rightarrow$ *OFF*

En donde, el evento 3 tendrá la máxima prioridad, seguido del evento 2 y, por último, el evento 1.