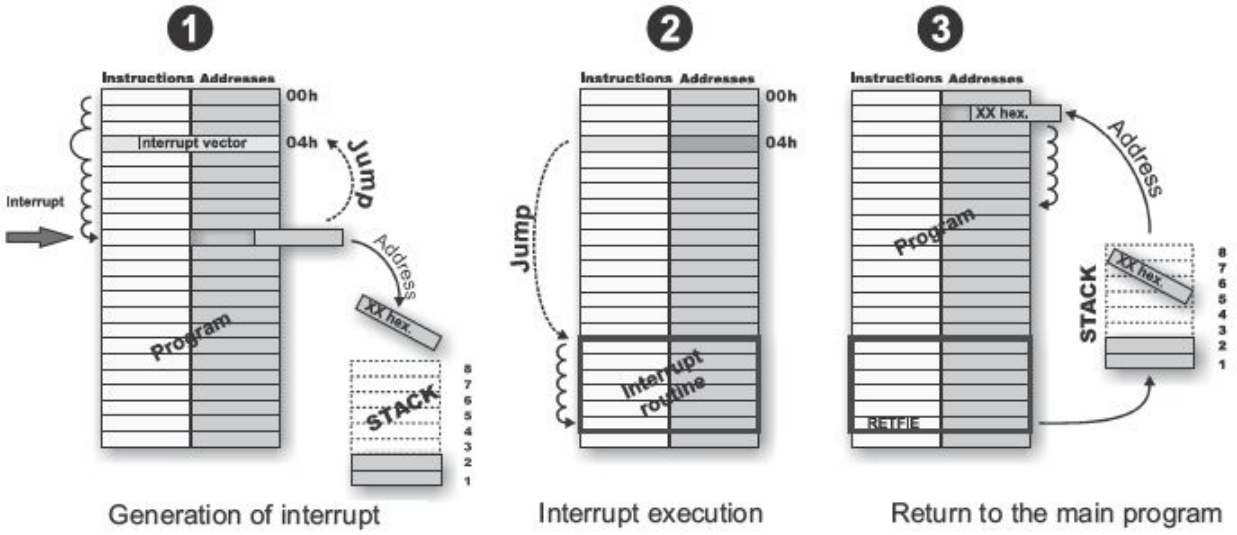
NVIC
Nested Vectored Interrupt Controller

Características

- Soporte de Interrupciones Vectorizadas
- Baja Latencia de Interrupción
- En LPC176x/5x soporta 35 interrupciones Vectorizadas
- Soporte de Interrupciones anidadas
- Soporta 32 niveles de prioridad
- Interrupción no enmascarable
- Generación de interrupción por software

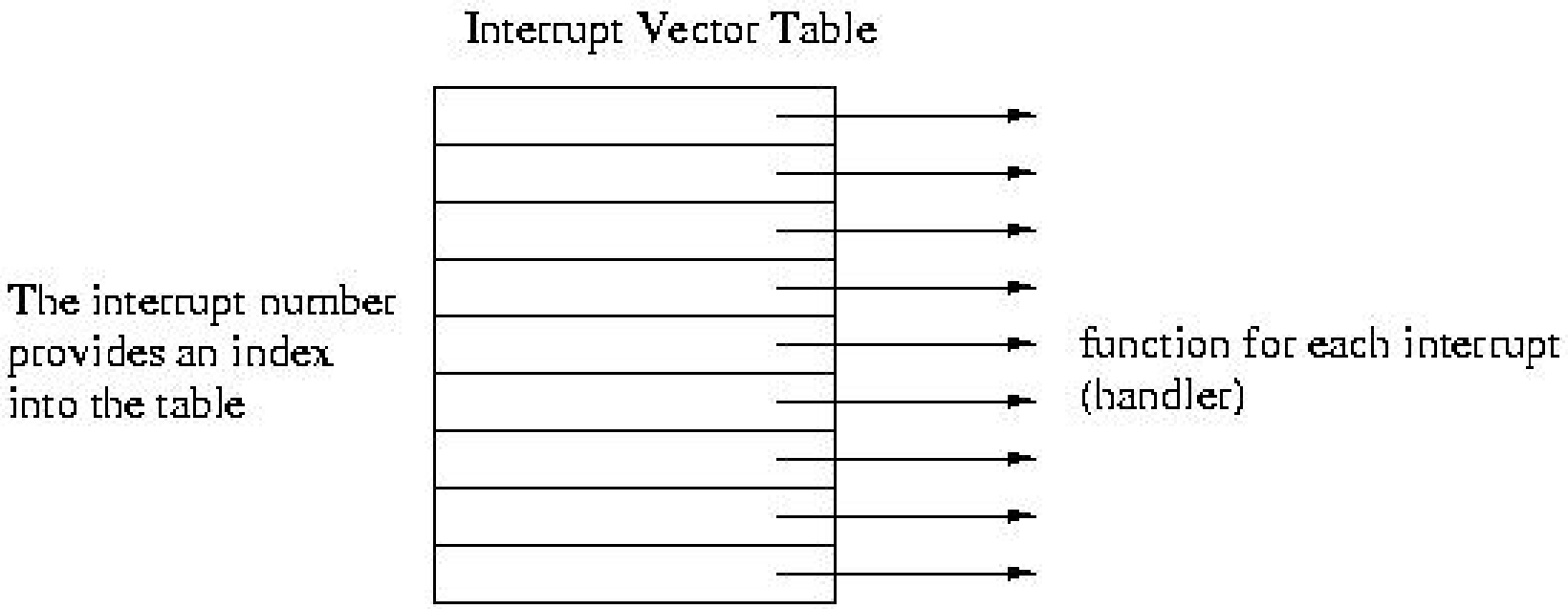
Manejo de Interrupciones

Existen varias formas de gestionar la dirección a la que se dirigirá el contador de programa en una interrupción, una es cargar el contador de programa con una dirección fija al momento de dispararse la interrupción, este es el caso de AVR de Atmel, 8051 de Intel o PIC de Microchip.

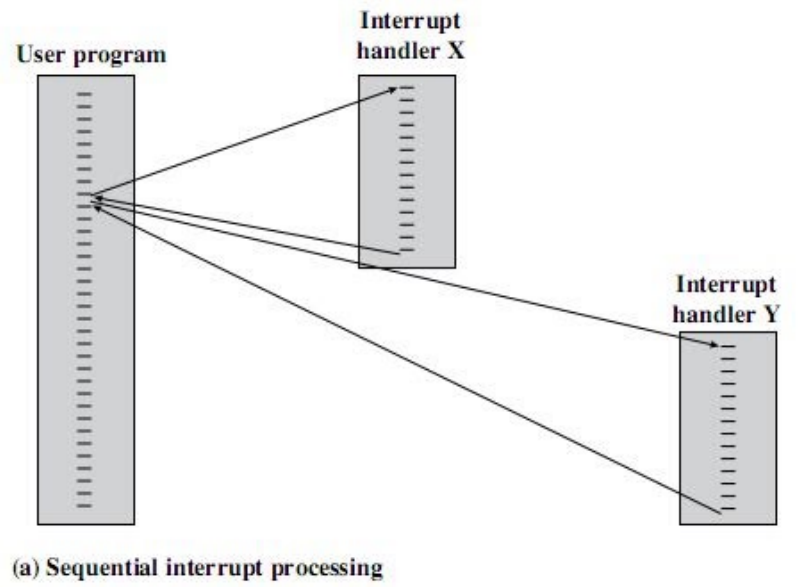


Manejo de Interrupciones

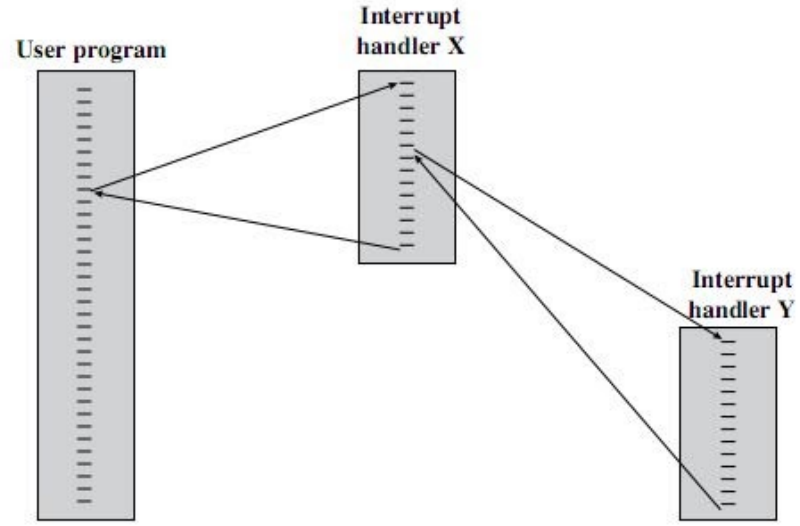
Otra forma, es tener una tabla o arreglo con las direcciones de memoria a donde dirigirse en cada caso de interrupción, entonces dependiendo de la fuente de interrupción se apunta a un elemento de esa tabla y se carga en el PC con la dirección del handler correspondiente, como es nuestro caso.



Concepto de interrupción Anidada



(a) Sequential interrupt processing



(b) Nested interrupt processing

En el caso de PIC tenemos que en el momento de dispararse una interrupción se deja de atender las otras, por lo que buscábamos que todas las tareas dentro de la interrupción sean lo más cortas posibles debido a que todo el tiempo que permanecemos dentro de una interrupción es tiempo que no podemos atender las otras.

Podemos anidar interrupciones, lo cual le da mucha mayor versatilidad a nuestras aplicaciones

Prioridad

El hardware de nuestro microprocesador (recordemos que estamos hablando de periféricos que están incluidos en el núcleo) nos permite, además de lo mencionado anteriormente, configurar prioridades. Así, las interrupciones podrán ser interrumpidas solo por eventos de mayor prioridad.

De lo visto anteriormente se desprende que debemos:

- Configurar la prioridad de la interrupción
- Configurar el periférico que queremos que genere una interrupción (EXTINT, UART, ADC, DAC, I2C, SPI, etc.)
- Habilitar la interrupción
- Escribir el código del handler que se ejecutará al momento de dispararse la interrupción

Registros asociados

Interrupt Set Enable Registers ISER0-ISER1:

Se encargan de habilitar las interrupciones y se pueden utilizar para leer cuales son las interrupciones habilitadas.

Interrupt Clear Enable Registers ICER0-ICER1:

Encargados de deshabilitar las interrupciones se puede utilizar para leer cuales son las interrupciones habilitadas.

Interrupt Set Pending Registers ISPR0-ISPR1:

Permiten determinar si una interrupción está pendiente de ser atendida o forzarla a estar pendiente.

Interrupt Clear Pending Registers ICPR0-ICPR1:

Permite determinar si una interrupción está pendiente de ser atendida o forzarla a dejar de estar pendiente.

Registros asociados

Interrupt Set Enable Registers ISER0-ISER1:

Se encargan de habilitar las interrupciones y se pueden utilizar para leer cuales son las interrupciones habilitadas.

6.5.1 Interrupt Set-Enable Register 0 register (ISER0 - 0xE000 E100)

The ISER0 register allows enabling the first 32 peripheral interrupts, or for reading the enabled state of those interrupts. The remaining interrupts are enabled via the ISER1 register (Section 6.5.2). Disabling interrupts is done through the ICER0 and ICER1 registers (Section 6.5.3 and Section 6.5.4).

Table 52. Interrupt Set-Enable Register 0 register (ISER0 - 0xE000 E100)

Bit	Name	Function
0	ISE_WDT	Watchdog Timer Interrupt Enable. Write: writing 0 has no effect, writing 1 enables the interrupt. Read: 0 indicates that the interrupt is disabled, 1 indicates that the interrupt is enabled.
1	ISE_TIMER0	Timer 0 Interrupt Enable. See functional description for bit 0.
2	ISE_TIMER1	Timer 1. Interrupt Enable. See functional description for bit 0.
3	ISE_TIMER2	Timer 2 Interrupt Enable. See functional description for bit 0.
4	ISE_TIMER3	Timer 3 Interrupt Enable. See functional description for bit 0.
5	ISE_UART0	UART0 Interrupt Enable. See functional description for bit 0.
6	ISE_UART1	UART1 Interrupt Enable. See functional description for bit 0.
7	ISE_UART2	UART2 Interrupt Enable. See functional description for bit 0.
8	ISE_UART3	UART3 Interrupt Enable. See functional description for bit 0.
9	ISE_PWM	PWM1 Interrupt Enable. See functional description for bit 0.
10	ISE_I2C0	I^2C0 Interrupt Enable. See functional description for bit 0.
11	ISE_I2C1	I^2C1 Interrupt Enable. See functional description for bit 0.
12	ISE_I2C2	I^2C2 Interrupt Enable. See functional description for bit 0.
13	ISE_SPI	SPI Interrupt Enable. See functional description for bit 0.

Registros asociados

Interrupt Active Bits Registers IABR0-IABR1:

Registros de solo lectura que permiten saber si una interrupción está activa (Atendiéndose).

Interrupt Priority Registers IPR0-IPR8:

Permiten establecer el nivel de prioridad de las diferentes fuentes de interrupción, tenemos 32 niveles de prioridad, por lo que contaremos con 5 bits por fuente de interrupción para configurar su nivel de prioridad.

Handlers

El archivo cr_startup_lpc175x_6x.c tiene todas las definiciones de los handler de que dispone el microcontrolador

```
    void ResetISR(void);
WEAK void NMI_Handler(void);
WEAK void HardFault_Handler(void);
WEAK void MemManage_Handler(void);
WEAK void BusFault_Handler(void);
WEAK void UsageFault_Handler(void);
WEAK void SVC_Handler(void);
WEAK void DebugMon_Handler(void);
WEAK void PendSV_Handler(void);
WEAK void SysTick_Handler(void);
WEAK void IntDefaultHandler(void);
```

A la Izquierda las declaraciones de los handler de las diferentes interrupciones del nucleo, abajo las de los periféricos.

```
void WDT_IRQHandler(void) ALIAS(IntDefaultHandler);
void TIMER0_IRQHandler(void) ALIAS(IntDefaultHandler);
void TIMER1_IRQHandler(void) ALIAS(IntDefaultHandler);
void TIMER2_IRQHandler(void) ALIAS(IntDefaultHandler);
void TIMER3_IRQHandler(void) ALIAS(IntDefaultHandler);
void UART0_IRQHandler(void) ALIAS(IntDefaultHandler);
void UART1_IRQHandler(void) ALIAS(IntDefaultHandler);
void UART2_IRQHandler(void) ALIAS(IntDefaultHandler);
void UART3_IRQHandler(void) ALIAS(IntDefaultHandler);
void PWM1_IRQHandler(void) ALIAS(IntDefaultHandler);
void I2C0_IRQHandler(void) ALIAS(IntDefaultHandler);
void I2C1_IRQHandler(void) ALIAS(IntDefaultHandler);
```

Cuando escribimos estos handlers en nuestro programa el compilador omite estas definiciones y compila nuestro código