第5章　中断技术

第1节　中断的基本概念

第2节　MSP430的中断系统

第3节　MSP430可屏蔽中断程序设计

第5章 中断技术

第1节 中断的基本概念

一、什么是中断
二、中断源和中断优先级
三、中断服务程序
四、断点和中断现场
五、硬件中断和软件中断

第2节 MSP430的中断系统

一、 MSP430的中断源类型
二、MSP430F638的中断源、中断优先级和中断标志
三、MSP430F6638的中断类型号和中断向量表
四、MSP430非屏蔽和可屏蔽中断响应过程
五、端口4~P12外中断

第3节 MSP430的可屏蔽中断程序设计

一、编程步骤
1. 主程序流程
2. 中断子程流程
3. 中断向量设置

二、C语言中断程序举例

第1节　中断的基本概念

一、什么是中断

二、中断源和中断优先级

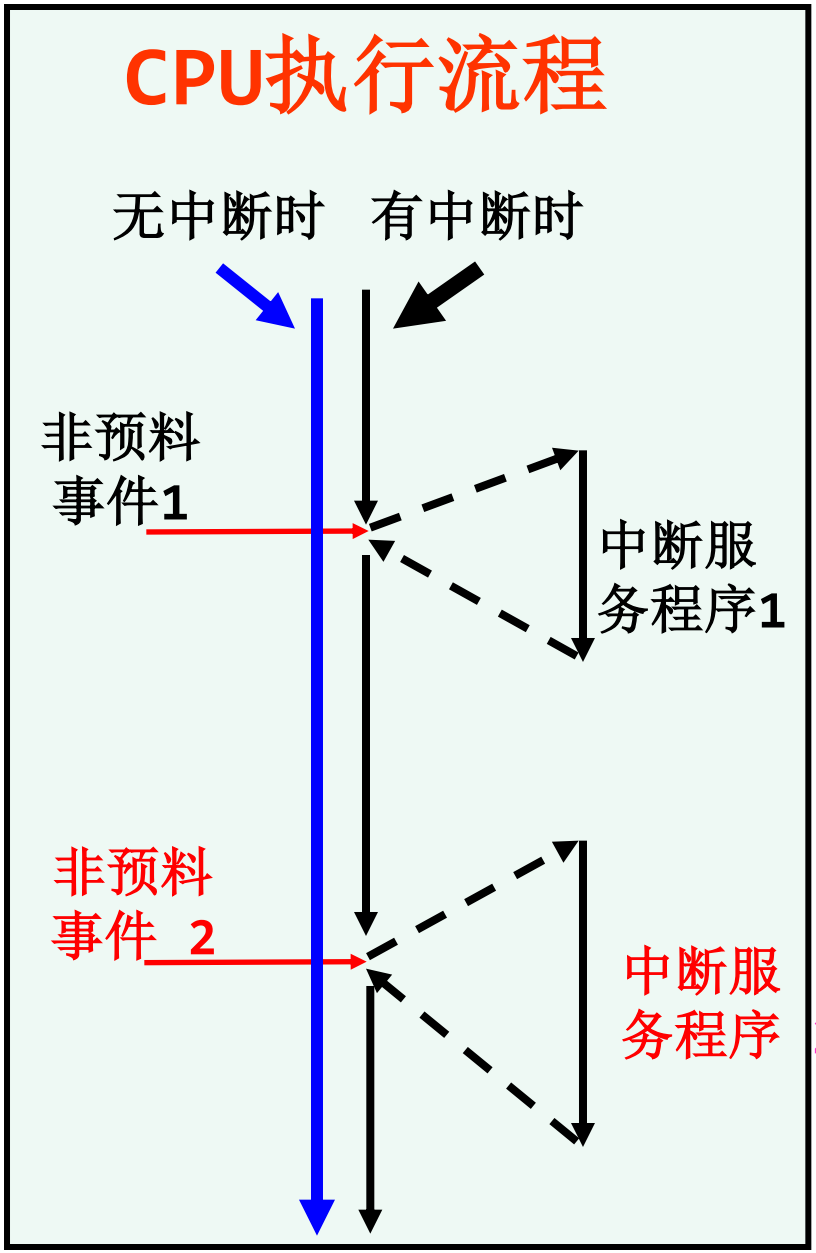
三、中断服务程序

四、断点和中断现场

五、硬件中断和软件中断

一、什么是中断

在CPU正常运行程序时，
由于内部或外部某个非预料事件的发生，
使CPU暂停正在运行的程序，
转去执行处理引起中断事件的程序，
完毕后返回被中断的程序继续运行，
这个过程就是中断。



二、中断源和中断优先级

- 引起中断的因素很多，

  将发出中断申请的外设或内部原因,称为中断源

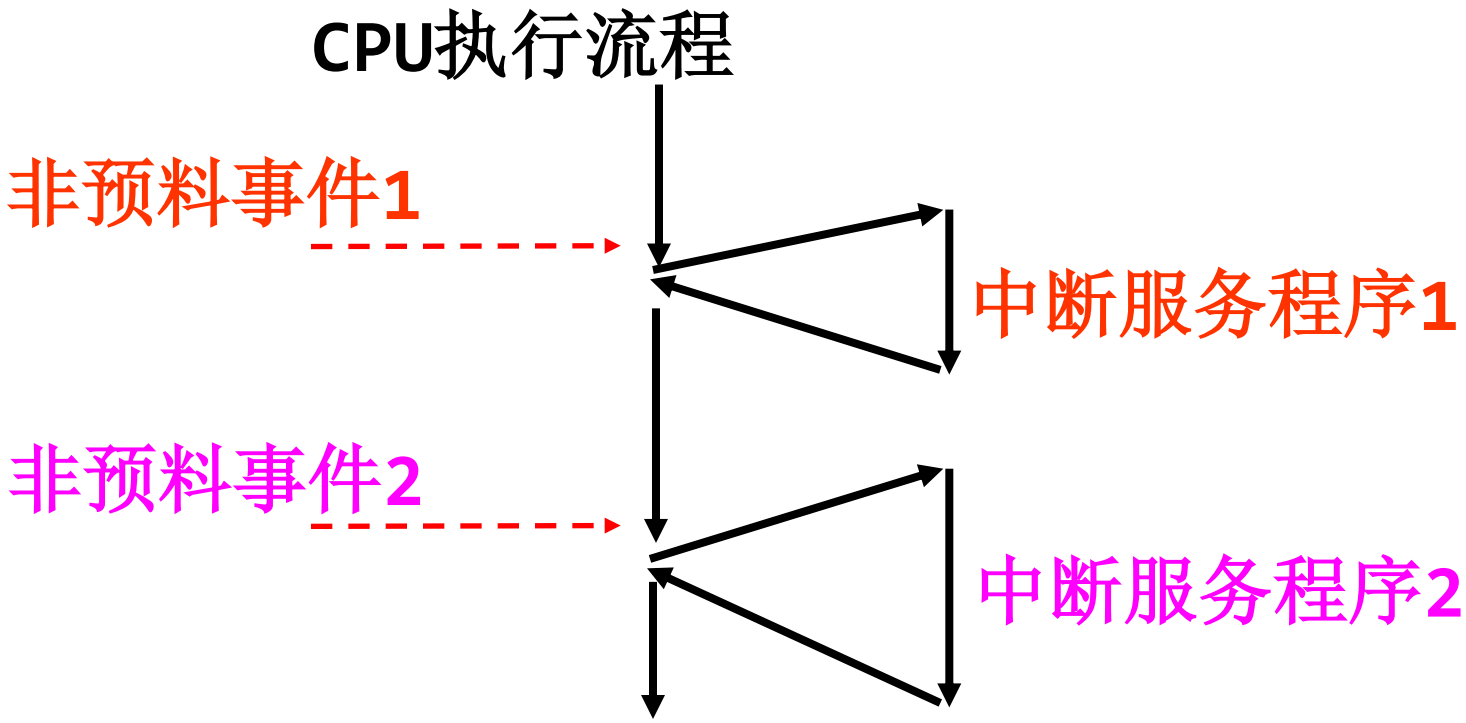
- 给每个中断源设定一个优先权级别,称为中断优先级

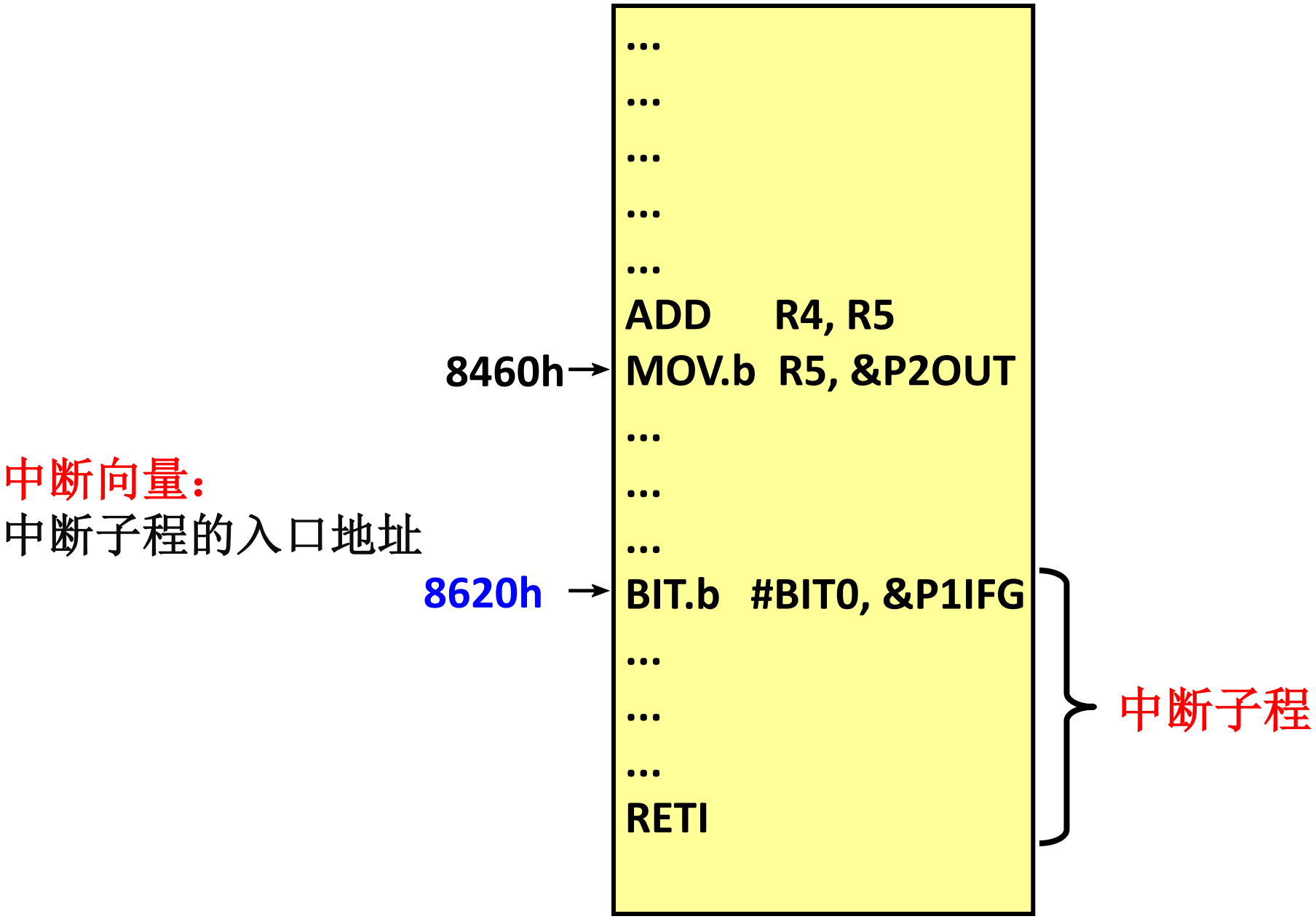
- 当多个中断源同时发出中断请求时，

  CPU按照中断优先权的高低，顺序依次响应。

三、中断服务程序和中断向量

- 处理中断源，完成其所要求功能的程序，
  称中断服务程序(中断例行程序、中断子程)。
- 中断程序的入口地址称为中断向量
  即中断程序第1条指令存放在存储器的位置




...
...
...
...
...
ADD R4, R5
8460h
MOV.b R5, &P2OUT
...
...
...
中断向量：
中断子程的入口地址
8620h
BIT.b #BIT0, &P1IFG
...
...
...
RETI
中断子程

非预料事件是指事件发生的时间无法预知，

即中断源何时产生中断不确定，是随机的。

但事件的性质及处理方法则是已知的， 确定的，

即中断服务程序是事先编写好的，

只是何时执行未知。

中断源产生中断的随机性，

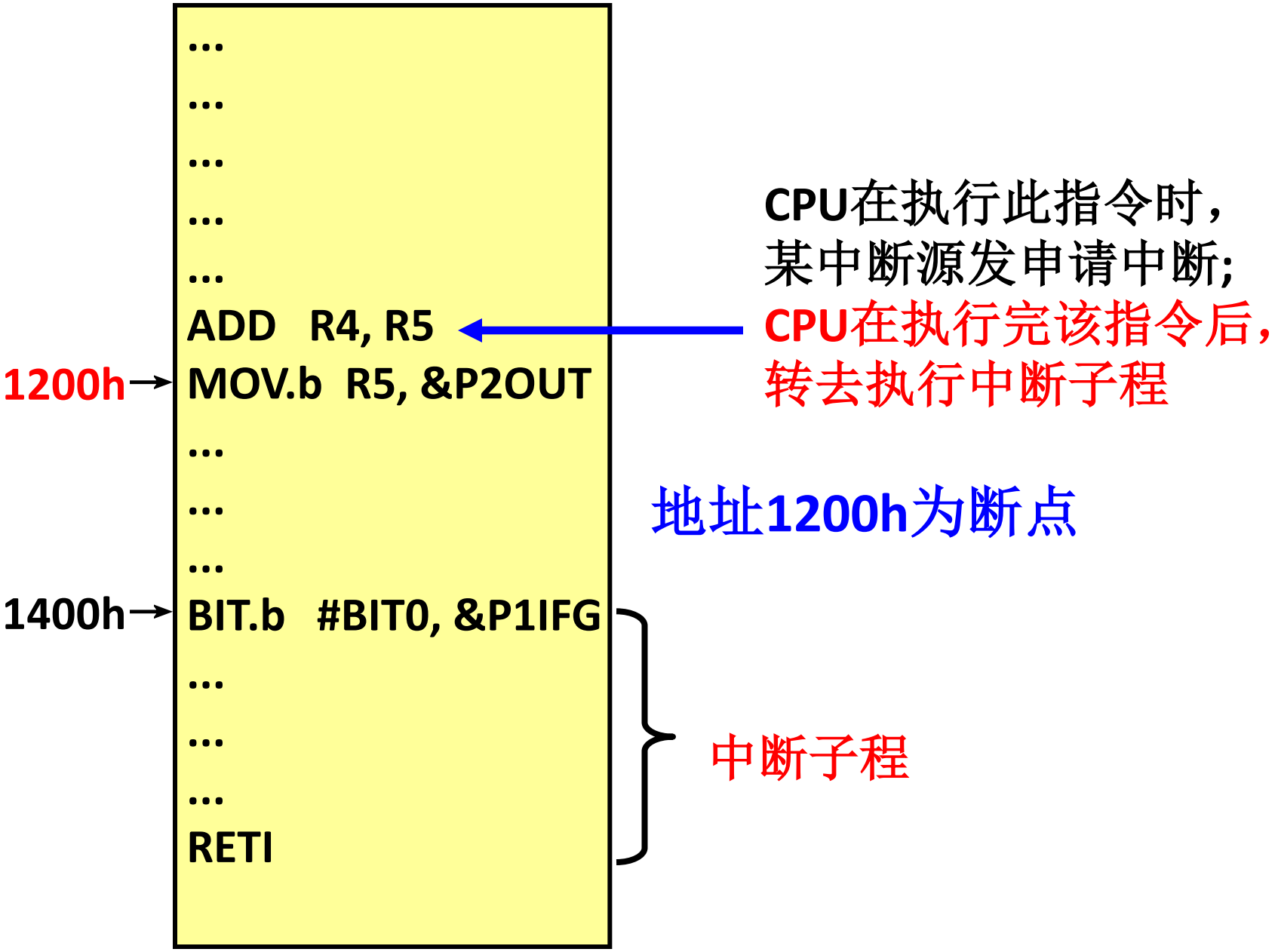
使中断服务程序的执行也具有随机性，

即何时执行中断服务程序不是在程序中安排好的。

四、断点和中断现场

断点是指CPU执行的现行程序被中断时的下一条指令的地址，又称断点地址。

中断现场是指CPU转去执行中断服务程序前的运行状态，包括CPU内部各寄存器、断点地址等。


...
...
...
...
...
ADD R4, R5
1200h
MOV.b R5, &P2OUT
...
...
...
1400h
BIT.b #BIT0, &P1IFG
...
...
...
RETI
CPU在执行此指令时，
某中断源发申请中断;
CPU在执行完该指令后，
转去执行中断子程
地址1200h为断点
中断子程

五. 硬件中断和软件中断

- 早期中断概念的引入，

  是为解决CPU与外设间的速度匹配问题，

  提高CPU 的工作效率。

  中断源主要是由外部硬件产生

- 当今的中断技术，

  不再限于外部硬件产生中断(称硬件中断或外中断)，

  还可由CPU内部产生 (如除零操作)，

或者由程序预先安排，即用指令调用中断服务程序，

(如PC机的DOS系统的中断功能调用)，称软件中断或内中断.

```
……
……
……
……
……
MOV   AH, 01
INT     21H
CMP  AL, 0Dh
……
……
……
……
PUSH  AX
……
……
IRET
```

80x86微机汇编语言
中断调用指令

← 用指令INT调用中断程序

第2节　MSP430的中断系统

一、MSP430的中断源类型

二、MSP430F6638的中断源、中断优先级

三、MSP430F6638的中断类型号和中断向量表

四、MSP430F6638的中断标志

五、MSP430的可屏蔽中断响应过程

六、MSP430F6638端口P1和P2外部中断

一、MSP430的中断源类型

两种分类：

1. 按中断源的响应是否受控分类
2. 按中断源来自MCU外部引脚还是内部分类

1. 按中断源的响应是否受控分类

与中断响应有关的控制信号

- 可屏蔽中断的总中断控制位 GIE(General Interrupt Enable Bit)

状态寄存器SR (Status Register)

15~9	8	7	6	5	4	3	2	1	0
保留	V	SCG1	SCG0	OSCoff	CPUoff	GIE	N	Z	C

GIE ： 置位1，允许可屏蔽中断
复位0，禁止可屏蔽中断

- 分中断控制位

各外围模块自己的中断响应控制位

按中断源的响应是否受控分类

MSP430的中断源分为三大类型

- 系统复位中断 system reset

  (也称不可屏蔽中断， Nonmaskable interrupts）

  ——不能被屏蔽的中断

  (不受被总控位GIE、分控位IE位的控制)

- 非屏蔽中断(Non)maskable interrupts

  ——不能被总控位GIE屏蔽，但能被分控位IE位屏蔽的中断

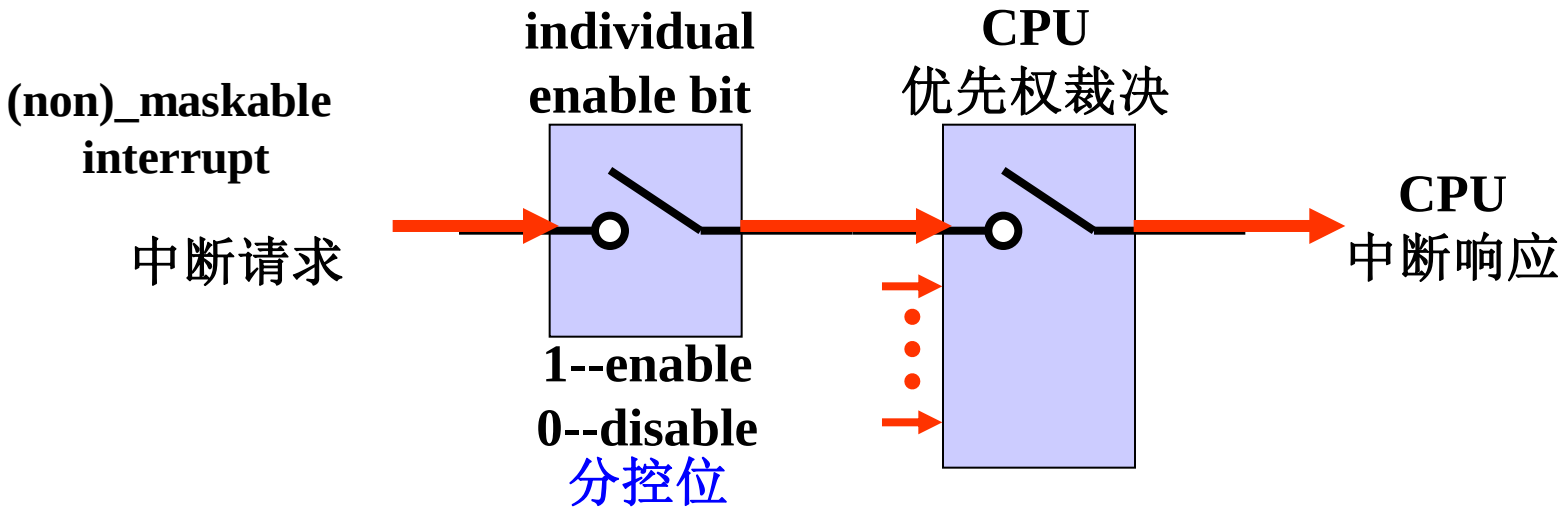
  (不受总控位GIE控制，但受分控位IE位控制)

- 可屏蔽中断maskable interrupts

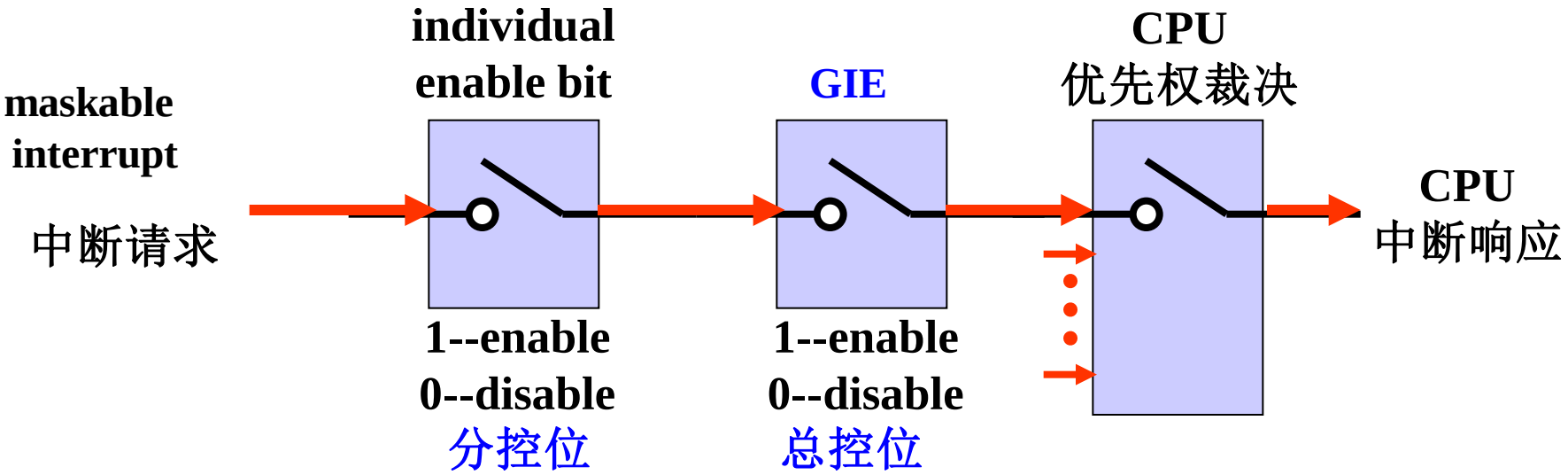
  ——能被总控位GIE或分控位IE位屏蔽的中断

  (既受总控位GIE控制，还受分控位IE位控制)

非屏蔽中断的控制机制(分控位)



可屏蔽中断的控制机制(分控位、总控位)



汇编语言：开/关总中断控制位指令
(disable/enable general interrupt bit)

指令格式	执行操作	V	Z	N	C
DINT	0→ GIE	-	-	-	-
EINT	1→ GIE	-	-	-	-

状态寄存器SR (Status Register)

15~9	8	7	6	5	4	3	2	1	0
保留	V	SCG1	SCG0	OSCoff	CPUoff	GIE	N	Z	C

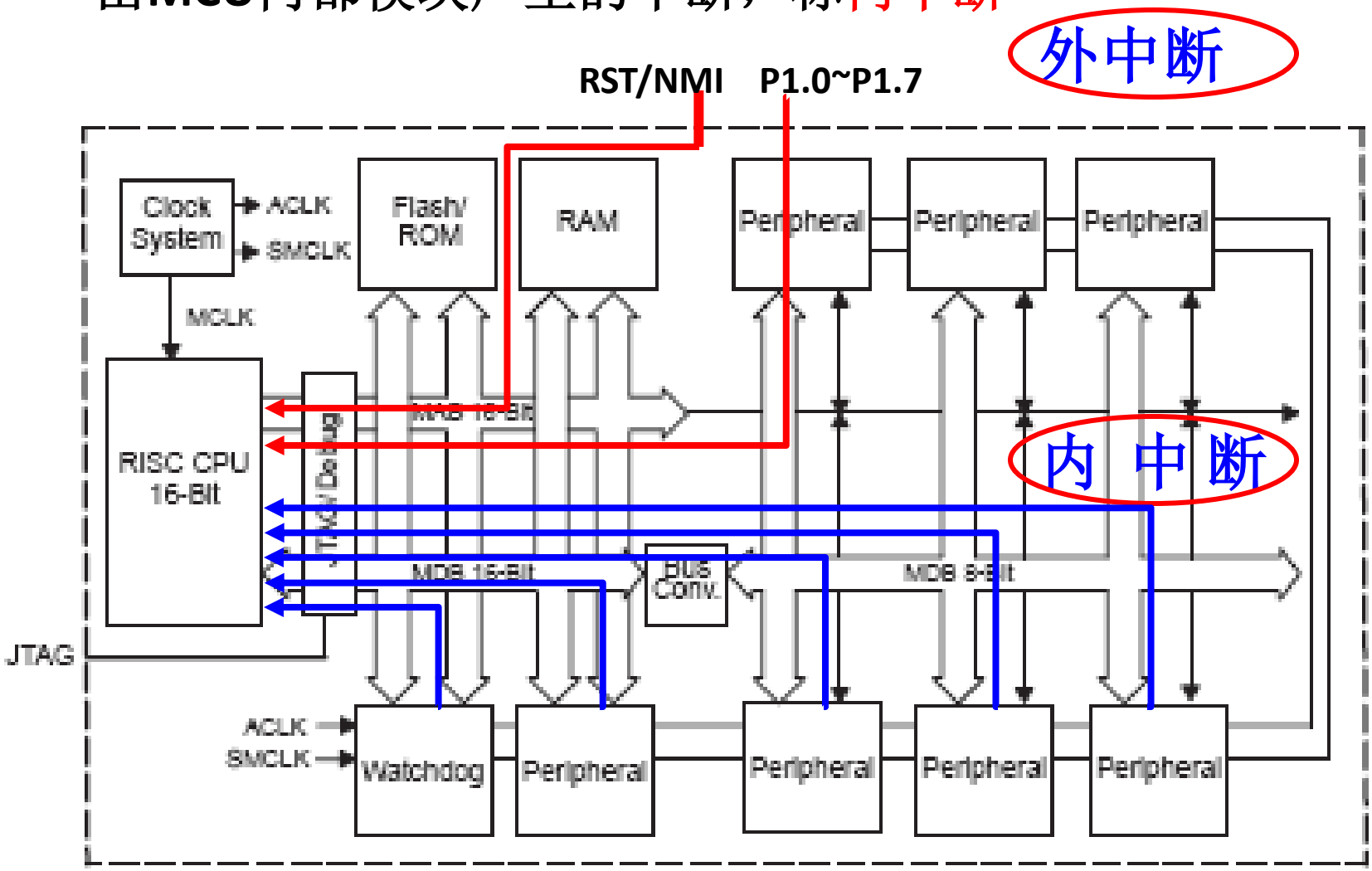
GIE :可屏蔽中断屏蔽位 (General Interrupt Enable Bit)

置位1: 允许所有可屏蔽中断　　复位0: 禁止所有可屏蔽中断

2. 按中断源来自MCU外部引脚还是内部分类

由外部引脚(如RST/NMI)产生的中断, 为外中断，

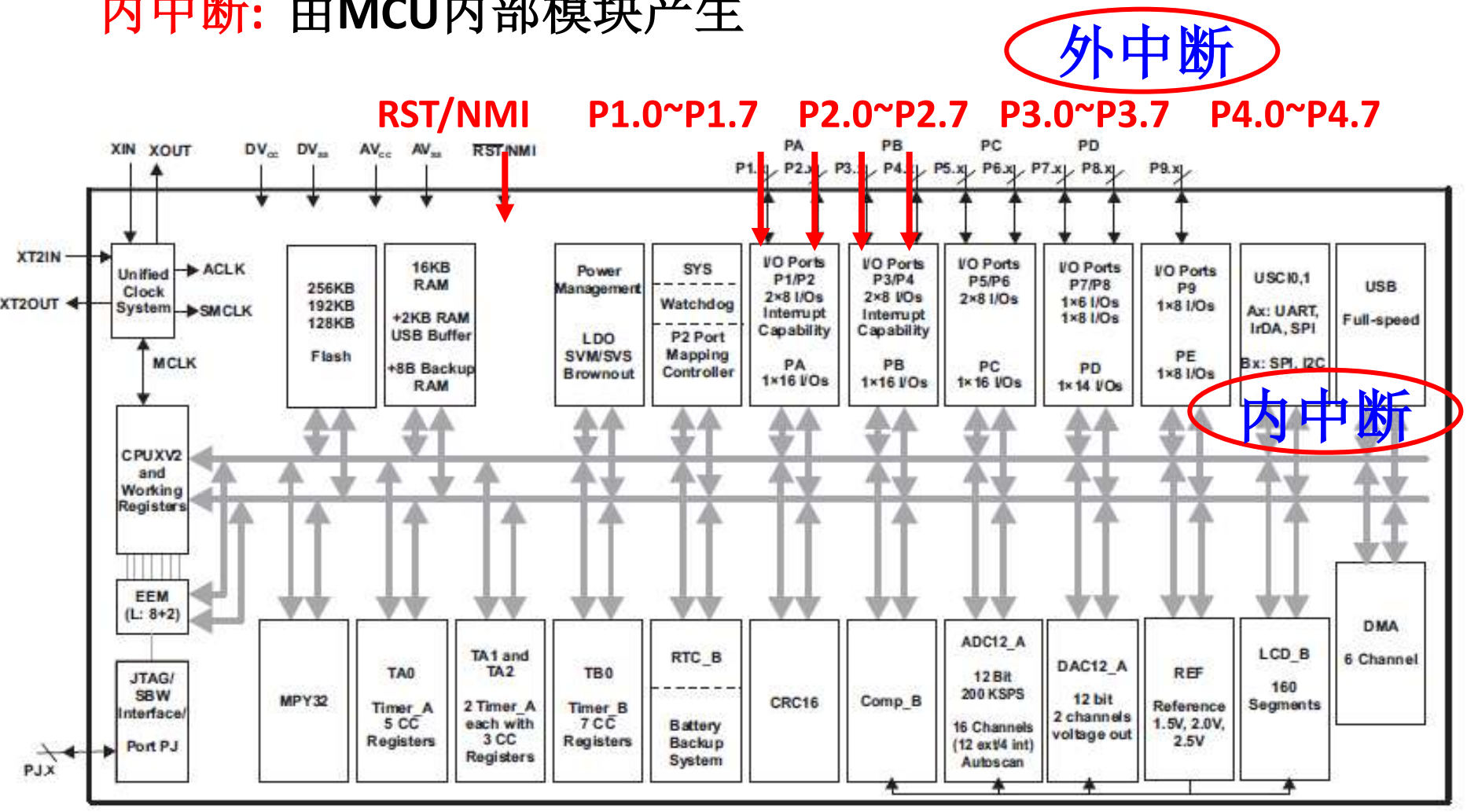
由MCU内部模块产生的中断，称内中断



MSP430F6638的外中断和内中断

外中断：引脚RST/NMI、P1.0~P1.7、P2.0~P2.7
P3.0~P3.7、P4.0~P4.7产生的中断

内中断：由MCU内部模块产生



二、MSP430F6638中断源和中断优先级 MSP430F6638.pdf

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
System Reset Power-Up, External Reset Watchdog Timeout, Key Violation Flash Memory Key Violation	WDTIFG, KEYV (SYSRSTIV)(1) (2)	Reset	0FFFEh	63, highest
System NMI PMM Vacant Memory Access JTAG Mailbox	SVMLIFG, SVMHIFG, DLYLIFG, DLYHIFG, VLRLIFG, VLRHIFG, VMAIFG, JMBNIFG, JMBOUTIFG (SYSSNIV)(1)	(Non)maskable	0FFFCh	62
User NMI NMI Oscillator Fault Flash Memory Access Violation	NMIIFG, OFIFG, ACCVIFG, BUSIFG (SYSUNIV)(1) (2)	(Non)maskable	0FFFAh	61
Comp_B	Comparator B interrupt flags (CBIV)(1) (3)	Maskable	0FFF8h	60
Timer TB0	TB0CCR0 CCIFG0 (3)	Maskable	0FFF6h	59
Timer TB0	TB0CCR1 CCIFG1 to TB0CCR6 CCIFG6, TB0IFG (TBIV)(1) (3)	Maskable	0FFF4h	58
Watchdog Interval Timer Mode	WDTIFG	Maskable	0FFF2h	57
USCI_A0 Receive/Transmit	UCA0RXIFG, UCA0TXIFG (UCA0IV)(1) (3)	Maskable	0FFF0h	56
USCI_B0 Receive/Transmit	UCB0RXIFG, UCB0TXIFG (UCB0IV)(1) (3)	Maskable	0FFEEh	55
ADC12_A(4)	ADC12IFG0 to ADC12IFG15 (ADC12IV)(1) (3)	Maskable	0FFECh	54
Timer TA0	TA0CCR0 CCIFG0(3)	Maskable	0FFEAh	53
Timer TA0	TA0CCR1 CCIFG1 to TA0CCR4 CCIFG4, TA0IFG (TA0IV)(1) (3)	Maskable	0FFE8h	52

MSP430F6638中断源和中断优先级(续)

MSP430F6638.pdf

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
USB_UBM	USB interrupts (USBIV)(1)(3)	Maskable	0FFE6h	51
DMA	DMA0IFG, DMA1IFG, DMA2IFG, DMA3IFG, DMA4IFG, DMA5IFG (DMAIV)(1) (3)	Maskable	0FFE4h	50
Timer TA1	TA1CCR0 CCIFG0(3)	Maskable	0FFE2h	49
Timer TA1	TA1CCR1 CCIFG1 to TA1CCR2 CCIFG2, TA1IFG (TA1IV)(1) (3)	Maskable	0FFE0h	48
I/O Port P1	P1IFG.0 to P1IFG.7 (P1IV)(1) (3)	Maskable	0FFDEh	47
USCI_A1 Receive/Transmit	UCA1RXIFG, UCA1TXIFG (UCA1IV)(1) (3)	Maskable	0FFDCh	46
USCI_B1 Receive/Transmit	UCB1RXIFG, UCB1TXIFG (UCB1IV)(1) (3)	Maskable	0FFDAh	45
I/O Port P2	P2IFG.0 to P2IFG.7 (P2IV)(1) (3)	Maskable	0FFD8h	44
LCD_B	LCD_B Interrupt Flags (LCDBIV)(1)	Maskable	0FFD6h	43
RTC_B	RTCRDYIFG, RTCTEVIFG, RTCAIFG, RT0PSIFG, RT1PSIFG, RTCOFIFG (RTCIV)(1) (3)	Maskable	0FFD4h	42
DAC12_A(5)	DAC12_0IFG, DAC12_1IFG(1) (3)	Maskable	0FFD2h	41
Timer TA2	TA2CCR0 CCIFG2(3)	Maskable	0FFD0h	40
Timer TA2	TA2CCR1 CCIFG1 to TA2CCR2, TA2IFG (TA2IV)(1) (3)	Maskable	0FFCEh	39
I/O Port P3	P3IFG.0 to P3IFG.7 (P3IV)(1) (3)	Maskable	0FFCCh	38
I/O Port P4	P4IFG.0 to P4IFG.7 (P4IV)(1) (3)	Maskable	0FFCAh	37

msp430F6638中断源:

- 系统复位中断 system reset: 类型号63
- 非屏蔽中断(Non)maskable interrupts: 类型号62和61
- 可屏蔽中断maskable interrupts： 类型号60和37

三、MSP430F149的中断类型号和中断向量表

- 为方便编程，给MSP430的每个中断源一个类型号，类型号的值与优先级的大小相同

- 用中断类型号区分各中断子程(中断源)

  MSP430F6638共有26个中断类型号（37~63）

  类型号63：　复位中断子程

  类型号62：　非屏蔽中断子程

  ……

  类型号47　：　端口P1中断

  ……

  类型号44　：　端口P2中断

- 中断向量

  指中断子程的入口地址

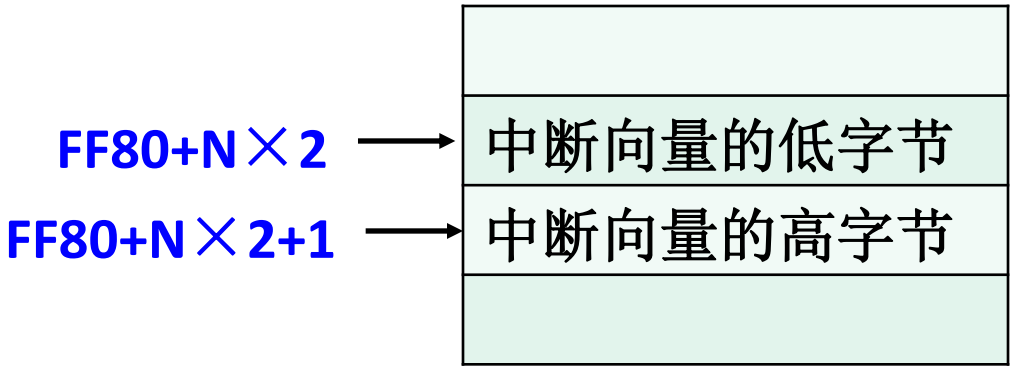
  在MSP430 中段程序的地址是一个16位二进制数，只能存在0000~FFFh的64K的范围里。

- 类型N的中断向量存放在存储器的固定单元中，

  起始地址是FF80+N×2

- 存放一个中断向量占用2个存储器单元，

  地址分别是：FF80+N×2，FF80+N×2+1



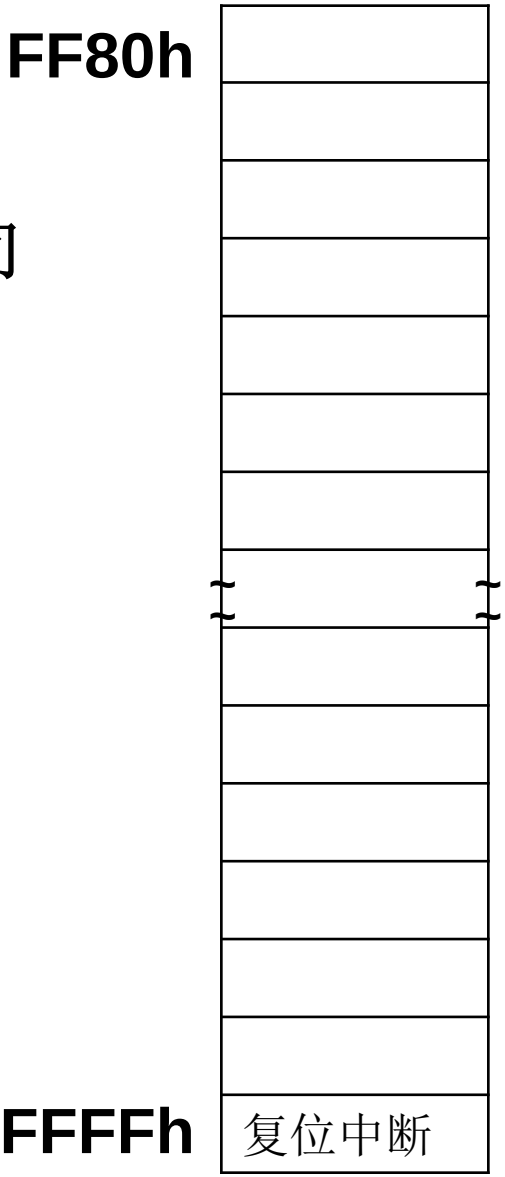
中断子程N的入口地址(中断向量)在存储器中的位置

中断向量表：

指存放 所有中断向量的存储器区域

- MSP430中断向量表在FF80~FFFFh的0x80B空间
- 不同的MCU型号实际存放的中断向量个数

  可能不同



MSP430F6638的中断向量表

INTERRUPT SOURCE	WORD ADDRESS
System Reset Power-Up, External Reset Watchdog Timeout, Key Violation Flash Memory Key Violation	0FFFEh
System NMI PMM Vacant Memory Access JTAG Mailbox	0FFFCh
User NMI NMI Oscillator Fault Flash Memory Access Violation	0FFFAh
Comp_B	0FFF8h
Timer TB0	0FFF6h
Timer TB0	0FFF4h
Watchdog Interval Timer Mode	0FFF2h
USCI_A0 Receive/Transmit	0FFF0h
USCI_B0 Receive/Transmit	0FFEEh
ADC12_A(4)	0FFECh
Timer TA0	0FFEAh
Timer TA0	0FFE8h

INTERRUPT SOURCE	WORD ADDRESS
USB_UBM	0FFE6h
DMA	0FFE4h
Timer TA1	0FFE2h
Timer TA1	0FFE0h
I/O Port P1	0FFDEh
USCI_A1 Receive/Transmit	0FFDCh
USCI_B1 Receive/Transmit	0FFDAh
I/O Port P2	0FFD8h
LCD_B	0FFD6h
RTC_B	0FFD4h
DAC12_A(5)	0FFD2h
Timer TA2	0FFD0h
Timer TA2	0FFCEh
I/O Port P3	0FFCCh
I/O Port P4	0FFCAh

- 头文件 **msp430F6638.h**

  用符号表示各中断源在中断向量表的偏移地址
- 中断向量表的首地址为**0xFF80**

msp430F6638.h 中断向量相关的符号定义

```
#define PORT4_VECTOR        (37 * 1u)        /* 0xFFCA Port 4 */
#define PORT3_VECTOR        (38 * 1u)        /* 0xFFCC Port 3 */
#define TIMER2_A1_VECTOR (39 * 1u)       /* 0xFFCE Timer0_A5 CC1-4, TA */
#define TIMER2_A0_VECTO   (40 * 1u)           /* 0xFFD0 Timer0_A5 CC0 */
#define DAC12_VECTOR        (41 * 1u)           /* 0xFFD3 DAC12 */
#define RTC_VECTOR          (42 * 1u)           /* 0xFFD4 RTC */
#define LCD_B_VECTOR        (43 * 1u)           /* 0xFFD6 LCD B */
#define PORT2_VECTOR        (44 * 1u)           /* 0xFFD8 Port 2 */
#define USCI_B1_VECTOR      (45 * 1u)           /* 0xFFDA USCI B1 Receive/Transmit */
#define USCI_A1_VECTOR      (46 * 1u)           /* 0xFFDC USCI A1 Receive/Transmit */
#define PORT1_VECTOR        (47 * 1u)           /* 0xFFDE Port 1 */
#define TIMER1_A1_VECTOR (48 * 1u)           /* 0xFFE0 Timer1_A3 CC1-2, TA1 */
#define TIMER1_A0_VECTOR (49 * 1u)           /* 0xFFE2 Timer1_A3 CC0 */
```

msp430F6638.h 中断向量相关的符号定义(续)

```
#define DMA_VECTOR          (50 * 1u)          /* 0xFFE4 DMA */
#define USB_UBM_VECTOR      (51 * 1u)      /* 0xFFE6 USB Timer / cable event / USB reset */
#define TIMER0_A1_VECTOR    (52 * 1u)          /* 0xFFE8 Timer0_A5 CC1-4, TA */
#define TIMER0_A0_VECTOR    (53 * 1u)          /* 0xFFEA Timer0_A5 CC0 */
#define ADC12_VECTOR        (54 * 1u)          /* 0xFFEC ADC */
#define USCI_B0_VECTOR      (55 * 1u)          /* 0xFFEE USCI B0 Receive/Transmit */
#define USCI_A0_VECTOR      (56 * 1u)          /* 0xFFF0 USCI A0 Receive/Transmit */
#define WDT_VECTOR          (57 * 1u)          /* 0xFFF2 Watchdog Timer */
#define TIMER0_B1_VECTOR    (58 * 1u)          /* 0xFFF4 Timer0_B7 CC1-6, TB */
#define TIMER0_B0_VECTOR    (59 * 1u)          /* 0xFFF6 Timer0_B7 CC0 */
#define COMP_B_VECTOR       (60 * 1u)          /* 0xFFF8 Comparator B */
#define UNMI_VECTOR         (61 * 1u)          /* 0xFFFA User Non-maskable */
#define SYSNMI_VECTOR       (62 * 1u)          /* 0xFFFC System Non-maskable */
#define RESET_VECTOR        (63 * 1u)          /* 0xFFFE Reset [Highest Priority] */
```

类型37的中断向量存放在向量表的地址：

0FF80 + 37*2 = 0FFCAh

0FF80h + PORT4_VECTOR*2

类型63(复位中断)的中断向量存放在向量表的地址：

0FF80 + 63*2 = 0FFFEh

0FF80h + RESET_VECTOR*2

四、MSP430F6638的中断标志

- 为了保存中断源发出的中断申请，
  MSP430内部对应有一个标志被置位，称为中断标志位
  这些标志存放在外围模块内部或特殊功能寄存器中
- 在MSP430F6638.h中，对这些寄存器的各位用符号做了定义，以便编程使用。

INTERRUPT SOURCE	INTERRUPT FLAG
System Reset Power-Up, External Reset Watchdog Timeout, Key Violation Flash Memory Key Violation	WDTIFG, KEYV (SYSRSTIV)(1) (2)
System NMI PMM Vacant Memory Access JTAG Mailbox	SVMLIFG, SVMHIFG, DLYLIFG, DLYHIFG, VLRLIFG, VLRHIFG, VMAIFG, JMBNIFG, JMBOUTIFG (SYSSNIV)(1)
User NMI NMI Oscillator Fault Flash Memory Access Violation	NMIIFG, OFIFG, ACCVIFG, BUSIFG (SYSUNIV)(1) (2)
Comp_B	Comparator B interrupt flags (CBIV)(1) (3)
Timer TB0	TB0CCR0 CCIFG0 (3)
Timer TB0	TB0CCR1 CCIFG1 to TB0CCR6 CCIFG6, TB0IFG (TBIV)(1) (3)
Watchdog Interval Timer Mode	WDTIFG
USCI_A0 Receive/Transmit	UCA0RXIFG, UCA0TXIFG (UCA0IV)(1) (3)
USCI_B0 Receive/Transmit	UCB0RXIFG, UCB0TXIFG (UCB0IV)(1) (3)
ADC12_A(4)	ADC12IFG0 to ADC12IFG15 (ADC12IV)(1) (3)
Timer TA0	TA0CCR0 CCIFG0(3)
Timer TA0	TA0CCR1 CCIFG1 to TA0CCR4 CCIFG4, TA0IFG (TA0IV)(1) (3)

INTERRUPT SOURCE	INTERRUPT FLAG
USB_UBM	USB interrupts (USBIV)[1][3]
DMA	DMA0IFG, DMA1IFG, DMA2IFG, DMA3IFG, DMA4IFG, DMA5IFG (DMAIV)[1] [3]
Timer TA1	TA1CCR0 CCIFG0[3]
Timer TA1	TA1CCR1 CCIFG1 to TA1CCR2 CCIFG2, TA1IFG (TA1IV)[1] [3]
I/O Port P1	P1IFG.0 to P1IFG.7 (P1IV)[1] [3]
USCI_A1 Receive/Transmit	UCA1RXIFG, UCA1TXIFG (UCA1IV)[1] [3]
USCI_B1 Receive/Transmit	UCB1RXIFG, UCB1TXIFG (UCB1IV)[1] [3]
I/O Port P2	P2IFG.0 to P2IFG.7 (P2IV)[1] [3]
LCD_B	LCD_B Interrupt Flags (LCDBIV)[1]
RTC_B	RTCRDYIFG, RTCTEVIFG, RTCAIFG, RT0PSIFG, RT1PSIFG, RTCOFIFG (RTCIV)[1] [3
DAC12_A[5]	DAC12_0IFG, DAC12_1IFG[1] [3]
Timer TA2	TA2CCR0 CCIFG2[3]
Timer TA2	TA2CCR1 CCIFG1 to TA2CCR2, TA2IFG (TA2IV)[1] [3]
I/O Port P3	P3IFG.0 to P3IFG.7 (P3IV)[1] [3]
I/O Port P4	P4IFG.0 to P4IFG.7 (P4IV)[1] [3]

中断标志

由于某些原因，如CPU正在执行中断子程，此时GIE=0、新中断的优先级不够高等，CPU并不一定能够马上响应中断源发出的中断请求，因此，需要一些寄存器来标志 (保存) 中断请求信号，直到被CPU响应。这些寄存器被称之为中断标志寄存器，其中的位称之为中断标志位，并与中断源一一对应。

例 定时器A的控制寄存器中的中断标志位 CCIFG

15	14	13	12	11	10	9	8
CMx		CCISx		SCS	SCCI	Unused	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	r0	rw-(0)

7	6	5	4	3	2	1	0
OUTMODx			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

比较匹配中断标志
1: 有中断
0: 无中断

- 引起MSP430的中断源可以有64级，每一级对应1个中断向量
  有的一级中断内还有多个子中断源

- 有的同一级中断源可包含多个子中断源，
  每个子中断源对应一位中断标志位，
  这些子中断源共享同一个中断向量（即共享同一个中断子程）
- 在中断子程中，可通过查询中断标志位来确定中断请求到底是由哪一个子中断源发出的。

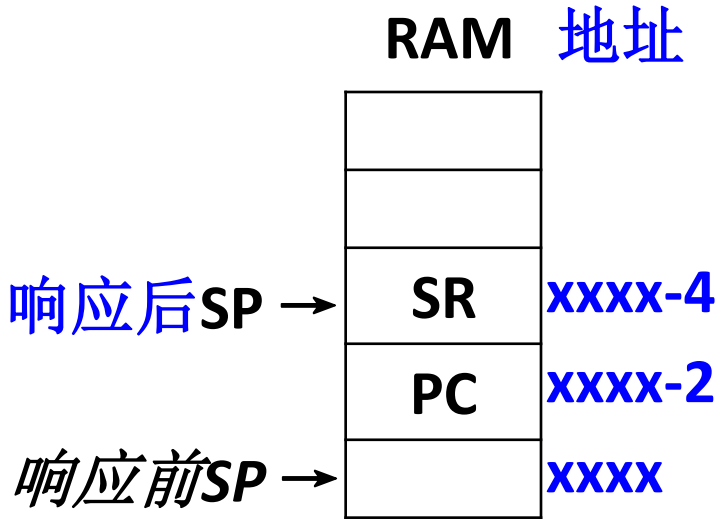
五、MSP430非屏蔽和可屏蔽中断响应过程

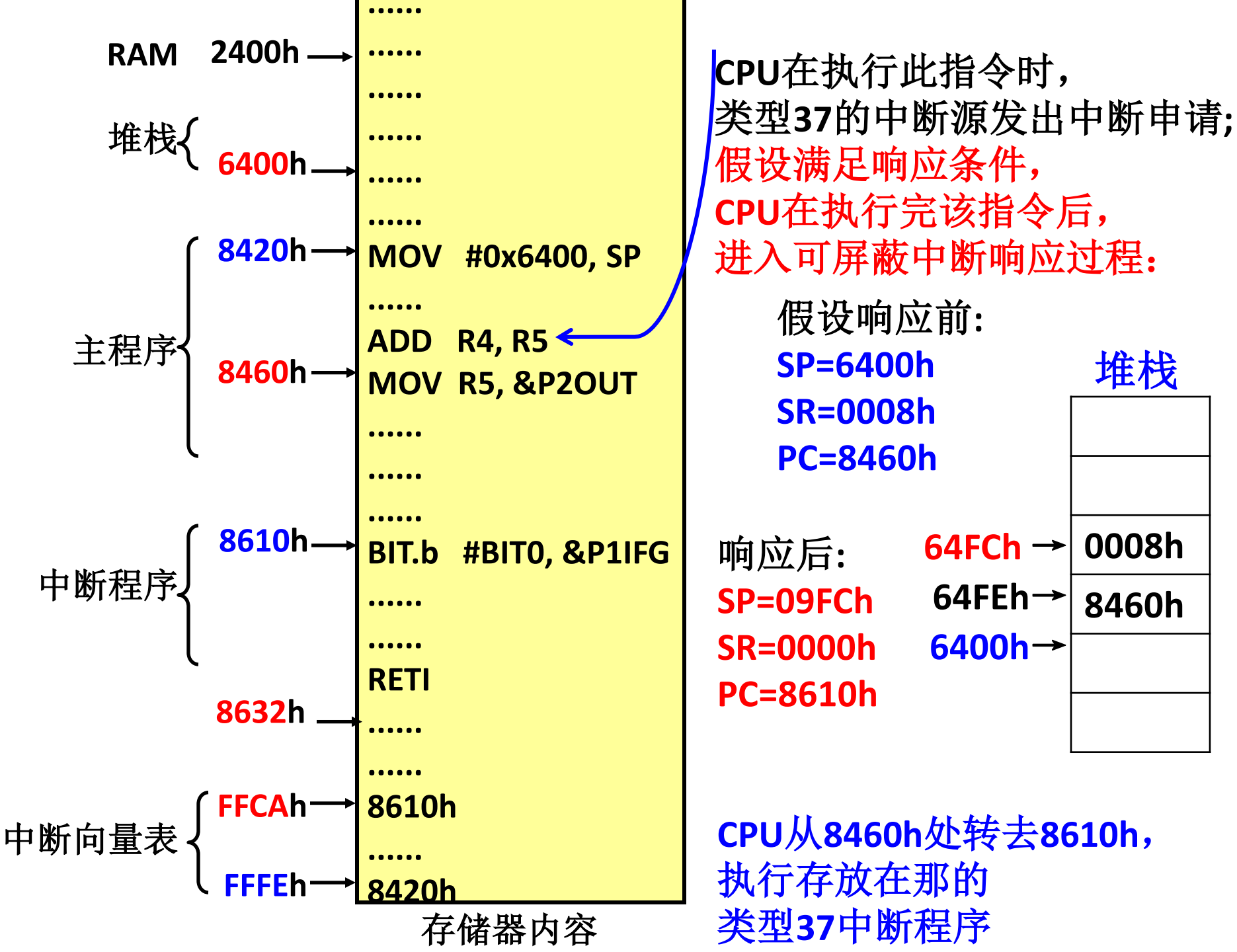
当有N类型号的非屏蔽或可屏蔽中断源产生，
满足响应条件时（???），
CPU在执行完当前指令后，
硬件自动完成下面操作：

1. 入栈保存断点
   (相当于执行 PUSH PC)
2. 入栈保存SR
   (相当于执行 PUSH SR)

保护现场

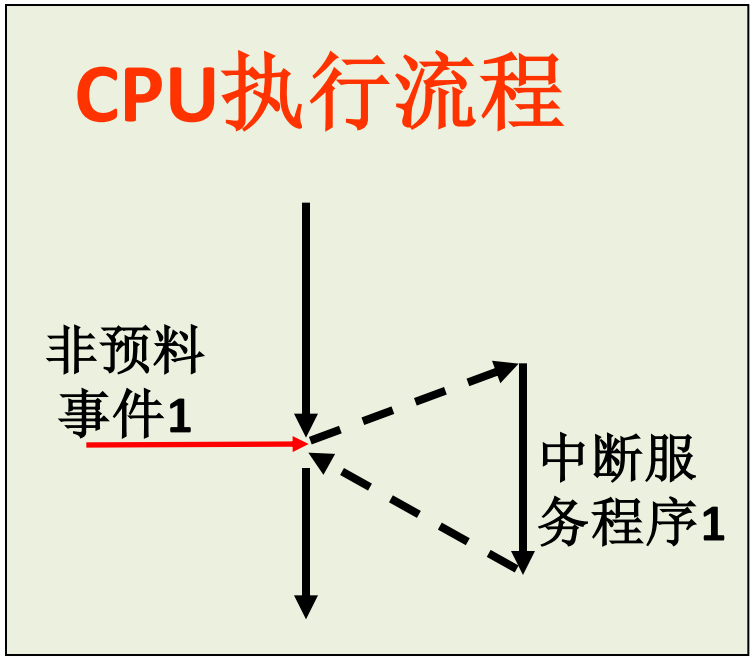
3. 清零SR
   (置GIE=0，屏蔽可屏蔽中断，
   并结束低功耗方式)
4. 从中断向量表取中断向量至PC
   (相当于 执行MOV &(0xFF80+N*2), PC
5. 转去执行中断服务子程




RAM
2400h
堆栈
6400h
8420h
主程序
8460h
8610h
中断程序
8632h
FFCAh
中断向量表
FFFEh
MOV #0x6400, SP
ADD R4, R5
MOV R5, &P2OUT
BIT.b #BIT0, &P1IFG
RETI
8610h
8420h
存储器内容
CPU在执行此指令时，
类型37的中断源发出中断申请;
假设满足响应条件，
CPU在执行完该指令后，
进入可屏蔽中断响应过程：
假设响应前:
SP=6400h
SR=0008h
PC=8460h
堆栈
响应后:
SP=09FCh
SR=0000h
PC=8610h
64FCh
64FEh
6400h
0008h
8460h
CPU从8460h处转去8610h，
执行存放在那的
类型37中断程序

中断返回：

在中断子程中执行 RETI 指令，
实现恢复中断现场，
并从断点处继续运行被中断的程序。

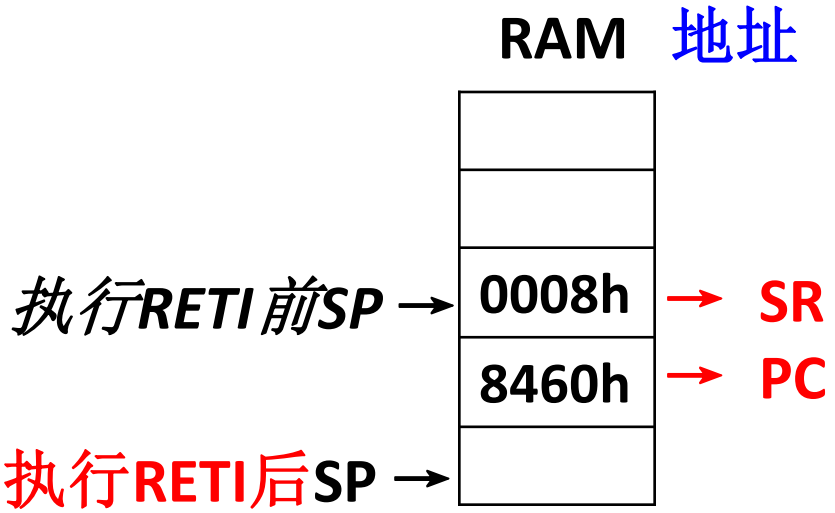


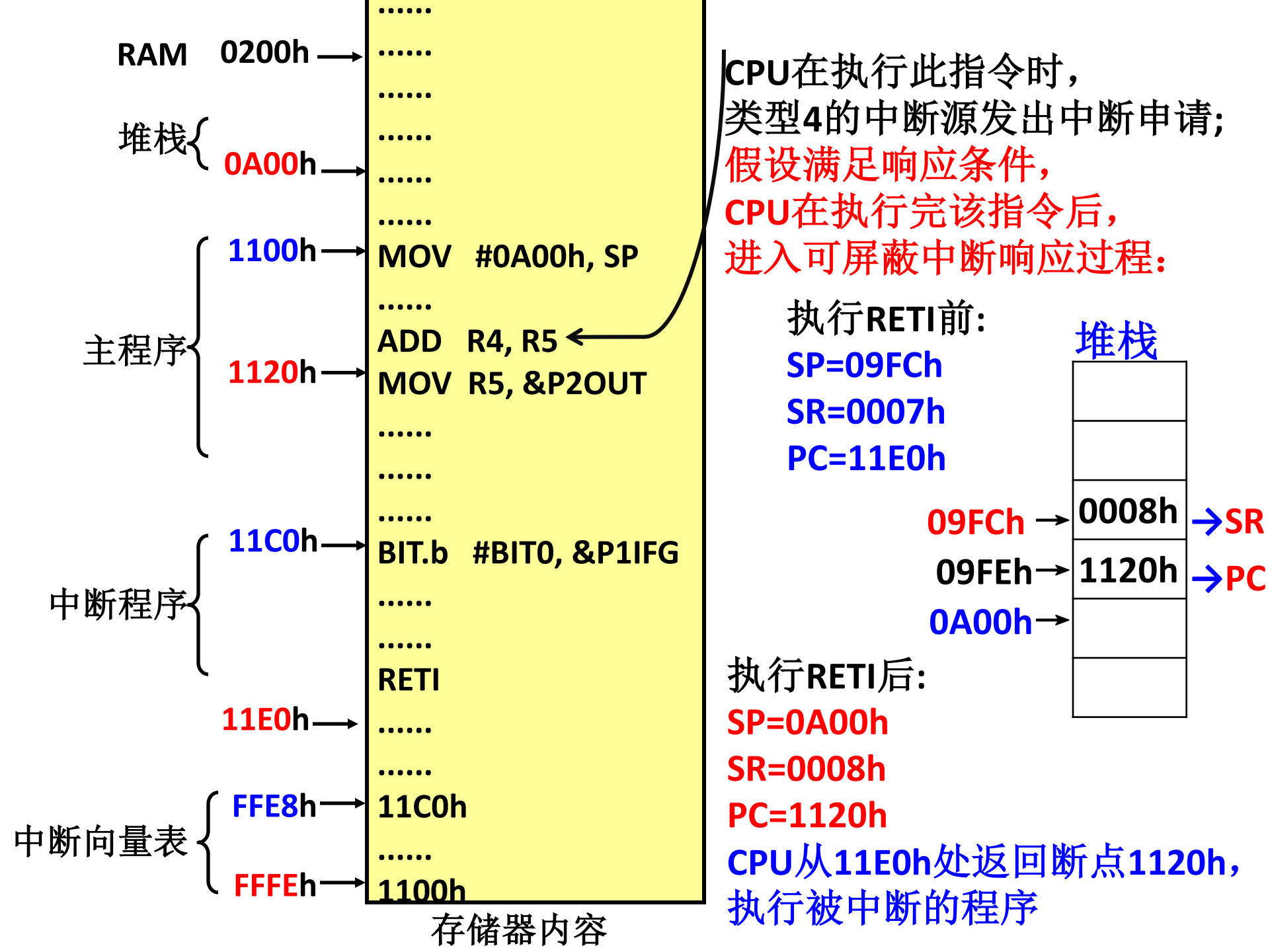
中断子程返回指令 RETI

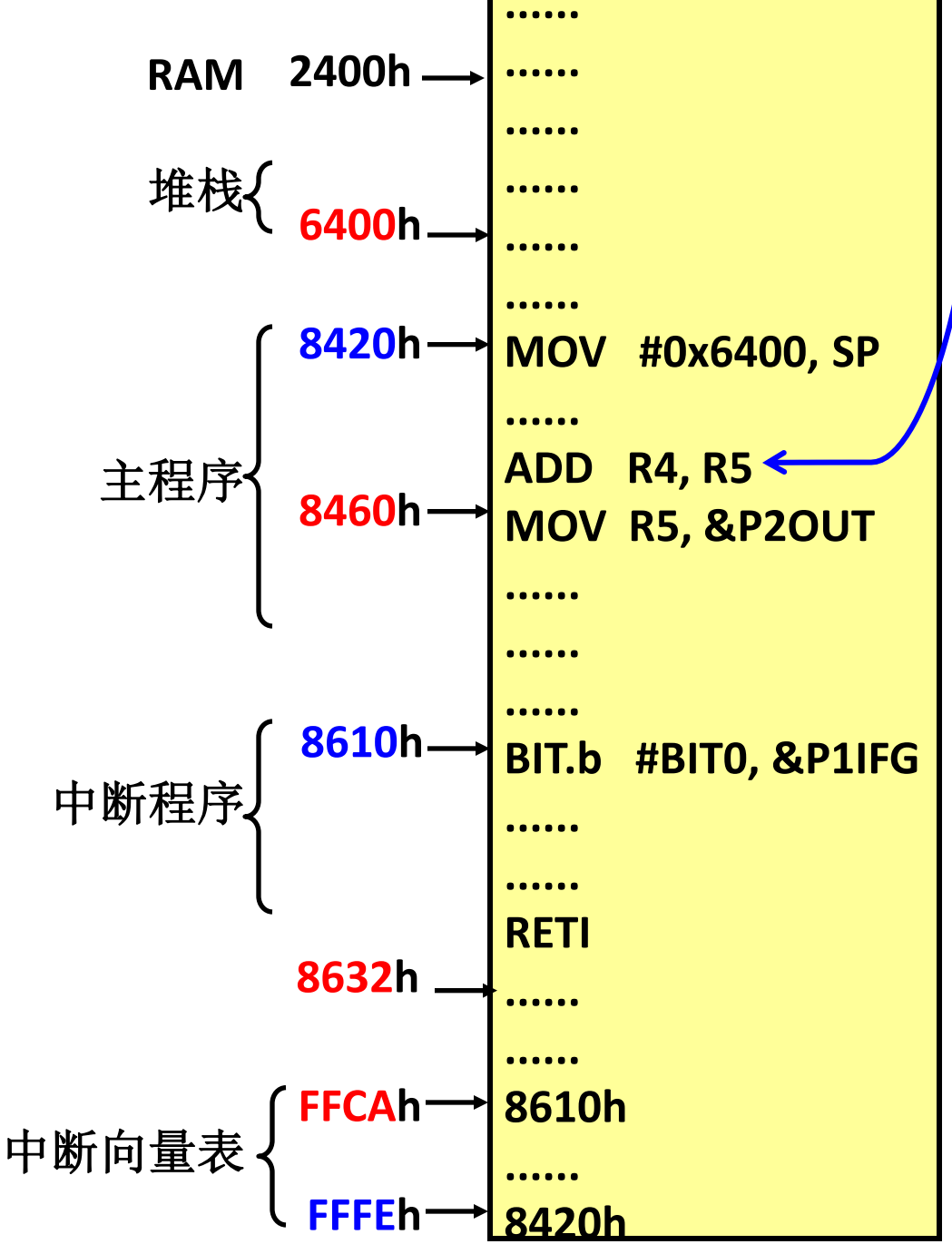
执行操作：

@SP → SR
SP+2 → SP
} 相当于POP SR

@SP → PC
SP+2 → SP
} 相当于POP PC




RAM
0200h
堆栈
0A00h
1100h
主程序
1120h
11C0h
中断程序
11E0h
FFE8h
中断向量表
FFFEh
.....
MOV #0A00h, SP
ADD R4, R5
MOV R5, &P2OUT
BIT.b #BIT0, &P1IFG
RETI
11C0h
1100h
存储器内容
CPU在执行此指令时，
类型4的中断源发出中断申请；
假设满足响应条件，
CPU在执行完该指令后，
进入可屏蔽中断响应过程：
执行RETI前：
SP=09FCh
SR=0007h
PC=11E0h
堆栈
09FCh
0008h
SR
09FEh
1120h
PC
0A00h
执行RETI后：
SP=0A00h
SR=0008h
PC=1120h
CPU从11E0h处返回断点1120h，
执行被中断的程序



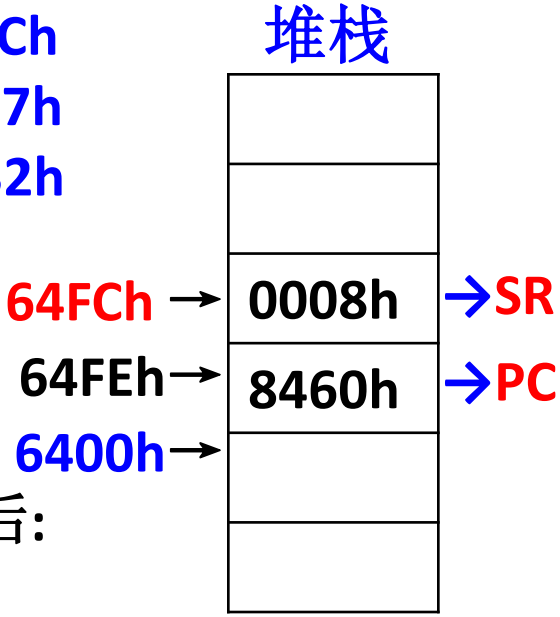
CPU在执行此指令时，
类型37的中断源发出中断申请；
假设满足响应条件，
CPU在执行完该指令后，
进入可屏蔽中断响应过程：

执行RETI前：

SP=64FCh

SR=0007h

PC=8632h



执行RETI后：

SP=6400h

SR=0008h

PC=8460h

CPU从8632h处返回断点8460h，
执行被中断的程序

六、MSP430F6638端口P4 ~P1外中断

- 端口P4~P1的Px.0~Px.7引脚均可发出中断申请，
  即具有共8x 4=32个外部输入可屏蔽中断源；

- P4~P1端口的8个引脚中断源共用一个向量地址，
  P4~P1的中断类型号分别为37、38、44、47
  与中断有关的寄存器有PxDIR, PxSEL, PxIV, PxIES,PxIFG和PxIE
  x：4~1

中断源	中断标志	向量地址	优先级,类型号
…	…	…	…
P1的8个引脚	P1IFG.0~P1IFG.7 (8)	0FFDEh	47
…	…	…	…
P2的8个引脚	P2IFG.0~P2IFG.7 (8)	0FFD8h	44
…	…	…	…
P3的8个引脚	P3IFG.0~P3IFG.7 (8)	0FFCCh	38
P4的8个引脚	P4IFG.0~P4IFG.7 (8)	0FFCAh	37

Table 31. Port P1/P2 Registers (Base Address: 0200h)

REGISTER DESCRIPTION	REGISTER	OFFSET
Port P1 input	P1IN	00h
Port P1 output	P1OUT	02h
Port P1 direction	P1DIR	04h
Port P1 pullup/pulldown enable	P1REN	06h
Port P1 drive strength	P1DS	08h
Port P1 selection	P1SEL	0Ah
Port P1 interrupt vector word	P1IV	0Eh
Port P1 interrupt edge select	P1IES	18h
Port P1 interrupt enable	P1IE	1Ah
Port P1 interrupt flag	P1IFG	1Ch
Port P2 input	P2IN	01h
Port P2 output	P2OUT	03h
Port P2 direction	P2DIR	05h
Port P2 pullup/pulldown enable	P2REN	07h
Port P2 selection	P2SEL	0Bh
Port P2 interrupt vector word	P2IV	1Eh
Port P2 interrupt edge select	P2IES	19h
Port P2 interrupt enable	P2IE	1Bh
Port P2 interrupt flag	P2IFG	1Dh

Table 32. Port P3/P4 Registers (Base Address: 0220h)

REGISTER DESCRIPTION	REGISTER	OFFSET
Port P3 input	P3IN	00h
Port P3 output	P3OUT	02h
Port P3 direction	P3DIR	04h
Port P3 pullup/pulldown enable	P3REN	06h
Port P3 drive strength	P3DS	08h
Port P3 selection	P3SEL	0Ah
Port P3 interrupt vector word	P3IV	0Eh
Port P3 interrupt edge select	P3IES	18h
Port P3 interrupt enable	P3IE	1Ah
Port P3 interrupt flag	P3IFG	1Ch
Port P4 input	P4IN	01h
Port P4 output	P4OUT	03h
Port P4 direction	P4DIR	05h
Port P4 pullup/pulldown enable	P4REN	07h
Port P4 drive strength	P4DS	09h
Port P4 selection	P4SEL	0Bh
Port P4 interrupt vector word	P4IV	1Eh
Port P4 interrupt edge select	P4IES	19h
Port P4 interrupt enable	P4IE	1Bh
Port P4 interrupt flag	P4IFG	1Dh

● **PxSEL功能选择寄存器**

端口P4~P1的引脚有多种功能，即被MCU内的多个模块复用，功能选择寄存器用于选择引脚的功能

● **PxDIR 方向选择寄存器**

端口4~P1为单向端口，

通过方向寄存器选择各位端口的输入/输出方向

✓ 端口P4~P1作为中断源的引脚功能，
必须设置对应位的PxSEL.y=0和PxDIR.y=0

● PxIES中断边沿选择寄存器 (Interrupt Edge Select)

用于选择端口引脚作为中断申请的有效信号类型

位	7	6	5	4	3	2	1	0
	PxIES.7	PxIES.6	PxIES.5	PxIES.4	PxIES.3	PxIES.2	PxIES.1	PxIES.0
初始值	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

PxIES.y = 0，端口x的引脚y上升沿作为中断申请

PxIES.y = 1，端口x的引脚y下降沿作为中断申请

例 如果P4的8个引脚均作为外部中断源，P4IES的内容为0Fh，

表示端口P4的引脚P4.7~P4.4有上升沿信号时，发出中断申请

端口P4的引脚P4.3~P4.0有下降沿信号时，发出中断申请

● PxIFG中断标志寄存器(Interrupt Flag)

当作为中断功能的端口P4~P1的某引脚上有中断申请信号时，将置PxIFG相应位为1， 相当于锁存该中断申请信号

位	7	6	5	4	3	2	1	0
	PxIFG.7	PxIFG.6	PxIFG.5	PxIFG.4	PxIFG.3	PxIFG.2	PxIFG.1	PxIFG.0
初始值	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

PxIFG.y = 0，端口x的引脚y 无中断申请

PxIFG.y = 1，端口x的引脚y 有中断申请

例 如果P4的8个引脚均作为外部中断源，P4IFG的内容为F0h，

表示端口P4的引脚P4.7~P4.4上有中断申请

端口P4的引脚P4.3~P4.0上无中断申请

● PxIV中断矢量寄存器 (Interrupt Vector) (16位)

P4~P1端口的8个引脚中断源共用一个向量地址，
当有中断产生时，按照中断的来源，PxIV被设置为相应的值。
当多个同时产生，优先设优先级高的中断源值，响应该级中断后，自动设为下一个中断源的值，直至各级中断响应完。

PxIV内容	中断源	中断标志	优先级
0000h	无中断		
0002h	Px.0上有中断	PxIFG.0	最高
0004h	Px.1上有中断	PxIFG.1	
0006h	Px.2上有中断	PxIFG.2	
0008h	Px.3上有中断	PxIFG.3	
000Ah	Px.4上有中断	PxIFG.4	
000Ch	Px.5上有中断	PxIFG.5	
000Eh	Px.6上有中断	PxIFG.6	
0010h	Px.6上有中断	PxIFG.7	最低

● PxIE中断允许寄存器(Interrupt Enable)

是端口P4~P1的分中断允许控制寄存器

位	7	6	5	4	3	2	1	0
	PxIE.7	PxIE.6	PxIE.5	PxIE.4	PxIE.3	PxIE.2	PxIE.1	PxIE.0
初始值	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

PxIE.y = 0，禁止响应端口x的引脚y上的中断

PxIE.y = 1，允许响应端口x的引脚y上的中断

例 当PxSEL.y为0，PxDIR.y为0，Px.y引脚上有中断申请信号，则PxIFG.y置1，

如果PxIE.y=1，表示允许引脚Px.y上的中断申请发向CPU，

若CPU内的GIE=1，则 CPU将响应该中断

如果PxIE.y=0，表示禁止引脚Px.y上的中断申请发向CPU，

则 CPU将不响应该中断

端口P4~P1外部中断过程(以P4.3 为中断源为例)

- 必须设置P4SEL.3=0， P4DIR.3=0
- 根据中断信号设置P4IES.3, 假设置P4IES.3=1,下降沿有效
- 当引脚P4.3有中断申请(如下降沿信号), 则置P4IFG.3=1
- 如果P4IE.3=1, 则允许该中断申请发向CPU,

  若CPU内的GIE=1, 则 CPU将响应该中断;

  若CPU内的GIE=0, 则该中断被屏蔽;

  如果P4IE.3=0, 则禁止该中断申请发向CPU,则该中断被屏蔽

作为中断申请的端口~P1引脚，应设置下列相应寄存器：

1) 设置功能选择寄存器PxSEL.y对应位为0(基本I/O功能)
2) 设置方向选择寄存器PxDIR.y对应位为0(输入)
3) 设置PxIES.y选择中断源有效信号类型是上升还是下降沿
4) 设置PxIE.y打开分中断允许位
5) 设置GIE=1打开总中允许位

在中断程序中：

1) 由于端口的8个引脚共用一个中断向量，

   当有多个引脚做中断源时，

   需利用PxIFG判断产生中断的中断源引脚

2) 在中断子程中应清除PxIFG相应的中断标志位

第3节 MSP430的可屏蔽中断程序设计

一、 编程步骤

1. 主程序流程

2. 中断程序流程

3. 设置中断向量

二、 C语言中断程序举例

一、编程步骤

编程前应了解可屏蔽硬中断的响应过程，

了解有关的寄存器和引脚与中断响应过程的关系

1. 主程序

做好相关设置，

使中断源发出中断申请时CPU能够响应的准备工作

2. 中断程序

处理与中断源有关的关键任务

3. 设置中断向量

根据中断源的类型号在中断向量表相应位置，设置中断向量

1. 主程序

做好相关设置，使中断源发出中断申请时CPU能够响应的准备工作

主程序流程

2. C语言中断程序结构

```
__interrupt  void intName(void)
{
    ……
    ……
    ……
}
```

1. 定义了一个函数名为intName的中断程序
2. 结构上与普通函数的区别是?

   使用了关键字__interrupt

   使得对中断程序进行汇编时,

   返回的语句是RETI, 而不是RET

3. 设置中断向量

确定根据中断源确定中断类型号N,
将中断程序的入口地址放在中断向量表0FFE0h+N*2处

中断源	中断标志	向量地址	优先级 中断类型号
上电,外部复位 看门狗复位 FLASH密码错	WDTIFG, KEYV	0FFFEh	15
…	…	…	…
引脚P1.0~P1.7	P1IFG.0~P1IFG.7	0FFE8h	4
…	…	…	…
引脚P2.0~P2.7	P1IFG.0~P1IFG.7	0FFE2h	1

➢ 头文件 io430x14x.h和 msp430x14x.h

用符号表示各中断源在中断向量表的偏移地址

```
// Interrupt Vectors (offset from 0xFF80)
#define PORT2_VECTOR          (1 * 2u)  /* 0xFFE2 Port 2 */
#define USART1TX_VECTOR       (2 * 2u)  /* 0xFFE4 USART 1 Transmit */
#define USART1RX_VECTOR       (3 * 2u)  /* 0xFFE6 USART 1 Receive */
#define PORT1_VECTOR          (4 * 2u)  /* 0xFFE8 Port 1 */
#define TIMERA1_VECTOR        (5 * 2u)  /* 0xFFEA Timer A CC1-2, TA */
#define TIMERA0_VECTOR        (6 * 2u)  /* 0xFFEC Timer A CC0 */
#define ADC12_VECTOR          (7 * 2u)  /* 0xFFEE ADC */
#define USART0TX_VECTOR       (8 * 2u)  /* 0xFFF0 USART 0 Transmit */
#define USART0RX_VECTOR       (9 * 2u)  /* 0xFFF2 USART 0 Receive */
#define WDT_VECTOR            (10 * 2u) /* 0xFFF4 Watchdog Timer */
#define COMPARATORA_VECTOR    (11 * 2u) /* 0xFFF6 Comparator A */
#define TIMERB1_VECTOR        (12 * 2u) /* 0xFFF8 Timer B CC1-6, TB */
#define TIMERB0_VECTOR        (13 * 2u) /* 0xFFFA Timer B CC0 */
#define NMI_VECTOR            (14 * 2u) /* 0xFFFC Non-maskable */
#define RESET_VECTOR          (15 * 2u) /* 0xFFFE Reset [Highest Priority] */
```

msp430F6638.h 中断向量相关的符号定义

```
#define PORT4_VECTOR        (37 * 1u)          /* 0xFFCA Port 4 */
#define PORT3_VECTOR        (38 * 1u)          /* 0xFFCC Port 3 */
#define TIMER2_A1_VECTOR (39 * 1u)         /* 0xFFCE Timer0_A5 CC1-4, TA */
#define TIMER2_A0_VECTOR   (40 * 1u)             /* 0xFFD0 Timer0_A5 CC0 */
#define DAC12_VECTOR        (41 * 1u)            /* 0xFFD3 DAC12 */
#define RTC_VECTOR          (42 * 1u)            /* 0xFFD4 RTC */
#define LCD_B_VECTOR        (43 * 1u)            /* 0xFFD6 LCD B */
#define PORT2_VECTOR        (44 * 1u)            /* 0xFFD8 Port 2 */
#define USCI_B1_VECTOR      (45 * 1u)            /* 0xFFDA USCI B1 Receive/Transmit */
#define USCI_A1_VECTOR      (46 * 1u)            /* 0xFFDC USCI A1 Receive/Transmit */
#define PORT1_VECTOR        (47 * 1u)            /* 0xFFDE Port 1 */
#define TIMER1_A1_VECTOR (48 * 1u)             /* 0xFFE0 Timer1_A3 CC1-2, TA1 */
#define TIMER1_A0_VECTOR (49 * 1u)             /* 0xFFE2 Timer1_A3 CC0 */
```

msp430F6638.h 中断向量相关的符号定义(续)

```
#define DMA_VECTOR          (50 * 1u)          /* 0xFFE4 DMA */
#define USB_UBM_VECTOR      (51 * 1u)      /* 0xFFE6 USB Timer / cable event / USB reset */
#define TIMER0_A1_VECTOR    (52 * 1u)          /* 0xFFE8 Timer0_A5 CC1-4, TA */
#define TIMER0_A0_VECTOR    (53 * 1u)          /* 0xFFEA Timer0_A5 CC0 */
#define ADC12_VECTOR        (54 * 1u)          /* 0xFFEC ADC */
#define USCI_B0_VECTOR      (55 * 1u)          /* 0xFFEE USCI B0 Receive/Transmit */
#define USCI_A0_VECTOR      (56 * 1u)          /* 0xFFF0 USCI A0 Receive/Transmit */
#define WDT_VECTOR          (57 * 1u)          /* 0xFFF2 Watchdog Timer */
#define TIMER0_B1_VECTOR    (58 * 1u)          /* 0xFFF4 Timer0_B7 CC1-6, TB */
#define TIMER0_B0_VECTOR    (59 * 1u)          /* 0xFFF6 Timer0_B7 CC0 */
#define COMP_B_VECTOR       (60 * 1u)          /* 0xFFF8 Comparator B */
#define UNMI_VECTOR         (61 * 1u)          /* 0xFFFA User Non-maskable */
#define SYSNMI_VECTOR       (62 * 1u)          /* 0xFFFC System Non-maskable */
#define RESET_VECTOR        (63 * 1u)          /* 0xFFFE Reset [Highest Priority] */
```

C语言程序设置中断向量方法

在中断程序前使用预编译命令#pragma vector=偏址 语句，
将中断程序的入口地址放入到FFE0+偏址的中断向量表中

```
#pragma  vector=N*2                    //使用中断类型号计算偏址
__interrupt  void intName(void)
{
        ……
        ……
}
```

```
#pragma  vector=PORT1_VECTOR    //使用符号表示的中断偏址
__interrupt  void intName(void)
{
        ……
        ……
}
```

C语言：开/关总中断控制位函数

(disable/enable general interrupt bit)

一些操作CPU内部状态寄存器的函数，包含在CCS 编译器里，称内部函数，在in430.h中做了声明，用户可以使用。

in430.h

```
void _enable_interrupts(void);
void _disable_interrupts(void);
unsigned short _bic_SR_register(unsigned short mask);
unsigned short _bic_SR_register_on_exit(unsigned short mask);
……
unsigned short _get_SR_register_on_exit(void);
unsigned short _swap_bytes(unsigned short src);
void _nop(void);
void _never_executed(void);
#define _EINT()                    _enable_interrupts()
#define _DINT()                    _disable_interrupts()
……
#define __enable_interrupt()       _enable_interrupts()
#define __disable_interrupt()      _disable_interrupts()
```

C语言：开/关总中断控制位函数

(disable/enable general interrupt bit)

函数名称	功能	包含在
_disable_interrupt( )	0→ GIE	in430.h
_ensable_interrupt( )	1→ GIE	in430.h
_DINT()	0→ GIE	in430.h
_EINT()	1→ GIE	in430.h
__disable_interrupt( )	0→ GIE	in430.h
__ensable_interrupt( )	1→ GIE	in430.h

头文件 msp430F6638.h 中包含了in430.h

```
……
#include   "in430.h "
……
```

用C语言编写中断程序方法1

使用_disable_interrupt( ) 和_enable_interrupt( )

```
#include        “msp430F6638.h"
int main( void )
{     //Stop watchdog timer to prevent time out reset
      WDTCTL = WDTPW + WDTHOLD;
            _disable_interrupt();   //关总中断控制(非必要)
            ……                                  //主程序初始化准备工作
            ……
            _enable_interrupt();                //开总中断控制
      while(1){     };                          //主程序循环
}

#pragma  vector=数字或符号表示的类型号  //中断向量设置
__interrupt void port_int(void)                 //中断子程
{
     ……
}
```

用C语言编写中断程序方法1

使用__disable_interrupt( ) 和__enable_interrupt( )

```
#include        “msp430F6638.h"
int main( void )
{     //Stop watchdog timer to prevent time out reset
      WDTCTL = WDTPW + WDTHOLD;
            __disable_interrupt();          //关总中断控制(非必要)
            ……                              //主程序初始化准备工作
            ……
            __enable_interrupt();           //开总中断控制
      while(1){    };                       //主程序循环
}

#pragma  vector=数字或符号表示的类型号  //中断向量设置
__interrupt void port_int(void)                 //中断子程
{
    ……
}
```

用C语言编写中断程序方法2

使用_DINT( ) 和_EINT ( )

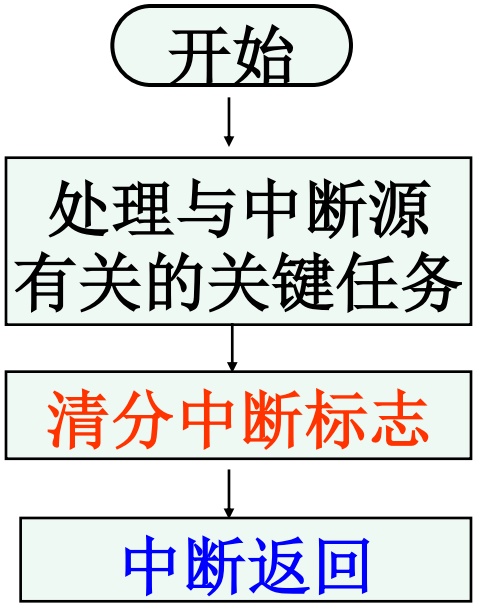
```
#include        “msp430F6638.h"
int main( void )
{     //Stop watchdog timer to prevent time out reset
      WDTCTL = WDTPW + WDTHOLD;
           _DINT( );                    //关总中断控制(非必要)
           ……                                  //主程序初始化准备工作
           ……
           _EINT( );                    //开总中断控制
      while(1){    };                          //主程序循环
}

#pragma  vector=数字或符号表示的类型号  //中断向量设置
__interrupt void port_int(void)                 //中断子程
{
    ……
}
```

可屏蔽中断程序设计

1. 主程序

2. 中断子程



3. 设置中断向量

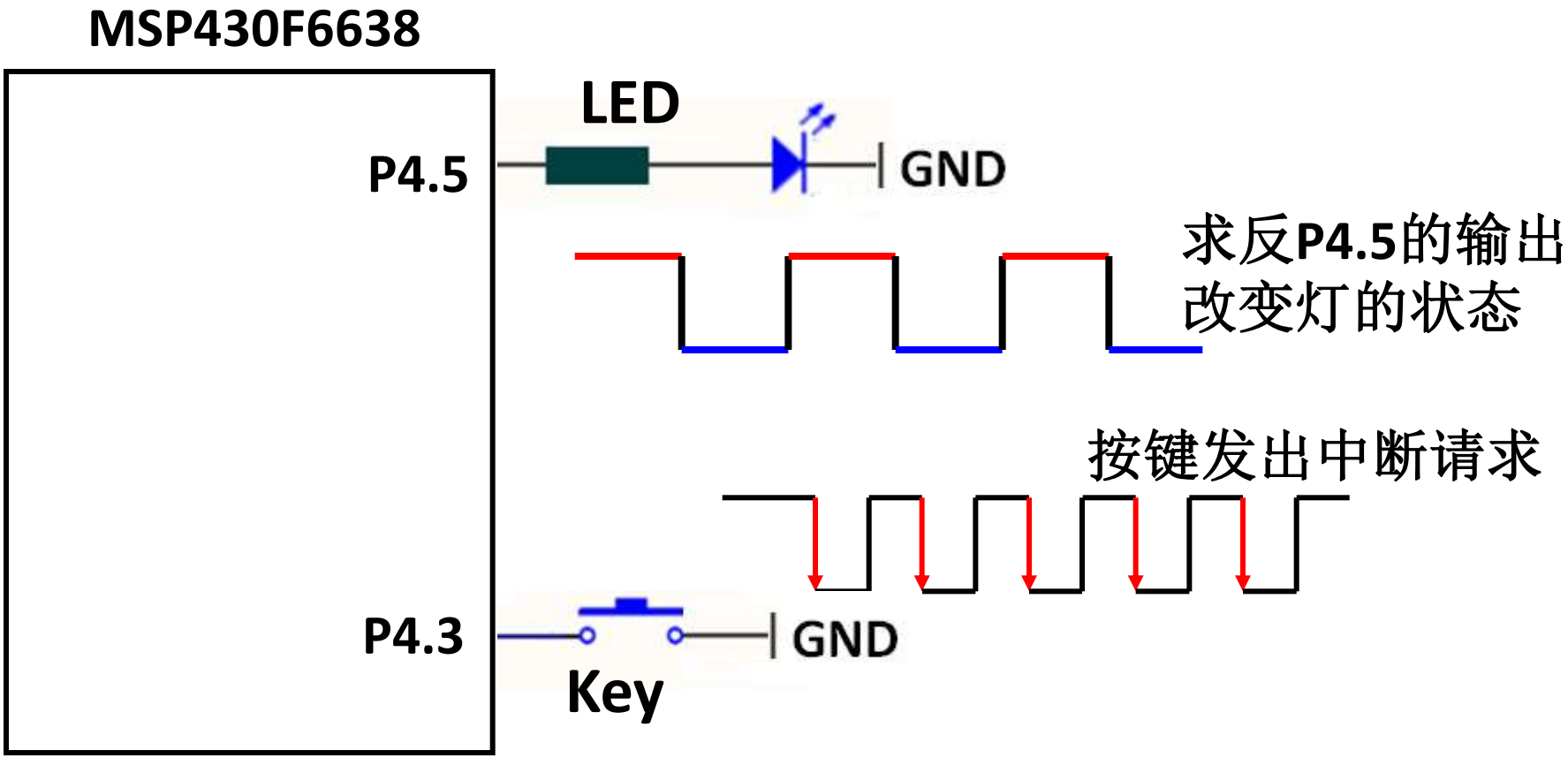
根据中断源在
中断向量表相应位置
设置中断向量

例： 中断编程举例 (以P4.3上的中断为例)

请用C语言编写程序，

以中断方式响应P4.3上的按键，

每按下一次键，改变一次4.5上的发光二极管状态。



例1 程序清单

```
#include        “msp430F6638.h"
int main( void )
{       WDTCTL = WDTPW + WDTHOLD;          //关闭看门狗
        _DINT();                           //禁止可屏蔽中断 GIE=0
        P4DIR |= BIT5;                     //设置P4.5口方向为输出
        P4DIR &= ~BIT3;  P4REN |= BIT3;          //使能P4.3上拉电阻
        P4OUT |= BIT3;                     //P4.3口置高电平
        P4IES |= BIT3;                     //中断沿设置（下降沿触发）
        P4IFG &= ~BIT3;                    //清P4.3中断标志
        P4IE |= BIT3;                      //使能P4.3口中断
        _EINT();                           //开中断
        while(1){ };                       //无限循环
}

#pragma vector=PORT4_VECTOR                // P4中断函数
__interrupt void Port_4(void)
{       P4OUT ^= BIT5;                     //改变LED5灯状态
        P4IFG &= ~BIT3;                    //清P4.3中断标志位
}
```

思考：为什么要使能P4.3的上拉电阻？

例1 程序清单

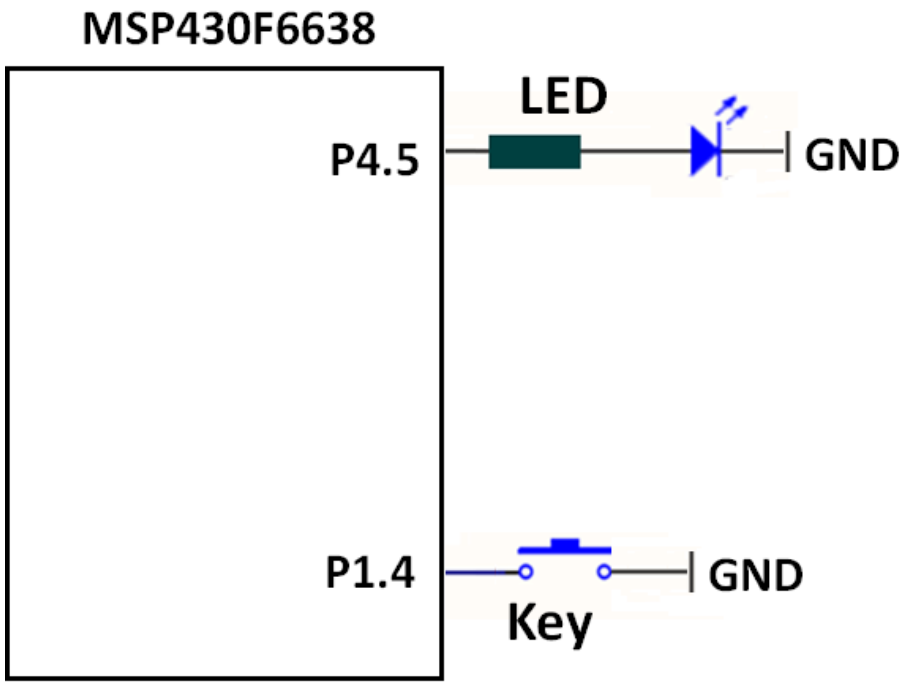
```
#include        “msp430F6638.h"
int main( void )
{       WDTCTL = WDTPW + WDTHOLD;          //关闭看门狗
        _DINT();                           //禁止可屏蔽中断 GIE=0
        P4DIR |= BIT5;                     //设置P4.5口方向为输出
        P4DIR &= ~BIT3;  P4REN |= BIT3;          //使能P4.3上拉电阻
        P4OUT |= BIT3;                     //P4.3口置高电平
        P4IES |= BIT3;                     //中断沿设置（下降沿触发）
        P4IFG &= ~BIT3;                    //清P4.3中断标志
        P4IE |= BIT3;                      //使能P4.3口中断
        _EINT();                           //开中断
        while(1){ };                       //无限循环
}

#pragma vector=PORT4_VECTOR                // P4中断函数
__interrupt void Port_4(void)
{       P4OUT ^= BIT5;                     //改变LED5灯状态
        P4IFG &= ~BIT3;                    //清P4.3中断标志位
}
```

思考：为什么要使能P4.3的上拉电阻？

思考:

1. 中断程序何时被执行?执行过程是怎样?
2. 如何将程序改写为查询方式, 即查询到P4.3 有键按下, 就对P4.5求反?
3. 比较中断方式和查询方式有何不同?
4. 若中断源来自引脚P1.4, 如何修改程序?



注意:

- 若有多个中断同时请求，CPU选择优先级最高的中断请求先进行响应
- 当中断源产生时，MSP430内部对应有一个标志被置位
  中断程序之后，应确保该标志的 值已清零，
  否则被当成又一次的中断申请
- 对于单一中断标志的中断源请求，
  CPU会自动清零该中断标志
- 对于有多个中断标志的中断源请求，
  用户在中断子程用这些标志判断产生的具体子中断源，
  中断标志的清零由用户在使用完后编程清零

中断源	中断标志	向量地址	优先级,类型号
…	…	…	…
P1的8个引脚	P1IFG.0~P1IFG.7 (8)	0FFDEh	47
…	…	…	…
P2的8个引脚	P2IFG.0~P2IFG.7 (8)	0FFD8h	44
…	…	…	…
P3的8个引脚	P3IFG.0~P3IFG.7 (8)	0FFCCh	38
P4的8个引脚	P4IFG.0~P4IFG.7 (8)	0FFCAh	37

思考:

在例子中的中断子程内不清P4IFG.3中断标志的后果？

```
#pragma vector=PORT4_VECTOR    // P4中断函数
__interrupt void Port_4(void)
{           P4OUT ^= BIT5;                    //改变LED5灯状态
//          P4IFG &= ~BIT3;                   //清P4.3中断标志位
}
```

思考:

当有多个按键中断时，需要在中断子程中加入判断是哪个中断源。

利用中断标志判断

```
#pragma vector=PORT4_VECTOR    // P4中断函数

__interrupt void Port_4(void)
{           if  (P1IFG_bit.P0==1)                    //判断是否是P1IFG.0中断标志
            {           P4OUT ^= BIT5;                          //改变LED5灯状态

                        P4IFG &= ~BIT3;                         //清P4.3中断标志位

            }

}
```

利用中断矢量寄存器判断

```
#pragma vector=PORT4_VECTOR    // P4中断函数

__interrupt void Port_4(void)
{           if  (P4IV==8)                  //判断是否是P1IFG.0中断标志
           {          P4OUT ^= BIT5;                          //改变LED5灯状态
           //         P4IFG &= ~BIT3;                         //清P4.3中断标志位
           }
}
```

进入中断子程后，读取一次中断矢量寄存器，
将自动将其对应的中断标志位清0，
故在中断子程中可不必再清中断标志。