第5章 中断技术

学习要点

1. 了解中断的基本概念
2. 掌握MSP430G2xxx中断系统
3. 掌握MSP430G2553端口P1和P2的外中断
4. 掌握MSP430可屏蔽中断程序设计方法

第五章 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

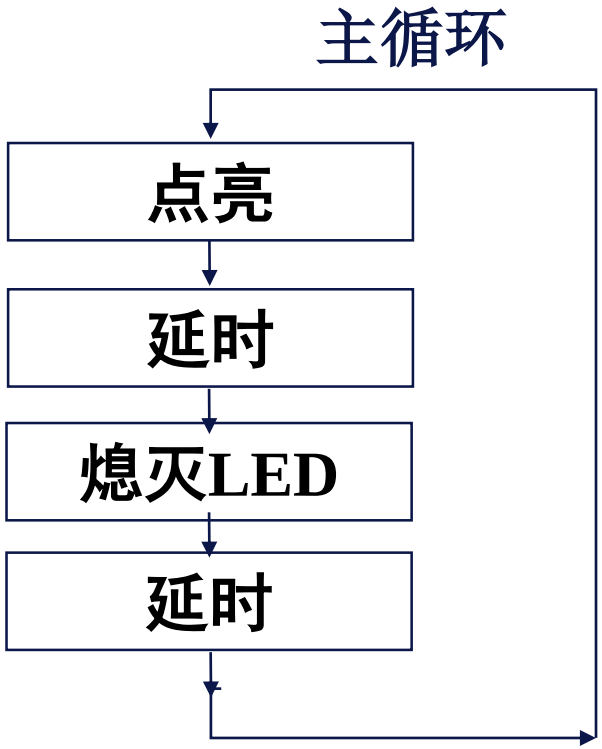
程序例：LED灯闪烁程序

```
//控制MSP430G2板（LaunchPad板）上LED1和LED2闪烁
#include "msp430.h"
void delay( );
int main ( void )
{ WDTCTL = WDTPW + WDTHOLD;          //关闭看门狗
  P1SEL=0x00;P1SEL2=0x00;//P1为基本输入输出功能
  P1DIR |= BIT0+BIT6;P1OUT &= ~(BIT0+BIT6);   //P1.0,P1.6为输
出方向，初始置0
  while(1)                           //无限循环
  { P1OUT|=BIT0+BIT6;                //点亮LED1和LED2
    delay();                         //调用延时子程序
    P1OUT&=~(BIT0+BIT6);//灭LED1和LED2
    delay();                         //调用延时子程
  }
}
 void delay( )
{ unsigned  long  i;                 //定义函数变量
  for ( i=0; i<100000;  i++ );   //延时
}
```

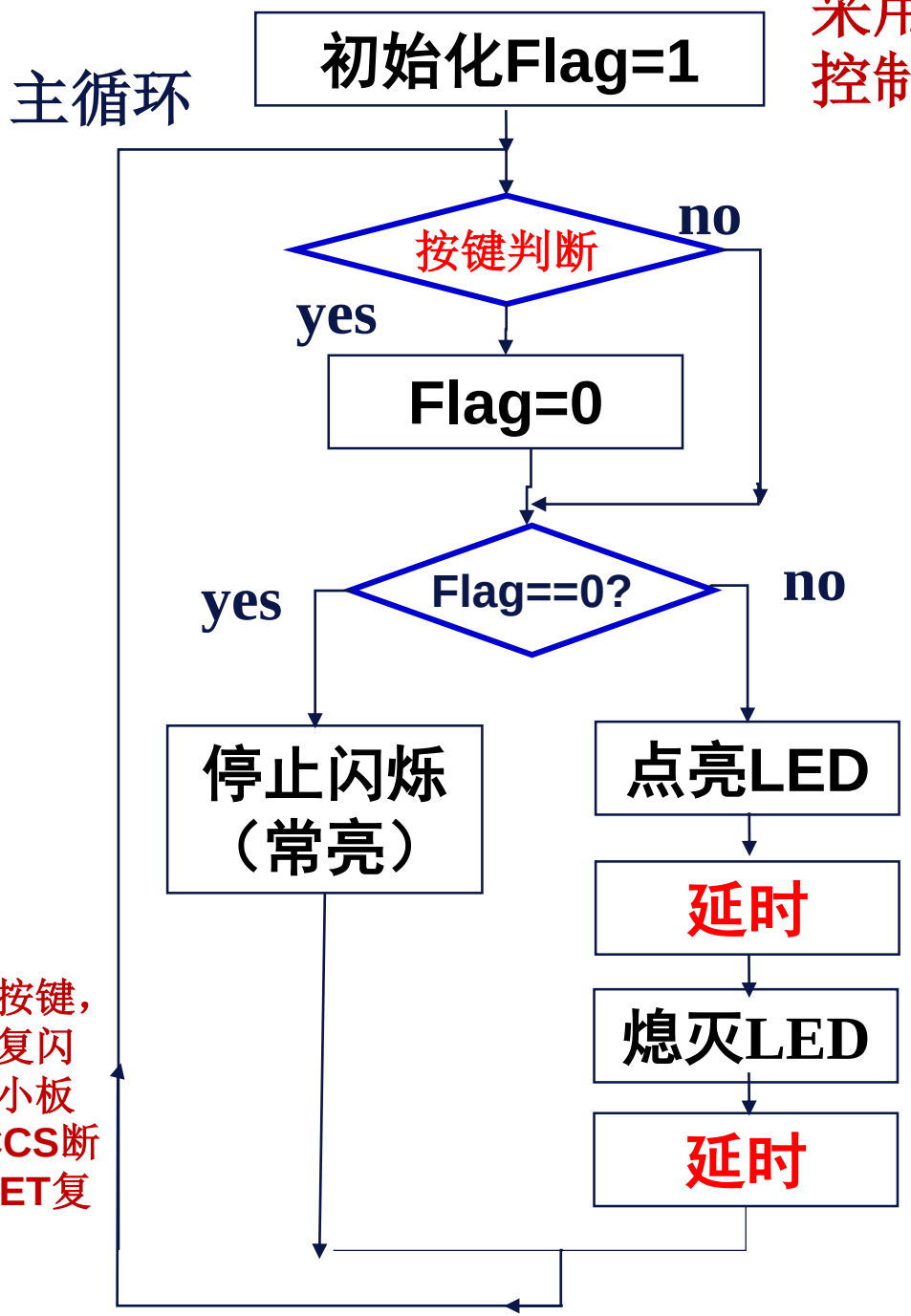
如果希望用按键控制LED等的闪烁应该怎么编程？

例如：按S2键(P1.3)时，LED灯停止闪烁，变成全亮

不用按键控制时



采用按键控制时



注意：此示例因为因为只采用了一个按键，按键按下后就停止闪烁，没有设计恢复闪烁的功能。如果恢复闪烁，需要通过小板上的RESET（S1）键复位程序（与CCS断开状态时），或者通过CCS中的RESET复位程序（与CCS连接状态时）

C5_0_a.c

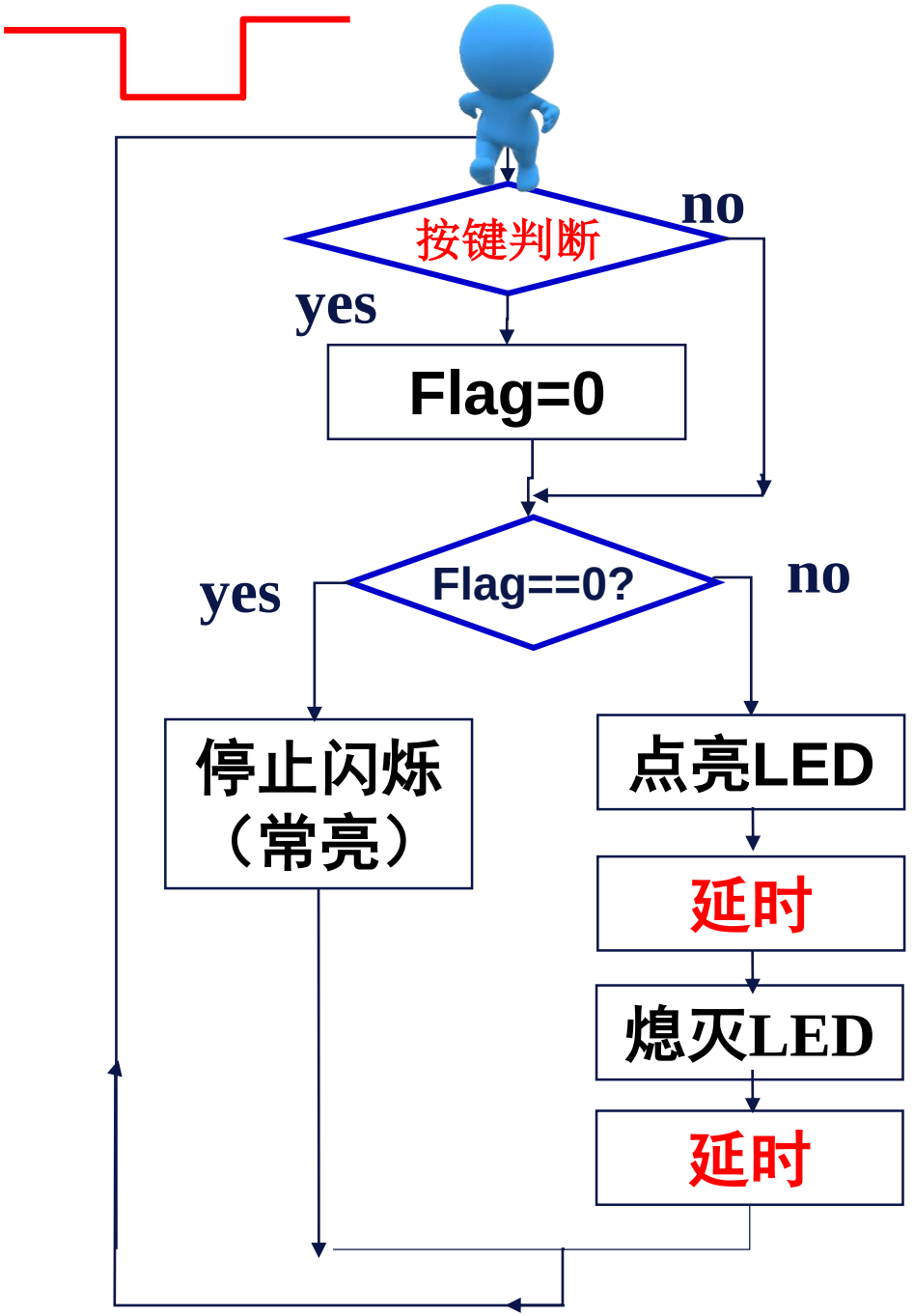
```
#include "msp430.h"
void delay( );
int main ( void )
{ unsigned  int  Flag; //定义闪烁标志位
  WDTCTL = WDTPW + WDTHOLD;  //关闭看门狗
  P1SEL=0x00;P1SEL2=0x00;//P1为基本输入输出功能
  P1DIR |= BIT0+BIT6;P1OUT &= ~(BIT0+BIT6);  //P1.0,P1.6为输出方向
  P1DIR &= ~BIT3;P1REN |=BIT3; P1OUT |=BIT3; //设置P1.3(S2键）为输
入，拉电阻使能，上拉方式
  Flag=1;
  while(1)                    //无限循环
  { if ( (P1IN&BIT3) ==0) //如果按下S2，标志位变化
      {Flag=0;}
    else
    {
      if (Flag==0) {P1OUT|=BIT0+BIT6;} //闪烁标志位为0，停止闪烁，全亮
      else          //闪烁标志位为1，闪烁
      { P1OUT|=BIT0+BIT6;  //点亮LED1和LED2
        delay();           //调用延时子程序
        P1OUT&=~(BIT0+BIT6);//灭LED1和LED2
        delay();           //调用延时子程
      }
    }
 }
}
void delay( )
{ unsigned  long  i;                  //定义函数变量
  for ( i=0; i<100000;  i++ );  //延时
}
```

用按键控制LED等的闪烁的程序例

这个程序能否实现所期望的功能？是否存在问题？

采用简单查询方式

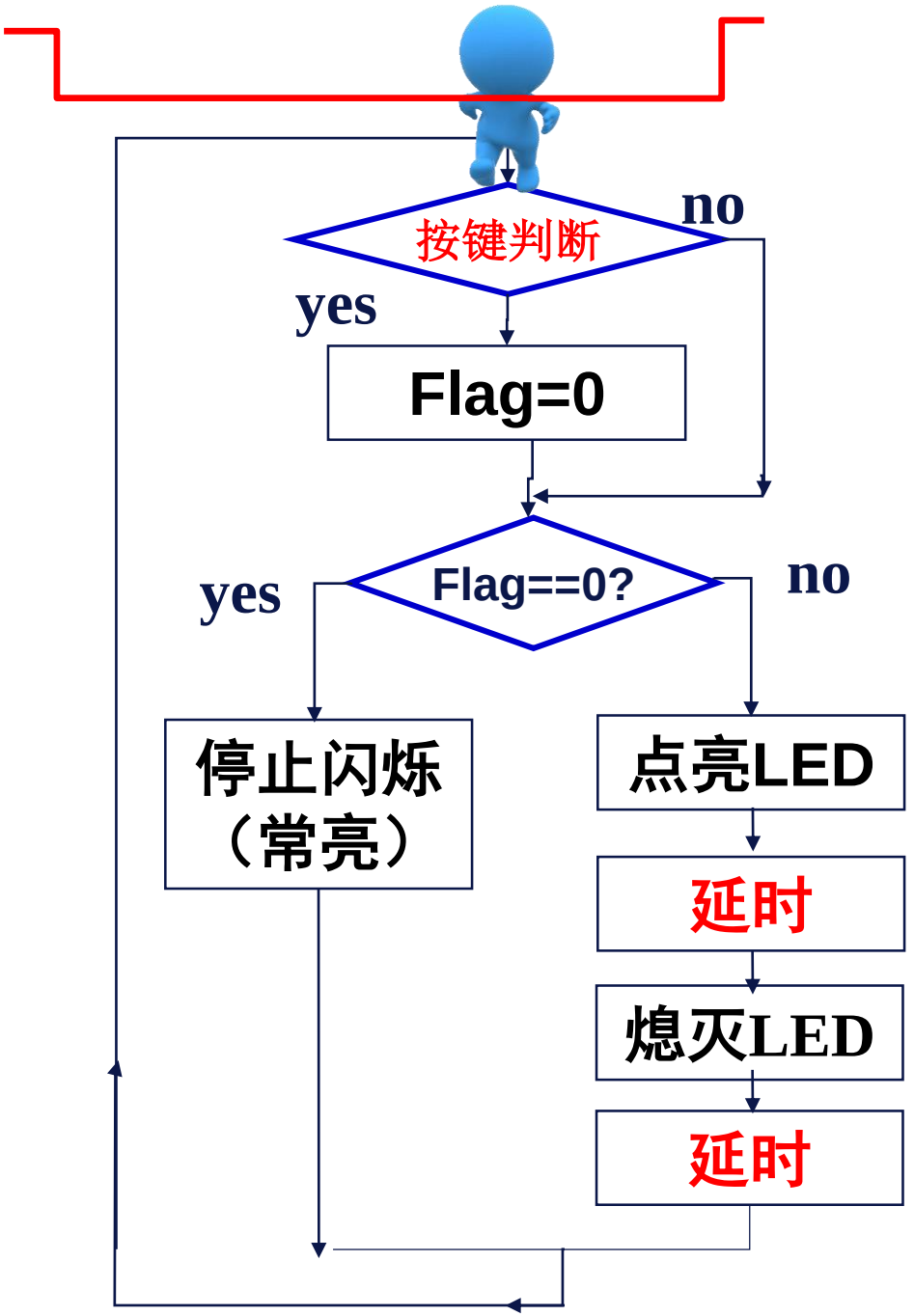
按键时间比较短时



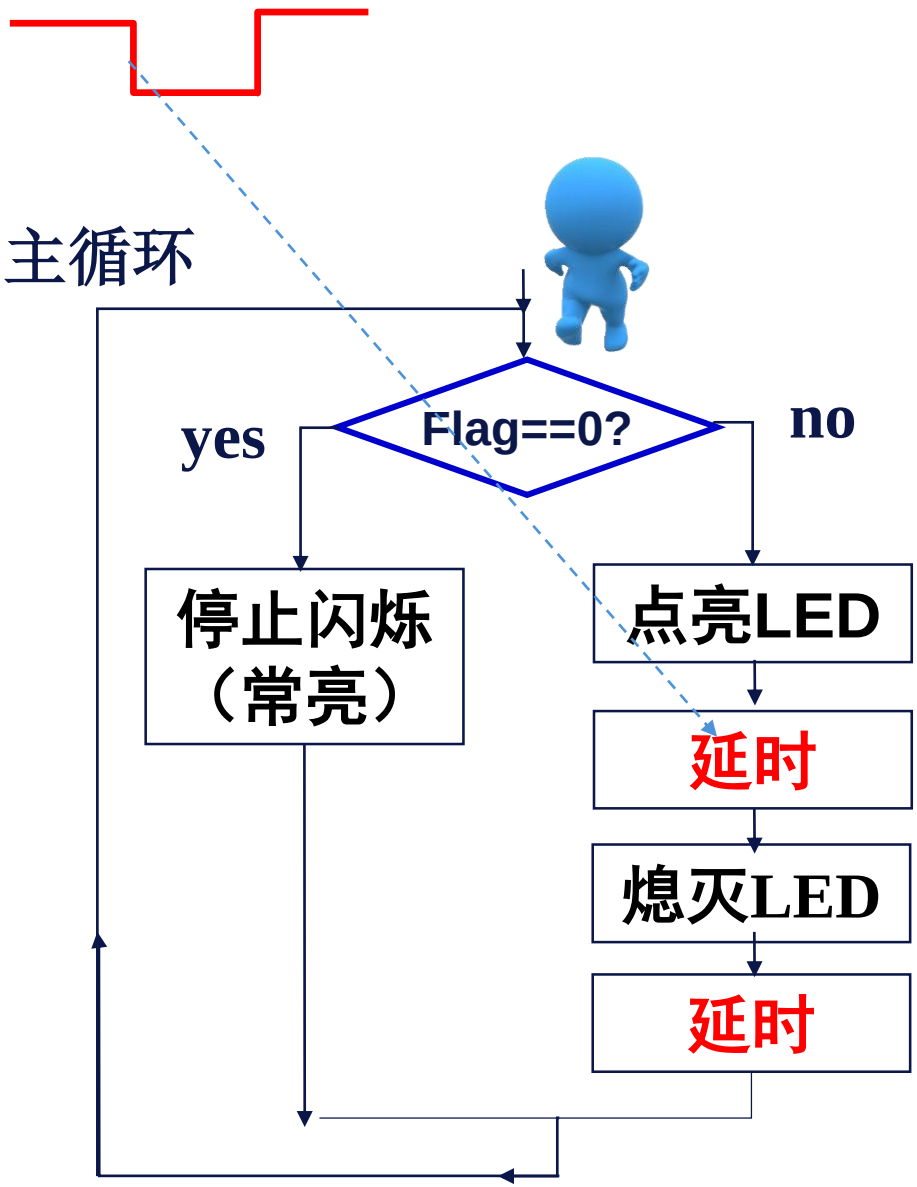
采用简单查询方式

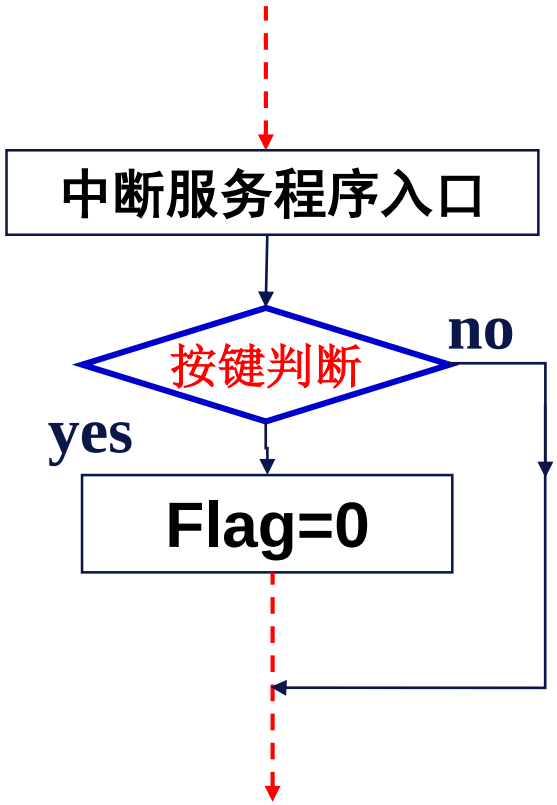
加长按键时间

有没有更好的方式？



中断方式


主循环
Flag==0?
yes
no
停止闪烁
（常亮）
点亮LED
延时
熄灭LED
延时


中断服务程序入口
按键判断
no
yes
Flag=0

一、什么是中断

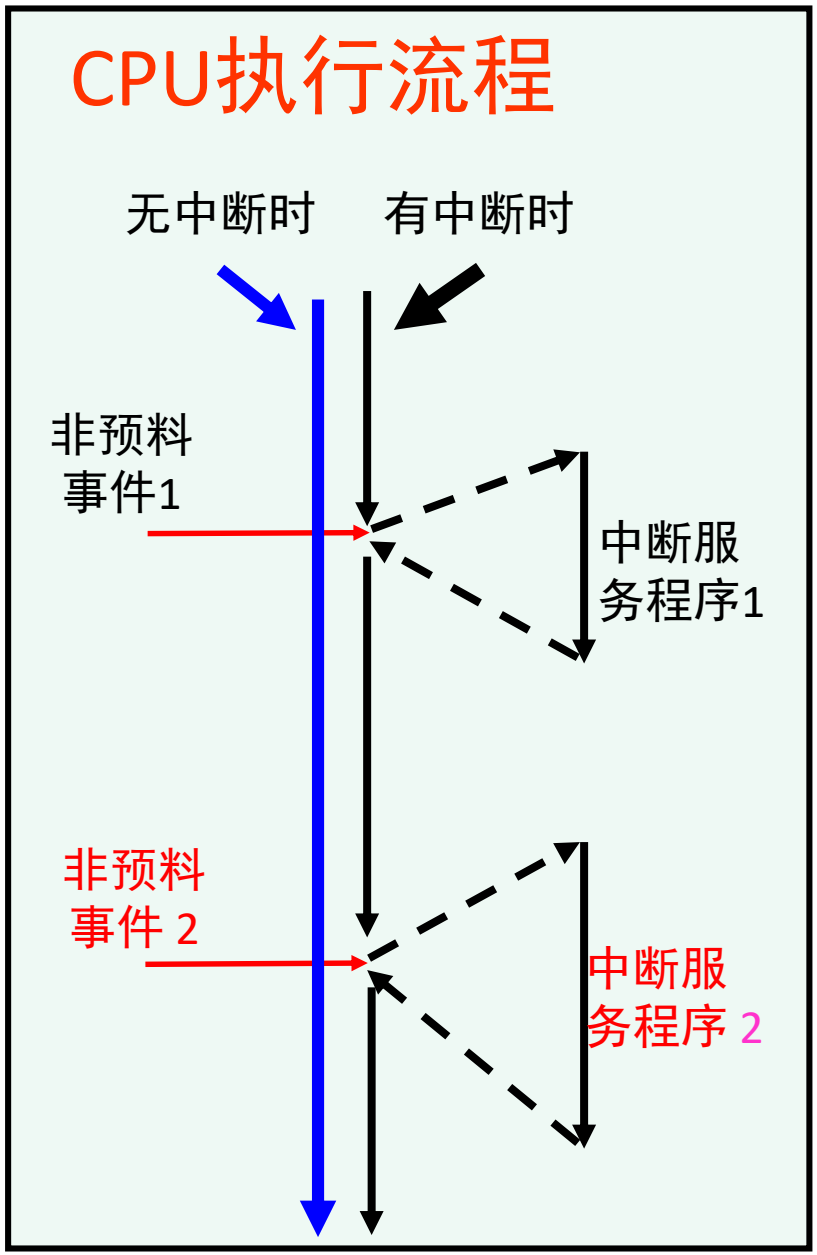
在**CPU**正常运行程序时，

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。



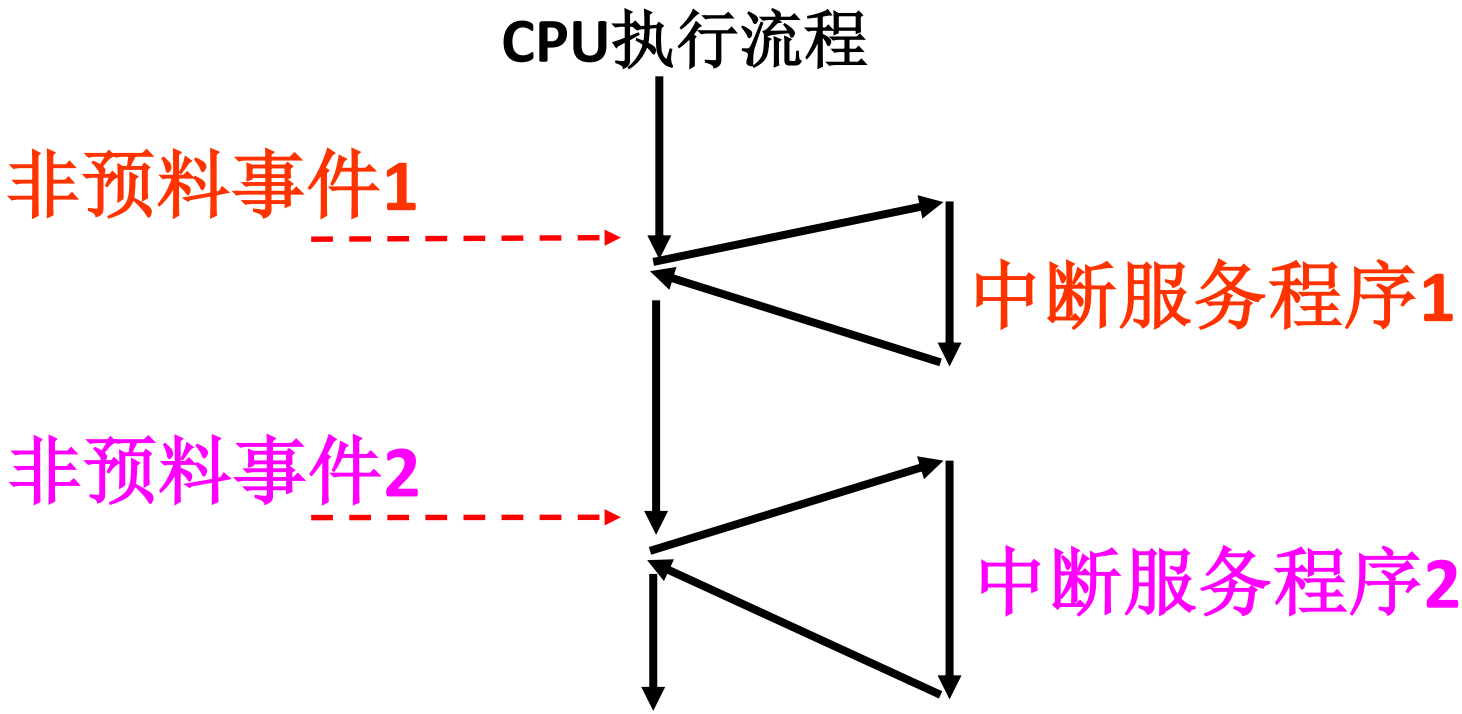
二、中断源和中断优先级

- **中断源：**发出中断申请的外设或内部因素（引起中断的因素很多）
- **中断优先级：**给每个中断源指定一个优先权级别
  - 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。

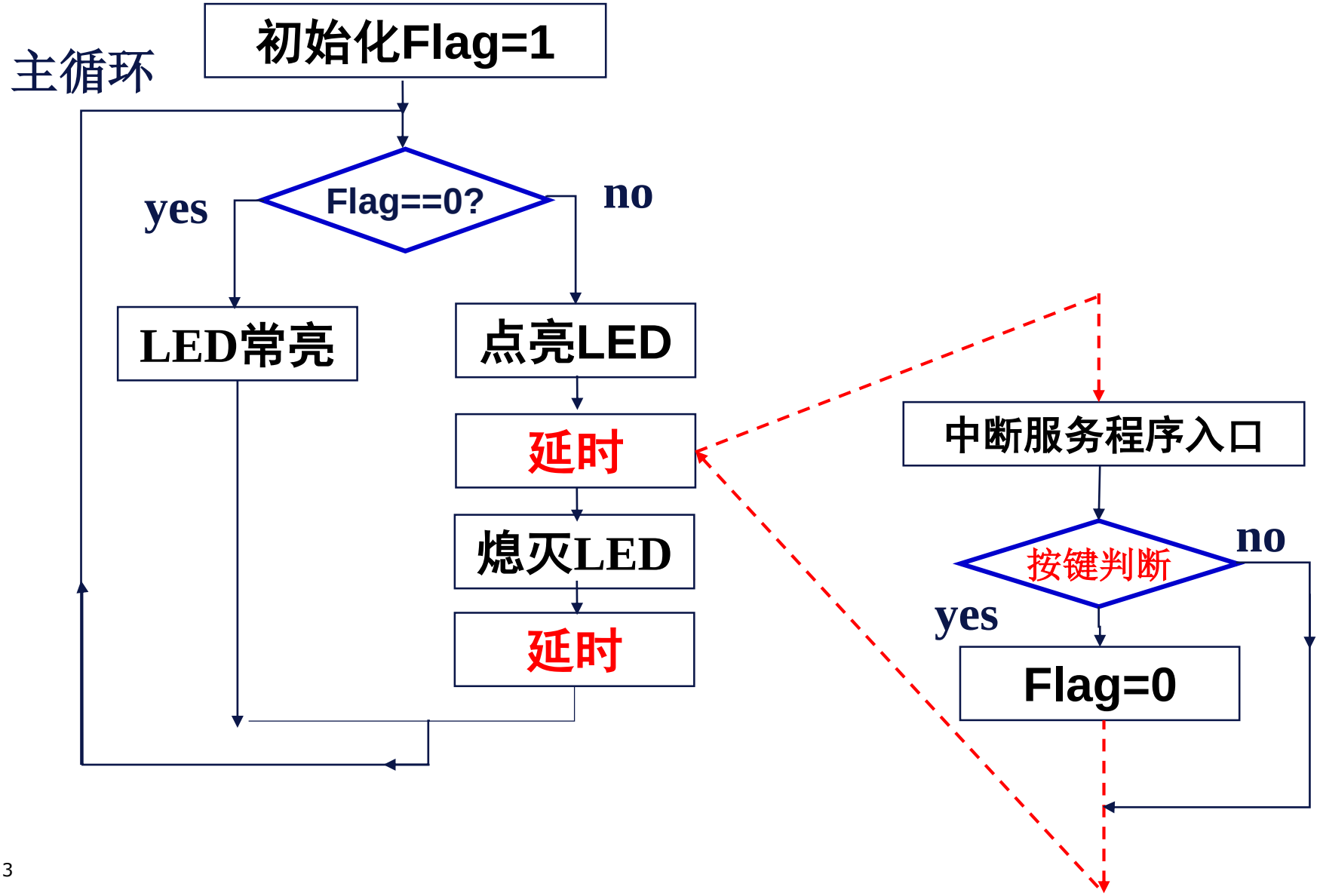
三、中断服务程序和中断向量

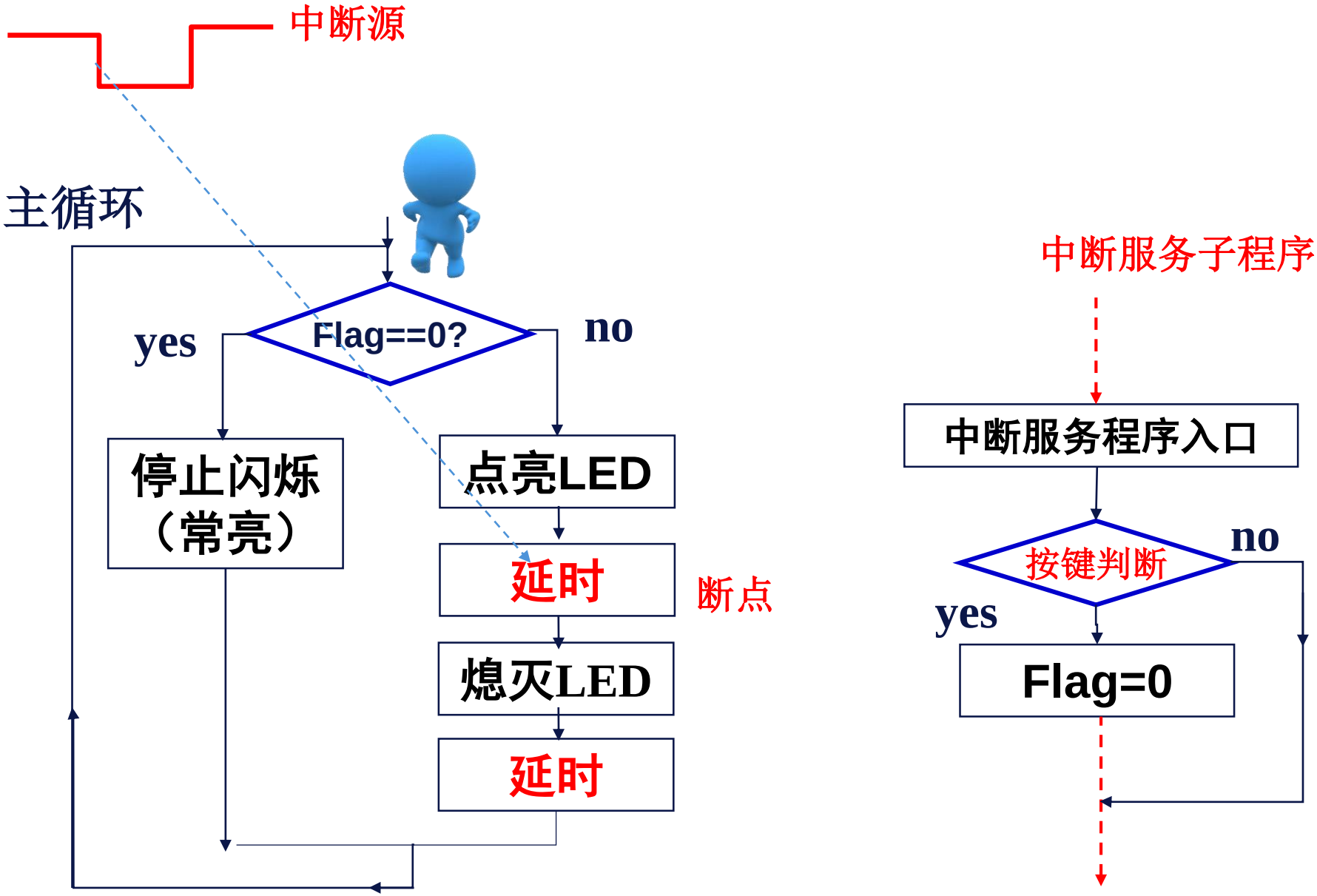
- **中断服务程序：**处理中断源，完成其所要求功能的程序，(中断例行程序、中断子程)。
- **中断向量：**中断程序的入口地址称为

即中断程序第1条指令存放在存储器的位置。



用中断实现按键控制LED闪烁功能：流程图




中断源
主循环
yes
Flag==0?
no
停止闪烁
（常亮）
点亮LED
延时
断点
熄灭LED
延时
中断服务子程序
中断服务程序入口
按键判断
no
yes
Flag=0

用中断实现按键控制MSP430G2板（LaunchPad板）LED闪烁功能的程序例

C5_0_b.c

```
#include "msp430.h"
void delay( );
unsigned  int  Flag; //定义闪烁标志位
int main ( void )
{ WDTCTL = WDTPW + WDTHOLD;          //关闭看门狗
  _DINT();                          //禁止可屏蔽中断 GIE=0
  P1SEL=0x00;P1SEL2=0x00;//P1为基本输入输出功能
  P1DIR |= BIT0+BIT6;  //P1.0,P1.6为输出方向，初始置0
  P1DIR &= ~BIT3;P1REN |=BIT3; P1OUT |=BIT3; //设置P1.3(S2键）为输入，拉电阻使能，上拉方式
  P1IES |=BIT3; //置P1.3下降沿作中断源
  P1IFG =0;       //清P1IFG中断标志
  P1IE |=BIT3;   //打开P1.3,
  _EINT();       //允许可屏蔽中断 GIE=1
  Flag=1;
  while(1)      //无限循环
  {
    if (Flag==0) {P1OUT|=BIT0+BIT6;} //闪烁标志位为0，停止闪烁，全亮
    else  //闪烁标志位为1，闪烁
    { P1OUT|=BIT0+BIT6;     //点亮LED1和LED2
      delay();              //调用延时子程序
      P1OUT&=~(BIT0+BIT6);//灭LED1和LED2
      delay();              //调用延时子程
    }
   }
}
```

```
void delay( )
{ unsigned  long  i;                    //定义函数变量
  for ( i=0; i<100000;  i++ );   //延时
}

#pragma  vector=PORT1_VECTOR   //置P1中断向量
__interrupt  void port_int(void)//中断子程
{  if  ( (P1IFG&BIT3)!=0 )      //判断是否是P1IFG.3中断标志
   {  Flag=0;                  //停止闪烁标志位
      P1IFG &=~BIT3;           //清P1.3中断标志
    }
 }
```

可以在CCS中观察反汇编指令和中断向量

程序存储区

```
      main():
c000:   40B2 5A80 0120   MOV.W   #0x5a80,&Watchdog_Time
c006:   C232             DINT
c008:   4303             NOP
c00a:   43C2 0026        CLR.B   &Port_1_2_P1SEL
c00e:   43C2 0041        CLR.B   &Port_1_2_P1SEL2
c012:   D0F2 0041 0022   BIS.B   #0x0041,&Port_1_2_P1DI
c018:   C2F2 0022        BIC.B   #8,&Port_1_2_P1DIR
c01c:   D2F2 0027        BIS.B   #8,&Port_1_2_P1REN
c020:   D2F2 0021        BIS.B   #8,&Port_1_2_P1OUT
c024:   D2F2 0024        BIS.B   #8,&Port_1_2_P1IES
c028:   43C2 0023        CLR.B   &Port_1_2_P1IFG

c0b4:   2C0C             JHS     ($C$L4)
      $C$L3:
c0b6:   5391 0000        INC.W   0x0000(SP)
c0ba:   6381 0002        ADC.W   0x0002(SP)
c0be:   9391 0002        CMP.W   #1,0x0002(SP)
c0c2:   2BF9             JLO     ($C$L3)
c0c4:   2004             JNE     ($C$L4)
c0c6:   90B1 86A0 0000   CMP.W   #0x86a0,0x0000(SP)
c0cc:   2BF4             JLO     ($C$L3)
      $C$L4:
c0ce:   5221             ADD.W   #4,SP
c0d0:   4130             RET
```

中断向量表

0xffff

0xffde - 0xffff(-0x2...

16-Bit Hex - C Style

0xFFDE	0xFFFF
0xFFE0	0xC122
0xFFE2	0xFFFF
0xFFE4	0xC102
0xFFE6	0xC122
0xFFE8	0xFFFF
0xFFEA	0xC122
0xFFEC	0xC122
0xFFEE	0xC122
0xFFF0	0xC122

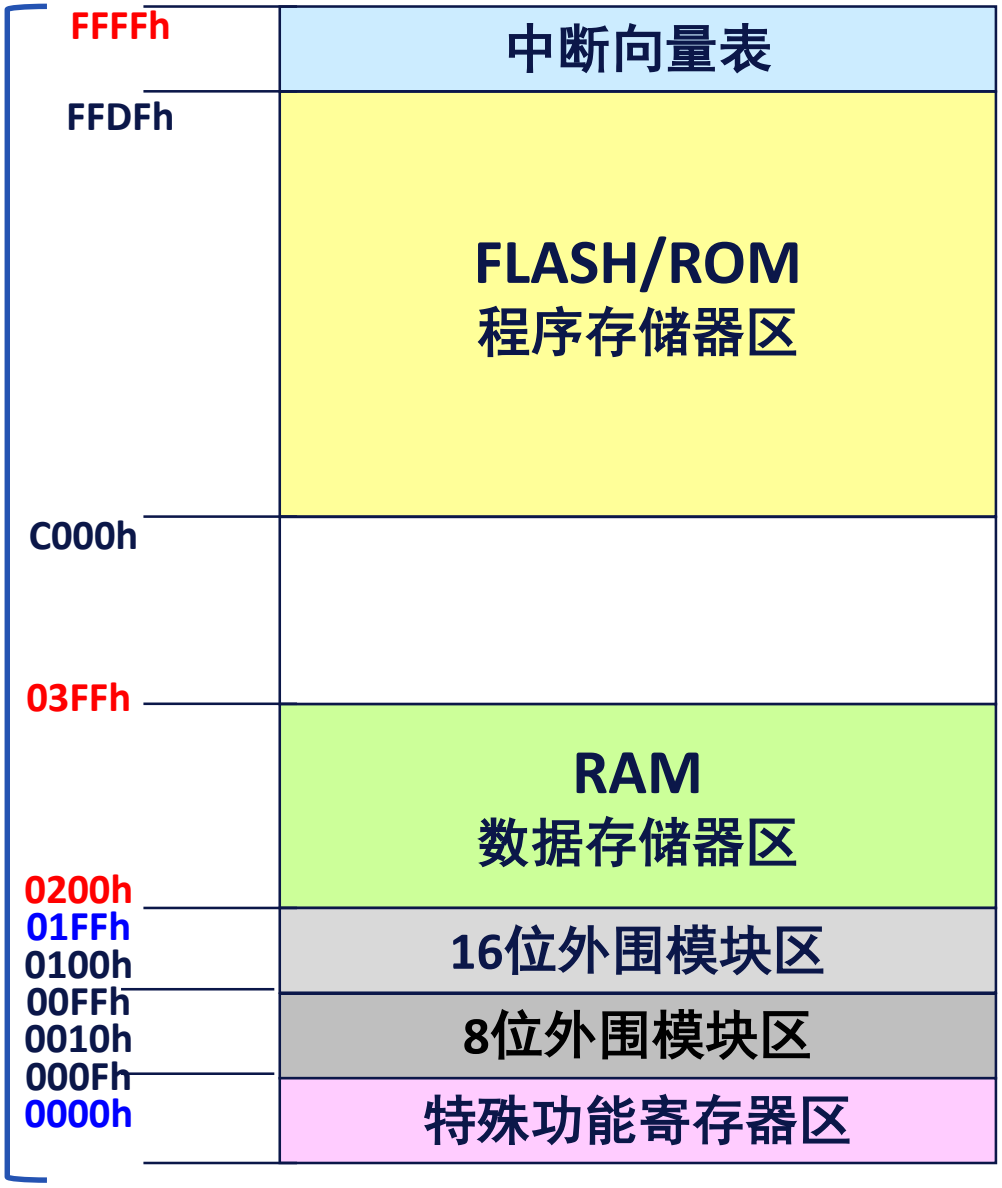
PC

程序存储区

```
      port_int():
c102:   B2F2 0023        BIT.B   #8,&Port_1_2_P1IFG
c106:   2404             JEQ     ($C$L5)
c108:   4382 0200        CLR.W   &Flag
c10c:   C2F2 0023        BIC.B   #8,&Port_1_2_P1IFG
      $C$L5:
c110:   1300             RETI
```

msp430G2553

的存储器结构



可以在CCS中观察反汇编指令和中断向量

程序存储区

```
      main():
c000:   40B2 5A80 0120   MOV.W   #0x5a80,&Watchdog_Time
c006:   C232             DINT
c008:   4303             NOP
c00a:   43C2 0026        CLR.B   &Port_1_2_P1SEL
c00e:   43C2 0041        CLR.B   &Port_1_2_P1SEL2
c012:   D0F2 0041 0022   BIS.B   #0x0041,&Port_1_2_P1DI
c018:   C2F2 0022        BIC.B   #8,&Port_1_2_P1DIR
c01c:   D2F2 0027        BIS.B   #8,&Port_1_2_P1REN
c020:   D2F2 0021        BIS.B   #8,&Port_1_2_P1OUT
c024:   D2F2 0024        BIS.B   #8,&Port_1_2_P1IES
c028:   43C2 0023        CLR.B   &Port_1_2_P1IFG

        ⋮                ⋮              ⋮

c0b4:   2C0C             JHS     ($C$L4)
      $C$L3:
c0b6:   5391 0000        INC.W   0x0000(SP)
c0ba:   6381 0002        ADC.W   0x0002(SP)
c0be:   9391 0002        CMP.W   #1,0x0002(SP)
c0c2:   2BF9             JLO     ($C$L3)
c0c4:   2004             JNE     ($C$L4)
c0c6:   90B1 86A0 0000   CMP.W   #0x86a0,0x0000(SP)
c0cc:   2BF4             JLO     ($C$L3)
      $C$L4:
c0ce:   5221             ADD.W   #4,SP
c0d0:   4130             RET
```

中断向量表

0xffff

0xffde - 0xffff(-0x2...

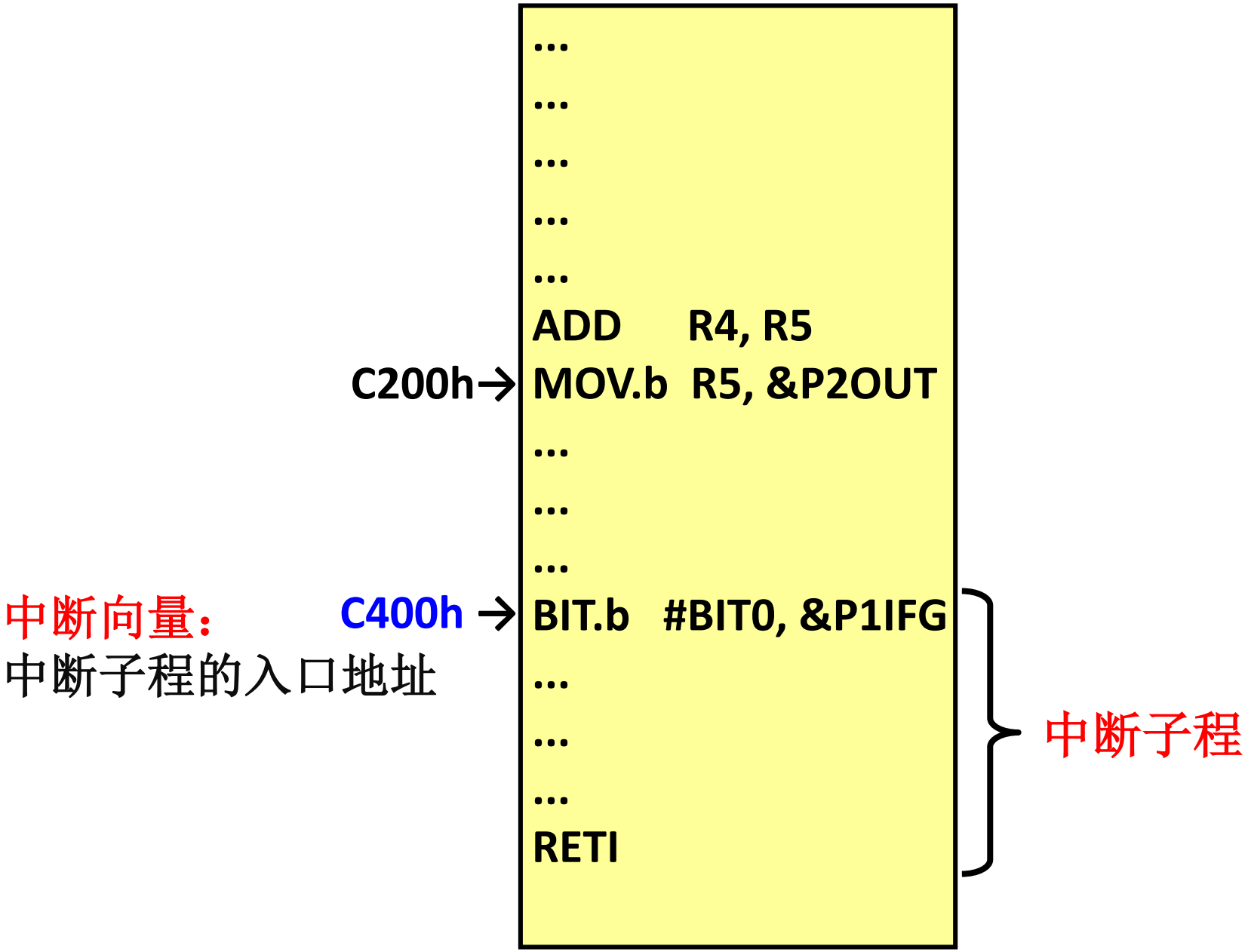
16-Bit Hex - C Style

0xFFDE	0xFFFF
0xFFE0	0xC122
0xFFE2	0xFFFF
0xFFE4	0xC102
0xFFE6	0xC122
0xFFE8	0xFFFF
0xFFEA	0xC122
0xFFEC	0xC122
0xFFEE	0xC122
0xFFF0	0xC122

PC

程序存储区

```
      port_int():
c102:   B2F2 0023        BIT.B   #8,&Port_1_2_P1IFG
c106:   2404             JEQ     ($C$L5)
c108:   4382 0200        CLR.W   &Flag
c10c:   C2F2 0023        BIC.B   #8,&Port_1_2_P1IFG
      $C$L5:
c110:   1300             RETI
```


…
…
…
…
…
ADD R4, R5
C200h→ MOV.b R5, &P2OUT
…
…
…
中断向量：
中断子程的入口地址
C400h → BIT.b #BIT0, &P1IFG
…
…
…
RETI
中断子程

中断的确定性和非确定性

- 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。

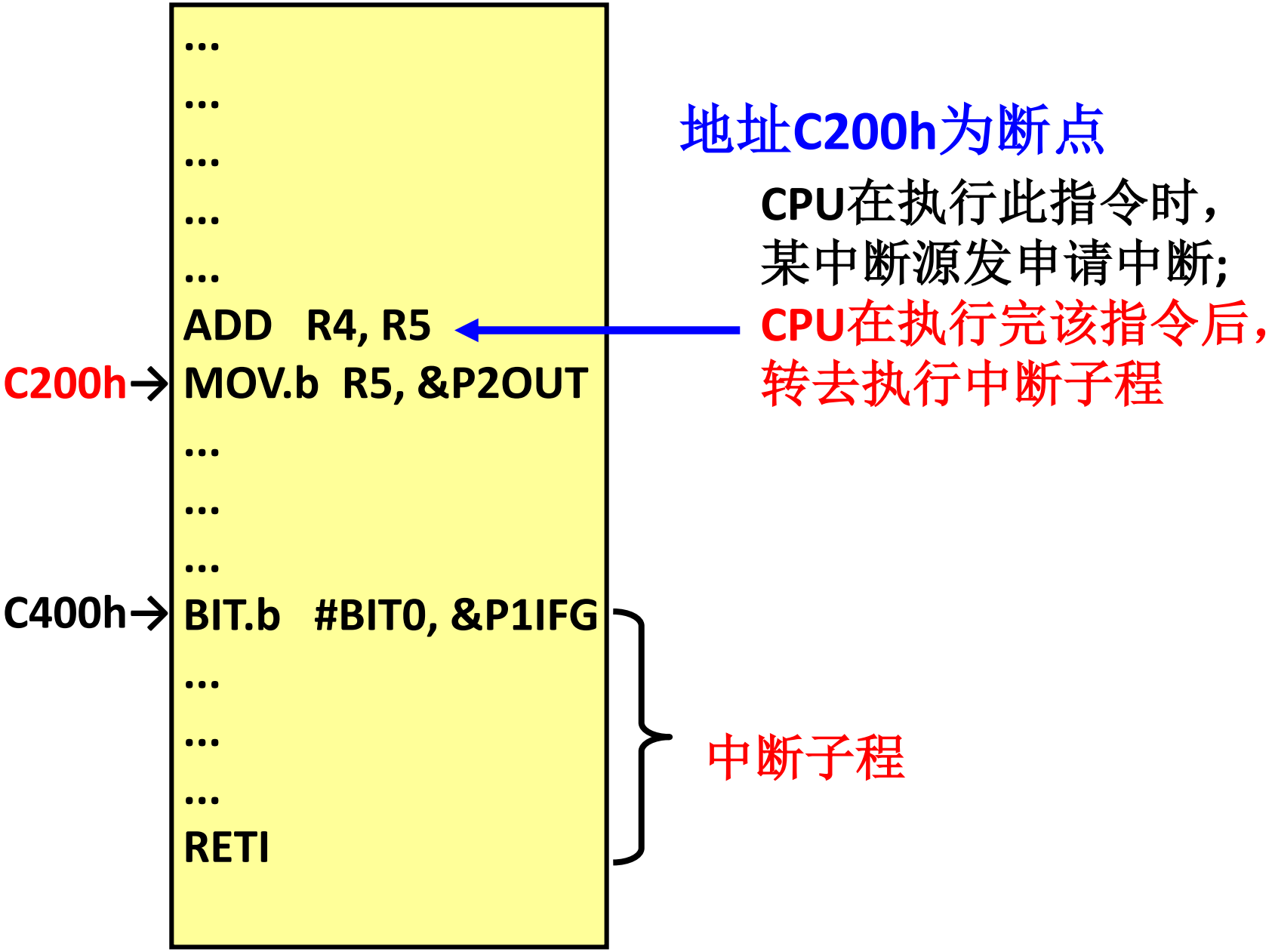
- 中断源产生中断的随机性，使中断服务程序的执行也具有随机性，

  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
C200h→ MOV.b R5, &P2OUT
...
...
...
C400h→ BIT.b #BIT0, &P1IFG
...
...
...
RETI
地址C200h为断点
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
中断子程

第2节 MSP430的中断系统

第3节 MSP430的可屏蔽中断程序设计

- **掌握MSP430的中断类型**
- **理解中断响应过程**
- **掌握P1和P2中断寄存器的操作方法**
- **掌握C语言的中断程序设计方法**

第2节 MSP430的中断系统

一、**MSP430**的中断源类型

二、**MSP430G2553**的中断源、中断优先级

三、**MSP430G2553**的中断类型号和中断向量表

四、**MSP430G2553**的中断标志

五、**MSP430**的可屏蔽中断响应过程

六、**MSP430G2553**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

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**）

——不能被总控位**GIE**和自己的分控位**IE**位屏蔽的中断

●**非屏蔽中断(Non)maskable interrupts**

——不能被总控位**GIE**屏蔽，

但能被自己的分控位**IE**位屏蔽的中断

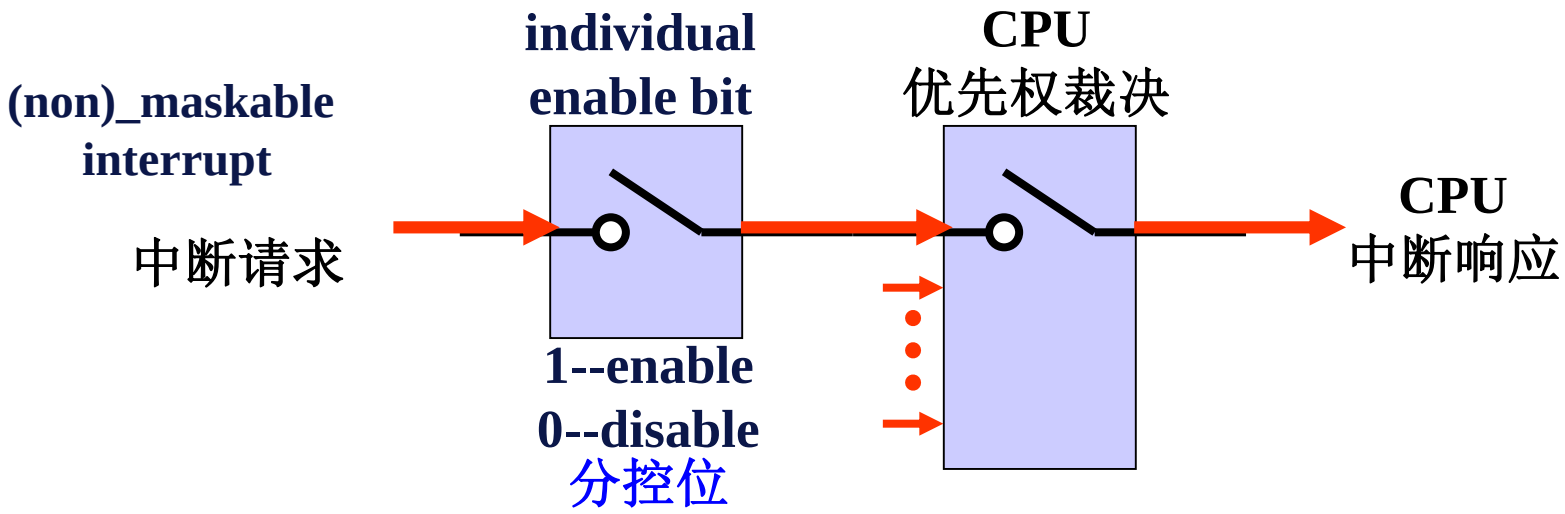
●**可屏蔽中断maskable interrupts**

——能被总控位**GIE**和自己的分控位**IE**位屏蔽的中断

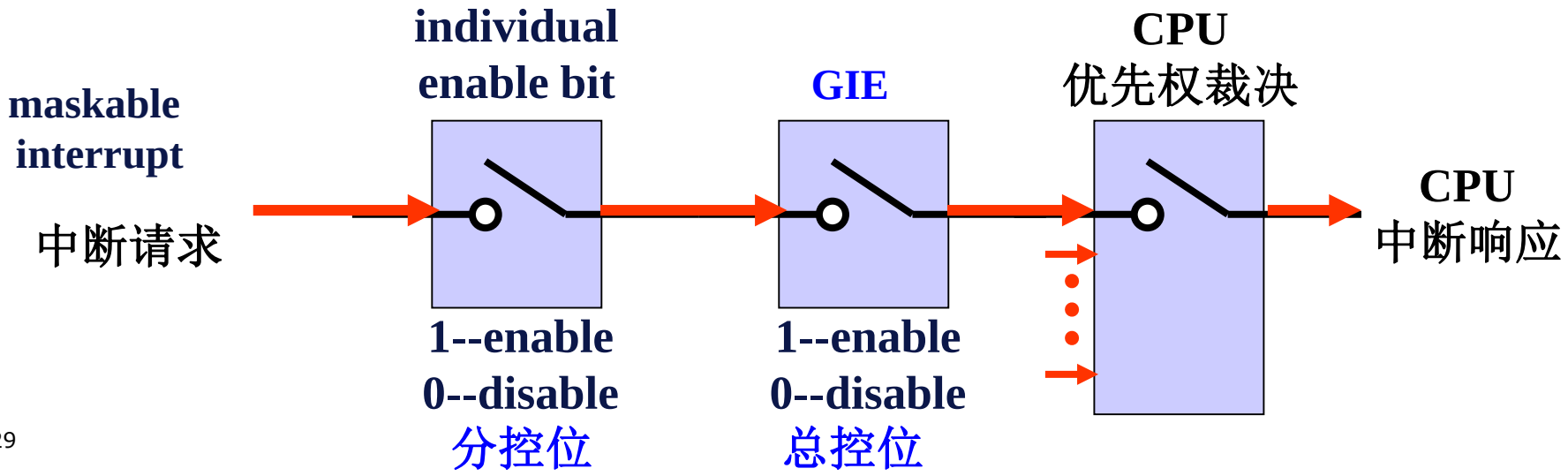
MSP430G2553中断源MSP430G2553.pdf P11

类型	中断源		优先级
复位	Power up, External Reset、 Watchdog, Flash memory， PC out of range	上电,外部复位,看门狗复位,FLASH密码错、PC跑飞	15 (highest)
非屏蔽	NMI , Oscillator Fault, Flash memory access violation	NMI引脚、振荡器失效, FCTlx访问错	14
可屏蔽	Timer1_A3	定时器TA1	13
	Timer1_A3	定时器TA1	12
	Comparator_A+	比较器A	11
	Watchdog timer+	看门狗定时器	10
	Timer0_A3	定时器TA0	9
	Timer0_A3	定时器TA0	8
	USCI_A0/USCI_B0 receive USCI_B0 I2C status	串行通信接口A0/B0	7
	USCI_A0/USCI_B0 transmit USCI_B0 I2C receive/transmit	串行通信接口A0/B0	6
	ADC10	模/数转换器ADC10	5
	无	无	4
	I/O port 2	P2.0~P2.7的8个引脚	3
	I/O port 1	P1.0~P1.7的8个引脚	2
	无	无	1，0

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

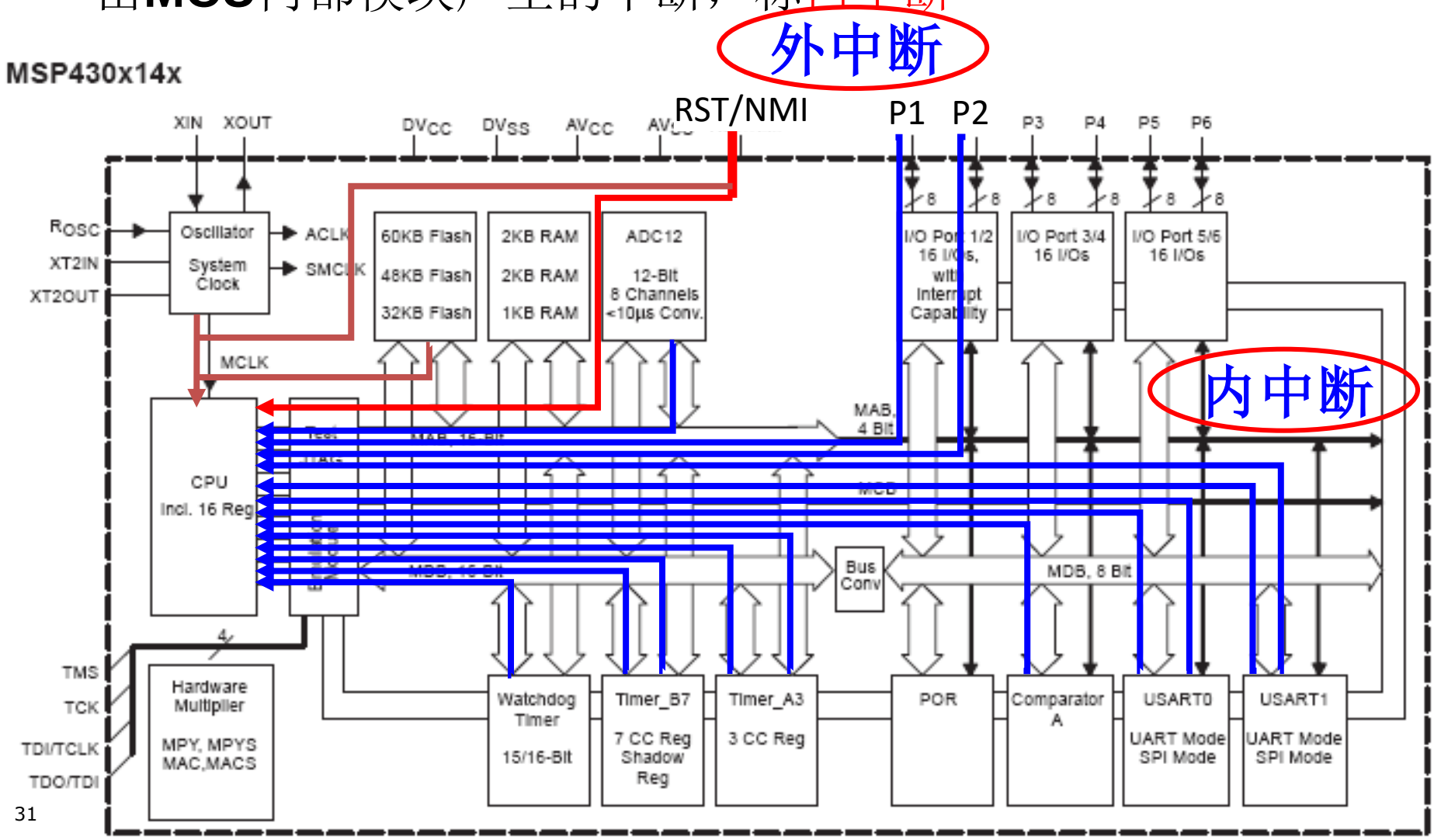
GIE :可屏蔽中断屏蔽位(**General Interrupt Enable Bit**)

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



二、MSP430G2553中断优先级

MSP430G2553.pdf P11

类型	中断源		优先级
复位	Power up, External Reset、 Watchdog, Flash memory， PC out of range	上电,外部复位,看门狗复位,FLASH密码错、PC跑飞	15 (highest)
非屏蔽	NMI , Oscillator Fault, Flash memory access violation	NMI引脚、振荡器失效, FCTIx访问错	14
可屏蔽	Timer1_A3	定时器TA1	13
	Timer1_A3	定时器TA1	12
	Comparator_A+	比较器A	11
	Watchdog timer+	看门狗定时器	10
	Timer0_A3	定时器TA0	9
	Timer0_A3	定时器TA0	8
	USCI_A0/USCI_B0 receive USCI_B0 I2C status	串行通信接口A0/B0	7
	USCI_A0/USCI_B0 transmit USCI_B0 I2C receive/transmit	串行通信接口A0/B0	6
	ADC10	模/数转换器ADC10	5
	无	无	4
	I/O port 2	P2.0~P2.7的8个引脚	3
	I/O port 1	P1.0~P1.7的8个引脚	2
	无	无	1，0

三、MSP430G2553的中断类型号和中断向量表

- 中断类型号： 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)

  MSP430G2553共有15个中断类型号（15~0）

  类型号15： 复位中断子程

  类型号14： 非屏蔽中断子程

  ……

  类型号3 ： 端口P2中断

  类型号2 ： 端口P1中断

MSP430G2553的中断类型号

类型	中断源	优先级	类型号
复位	上电、外部复位、看门狗复位、FLASH密码错、PC跑飞	15 (highest)	15 (highest)
非屏蔽	NMI引脚、振荡器失效、FCTlx访问错	14	14
可屏蔽	定时器TA1	13	13
	定时器TA1	12	12
	比较器A	11	11
	看门狗定时器	10	10
	定时器TA0	9	9
	定时器TA0	8	8
	串行通信接口A0/B0	7	7
	串行通信接口A0/B0	6	6
	模/数转换器ADC10	5	5
	无	4	4
	P2.0~P2.7的8个引脚	3	3
	P1.0~P1.7的8个引脚	2	2
	无	1	1
	无	0	0

发生中断时的指令执行情况

第一条指令地址

主函数

主循环
while(1)

PC

中断程序入口地址
（中断向量）

PC

中断子程序

地址	指令内容
C000	SUB.W #4,SP
C002	MOV.W #0x5a80 …
C008	DINT
C00A	NOP
C00C	CLR.B &Port_1_2_P1SEL
…	……
C058	CMP.W &0x0206,0x0002(SP)
C05E	JLO (CL2)
…	……
C08E	JMP (CL1)
C090	CLR.W 0x0000(SP)
C094	CLR.W 0x0002(SP)
C098	JMP (CL1)
C09A	BIT.B #2,&Port_1_2_P1IFG
C09E	JEQ (CL9)
C0A0	CMP.W &0x0202,&0x0206
C0A6	JLO (CL7)
…	……
C0D0	DEC.W &Tduty
C0D4	SBC.W &0x0206
C0D8	BIC.B #4,&Port_1_2_P1IFG
C0DC	reti

- 中断事件发生
- 跳转到中断子程序入口地址

CPU是怎么知道要跳转到这个入口地址的？

中断向量

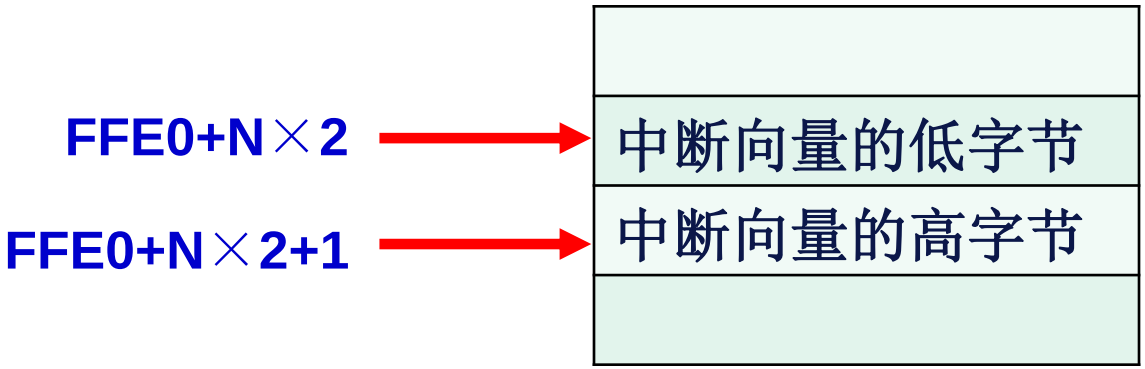
- 中断向量

  指中断子程的入口地址

  在**MSP430G2553** 中地址是一个**16**位二进制数
- 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**FFE0+N×2**
- 存放一个中断向量占用**2**个存储器单元，

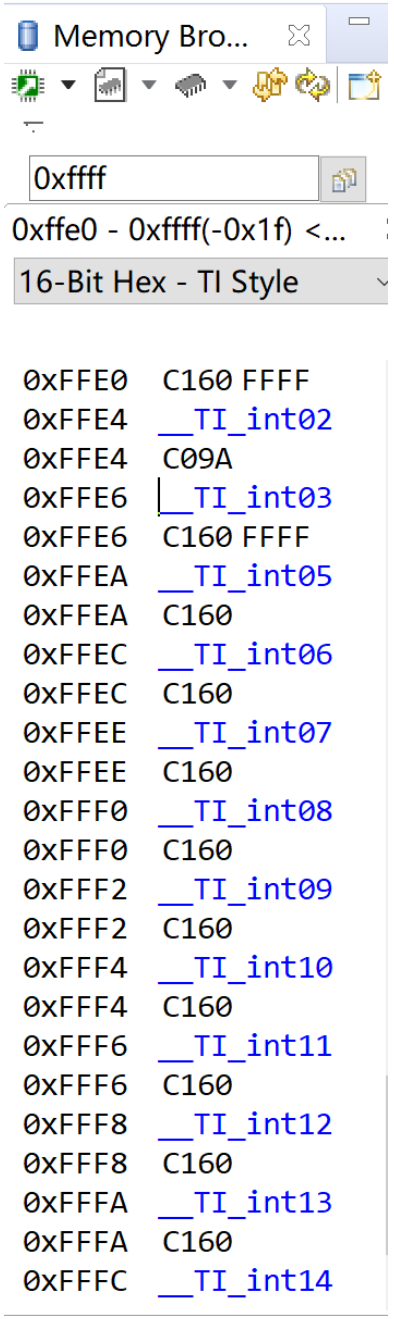
  地址分别是: **FFE0+N×2, FFE0+N×2+1**



中断子程**N**的入口地址**(**中断向量**)**在存储器中的位置

例： 计算中断类型号2的中断向量存放在存储器的位置

例： 如右图用CCS的View>Memory Browser
可查看中断向量表，
问类型2的中断向量是？



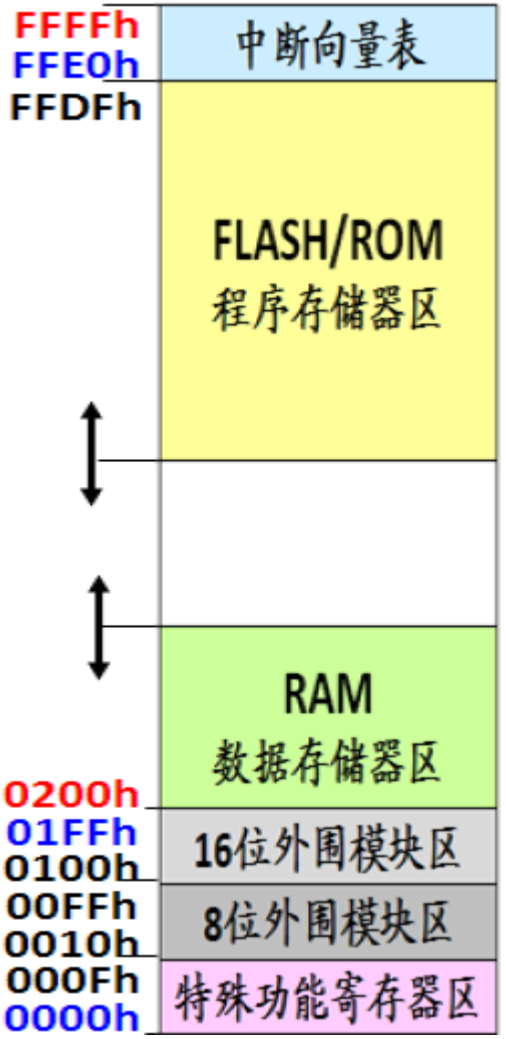
思考：如何在CCS的DEBUG下查看到中断函数?

中断向量表：

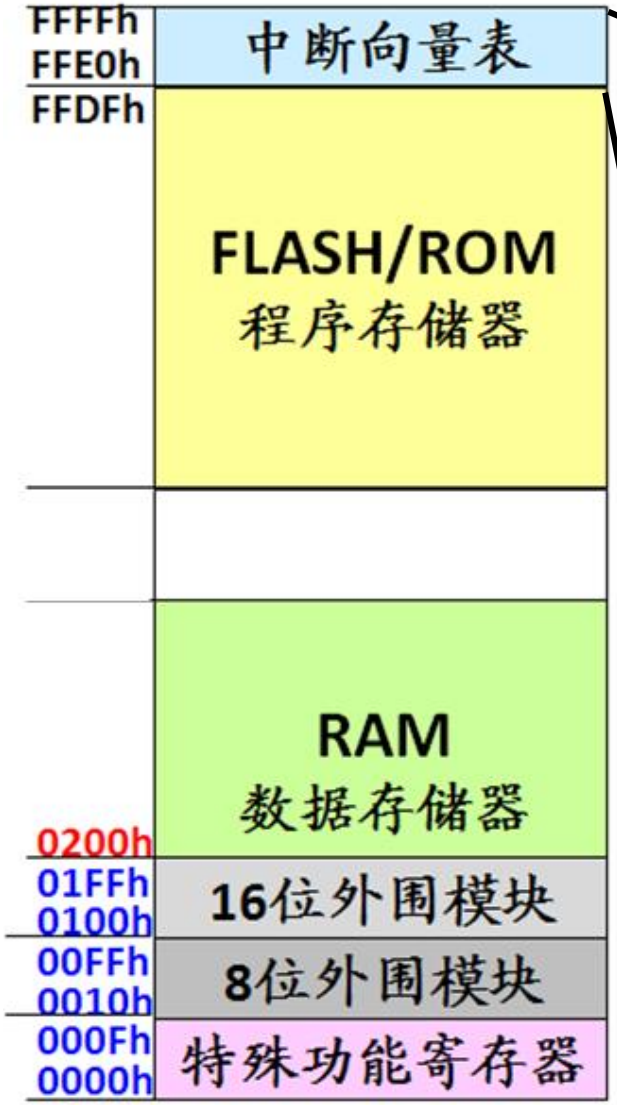
指存放所有中断向量的存储器区域

- **MSP430**中断向量表在**FFE0~FFFFh**的**32B**空间起始地址是**FFE0h,** 可存放**16**个中断向量

- 不同的**MCU**型号实际存放的中断向量个数可能不同



MSP430G2553的中断向量表



地址	类型号（N）	对应的中断源的
0FFFEh	15	上电,外部复位,看门狗复位,FLASH密码错、PC跑飞
0FFFCh	14	NMI引脚、振荡器失效,FCTlx访问错
0FFFAh	13	定时器TA1
0FFF8h	12	定时器TA1
0FFF6h	11	比较器A
0FFF4h	10	看门狗定时器
0FFF2h	9	定时器TA0
0FFF0h	8	定时器TA0
0FFEEh	7	串行通信接口A0/B0
0FFECh	6	串行通信接口A0/ B0
0FFEAh	5	模/数转换器ADC10
0FFE8h	4	无
0FFE6h	3	P2.0~P2.7的8个引脚
0FFE4h	2	P1.0~P1.7的8个引脚
0FFE2h	1	
0FFE0h	0	

类型N的中断向量存放地址，

0xFFE0+N×2

发生中断时的指令执行情况

第一条指令地址

主函数

主循环
while(1)

中断程序入口地址
（中断向量）

中断子程序

地址	指令内容
C000	SUB.W #4,SP
C002	MOV.W #0x5a80 …
C008	DINT
C00A	NOP
C00C	CLR.B &Port_1_2_P1SEL
…	……
C058	CMP.W &0x0206,0x0002(SP)
C05E	JLO (CL2)
…	……
C08E	JMP (CL1)
C090	CLR.W 0x0000(SP)
C094	CLR.W 0x0002(SP)
C098	JMP (CL1)
C09A	BIT.B #2,&Port_1_2_P1IFG
C09E	JEQ (CL9)
C0A0	CMP.W &0x0202,&0x0206
C0A6	JLO (CL7)
…	……
C0D0	DEC.W &Tduty
C0D4	SBC.W &0x0206
C0D8	BIC.B #4,&Port_1_2_P1IFG
C0DC	reti

PC

PC

PC

地址	数据
FFE6	….
FFE4	C09A
FFE2	….
FFE0	….

中断向量表：
（中断类型2）

FFE0+2*2=FFE4

- 头文件 **msp430G2553.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**0xFFE0**

```
/************************************************************
* Interrupt Vectors (offset from 0xFFE0)
************************************************************/
#define TRAPINT_VECTOR          (0 * 1u)  /* 0xFFE0 TRAPINT */
#define  PORT1_VECTOR           (2 * 2u)  /* 0xFFE4 Port 1 */
#define  PORT2_VECTOR           (3 * 2u)  /* 0xFFE6 Port 2 */
#define  ADC10_VECTOR           (5 * 2u)  /* 0xFFEA ADC10 */
#define  USCIAB0TX_VECTOR       (6 * 2u)  /* 0xFFEC USCI A0/B0 Transmit */
#define  USCIAB0RX_VECTOR       (7 * 2u)  /* 0xFFEE USCI A0/B0 Receive */
#define  TIMER0_A1_VECTOR       (8 * 2u)  /* 0xFFF0 Timer0)A CC1, TA0 */
#define  TIMER0_A0_VECTOR       (9 * 2u)  /* 0xFFF2 Timer0_A CC0 */
#define  WDT_VECTOR             (10 * 2u) /* 0xFFF4 Watchdog Timer */
#define  COMPARATORA_VECTOR     (11 * 2u) /* 0xFFF6 Comparator A */
#define  TIMER1_A1_VECTOR       (12 * 2u) /* 0xFFF8 Timer1_A CC1-4, TA1 */
#define  TIMER1_A0_VECTOR       (13 * 2u) /* 0xFFFA Timer1_A CC0 */
#define NMI_VECTOR              (14 * 2u) /* 0xFFFC Non-maskable */
#define RESET_VECTOR            (15 * 2u) /* 0xFFFE Reset [Highest Priority] */
```

类型2的中断向量存放在向量表的地址：

0FFE0 + 2*2 = 0FFE4h

0FFE0h + PORT1_VECTOR

类型15(复位中断)的中断向量存放在向量表的地址：

0FFE0h + RESET_VECTOR

0FFE0 + 15*2 = 0FFFEh

四、MSP430G2553的中断标志

■ 中断标志位

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（存放在外围模块内部或特殊功能寄存器中）

■ 中断标志位功能

➢ 保存中断请求信号

当CPU不能够马上响应中断源发出的中断请求，需要一些寄存器标志保存中断请求信号，直到被CPU响应。

➢ 区分多个子中断源：

多个子中断源（例如P1和P2的8个管脚）共用一个中断类型号

- 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)
可通过在中断函数中查询中断标志位，确定产生中断的是哪个子中断源

类型	中断源	中断标志 (数目)		优先级	类型号
复位	上电,外部复位,看门狗复位,FLASH密码错、PC跑飞	PORIFG, RSTIFG,WDTIFG, KEYV	(4)	15	15
非屏蔽	NMI引脚、振荡器失效, FCTlx访问错	NMIFG, OFIFG, ACCVIFG	(3)	14	14
可屏蔽	定时器TA1	TA1CCR2、 TA1CCR1 CCIFG, TAIFG	(3)	12	12
	定时器TA0	TA0CCR2 、TA0CCR1 CCIFG, TAIFG	(3)	8	8
	串行通信接口A0, B0	UCA0RXIFG, UCB0RXIFG	(2)	7	7
	串行通信接口A1, B1	UCA0TXIFG, UCB0TXIFG	(2)	6	6
	P2.0~P2.7的8个引脚	P2IFG.0 to P2IFG.7	(8)	3	3
	P1.0~P1.7的8个引脚	P1IFG.0 to P1IFG.7	(8)	2	2

MSP430G2553中断信息总表 (MSP430G2553.pdf P11)

类型	中断源	中断标志 (数目)		优先级	类型号
复位	上电,外部复位,看门狗复位,FLASH密码错、PC跑飞	PORIFG, RSTIFG,WDTIFG, KEYV	(4)	15	15
非屏蔽	NMI引脚、振荡器失效, FCTlx访问错	NMIFG, OFIFG, ACCVIFG	(3)	14	14
可屏蔽	定时器TA1	TA1CCR0 CCIFG	(1)	13	13
	定时器TA1	TA1CCR2 TA1CCR1 CCIFG, TAIFG	(3)	12	12
	比较器A	CAIFG	(1)	11	11
	看门狗定时器	WDTIFG	(1)	10	10
	定时器TA0	TA0CCR0 CCIFG	(1)	9	9
	定时器TA0	TA0CCR2 TA0CCR1 CCIFG, TAIFG	(3)	8	8
	串行通信接口A0/B0	UCA0RXIFG, UCB0RXIFG	(2)	7	7
	串行通信接口A0/B0	UCA0TXIFG, UCB0TXIFG	(2)	6	6
	模/数转换器ADC10	ADC10IFG	(1)	5	5
	无	无	无	4	4
	P2.0~P2.7的8个引脚	P2IFG.0 to P2IFG.7	(8)	3	3
	P1.0~P1.7的8个引脚	P1IFG.0 to P1IFG.7	(8)	2	2

五、MSP430非屏蔽和可屏蔽中断响应过程

第一条指令地址

主函数

主循环
while(1)

中断程序入口地址
（中断向量）

中断子程序

地址	指令内容
C000	SUB.W #4,SP
C002	MOV.W #0x5a80 …
C008	DINT
C00A	NOP
C00C	CLR.B &Port_1_2_P1SEL
…	……
C058	CMP.W &0x0206,0x0002(SP)
C05E	JLO (CL2)
…	……
C08E	JMP (CL1)
C090	CLR.W 0x0000(SP)
C094	CLR.W 0x0002(SP)
C098	JMP (CL1)
C09A	BIT.B #2,&Port_1_2_P1IFG
C09E	JEQ (CL9)
C0A0	CMP.W &0x0202,&0x0206
C0A6	JLO (CL7)
…	……
C0D0	DEC.W &Tduty
C0D4	SBC.W &0x0206
C0D8	BIC.B #4,&Port_1_2_P1IFG
C0DC	reti

地址	数据
FFE6	….
FFE4	C09A
FFE2	….
FFE0	….

进中断时：从中断向量表中获取跳转地址

出中断时，返回的地址从哪里获得？

五、MSP430非屏蔽和可屏蔽中断响应过程

地址	数据
FFE6	….
FFE4	C09A
FFE2	….
FFE0	….

第一条指令地址

主函数

主循环
while(1)

中断程序入口地址
（中断向量）

中断子程序

地址	指令内容
C000	SUB.W #4,SP
C002	MOV.W #0x5a80 …
C008	DINT
C00A	NOP
C00C	CLR.B &Port_1_2_P1SEL
…	……
C058	CMP.W &0x0206,0x0002(SP)
C05E	JLO (CL2)
…	……
C08E	JMP (CL1)
C090	CLR.W 0x0000(SP)
C094	CLR.W 0x0002(SP)
C098	JMP (CL1)
C09A	BIT.B #2,&Port_1_2_P1IFG
C09E	JEQ (CL9)
C0A0	CMP.W &0x0202,&0x0206
C0A6	JLO (CL7)
…	……
C0D0	DEC.W &Tduty
C0D4	SBC.W &0x0206
C0D8	BIC.B #4,&Port_1_2_P1IFG
C0DC	reti

CPU自动完成
1）保存断点地址
2）保存SR
3）清零SR
4）从中断向量表获取跳转地址
5）跳转到中断程序入口地址

CPU自动完成
1）恢复SR
2）恢复断点地址
3）从断点继续执行

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

六、MSP430G2553端口P1 和 P2 外中断

- 16个外部输入可屏蔽中断源
  - 端口P1的P1.0~P1.7
  - 端口P2的P2.0~P2.7

- 每个端口的8个引脚中断源共用一个向量地址
  - P1中断有关的寄存器：P1SEL, P1SEL2, P1DIR, P1IES,P1IFG和P1IE
  - P2中断有关的寄存器：P2SEL, P2SEL2, P2DIR, P2IES, P2IFG和P2IE

中断源	中断标志	向量地址	优先级,类型号
…	…	…	…
P2的8个引脚	P2IFG.0~P2IFG.7 (8)	0FFE6h	3
P1的8个引脚	P1IFG.0~P1IFG.7 (8)	0FFE4h	2
…	…	…	…

P1,P2外中断使用方法

1）中断寄存器设置方法

使用输入中断时，应设置为0

应选为选择输入方向，设置为0

根据电路情况选择是否用内部拉电阻，以及上拉或者下拉

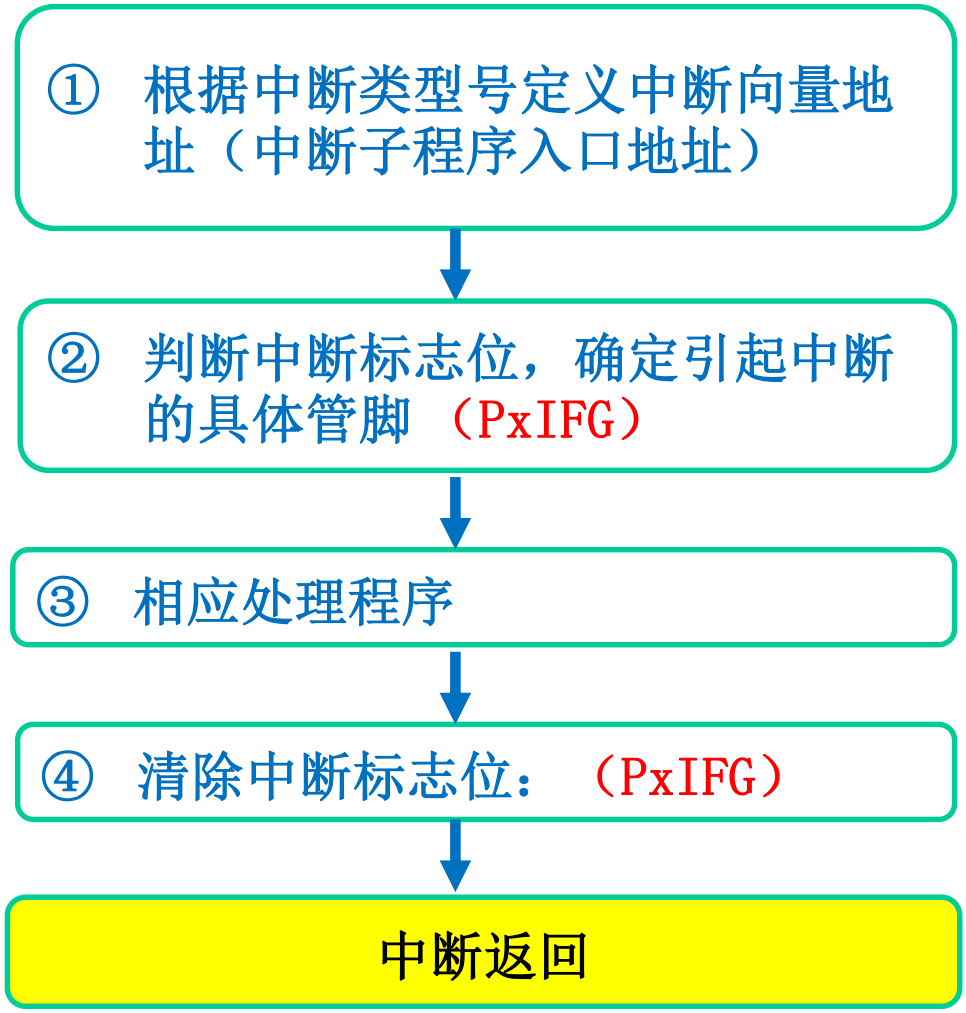
0：上升沿
1：下降沿

使用中断的管脚设为1
不使用中断的管脚设为0

总中断允许设为1

P1,P2外中断使用方法

2）中断子程序处理方法



#pragma vector=N*2

判断管脚对应位是否为1

将PxIFG中的对应位置为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P1的8个引脚均作为外部中断源，P1IES的内容为0Fh，

表示端口P1的引脚P1.7~P1.4有上升沿信号时，发出中断申请

端口P1的引脚P1.3~P1.0有下降沿信号时，发出中断申请

● PxIE中断允许寄存器(Interrupt Enable)

是端口P1和P2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, PxSEL2.y为0, PxDIR.y为0,
若Px.y引脚上有中断申请信号，则PxIFG.y自动置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

● PxIFG中断标志寄存器(Interrupt Flag)

当作为中断功能的端口P1和P2的某引脚上有中断申请信号时，硬件自动置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P1的8个引脚均作为外部中断源，P1IFG的内容为F0h，

表示端口P1的引脚P1.7~P1.4上有中断申请

端口P1的引脚P1.3~P1.0上无中断申请

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语言中端程序举例

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

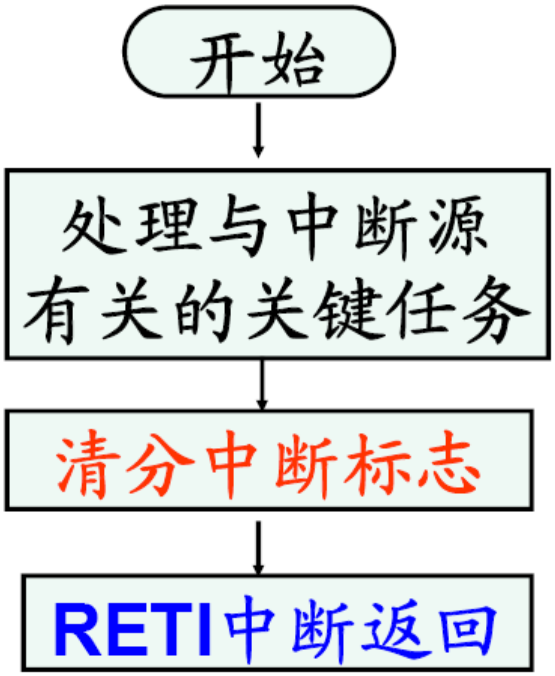
做好相关设置，使中断源发出中断申请时CPU能够响应的准备工作

主程序流程

2. 中断程序

处理与中断源有关的关键任务

中断程序流程



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**

   使得反汇编中断程序时,

   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、PC跑飞	PORIFG, RSTIFG,WDTIFG, KEYV	0FFFEh	15
…	…	…	…
引脚P2.0~P2.7	P2IFG.0~P2IFG.7	0FFE6h	3
引脚P1.0~P1.7	P1IFG.0~P1IFG.7	0FFE4h	2
…	…	…	…

➢**msp430G2553.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0xFFE0**

```
/************************************************************
* Interrupt Vectors (offset from 0xFFE0)
************************************************************/
#define TRAPINT_VECTOR          (0 * 1u)  /* 0xFFE0 TRAPINT */
#define  PORT1_VECTOR           (2 * 2u)  /* 0xFFE4 Port 1 */
#define  PORT2_VECTOR           (3 * 2u)  /* 0xFFE6 Port 2 */
#define  ADC10_VECTOR           (5 * 2u)  /* 0xFFEA ADC10 */
#define  USCIAB0TX_VECTOR       (6 * 2u)  /* 0xFFEC USCI A0/B0 Transmit */
#define  USCIAB0RX_VECTOR       (7 * 2u)  /* 0xFFEE USCI A0/B0 Receive */
#define  TIMER0_A1_VECTOR       (8 * 2u)  /* 0xFFF0 Timer0)A CC1, TA0 */
#define  TIMER0_A0_VECTOR       (9 * 2u)  /* 0xFFF2 Timer0_A CC0 */
#define  WDT_VECTOR             (10 * 2u) /* 0xFFF4 Watchdog Timer */
#define  COMPARATORA_VECTOR     (11 * 2u) /* 0xFFF6 Comparator A */
#define  TIMER1_A1_VECTOR       (12 * 2u) /* 0xFFF8 Timer1_A CC1-4, TA1 */
#define  TIMER1_A0_VECTOR       (13 * 2u) /* 0xFFFA Timer1_A CC0 */
#define NMI_VECTOR              (14 * 2u) /* 0xFFFC Non-maskable */
#define RESET_VECTOR            (15 * 2u) /* 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)

函数名称	功能	包含在
__disable_interrupt( )	0→ GIE	intrinsics.h
__ensable_interrupt( )	1→ GIE	intrinsics.h

intrinsics.h 文件中：

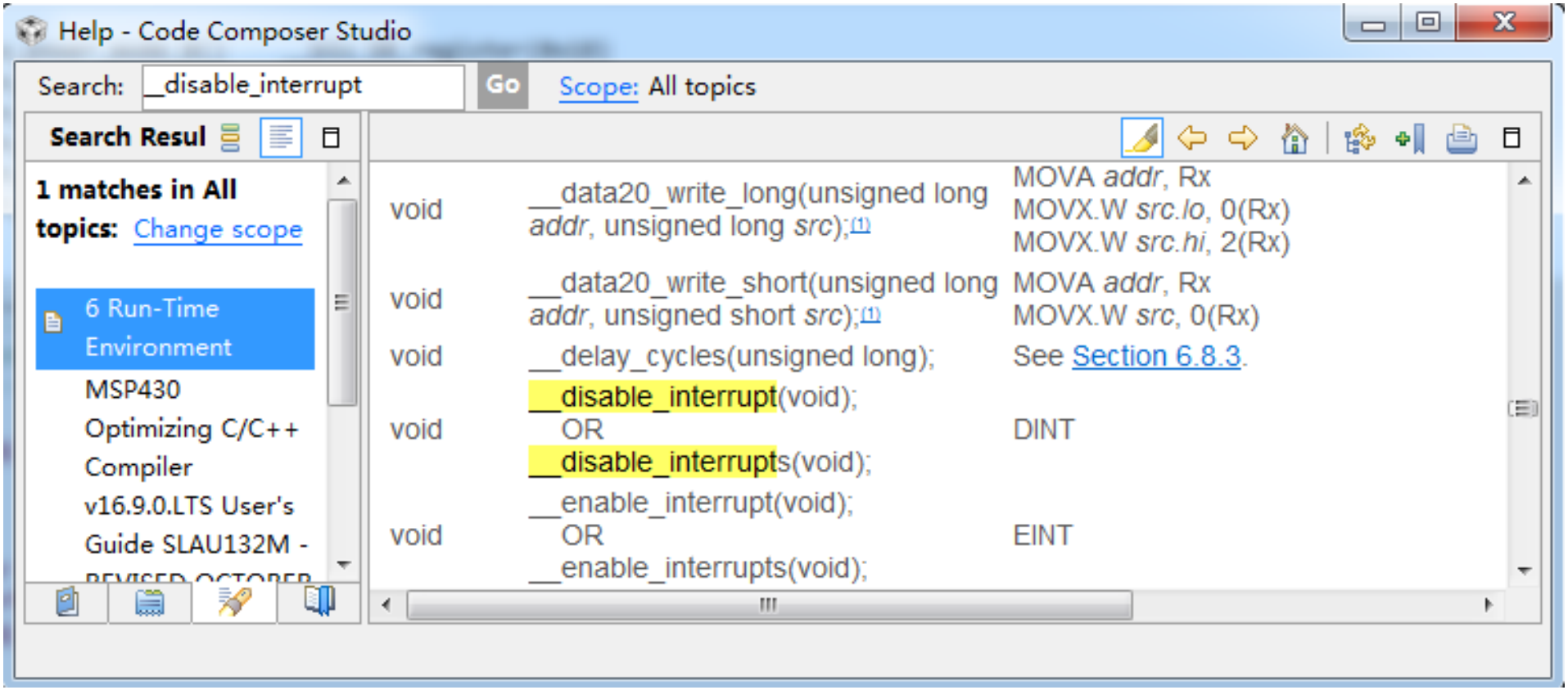
声明了一些包含在**CCS** 编译器的内部函数，方便用户使用

如：

```
__intrinsic   void   __enable_interrupt(void);
__intrinsic   void   __disable_interrupt(void);
```

在**CCS**下点击**Help>Help Contents**, 了解这些内联函数实现的功能

在CCS下点击Help>Help Contents，了解这些内部函数实现的功能



内联函数由编译器插入完成该函数功能的相应指令或指令段，比通过调用函数方式节省时间开支。

例如，在intrinsics.h中声明了一个__enable_interrupt(void)：

```
void        __enable_interrupt(void);
```

编译器编译用户源程序时，在程序用到了__enable_interrupt( )这个内联函数的地方，就用一条 eint 指令 代替。

可以看出这个内联函数很简单，就一条指令。

in430.h

对intrinsics.h中的一些函数用#define 做了简化定义

```
#define   _EINT()              __enable_interrupt()
#define   _DINT()              __disable_interrupt()
#define   _BIC_SR(x)           __bic_SR_register(x)
#define   _BIC_SR_IRQ(x)       __bic_SR_register_on_exit(x)
#define   _BIS_SR(x)           __bis_SR_register(x)
#define   _BIS_SR_IRQ(x)       __bis_SR_register_on_exit(x)
#define   _SWAP_BYTES(x)       __swap_bytes(x)
#define   _NOP()               __no_operation()
```

在msp430g2553.h中包含有in430.h 和 intrinsics.h

所以，程序中只用包含msp430.h，

就可以使用in430.h 和 intrinsics.h中的定义

msp430.h

```
#if defined (__MSP430C111__)
#include "msp430c111.h"
……
#elif defined (__MSP430G2553__)
#"include   "m430g2553.h "
……
```

msp430g2553.h

```
……
#include "in430.h"
#include <intrinsics.h>
……
```

用C语言编写中断程序方法1

1. 使用__disable_interrupt( ) 和__enable_interrupt( )

```
#include  "msp430.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=数字或符号表示的偏址     //中断向量设置
__interrupt void port_int(void)          //中断函数，函数名可自取
{
    ……
}
```

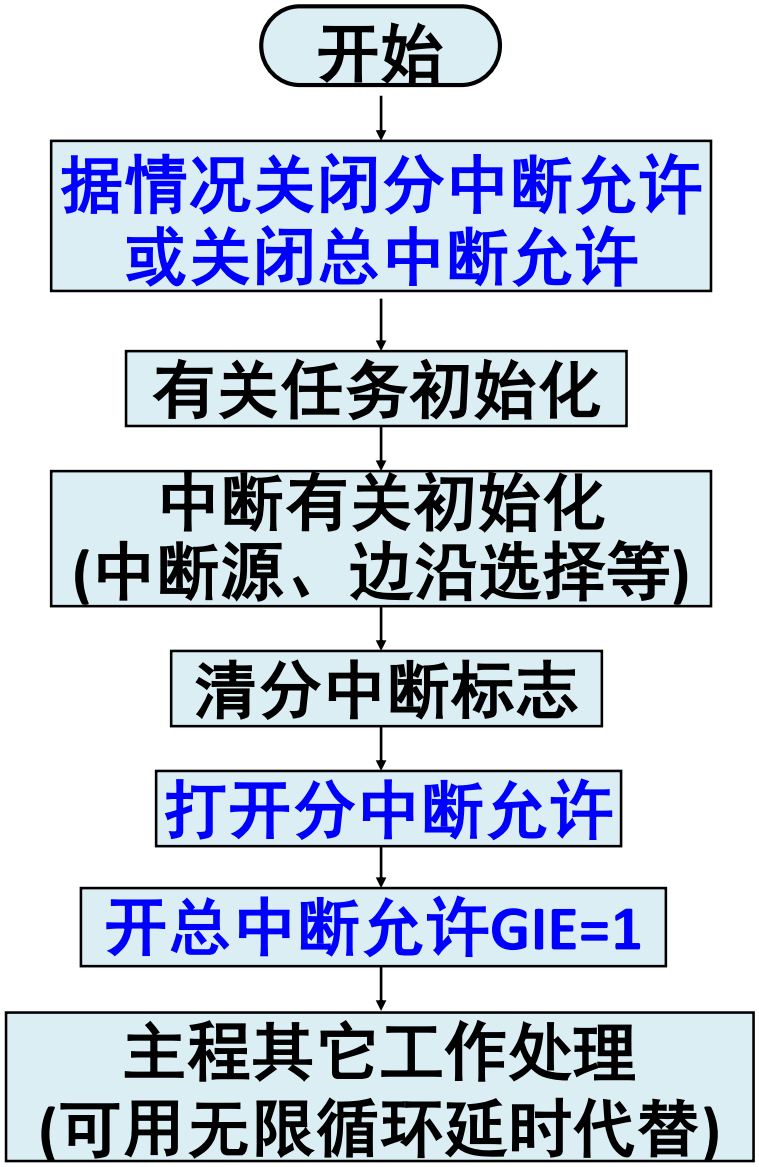
用C语言编写中断程序方法2

使用_DINT( ) 和_EINT ( )

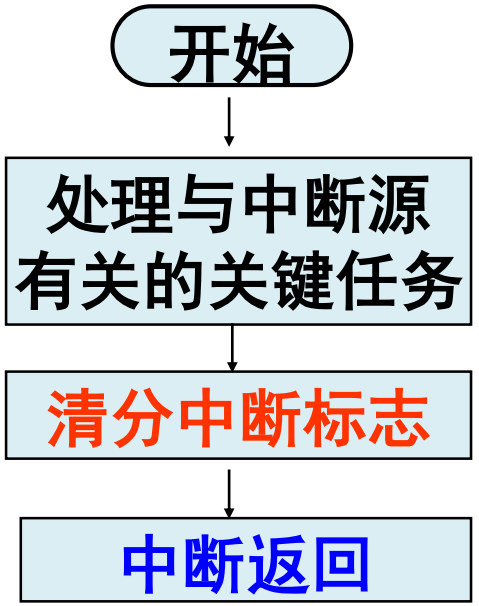
```
#include        “msp430.h"

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=数字或符号表示的偏址     //中断向量设置
__interrupt void port_int(void)          //中断函数，函数名可自取{
    ……
}
```

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

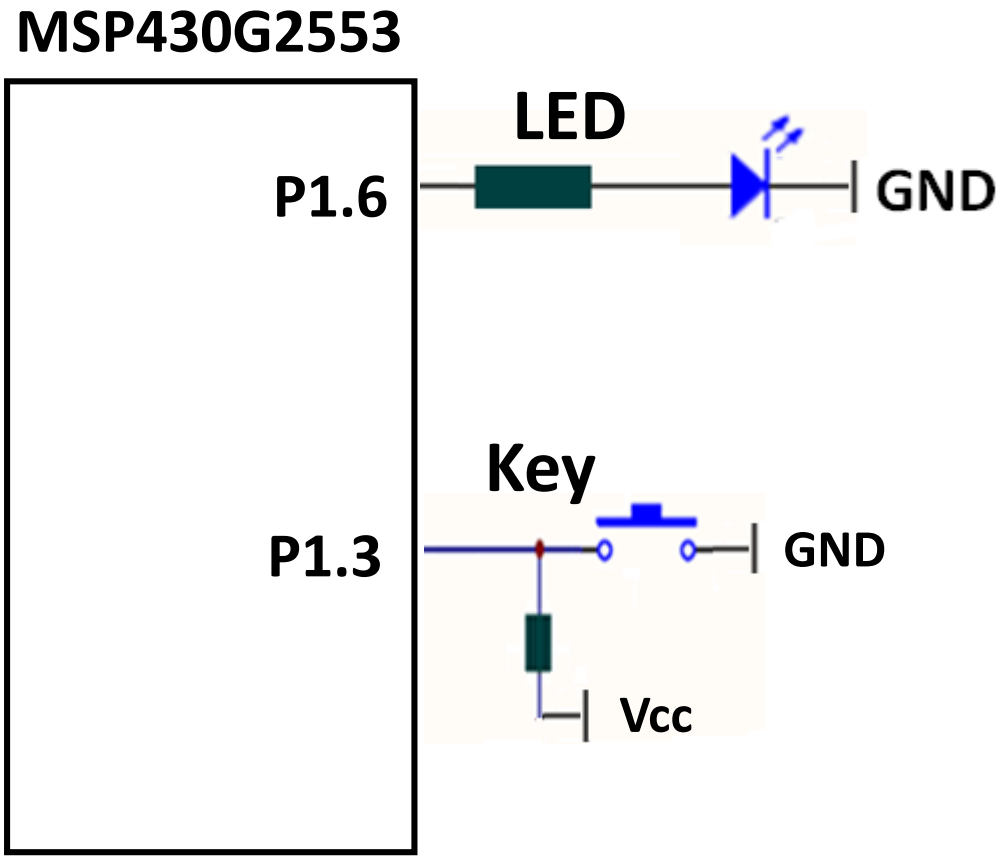
根据中断源在
中断向量表相应位置
设置中断向量

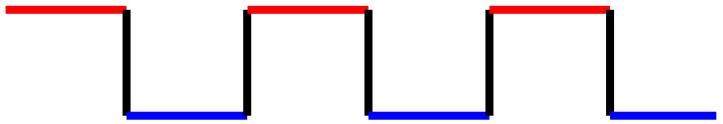
例： 中断编程举例 (以P1.3上的中断为例)

请用C语言编写程序，

以中断方式响应P1.3上的按键，

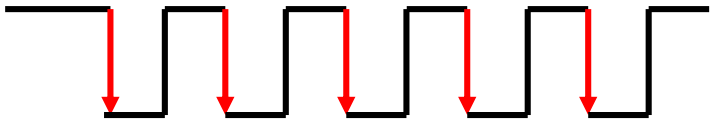
每按下一次键，改变一次发光二极管状态





求反P1.6的输出
改变灯的状态

按键中断请求



P1.3中断C语言程序清单 C5_1_1.c

外部按键有上拉电阻

```
#include "msp430.h"
int main( void )
{  WDTCTL = WDTPW + WDTHOLD;            //关闭看门狗
   _DINT();                             //禁止可屏蔽中断 GIE=0
   P1IE &=~BIT3;                        // 关闭P1.3中断允许
    P1SEL &=~BIT6;                      //设置P1.6为基本I/O功能
    P1SEL2 &=~BIT6;                     //
    P1OUT &=~BIT6;                      //置P1.6输出初值
    P1DIR |=BIT6;                       //设置P1.6为输出

    P1SEL &=~BIT3;                      //置P1.3作为基本I/O端口
    P1SEL2 &=~BIT3;                     //
    P1DIR &=~BIT3;                      //置P1.3为输入
    P1IES |=BIT3;                       //置P1.3下降沿作中断源
    P1IFG =0;                           //清P1IFG中断标志
    P1IE |=BIT3;                        //打开P1.3中断允许
    _EINT();                            //允许可屏蔽中断 GIE=1
   while(1) {   };                      //主循环
}
#pragma  vector=PORT1_VECTOR      //置P1中断向量
__interrupt  void port_int(void)          //中断函数
{    if  ( (P1IFG&BIT3)!=0 )              //判断是否是P1IFG.3中断标志
     {       P1OUT ^=BIT6;                //对P1.6取反
              P1IFG &=~BIT3;              //清P1.3中断标志
      }
}
```

P1.3中断C语言程序反汇编代码：

```
main():
c000:   40B2 5A80 0120      MOV.W   #0x5a80,&Watchdog_Timer_WDTCTL
 4          _DINT();            //禁止可屏蔽中断GIE=0
c006:   C232                DINT
c008:   4303                NOP
 5          P1IE &=~BIT3;       // 关闭P1.3中断允许
c00a:   C2F2 0025           BIC.B   #8,&Port_1_2_P1IE
 6              P1SEL &=~BIT6;          //设置P1.6为基本I/O功能
c00e:   F0F2 00BF 0026      AND.B   #0x00bf,&Port_1_2_P1SEL
 7          P1SEL2 &=~BIT6;         //
c014:   F0F2 00BF 0041      AND.B   #0x00bf,&Port_1_2_P1SEL2
 8          P1OUT &=~BIT6;      //置P1.6输出初值
c01a:   F0F2 00BF 0021      AND.B   #0x00bf,&Port_1_2_P1OUT
 9              P1DIR |=BIT6;           //设置P1.6为输出
c020:   D0F2 0040 0022      BIS.B   #0x0040,&Port_1_2_P1DIR
11          P1SEL &=~BIT3;          //置P1.0作为基本I/O端口
c026:   C2F2 0026           BIC.B   #8,&Port_1_2_P1SEL
12          P1SEL2 &=~BIT3;         //
c02a:   C2F2 0041           BIC.B   #8,&Port_1_2_P1SEL2
13          P1DIR &=~BIT3;          //置P1.0为输入
c02e:   C2F2 0022           BIC.B   #8,&Port_1_2_P1DIR
14          P1IES |=BIT3;               //置P1.0下降沿作中断源
c032:   D2F2 0024           BIS.B   #8,&Port_1_2_P1IES
15          P1IFG =0;           //清P1IFG中断标志
c036:   43C2 0023           CLR.B   &Port_1_2_P1IFG
16              P1IE |=BIT3;        //打开P1.0中断允许
c03a:   D2F2 0025           BIS.B   #8,&Port_1_2_P1IE
17        _EINT();                  //允许可屏蔽中断GIE=1
c03e:   D232                EINT
18        while(1)  {   };                  //主循环
$C$L1:
c040:   3FFF                JMP     ($C$L1)
```

_DINT(); → DINT

P1IFG &=~BIT3;

→ BIC.B #0x8,&P1IFG

P1IE &=~BIT3;

→ BIC.B #0x8, &P1IE

_EINT(); → EINT

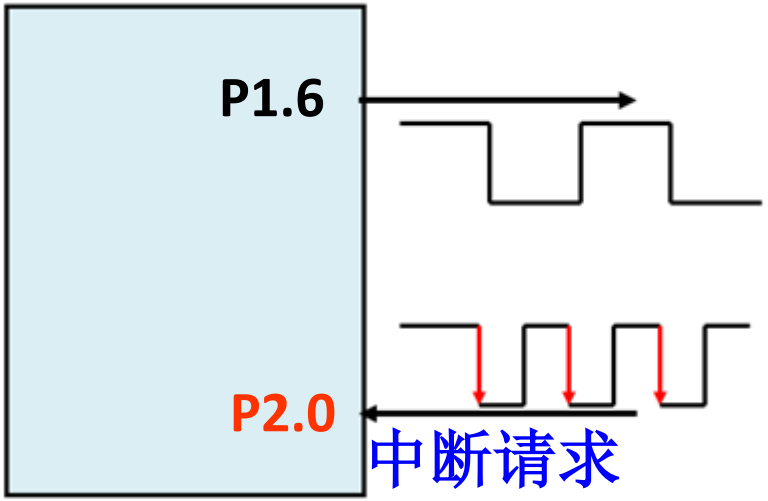
在CCS下C中断函数反汇编代码：

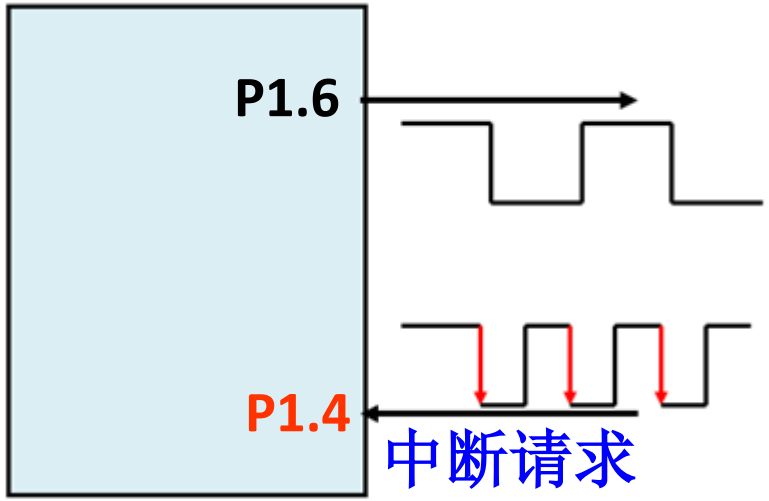
```
        port_int():
c054:     B2F2 0023             BIT.B   #8,&Port_1_2_P1IFG
c058:     2405                  JEQ     ($C$L2)
23              { P1OUT ^=BIT6;         //对P1.6取反
c05a:     E0F2 0040 0021        XOR.B   #0x0040,&Port_1_2_P1OUT
24              P1IFG &=~BIT3;            //清P1.3中断标志
c060:     C2F2 0023             BIC.B   #8,&Port_1_2_P1IFG
26      }
        $C$L2:
c064:     1300                  RETI
```

中断函数结尾是一条 RETI 指令

思考:

- ✓ 中断程序何时被执行? 执行过程是怎样?
- ✓ 如何将程序改写为查询方式,
  即查询到P1.3从1变到0, 就对P1.6求反?
- ✓ 比较中断方式和查询方式有何不同?
- ✓ 若中断源来自引脚P2.0, 如何修改程序?
- ✓ 若中断源来自引脚P1.4, 如何修改程序?





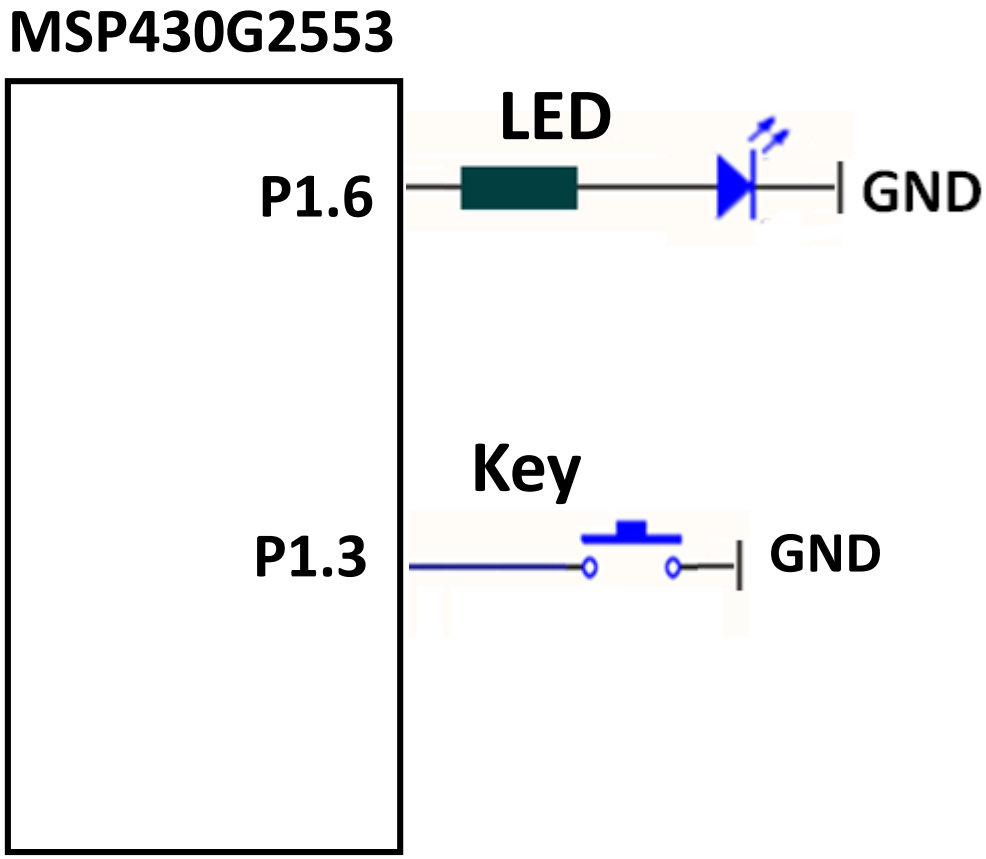
思考:

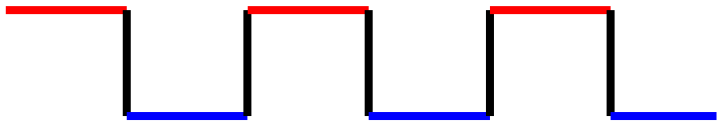
✓ if ((P1IFG&BIT3)!=0) 可否写成 if ((P1IFG&BIT3)==1)

✓如果P1.3用内部上拉电阻，如何编程？

✓如果P1.3上的脉冲以1Hz的频率出现， 可实现什么功能？

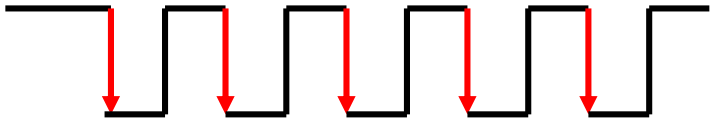
如果用单片机内部的上拉电阻





求反P1.6的输出
改变灯的状态

按键中断请求



P1.3中断C语言程序清单 C5_1_2.c

外部按键无上拉电阻

```
#include "msp430.h"
int main( void )
{   WDTCTL = WDTPW + WDTHOLD;          //关闭看门狗
    _DINT();                          //禁止可屏蔽中断 GIE=0
    P1IE &=~BIT3;                     // 关闭P1.3中断允许
    P1SEL &=~BIT6;                    //设置P1.6为基本I/O功能
    P1SEL2 &=~BIT6;                   //
    P1OUT &=~BIT6;                    //置P1.6输出初值
    P1DIR |=BIT6;                     //设置P1.6为输出
    P1SEL &=~BIT3;                    //置P1.3作为基本I/O端口
    P1SEL2 &=~BIT3;                   //
    P1DIR &=~BIT3;                    //置P1.3为输入
    P1REN |=BIT3;                     //置P1.3 内部电阻允许
    P1OUT |=BIT3;                     //置P1.3 为上拉电阻
    P1IES |=BIT3;                     //置P1.3下降沿作中断源
    P1IFG =0;                         //清P1IFG中断标志
    P1IE |=BIT3;                      //打开P1.3中断允许
   _EINT();                           //允许可屏蔽中断 GIE=1
   while(1) {   };                    //主循环
}
#pragma  vector=PORT1_VECTOR                    //置P1中断向量
__interrupt void port_int(void)                 //中断子程
{    if  ( (P1IFG&BIT3)!=0 )          //判断是否是P1IFG.3中断标志
     {       P1OUT ^=BIT6;           //对P1.6取反
             P1IFG &=~BIT3;                     //清P1.3中断标志
      }
}
```

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
…	…
定时器A1	TACCR1 to TACCR2 CCIFGs, TAIFG (3)
P2的8个引脚	P2IFG.0~P2IFG.7 (8)
P1的8个引脚	P1IFG.0~P1IFG.7 (8)

思考:

在例子中的中断子程内不清P1IFG.P3中断标志的后果？

```
#pragma  vector=PORT1_VECTOR                    //置P1中断向量
__interrupt void port_int(void)                 //中断子程
{    if  ( (P1IFG&BIT3)!=0 )     //判断是否是P1IFG.3中断标志
     {     P1OUT ^=BIT6;                        //对P1.6取反
//          P1IFG &=~BIT3;                      //清P1.3中断标志
      }
}
```