中断与异常1

程晨闻
东南大学电气工程学院

- 中断和异常基本概念
- 中断的响应流程
- 中断控制寄存器的访问

➢ 如果下面有一段程序

```
While(1){
        DoSomething();      //耗时3s

        CheckKey();         //检测是否有按键按下

        Display();          //显示数据
}
```

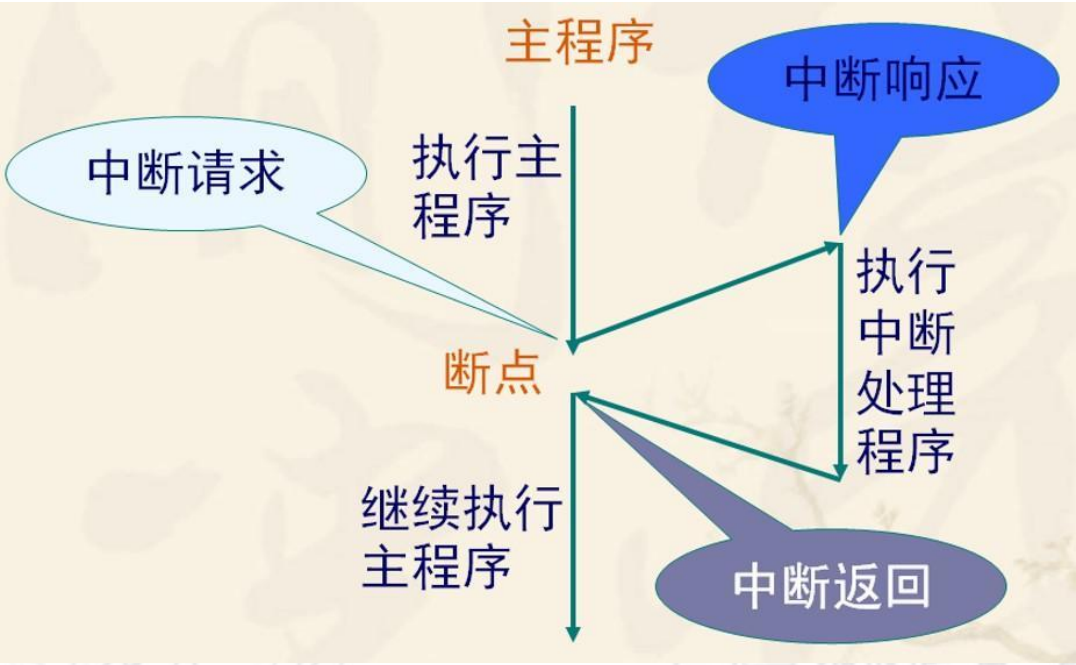
有可能检测不到按键

如何能够保证按键按下后能够及时检测到并响应?

➢ CPU的中断和异常

– 程序运行过程中，发生了不可预知的事件需要暂停当前的任务立刻处理

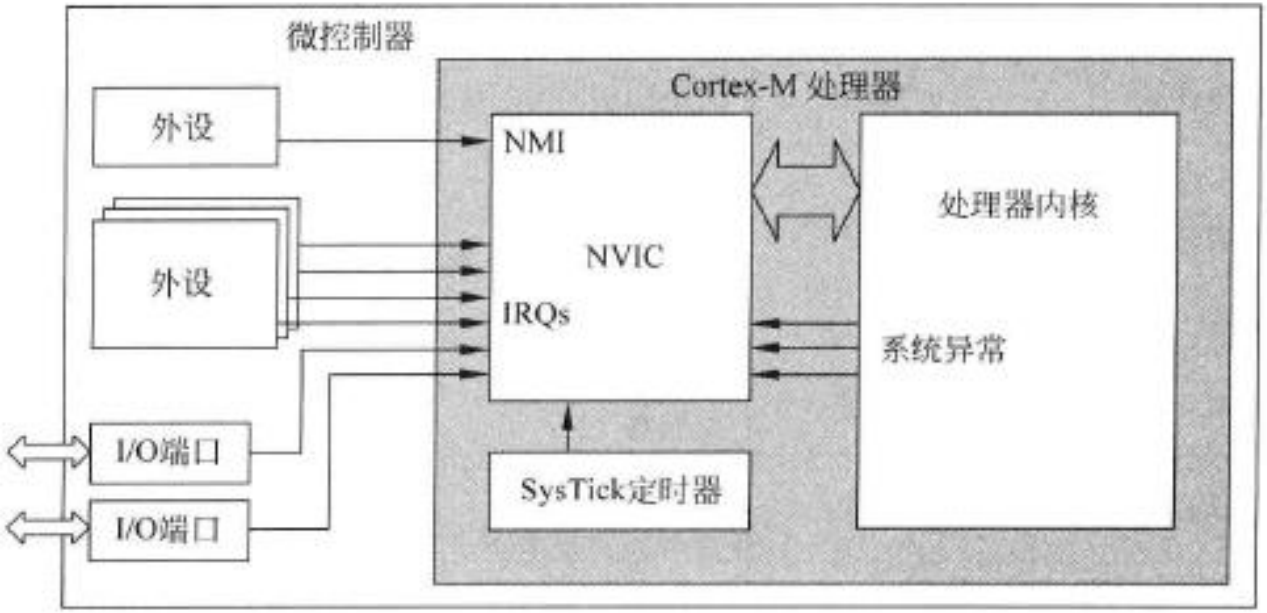
- 查看按键是否按下，查看定时器是否超时
- 除法除数为零、系统故障



➢ TM4C1294的中断和异常

– 由嵌套向量中断控制器NVIC管理

- 灵活的中断和异常管理：禁止、使能、软件触发，清除、电平触发、脉冲触发
- 支持嵌套：优先级管理与抢占
- 向量化入口：自动定位中断
- 中断也被看做是一种异常



TM4C1294的中断和异常

- 在ARM中，一般由处理器系统内部引发的事件，称作异常，如重启Reset，硬件错误等；由处理器外部，也就是外设引发的事件，称作中断，如GPIO模块等
- Cortex-M架构支持多种异常和外部中断，每个中断和异常都有编号，编号1~15为系统异常，16及以上的则为中断输入
- 包括中断在内的大多数的异常的优先级都是可编程的，一些系统异常具有固定的优先级。

系统异常列表

异常编号	异常类型	优先级	描述
1	复位	-3（最高）	复位
2	NMI	-2	不可屏蔽中断（外部NMI输入）
3	硬件错误	-1	所有错误都可能会引发，前提是相应的错误处理未使能
4	内存管理错误	可编程	存储器管理错误，存储器管理单元（MPU）冲突或访问非法位置
5	总线错误	可编程	当高级高性能总线（AHB）接口收到从总线的错误响应时产生
6	使用错误	可编程	程序错误或试图访问协处理器导致的错误
7~10	保留	NA	—
11	SVC	可编程	请求管理调用。一般用于OS环境且允许应用任务访问系统服务
12	调试监控	可编程	在使用基于软件的调试方案时，断点和监视点等调试事件的异常
13	保留	NA	—
14	PendSV	可编程	可挂起的服务调用。OS一般用该异常进行上下文切换
15	SYSTICK	可编程	当其在处理器中存在时，有定时器外设产生，可用于OS或简单的定时器外设
16	中断	可编程	由外设发出信号或软件请求产生，并通过NVIC（设置优先级）关联的异常

➢ TM4C1294中断列表

– 中断编号=异常编号-16

Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
0-15	-	0x0000.0000 - 0x0000.003C	Processor exceptions
16	0	0x0000.0040	GPIO Port A
17	1	0x0000.0044	GPIO Port B
18	2	0x0000.0048	GPIO Port C
19	3	0x0000.004C	GPIO Port D
20	4	0x0000.0050	GPIO Port E
21	5	0x0000.0054	UART0
22	6	0x0000.0058	UART1
23	7	0x0000.005C	SSI0
24	8	0x0000.0060	I^2C0
...			
118	102	0x0000.01D8	I^2C 6
119	103	0x0000.01DC	I^2C 7
120-124	104-108	-	Reserved
125	109	0x0000.01F4	I^2C 8
126	110	0x0000.01F8	I^2C 9
127-129	111-113	-	Reserved

➢ 中断的响应流程

– 接受异常请求的条件

- 处理器正在运行
- 异常处于使能状态
- 异常的优先级高于当前等级
- 异常未被屏蔽

中断产生

中断响应

➢ 中断的响应流程

– 异常进入

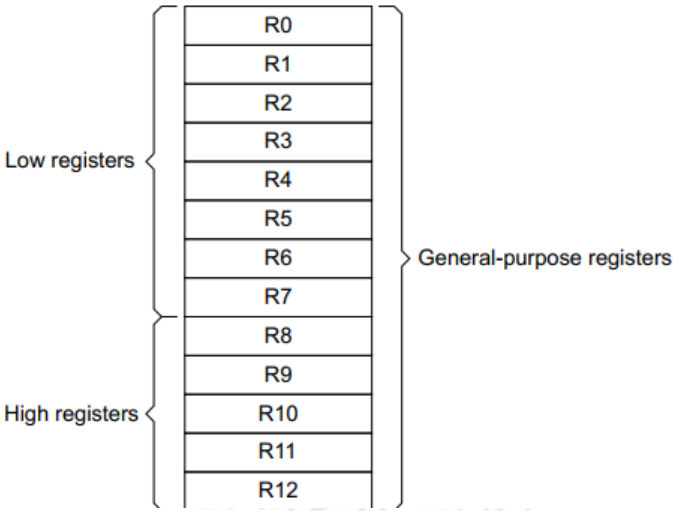
- CPU内核寄存器和返回地址被压入当前使用的栈
- 取出中断向量
- 取出待执行异常处理的指令
- 更新NVIC寄存器和内核寄存器

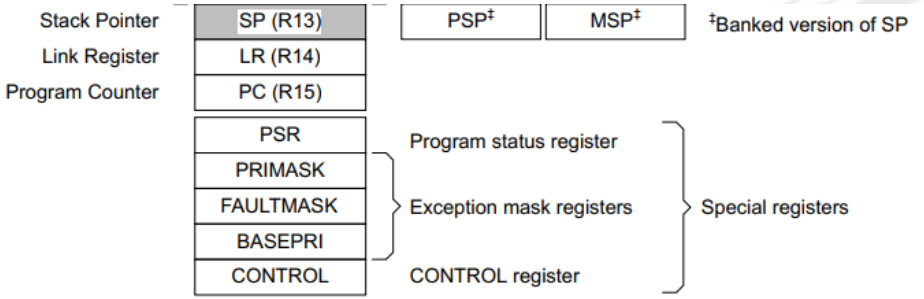
可以同时进行

更新中断状态

更新程序逻辑的执行状态：程序状态寄存器（PSR）、链接寄存器（LR）、程序计数器(PC)、堆栈指针(SP)

Figure 2-3. Cortex-M4F Register Set





中断向量表

- 堆栈指针复位值
- 起始地址

Figure 2-6. Vector Table

Exception number	IRQ number	Offset	Vector
(N+16)	(N)	0x040 + 0x(N*4)	IRQ N
.	.	.	.
.	.	.	.
.	.	.	.
		0x004C	
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
0		0x0000	Initial SP value

➢ 中断的响应流程

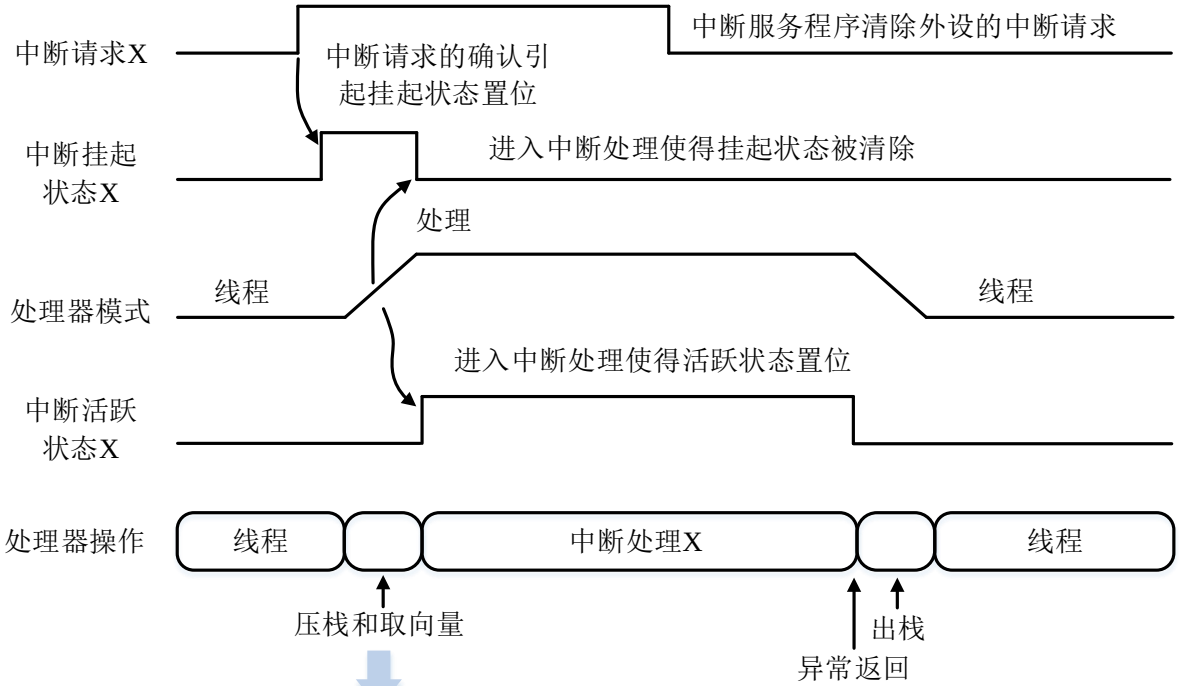
– 执行异常处理

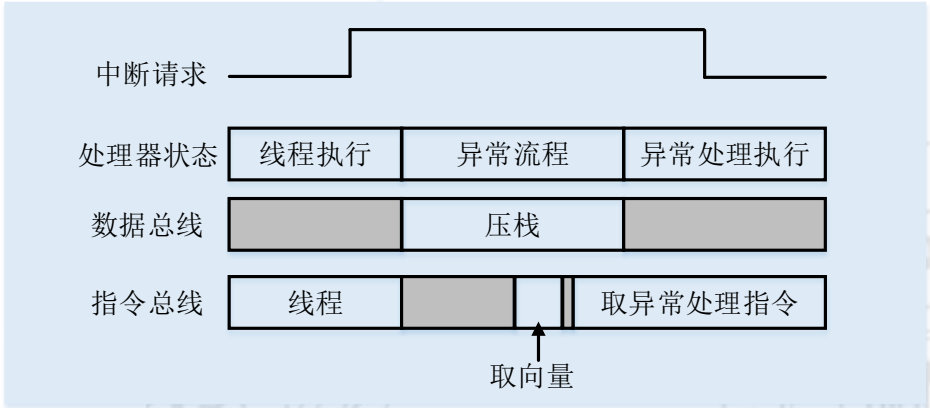
- 执行相应的中断服务函数
- 如果有更高优先级的异常产生，则接受新的异常，当前程序被抢占（异常嵌套）
- 如果新的异常的优先级等于或者低于当前的异常，则等到当前异常处理完成后才会得到处理

– 异常返回

- CPU内核寄存器出栈
- NVIC寄存器更新
- CPU取出之前被中断的程序指令，程序恢复执行

中断的响应流程

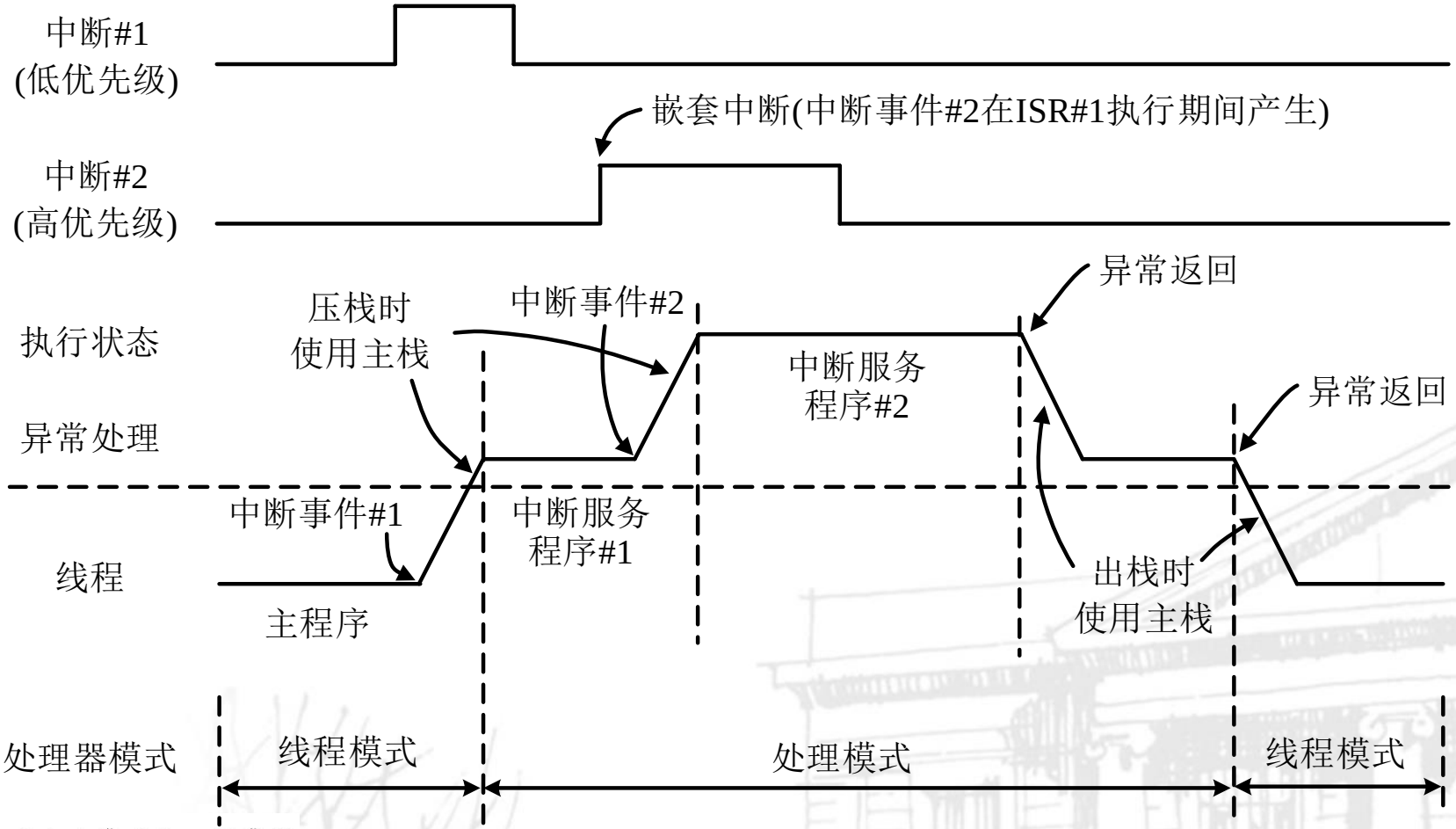




压栈和取向量可同时进行

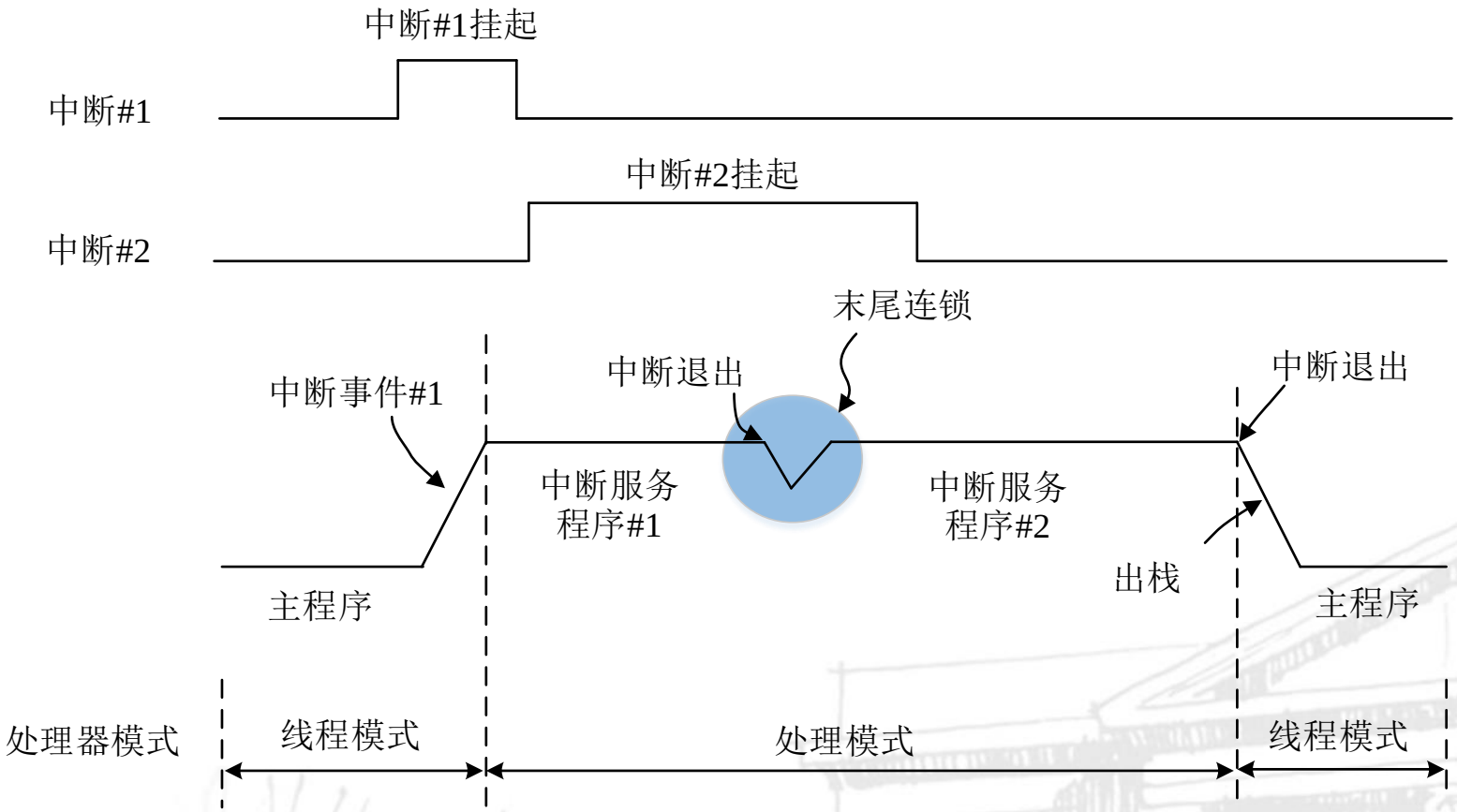
➢ 中断的响应流程

– 嵌套中断：两个中断同时发生时，优先服务优先级高的中断



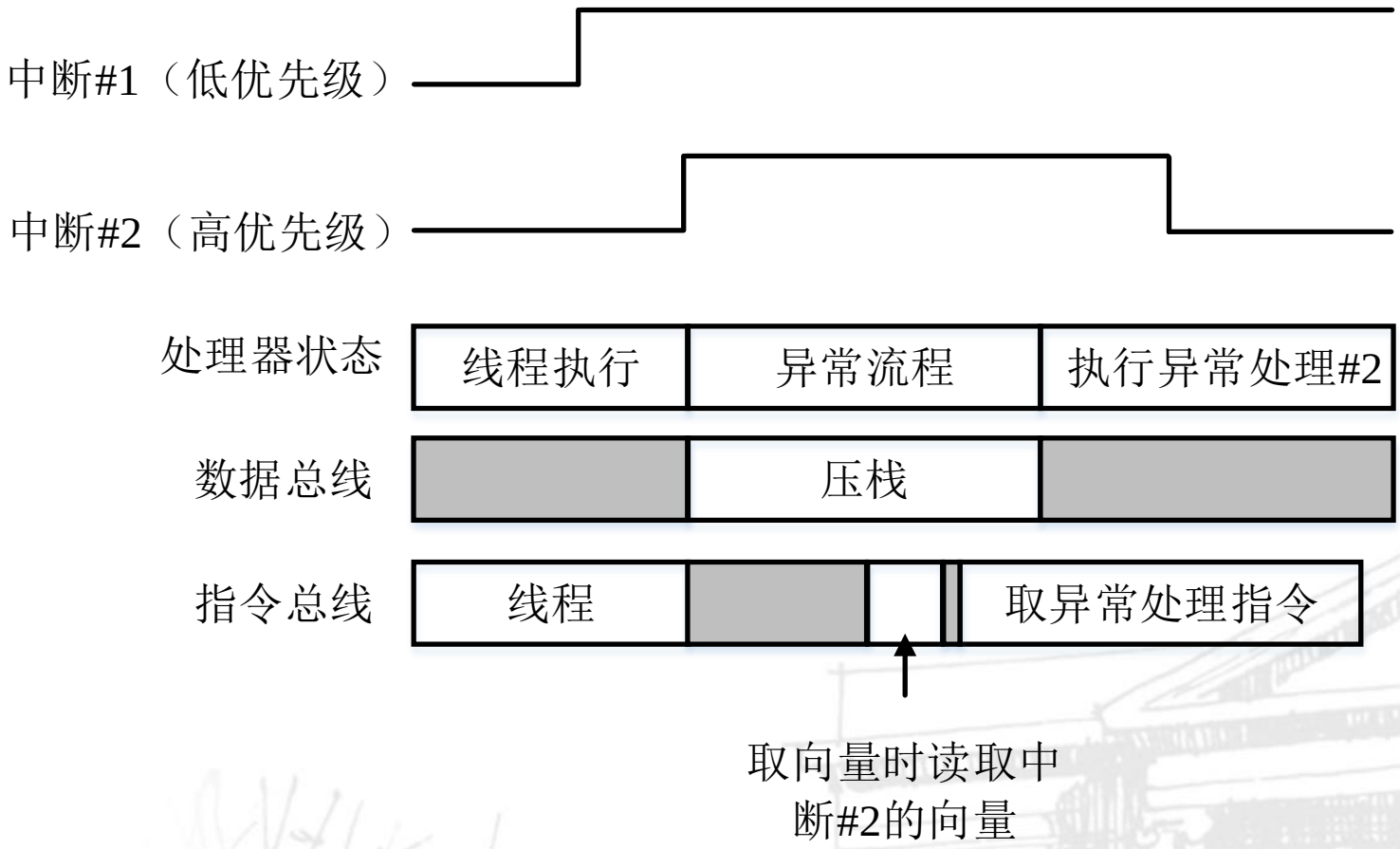
➢ 中断的响应流程

– 尾链：加速异常处理，跳过出栈



➢ 中断的响应流程

– 延迟到达：加速抢占，状态保存不被中断



外设		NVIC	CPU
产生中断	发出中断	接收中断	接收中断

➢ CPU中断的控制

– 总中断屏蔽寄存器（PRIMASK）（内核）

Priority Mask Register (PRIMASK)

Type RW, reset 0x0000.0000

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

Bit/Field	Name	Type	Reset	Description
31:1	reserved	RO	0x0000.000	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
0	PRIMASK	RW	0	Priority Mask

Value	Description
1	Prevents the activation of all exceptions with configurable priority.
0	No effect.

– **CPU寄存器组中的PRIMASK寄存器的最低位控制CPU是否接收中断请求。必须将PRIMASK设置为零，才能使CPU接收中断。**

CPU中断的控制

- 总中断屏蔽寄存器（PRIMASK）（内核）
- PRIMASK是CPU寄存器组中的寄存器，在总线上没有地址，因此必须使用汇编整理对其操作:

在汇编代码中，CPSID和CPSIE用于快速的开关中断:

CPSID	IRQ中断	PRIMASK=1	关中断
CPSIE	IRQ中断	PRIMASK=0	开中断
CPSID	FIQ中断	FAULTMASK=1	关异常
CPSIE	FIQ中断	FAULTMASK=0	开异常

➢ CPU中断的控制

– 总中断屏蔽寄存器（PRIMASK）（内核）

– 库函数提供了IntMasterEnable()函数开启CPU中断

IntMasterEnable

Enables the processor interrupt.

Prototype:

```
bool
IntMasterEnable(void)
```

Description:

This function allows the processor to respond to interrupts. This function does not affect the set of interrupts enabled in the interrupt controller; it just gates the single interrupt from the controller to the processor.

Example: Enable interrupts to the processor.

```
//
// Enable interrupts to the processor.
//
IntMasterEnable();
```

Returns:

Returns **true** if interrupts were disabled when the function was called or **false** if they were initially enabled.

➢ CPU中断的控制

– 总中断屏蔽寄存器（PRIMASK）（内核）

– 库函数提供了IntMasterEnable()函数开启CPU中断

TivaWare_C_Series-2.1.4.178\driverlib\interrupt.c

```
bool
IntMasterEnable(void)
{
    //
    // Enable processor interrupts.
    //
    return(CPUcpsie());
}
```

TivaWare_C_Series-2.1.4.178\driverlib\cpu.c

```
uint32_t
CPUcpsie(void)
{
    //
    // Read PRIMASK and enable interrupts.
    //
    __asm("    mrs     r0, PRIMASK\n"
          "    cpsie   i\n"
          "    bx      lr\n");//程序返回

    return(0);
}
```

➢ 中断使能寄存器（ENn）

Interrupt 0-31 Set Enable (EN0), offset 0x100

Interrupt 32-63 Set Enable (EN1), offset 0x104

Interrupt 64-95 Set Enable (EN2), offset 0x108

Interrupt 96-113 Set Enable (EN3), offset 0x10C

外设		NVIC	CPU
产生中断	发出中断	接收中断	接收中断

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.

➢ 中断禁止寄存器（DISn）

外设		NVIC	CPU
产生中断	发出中断	接收中断	接收中断

Interrupt 0-31 Clear Enable (DIS0), offset 0x180

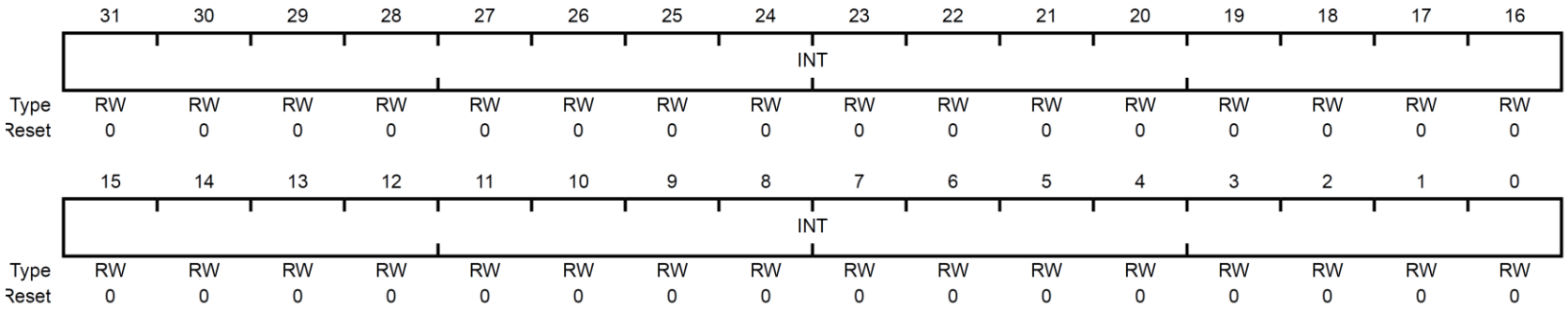
Interrupt 32-63 Clear Enable (DIS1), offset 0x184

Interrupt 64-95 Clear Enable (DIS2), offset 0x188

Interrupt 96-113 Clear Enable (DIS3), offset 0x18C

Interrupt 0-31 Clear Enable (DIS0)

Base 0xE000.E000
Offset 0x180
Type RW, reset 0x0000.0000



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

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, clears the corresponding `INT[n]` bit in the **EN0** register, disabling interrupt [n].

- 对ENn写1，开启对应的中断
- 对DISn写1，关闭对应的中断
- 库函数中，提供了IntEnable()和IntDisable()函数，来开启和关闭指定的中断和异常。

```
IntEnable(INT_GPIOJ);      //接收来自GIPOJ端口的中断请求
IntDisable(INT_UART0);     //关闭来自UART0模块的中断请求
```

➢ IntEnable

IntEnable

Enables an interrupt.

Prototype:

```
void
IntEnable(uint32_t ui32Interrupt)
```

Parameters:

ui32Interrupt specifies the interrupt to be enabled.

Description:

The specified interrupt is enabled in the interrupt controller. The *ui32Interrupt* parameter must be one of the valid **INT_*** values listed in Peripheral Driver Library User's Guide and defined in the inc/hw_ints.h header file. Other enables for the interrupt (such as at the peripheral level) are unaffected by this function.

Example: Enable the UART 0 interrupt.

```
//
// Enable the UART 0 interrupt in the interrupt controller.
//
IntEnable(INT_UART0);
```

Returns:

None.

➢ IntEnable

TivaWare_C_Series-2.1.4.178\driverlib\interrupt.c

```
void
IntEnable(uint32_t ui32Interrupt)
{
    //
    // Check the arguments.
    //
    ASSERT(ui32Interrupt < NUM_INTERRUPTS);

    //
    // Determine the interrupt to enable.
    //
    if(ui32Interrupt == FAULT_MPU)
    {
        //
        // Enable the MemManage interrupt.
        //
        HWREG(NVIC_SYS_HND_CTRL) |= NVIC_SYS_HND_CTRL_MEM;
    }
    else if(ui32Interrupt == FAULT_BUS)
    {
        //
        // Enable the bus fault interrupt.
        //
        HWREG(NVIC_SYS_HND_CTRL) |= NVIC_SYS_HND_CTRL_BUS;
    }
    else if(ui32Interrupt == FAULT_USAGE)
    {
        //
        // Enable the usage fault interrupt.
        //
        HWREG(NVIC_SYS_HND_CTRL) |=
NVIC_SYS_HND_CTRL_USAGE;
    }
    else if(ui32Interrupt == FAULT_SYSTICK)
    {
        //
        // Enable the System Tick interrupt.
        //
        HWREG(NVIC_ST_CTRL) |= NVIC_ST_CTRL_INTEN;
    }
    else if(ui32Interrupt >= 16)
    {
        //
        // Enable the general interrupt.
        //
        HWREG(g_pui32EnRegs[(ui32Interrupt - 16) / 32]) =
            1 << ((ui32Interrupt - 16) & 31);
    }
}
```

➢ IntEnable

```
else if(ui32Interrupt >= 16)
    {
        // Enable the general interrupt.
        HWREG(g_pui32EnRegs[(ui32Interrupt - 16) / 32]) =
            1 << ((ui32Interrupt - 16) & 31);
    }
```

- 输入参数ui32Interrupt为异常编号，事先在hw_ints.h文件中定义好

```
#define INT_GPIOA_TM4C129       16        // GPIO Port A
#define INT_GPIOB_TM4C129       17        // GPIO Port B
#define INT_GPIOC_TM4C129       18        // GPIO Port C
#define INT_GPIOD_TM4C129       19        // GPIO Port D
#define INT_GPIOE_TM4C129       20        // GPIO Port E
#define INT_GPIOJ_TM4C129       67        // GPIO Port J
#define INT_GPIOK_TM4C129       68        // GPIO Port K
#define INT_GPIOL_TM4C129       69        // GPIO Port L
```

- 写 IntEnable(INT_GPIOJ);时，INT_GPIOJ会通过相关的宏定义，在编译时自动转换为INT_GPIOJ_TM4C129，即67。这种方式是为了适应多种类型的芯片。

➢ IntEnable

```
else if(ui32Interrupt >= 16)
    {
        // Enable the general interrupt.
        HWREG(g_pui32EnRegs[(ui32Interrupt - 16) / 32]) =
            1 << ((ui32Interrupt - 16) & 31);
    }
```

```
static const uint32_t g_pui32EnRegs[] =
{
    NVIC_EN0, NVIC_EN1, NVIC_EN2, NVIC_EN3, NVIC_EN4
};
```

```
#define NVIC_EN0                0xE000E100  // Interrupt 0-31 Set Enable
#define NVIC_EN1                0xE000E104  // Interrupt 32-63 Set Enable
#define NVIC_EN2                0xE000E108  // Interrupt 64-95 Set Enable
#define NVIC_EN3                0xE000E10C  // Interrupt 96-127 Set Enable
#define NVIC_EN4                0xE000E110  // Interrupt 128-159 Set Enable
```

Interrupt 0-31 Set Enable (EN0)

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

Register 4: Interrupt 0-31 Set Enable (EN0), offset 0x100

Register 5: Interrupt 32-63 Set Enable (EN1), offset 0x104

Register 6: Interrupt 64-95 Set Enable (EN2), offset 0x108

Register 7: Interrupt 96-113 Set Enable (EN3), offset 0x10C

➢ IntDisable

– IntDisable函数与IntEnable的实现过程类似，只是将数组g_pui32EnRegs换成了g_pui32Dii16Regs

```
static const uint32_t g_pui32Dii16Regs[] =
{
    NVIC_DIS0, NVIC_DIS1, NVIC_DIS2, NVIC_DIS3, NVIC_DIS4
};
```

TivaWare_C_Series-2.1.4.178\inc\ hw_nvic.h

```
#define NVIC_DIS0               0xE000E180  // Interrupt 0-31 Clear Enable
#define NVIC_DIS1               0xE000E184  // Interrupt 32-63 Clear Enable
#define NVIC_DIS2               0xE000E188  // Interrupt 64-95 Clear Enable
#define NVIC_DIS3               0xE000E18C  // Interrupt 96-127 Clear Enable
#define NVIC_DIS4               0xE000E190  // Interrupt 128-159 Clear Enable
```

Interrupt 0-31 Clear Enable (DIS0)

Base 0xE000.E000
Offset 0x180
Type RW, reset 0x0000.0000

Register 8: Interrupt 0-31 Clear Enable (DIS0), offset 0x180

Register 9: Interrupt 32-63 Clear Enable (DIS1), offset 0x184

Register 10: Interrupt 64-95 Clear Enable (DIS2), offset 0x188

Register 11: Interrupt 96-113 Clear Enable (DIS3), offset 0x18C

谢谢！