提供COSMIC 代码供参考：

一. 中断函数声明 以及 中断向量设置：

```
//=====================================================================
// TIM4 溢出中断声明
//=====================================================================
@far @interrupt void irq_system_tim4_ovf (void);


//=====================================================================
// 中断向量表
//=====================================================================
struct interrupt_vector const _vectab[] = {                //
{0x82, (interrupt_handler_t)_stext},             //      RESET   启动复位
{0x82, NonHandledInterrupt},                     //      TRAP    软件中断
{0x82, NonHandledInterrupt},                     // 00   TLI     外部最高级中断
{0x82, NonHandledInterrupt},                     // 01   AWU     中停机模式自动唤醒
{0x82, irq_system_clk     },                     // 02   CLK     始终控制器
{0x82, NonHandledInterrupt},                     // 03   EXTI0   端口A外部中断
{0x82, irq_system_wake_up },                     // 04   EXTI1   端口B外部中断
{0x82, NonHandledInterrupt},                     // 05   EXTI2   端口C外部中断
{0x82, NonHandledInterrupt},                     // 06   EXTI3   端口D外部中断
{0x82, NonHandledInterrupt},                     // 07   EXTI4   端口E外部中断
{0x82, NonHandledInterrupt},                     // 08   CAN     CAN RX 中断
{0x82, NonHandledInterrupt},                     // 09   CAN     CAN TX/ER/SC 中断
{0x82, NonHandledInterrupt},                     // 10   SPI     传输结束
{0x82, NonHandledInterrupt},                     // 11   TIM1    定时器1 更新/上溢出/下溢出/触发/刹车
{0x82, NonHandledInterrupt},                     // 12   TIM1    定时器1 捕获/比较
{0x82, NonHandledInterrupt},                     // 13   TIM2    定时器2 更新/上溢出
{0x82, NonHandledInterrupt},                     // 14   TIM2    定时器2 捕获/比较
{0x82, NonHandledInterrupt},                     // 15   TIM3    定时器3 更新/上溢出
{0x82, NonHandledInterrupt},                     // 16   TIM3    定时器3 捕获/比较
{0x82, irq_uart1_txe      },                     // 17   UART1   发送完成
{0x82, irq_uart1_rxne     },                     // 18   UART1   接收寄存器数据满
{0x82, NonHandledInterrupt},                     // 19   I2C     I2C中断
{0x82, NonHandledInterrupt},                     // 20   UART2/3 发送完成
{0x82, NonHandledInterrupt},                     // 21   UART2/3 接收寄存器数据满
{0x82, NonHandledInterrupt},                     // 22   ADC1    ADC1转换结束/模拟看门狗中断
{0x82, irq_system_tim4_ovf},                     // 23   TIM4    定时器4 更新/上溢出
{0x82, NonHandledInterrupt},                     // 24   FLASH   编程结束/禁止编程
{0x82, NonHandledInterrupt},                     //
{0x82, NonHandledInterrupt},                     //
{0x82, NonHandledInterrupt},                     //
{0x82, NonHandledInterrupt},                     //
{0x82, NonHandledInterrupt},                     //
};
```

二. 初始化

```
//=====================================================================
// 函数名称：sub_system_tim4_init
// 函数功能：系统定时 1ms，针对 11.0592MHz 晶体
// 入口参数：无
// 出口参数：无
// 程序版本：1.0
// 编写日期：
// 程序作者：
// 修改次数：
// 修改作者：
// 修改日期：
// 修改内容：
// 版本升级：
//=====================================================================
void sub_system_tim4_init(void)
{
    TIM4_PSCR = 0x06;                                          // 1. MASTER 时钟 64 分频
    TIM4_ARR  = 173-1;                                         // 2. 设置自动重载寄存器
    TIM4_SR   = 0x00;                                          // 3. 清零中断标志
    TIM4_IER  = 0x01;                                          // 4. 开启溢出中断允许
    TIM4_CR1  = 0x01;                                          // 5. 开启定时器
}


三. 中断处理程序
//=====================================================================
// 函数名称：irq_system_tim4_ovf
// 函数功能：系统基础定时用，中断时间 1ms
```

```
// 入口参数：无
// 出口参数：无
// 程序版本：1.0
// 编写日期：
// 程序作者：
// 修改次数：
// 修改作者：
// 修改日期：
// 修改内容：
// 版本升级：
//=====================================================================
@far @interrupt void irq_system_tim4_ovf (void)
{
    if( system_delay != 0 ) {                                          // 系统定时，基于1ms
        system_delay--;                                                //
    }                                                                  //

    if( uart1_delay != 0 ) {                                           //
        uart1_delay --;                                                //
    }                                                                  //
}
```

这个定义语句中，typedef void @far (*interrupt_handler_t)(void); 它定义了一个函数指针类型，这个函数没有调用参数，也没有返回参数，这是一个指针长指针。这个指针类型的名字是interrupt_handler_t。

这里唯一一个非标准C语言的元素是@far。在STM8中，指定地址的长度可以有3种方式，@tiny使用1个字节表示地址，只能寻址地址范围0x00~0xFF；@near使用2个字节表示地址，只能寻址地址范围0x0000~0xFFFF；@far使用3个字节表示地址，寻址地址范围0x000000~0xFFFFFF。

1、typedef void @far (*interrupt_handler_t)(void); 定义了一个函数指针的别名。
2、用interrupt_handler_t给interrupt_handler做类型限制，并定义了一个结构名interrupt_vector
3、用interrupt_vector和const做_vectab[]的类型限制，定义了一个元素为常量的数组.
4、数组的某一个元素：0x82, NonHandledInterrupt，0X82相当于unsigned char interrupt_instruction=0x82;从interrupt_instruction字面意思看似乎是中断指令，但为什么是0X82还是没弄明白。
5、@far @interrupt void NonHandledInterrupt (void)的意思：
   所有未用到的中断，其名字在中断向量数组内的名字都是NonHandledInterrupt，如果执行时PC异常跳到这个地址后，都会去执行NonHandledInterrupt这个函数而自动退出。
6、_vectab[]是否为编译器内部已经规定好的一个关键字，否则它如何正确的指向到中断的地址空间？
7、_stext的定义还是没查到在哪里有？！
8、现在最主要的是还是没有明白vectab[]里面数值是如何和正确跳转到中断入口联系起来。

如果结合看第2个问题和第5个问题就容易了，第5个问题是定义了一个结构数组，这个结构就是第2个问题中定义的结构interrupt_vector，这里定义结构数组的同时，对这个结构数组进行了初始化，以后的每行都是数组中的一个分量。

就以数组中的第2行为例：
{0x82, NonHandledInterrupt}, /* trap */
0x82对应结构的第1个分量unsigned char interrupt_instruction，这正是你的第2个问题中找不到在哪里用到的东西。NonHandledInterrupt对应结构的第2个分量interrupt_handler_t interrupt_handler，而这个分量的类型正好是你的第1个问题中定义的指针类型，它是一个指向函数的指针，这里把这个指针初始化为指向NonHandledInterrupt函数，即你的第3个问题中提到的函数。

你的第3个问题中提到的函数，它内部除了一句return也什么也没有，这是为了在你没有定义它的内容时的一个默认动作，通常你需要自己定义每个处理函数的动作，然后在初始化数组_vectab[]的时候，用自己的函数名替换掉那个默认的函数NonHandledInterrupt。

最后关于你的第4个问题：_stext()似乎是一个汇编函数，是由编译器提供的，它的目的是在芯片复位后至进入你的main()之前，编译器需要进行一些预处理动作，如初始化某些编译器需要用到的变量等，具体有哪些内容，我没有看过，不太清楚。你可以不必理会它在哪里，它做什么。

关于__vectab，找到了一个‘最可信’的答案，请ST的各位大佬指正：
如附件两个图，从编译后生成的MAP文件看，__vectab被定位到0X8000地址，在lkf文件和其它所有文件中没有找到__vectab，但从图2的vector file name和vector addre对话框内的内容看，stm8_interrupt_vector.c这个文件名应该任意修改，但项目建立的时候应该默认的是这个文件名，只要把vector file name中文件修改为和重起的文件名字一致应该也没问题，但估计__vectab应该是个关键字，直接对应到了vector addre，不可修改。
关于0x82的问题，根据stm8_interrupt_vector.ls这个文件中的内容看：

```
  48  0000               __vectab:
  49  0000 82             dc.b 130
  51  0001 00             dc.b page(__stext)
  52  0002 0000           dc.w __stext
  53  0004 82             dc.b 130
  55  0005 00             dc.b page(f_NonHandledInterrupt)
  56  0006 0000           dc.w f_NonHandledInterrupt
```

```
  57  0008 82            dc.b 130
  59  0009 00            dc.b page(f_NonHandledInterrupt)
  60  000a 0000          dc.w f_NonHandledInterrupt
  61  000c 82            dc.b 130
  63  000d 00            dc.b page(f_NonHandledInterrupt)
  64  000e 0000          dc.w f_NonHandledInterrupt
  65  0010 82            dc.b 130
  67  0011 00            dc.b page(f_NonHandledInterrupt)
  68  0012 0000          dc.w f_NonHandledInterrupt
  69  0014 82            dc.b 130
  71  0015 00            dc.b page(f_NonHandledInterrupt)
  72  0016 0000          dc.w f_NonHandledInterrupt
  73  0018 82            dc.b 130
  75  0019 00            dc.b page(f_NonHandledInterrupt)
  76  001a 0000          dc.w f_NonHandledInterrupt
  77  001c 82            dc.b 130
  79  001d 00            dc.b page(f_NonHandledInterrupt)
  80  001e 0000          dc.w f_NonHandledInterrupt
  81  0020 82            dc.b 130
  83  0021 00            dc.b page(f_NonHandledInterrupt)
  84  0022 0000          dc.w f_NonHandledInterrupt
  85  0024 82            dc.b 130
  87  0025 00            dc.b page(f_NonHandledInterrupt)
  88  0026 0000          dc.w f_NonHandledInterrupt
  89  0028 82            dc.b 130
  91  0029 00            dc.b page(f_NonHandledInterrupt)
  92  002a 0000          dc.w f_NonHandledInterrupt
  93  002c 82            dc.b 130
  95  002d 00            dc.b page(f_NonHandledInterrupt)
  96  002e 0000          dc.w f_NonHandledInterrupt
  97  0030 82            dc.b 130
  99  0031 00            dc.b page(f_NonHandledInterrupt)
100  0032 0000          dc.w f_NonHandledInterrupt
101  0034 82            dc.b 130
```

终于彻底的搞清楚它了。。。
1、struct interrupt_vector const _vectab[]={}
2、+seg .const -b 0x8000 -k
原来被定位到0x8000还有const的功劳，去掉const就从0x0000开始了。。。

vectab[]只是随便定义了数组，这个数组在应用代码中根本不需要用到，所以这个数组名字自然就可以随便起了，不是什么关键字， vectab的连接定位完全由const决定的，当修改了项目设置中的vector addre出现LKF文件和project的settings中vector addre不一致的时候，优先使用了对话框内的设置。

冥想下找到的答案，估计这个答案应该比较接近真实根源了。

现在还差一个0x82的问题没有弄清楚，dc.b这个东西一时还没查到这个指令的操作码，还没搞清楚它。
其实你可以理解这条不公开的指令为一个特殊的CALL指令，它的作用就是跳转到紧跟着的3个字节指向的地址，同时把相应的寄存器压入堆栈，与IRET指令对应。

CALL与RET成对，而这个0x82指令与IRET成对。