;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; 全局变量

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IMPORT rt\_thread\_switch\_interrupt\_flag

IMPORT rt\_interrupt\_from\_thread

IMPORT rt\_interrupt\_to\_thread

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; 常量

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;-------------------------------------------------------------------------

;有关内核外设寄存器定义可参考官方文档：STM32F10xxx Cortex-M3 programming manual

;系统控制块外设SCB地址范围：0xE000ED00-0xE000ED3F

;-------------------------------------------------------------------------

SCB\_VTOR EQU 0xE000ED08 ; 向量表偏移寄存器

NVIC\_INT\_CTRL EQU 0xE000ED04 ; 中断控制状态寄存器

NVIC\_SYSPRI2 EQU 0xE000ED20 ; 系统优先级寄存器(2)

NVIC\_PENDSV\_PRI EQU 0x00FF0000 ; PendSV 优先级值 (lowest)

NVIC\_PENDSVSET EQU 0x10000000 ; 触发PendSV exception的值

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; 代码产生指令

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

AREA |.text|, CODE, READONLY, ALIGN=2

THUMB

REQUIRE8

PRESERVE8

;/\*

; \*-----------------------------------------------------------------------

; \* 函数原型：void rt\_hw\_context\_switch\_to(rt\_uint32 to);

; \* r0 --> to

; \* 该函数用于开启第一次线程切换

; \*-----------------------------------------------------------------------

; \*/

rt\_hw\_context\_switch\_to PROC

; 导出rt\_hw\_context\_switch\_to，让其具有全局属性，可以在C文件调用

EXPORT rt\_hw\_context\_switch\_to

; 设置rt\_interrupt\_to\_thread的值

LDR r1, =rt\_interrupt\_to\_thread ;将rt\_interrupt\_to\_thread的地址加载到r1

STR r0, [r1] ;将r0的值存储到rt\_interrupt\_to\_thread

; 设置rt\_interrupt\_from\_thread的值为0，表示启动第一次线程切换

LDR r1, =rt\_interrupt\_from\_thread ;将rt\_interrupt\_from\_thread的地址加载到r1

MOV r0, #0x0 ;配置r0等于0

STR r0, [r1] ;将r0的值存储到rt\_interrupt\_from\_thread

; 设置中断标志位rt\_thread\_switch\_interrupt\_flag的值为1

LDR r1, =rt\_thread\_switch\_interrupt\_flag ;将rt\_thread\_switch\_interrupt\_flag的地址加载到r1

MOV r0, #1 ;配置r0等于1

STR r0, [r1] ;将r0的值存储到rt\_thread\_switch\_interrupt\_flag

; 设置 PendSV 异常的优先级

LDR r0, =NVIC\_SYSPRI2

LDR r1, =NVIC\_PENDSV\_PRI

LDR.W r2, [r0,#0x00] ; 读

ORR r1,r1,r2 ; 改

STR r1, [r0] ; 写

; 触发 PendSV 异常 (产生上下文切换)

LDR r0, =NVIC\_INT\_CTRL

LDR r1, =NVIC\_PENDSVSET

STR r1, [r0]

; 开中断

CPSIE F

CPSIE I

; 永远不会到达这里

ENDP

;/\*

; \*-----------------------------------------------------------------------

; \* void rt\_hw\_context\_switch(rt\_uint32 from, rt\_uint32 to);

; \* r0 --> from

; \* r1 --> to

; \*-----------------------------------------------------------------------

; \*/

;rt\_hw\_context\_switch\_interrupt

;EXPORT rt\_hw\_context\_switch\_interrupt

rt\_hw\_context\_switch PROC

EXPORT rt\_hw\_context\_switch

; 设置中断标志位rt\_thread\_switch\_interrupt\_flag为1

LDR r2, =rt\_thread\_switch\_interrupt\_flag ; 加载rt\_thread\_switch\_interrupt\_flag的地址到r2

LDR r3, [r2] ; 加载rt\_thread\_switch\_interrupt\_flag的值到r3

CMP r3, #1 ; r3与1比较，相等则执行BEQ指令，否则不执行

BEQ \_reswitch

MOV r3, #1 ; 设置r3的值为1

STR r3, [r2] ; 将r3的值存储到rt\_thread\_switch\_interrupt\_flag，即置1

; 设置rt\_interrupt\_from\_thread的值

LDR r2, =rt\_interrupt\_from\_thread ; 加载rt\_interrupt\_from\_thread的地址到r2

STR r0, [r2] ; 存储r0的值到rt\_interrupt\_from\_thread，即上一个线程栈指针sp的指针

\_reswitch

; 设置rt\_interrupt\_to\_thread的值

LDR r2, =rt\_interrupt\_to\_thread ; 加载rt\_interrupt\_from\_thread的地址到r2

STR r1, [r2] ; 存储r1的值到rt\_interrupt\_from\_thread，即下一个线程栈指针sp的指针

; 触发PendSV异常，实现上下文切换

LDR r0, =NVIC\_INT\_CTRL

LDR r1, =NVIC\_PENDSVSET

STR r1, [r0]

; 子程序返回

BX LR

; 子程序结束

ENDP

;/\*

; \*-----------------------------------------------------------------------

; \* void PendSV\_Handler(void);

; \* r0 --> switch from thread stack

; \* r1 --> switch to thread stack

; \* psr, pc, lr, r12, r3, r2, r1, r0 are pushed into [from] stack

; \*-----------------------------------------------------------------------

; \*/

PendSV\_Handler PROC

EXPORT PendSV\_Handler

; 失能中断，为了保护上下文切换不被中断

MRS r2, PRIMASK

CPSID I

; 获取中断标志位，看看是否为0

LDR r0, =rt\_thread\_switch\_interrupt\_flag ; 加载rt\_thread\_switch\_interrupt\_flag的地址到r0

LDR r1, [r0] ; 加载rt\_thread\_switch\_interrupt\_flag的值到r1

CBZ r1, pendsv\_exit ; 判断r1是否为0，为0则跳转到pendsv\_exit

; r1不为0则清0

MOV r1, #0x00

STR r1, [r0] ; 将r1的值存储到rt\_thread\_switch\_interrupt\_flag，即清0

; 判断rt\_interrupt\_from\_thread的值是否为0

LDR r0, =rt\_interrupt\_from\_thread ; 加载rt\_interrupt\_from\_thread的地址到r0

LDR r1, [r0] ; 加载rt\_interrupt\_from\_thread的值到r1

CBZ r1, switch\_to\_thread ; 判断r1是否为0，为0则跳转到switch\_to\_thread

; 第一次线程切换时rt\_interrupt\_from\_thread肯定为0，则跳转到switch\_to\_thread

; ========================== 上文保存 ==============================

; 当进入PendSVC Handler时，上一个线程运行的环境即：

; xPSR，PC（线程入口地址），R14，R12，R3，R2，R1，R0（线程的形参）

; 这些CPU寄存器的值会自动保存到线程的栈中，剩下的r4~r11需要手动保存

MRS r1, psp ; 获取线程栈指针到r1

STMFD r1!, {r4 - r11} ;将CPU寄存器r4~r11的值存储到r1指向的地址(每操作一次地址将递减一次)

LDR r0, [r0] ; 加载r0指向值到r0，即r0=rt\_interrupt\_from\_thread

STR r1, [r0] ; 将r1的值存储到r0，即更新线程栈sp

; ========================== 下文切换 ==============================

switch\_to\_thread

LDR r1, =rt\_interrupt\_to\_thread ; 加载rt\_interrupt\_to\_thread的地址到r1

; rt\_interrupt\_to\_thread是一个全局变量，里面存的是线程栈指针SP的指针

LDR r1, [r1] ; 加载rt\_interrupt\_to\_thread的值到r1，即sp指针的指针

LDR r1, [r1] ; 加载rt\_interrupt\_to\_thread的值到r1，即sp

LDMFD r1!, {r4 - r11} ;将线程栈指针r1(操作之前先递减)指向的内容加载到CPU寄存器r4~r11

MSR psp, r1 ;将线程栈指针更新到PSP

pendsv\_exit

; 恢复中断

MSR PRIMASK, r2

ORR lr, lr, #0x04 ; 确保异常返回使用的堆栈指针是PSP，即LR寄存器的位2要为1

BX lr ; 异常返回，这个时候任务堆栈中的剩下内容将会自动加载到xPSR，PC（任务入口地址），R14，R12，R3，R2，R1，R0（任务的形参）

; 同时PSP的值也将更新，即指向任务堆栈的栈顶。在ARMC3中，堆是由高地址向低地址生长的。

; PendSV\_Handler 子程序结束

ENDP

ALIGN 4

END