摘要

Keil 自带的 RTX51—Tiny 系统有这样几个缺点：1、非占先式任务调度，这样系统的实时性就很难保证；2、提供的系统服务太少，只有 wait 与 signal。而 RTX51—Tiny 的优点是：1、Keil 公司自己开发的，使用_task_关键字区别每个任务，这样可以使得被不同任务调用的不同函数即使没有相互调用，他们的局部变量也不会相互覆盖。免去了在 SmallRTOS 中需要手动制止函数间局部变量的相互覆盖。2、内核小。整个 RTX51--Tiny 完整编译只需 900B 的空间。

lq51 系统整合了 RTX51—Tiny 的优点。1、在 lq51 系统中_task_关键依然有效，0 号任务依然是最高优先级的任务，1 号任务次之，依次类推。2、lq51 系统完整编译只需 2.5K 的 ROM，并且系统自带的 128B RAM 对 lq51 系统也是绰绰有余的。同时 lq51 系统弥补了 RTX51—Tiny 的缺点。首先 lq51 系统采用占先式任务调度，这样可以最大限度保证系统的实时性。其次 lq51 系统提供 lqDelay、lqWaitFlg、 lqWaitSem、lqWaitMsg 这样四种系统服务，并且每种系统服务都是可裁剪的。

lq51 系统中很多关于实时系统的概念及实现方法是从 ucosII 中学来的，同时借鉴了 SmallRTOS 及 RTX51—Tiny。

adream307@163.com

lq51 简介

一、从一个例子开始：

例子在 /lq51_demo/exp1：

1、使用 keil 新建一个工程 exp1，选择 Generic->8051(all variants)。

2、把 lq51_c.c 及 lq51_a.asm 添加到 exp1 这个工程中。

3、新建一个 exp1.c 文件，在文件中输入一下内容：

```
#include"lq51.h"
#include<reg52.h>
void Task0(void)  _task_  0
{
    char i=0;
    lqStart();
    while(1){
        ++i;
        P0=i;
        lqDelay(100);
    }
}

void Task1(void)  _task_  1
{
        char i=50;
        while(1){
            P1=i;
            ++i;
            lqDelay(100);
        }
}

void Task2(void) _task_  2
{
    char i=100;
    while(1){
        P2=i;
        ++i;
        lqDelay(100);
    }
}
```

4、修改 lq51.h 如下：

```
#define   lqRamTop              0x7F
#define   LQ_DELAY_EN           1
#define   LQ_FLG_EN             0
```

```
#define LQ_SEM_EN               0
#define LQ_MSG_EN               0
#define LQ_TASK_TMO_CHK_EN      0
#define LQ_CHK_MSG_EN           0
```

5、选择 Project->Option for Target ' Target1 '，在 target 项目栏中的 operating 中选择 RTX51—Tiny。

6、编译这个工程，在调试状态下打开 P0、P1、P2 观察。

二、解释上面的例子：

1、lq51 的启动部分参照了 RTX51—Tiny，所以在 keil C51 中的关键字 _task_ 在 lq51 中依然有效，并且 targe 中的 operating 不能是 none，RTX51—Tiny 或 RTX51--Full 均可。

2、lq51 是基于优先级调度的，0 号任务具有最高优先级。lqStart()是系统初始化代码，说以 lqStart()必须在 0 号任务中最先被调用。

3、任务必须是死循环，否则系统出错。所以以上三个任务均有 while(1)。

4、以上任务的 lqDelay(100)是当前任务延迟 100 个系统时钟。lq51 定义一次 0 号定时器中断为一个系统时钟，在 lq51.h 中：

```
#define   lqTimerTick    10000
```

定义了 lqTimerTick 个机器周期产生一次 0 号定时器中断。

三、代码解读

先来个测验：在 C 语言中以下三个有啥不一样（是标准 C 不是 C51）？

```
1、  const int * ptr;
2、  int const * ptr;
3、  int * const ptr;
```

1、打开 lq51_a.asm 定位到 284 行：

```
PUBLIC  ?RTX_TASKENTRY
?RTX?TASKENT?S  SEGMENT CODE
        RSEG    ?RTX?TASKENT?S
?RTX_TASKENTRY:    DS 2
```

这几行代码是从 RTX—51 Tiny 的源代码 Conf_tny.A51 中原封不动的抄下来的，因为它跟关键字 _task_ 有关，如果要使用这个关键建立任务，就必须的加上这几行代码。这段代码这 ROM 中建立一个数组，这个数组存储每个任务函数的入口地址。

2、全局变量。打开 lq51_c.c。

unsigned char data lqTaskStack[lqMaxID+1]：这个数组用来存储每个任务堆栈的栈顶位置。lq51 的堆栈系统仿照 RTX51 的堆栈系统：
http://www.keil.com/support/man/docs/tr51/tr51_stackmgmt.htm

unsigned char data lqSPtemp：记录 ID 号=当前任务 ID+1 的堆栈底部，也就是当前任务堆栈所能生长的最高位置。

unsigned char data lqTaskTimer[lqMaxID]：任务的时钟，用于当前任务延迟(lqDelay)或在一定的时间限内等待某系统事件(lqWaitFlg,lqWaitSem,lqWaitMsg),

在 0 号定时器的中断服务子程序中，对每个不为 0 的任务时钟做减 1 操作，当任务时钟为 0 时，延时结束，任务重新处于就绪状态。

unsigned char data lqTaskState[lqMaxID]：任务状态表。用字节上的位表示任务的状态，每个位表示的信息如下：

第 7 位：K_MSG，如果该位置位，表示当前任务在等待邮箱事件。

第 6 位：K_SEM，如果该位置位，表示当前任务在等待信号量事件。

第 5 位：K_FLG，如果该位置位，表示当前任务在等待标志量。

第 4 位：K_TMO，如果该位置位，表示当前任务在延迟等待。这一位可以与 K_MSG、K_SEM、K_FLG 中的任何一个同时置位，表示在规定的时间内等待某个事件，如果在规定的时间内这个事件未发生，那个这个任务依然被激活，不过 K_TMO 标志位依然置位。

第 0~3 位：用于记录任务等待事件的索引号。因此在 lq51 系统中每个事件最多只能有 16 个。

例子：如果某个状态如下(二进制) 10010010，表示这个任务要在规定的时间内等待 2 号邮箱事件。

unsigned char data lqRdyTbl：任务就绪表，0 号位置位表示 0 号任务处于就绪状态，依次类推。

unsigned char data lqSwitchType：任务切换类型表。某位置位表示相应的任务是通过中断被切换出去的，否则是因为有更高优先级的任务就绪被切换出去的。通过中断切换任务是存在压栈操作，保护寄存器；如果是因为更高优先级的任务已经就绪而被切换出去，那是不存在寄存器压栈操作的。

unsigned char data lqIntNum：中断号。这个数据的高三位(第 7~5 位)记录中断嵌套数，如果这三位为 000 那么没有中断；低 5 位(第 4~0 位)记录中断类型号，通过中断类型号查找 lqISREnter 数值，得到中断服务子程序的入口地址。

unsigned char data lqCrt：当前正在运行的任务。

unsigned char data lqFlgData[lqFlgMax]：标志量数据，最高位表示当前标志量是否有效，其余 7 位以位形式记录这个标志量的任务。标志量与信号量类似，只是信号量可以累加，而标志量不能累加，只是二值制的信号量；当信号量有效时激活等待这个信号量的最高优先级的任务，而当标志量有效时激活**所有**在等待这个标志量的任务。标志量的初值在初始化时给定，可以是 0x80 或 0x00；

unsigned char data lqSemData[lqSemMax*2]：信号量数据结构。每个信号量拥有两个字节，第一个字节位信号量值，这个值在初始化时给定在 0~255 之间；第二个字节以位的形式记录等待这个信号量的任务，这个值最初必须是 0。当信号量事件有效时，等待信号量的最高优先级任务就绪。如果没有人在等待这个信号量，那么当前信号值加 1。注意：系统不会提示信号量溢出。

unsigned char data lqMsgData[lqMsgMax*2]：消息邮箱数据结构。每个邮箱包括两个字节，第一个字节表示该邮箱的消息值，这个值在初始化时给定，可以是 0~255 之间；第二个字节的最高位表示该邮箱是否有消息，如果最高位置位，则这个邮箱有消息，否则没消息。第二个字节的其他 7 位以位的形式记录等待

这个邮箱的事件的任务，第二个字节的初值只能是 0x80 或 0x00。如果邮箱中有消息，那么再向这个邮箱发送消息，则发送失败，邮箱内的消息不会覆盖。当邮箱事件有效时，把邮箱中的消息返回给在等待这个消息的优先级最高的任务。

3、入口函数 main，定位到 lq51_a.asm 的 293 行。这个函数初始化堆栈，把?RTX?TASKENT?S 数组内的任务函数入口地址填入堆栈。假设 RAM 区的最高地址是 0xFF，进入 main 时 SP 寄存器的值为 0x40，一共有 4 个用户任务，再加一个空闲任务，那么运行 main 函数之后，的堆栈如下图所示：

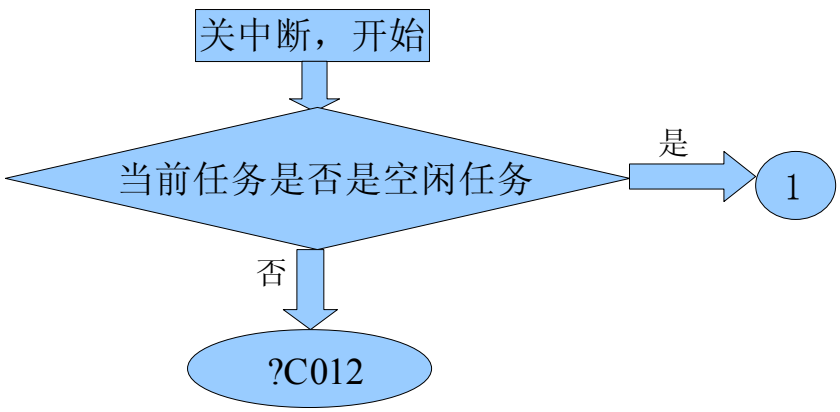
堆栈示意图

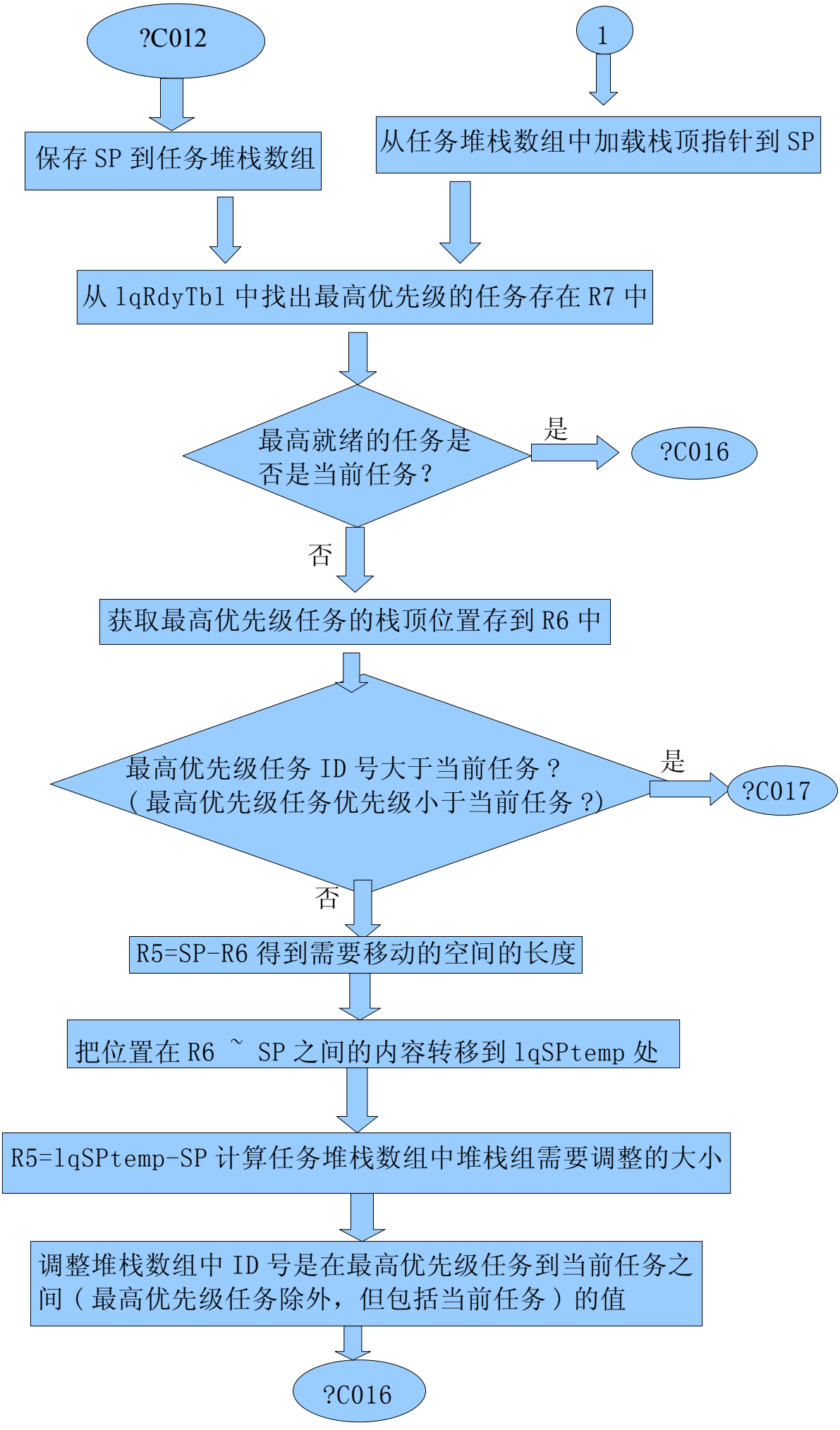
ROM 地址	地址内存储的值
0XFF	空闲任务入口地址高 8 位
0XFE	空闲任务入口地址低 8 位
0XFD	3 号任务入口地址高 8 位
0XFC	3 号任务入口地址低 8 位
0XFB	2 号任务入口地址高 8 位
0XFA	2 号任务入口地址低 8 位
0XF9	1 号任务入口地址高 8 位
0XF8	1 号任务入口地址低 8 位
0X43 ~ 0xF7	0 号任务堆栈的最多空间， lqSPtemp=0XF7
0X42	0 号任务入口地址高 8 位
0X41	0 号任务入口地址低 8 位

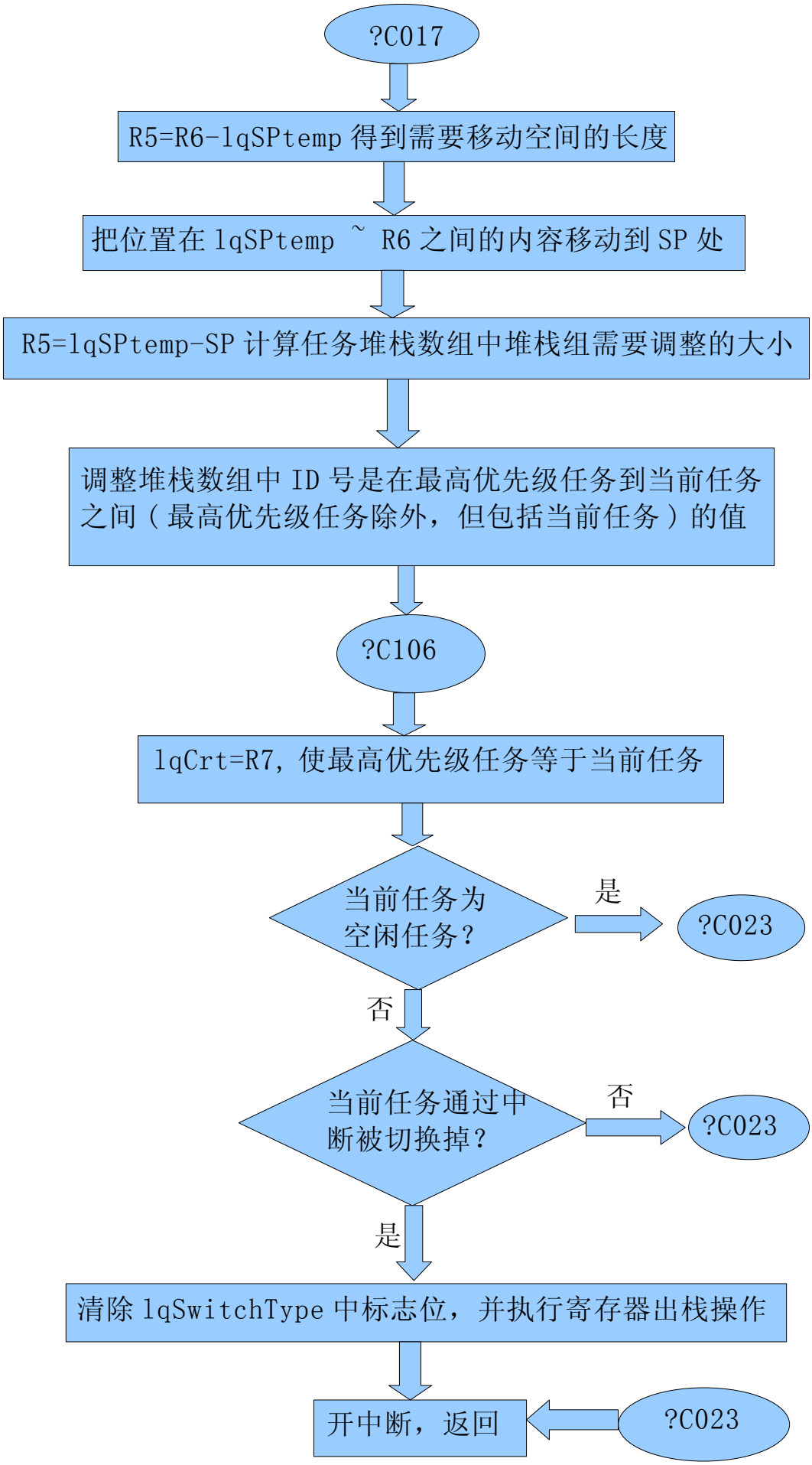
此时 SP=0X42，同时 main 函数初始化 0 号定时器并开中断，最后执行 RET 指令，把 0 号任务的入口地址出栈到 PC 寄存器，执行 0 号任务。

4、lqSche()任务切换函数(lq51_a.asm,p147)。这个函数按下面这个结构 http://www.keil.com/support/man/docs/tr51/tr51_stackmgmt.htm 调整堆栈区的结构，然后运行就绪表中优先级最高的任务。

lqSche()函数流程图




?C012
1
保存 SP 到任务堆栈数组
从任务堆栈数组中加载栈顶指针到 SP
从 lqRdyTbl 中找出最高优先级的任务存在 R7 中
最高就绪的任务是否是当前任务？
是
?C016
否
获取最高优先级任务的栈顶位置存到 R6 中
最高优先级任务 ID 号大于当前任务？（最高优先级任务优先级小于当前任务？）
是
?C017
否
R5=SP-R6 得到需要移动的空间的长度
把位置在 R6 ~ SP 之间的内容转移到 lqSPtemp 处
R5=lqSPtemp-SP 计算任务堆栈数组中堆栈组需要调整的大小
调整堆栈数组中 ID 号是在最高优先级任务到当前任务之间（最高优先级任务除外，但包括当前任务）的值
?C016


?C017
R5=R6-lqSPtemp 得到需要移动空间的长度
把位置在 lqSPtemp ~ R6 之间的内容移动到 SP 处
R5=lqSPtemp-SP 计算任务堆栈数组中堆栈组需要调整的大小
调整堆栈数组中 ID 号是在最高优先级任务到当前任务之间（最高优先级任务除外，但包括当前任务）的值
?C106
lqCrt=R7, 使最高优先级任务等于当前任务
当前任务为空闲任务?
是
?C023
否
当前任务通过中断被切换掉?
否
?C023
是
清除 lqSwitchType 中标志位，并执行寄存器出栈操作
开中断，返回
?C023

流程图中椭圆内的标号与源程序中的标号完全相同。

5、中断处理。见例子/lq51_demo/exp3

打开 lq51_a.asm 定位到 50 行，在这里我们设置了 0 号中断到 5 号中断的中断服务子程序。当系统发生中断时，PC 指针指到相应的中断服务子程序。这里的每个中断服务子程序都是相似的，进入中断服务子程序后，先关中断，然后把当前中断号存到 lqIntNum 的低 5 位，最后跳转到 lqInterruptISR。

lqInterruptISR：定位到 lq51_a.asm 文件的 361 行。

362 行 ~ 364 行：把当前寄存器压入堆栈，保护寄存器。

375 行 ~ 378 行：设置 lqSwitchType 中相应的位，表示当前任务是通过中断被切换掉，到恢复当前任务时，需要执行出栈操作，恢复寄存器。这个出栈操作在 lqSche()中。

379 行 ~ 382 行：把 lqISRNest 子程序的入口地址压入堆栈。这样可以确保从用户定义的中断服务子程序返回后执行 lqISRNest 这个子程序。

383 行：把 lqISREnter 数组的首地址存入 DPTR 中。lqISREnter 数值在 lq51.h 中定义：unsigned int code lqISREnter[]，#ifdef _IN_LQ51_C_可以保证这个数组只在 lq51_c.c 这个文件中产生实例。这个数组存储用户定义的中断服务子程序的入口地址。用户的中断服务子程序必须写成这种形式：

```
void 函数名(void) using 0
{
    函数体;
}
```

必须加上 using 0。然后在 lq51.h 的 extern void ISR_Timer0(void)后面加上这个函数的声明:

```
extern void 函数名(void);
```

最后在 lq51.h 内的 lqISREnter 数组内相应的位置上填入这个函数的函数名(函数名就是函数的入口地址)，如果这个中断服务子程序的中断号超过了 lqISREnter 中已存在的中断号，那么前面不足之处必须补 0，同时在 lq51_a.asm 文件的 93 行后加入这些：

```
CSEG    AT      中断入口地址
CLR     EA
ORL     lqIntNum,中断号
JMP     lqInterruptISR
```

384 行 ~ 385 行：从 lqIntNum 中获取当前中断的中断号。lqIntNum 的低 5 位用于存储中断号，因此 lq51 系统最多支持 32 个中断。

386 行 ~ 394 行：根据中断号，从 lqISREnter 数组中获取中断服务子程序的入口地址，并把它压入堆栈。这样可以保证在 lqInterruptISR 执行完毕返回后执行用户定义的中断服务子程序。

395 行 ~ 398 行：获取 lqIntNum 的高 3 位，同时清零低 5 位，高 3 位加 1，再把结果存会 lqIntNum 中。lqIntNum 的高 3 位存储中断嵌套数，当高

3 位为 0 时，没有中断在执行。因此 lq51 系统最多能有 7 重中断嵌套。

399 行 ～400 行：开中断，返回。返回之后执行用户定义的中断服务子程序。当从用户定义的中断服务子程序返回后执行 lqISRNest;

lqISRNest：定位到 lq51_a.asm 文件 402 行

403 行 ～406 行：关中断，并执行中断返回指令(RTEI)。这条指令必须被执行，否则单片机会认为以后的所以程序都是这个中断服务程序的一部分那么优先级小于等于当前中断的所有中断都不会被执行。

407 行 ～429 行：lqIntNum 的高 3 位减 1。如果高 3 位为 0 那么执行 lqSche()切换任务，因为用户的中断服务子程序有可能使更高优先级的任务处于就绪状态。如果高 3 位不为 0，那么表示当前中断是从低优先级的中断中抢过 CPU 的控制权，所以执行出栈操作，开中断并返回低优先级的中断。

6、消息事件 lqWaitMsg、lqSendMsg、lqSendMsgISR 、lqIsMsgEmpty、lqIsTaskTmo

lqWaitMsg :打开 lq51_c.c 定位 lqWaitMsg 函数。我逐条语句解析。

EA=0;

进入函数后首先关中断。如果我们没关中断，假设当程序执行到 lqRdyTbl &= ~lqMap[lqCrt]时，当前任务已经从任务就绪表中删除，但当前任务还没有加到消息邮箱的等待列表中，下一条语句才完成这个任务。如果这个时候发生中断，而在中断服务子程序结束后是有任务切换函数的。因为这个被中断了的任务已经从就绪表中删除了，所以这个任务不可能在任务切换函数后被运行，同时邮箱的任务等待列表中也没有这个任务，那么就不可能通过邮箱激活这个任务，那么这个任务就永远不可能运行。

当然你也可以说先把这个任务加到邮箱的任务等待列表，即先执行 lqMsgData[index+1] |= lqMap[lqCrt]，然后在将当前任务从任务就绪列表中删除。现在我们假设函数已经将任务添加到消息邮箱的任务等待列表，但还没有从任务就绪表中删除当前任务。这时发生中断，并且中断服务子程序向这个邮箱发送消息，因为任务已经加到邮箱的任务等待列表，假设这个任务在等待列表中拥有最高优先级，那么这个任务获得这个消息。当从中断服务子程序中返回后，接着执行当前任务。当前任务在 lqWaitMsg 中，而且这个语句已经过去了，

```
if(lqMsgData[index+1] & 0x80){      /*当前邮箱有邮件*/
        lqMsgData[index+1] &= 0x7F;
        EA=1;
        return lqMsgData[index];
}
```

当前任务任务邮箱没有消息，而实际上中断服务子程序已经向邮箱发送消息了，所以当前任务一定会错过这条消息。

或许你还可以说那么在进入函数之后关中断，在执行

lqRdyTbl &= ~lqMap[lqCrt];

```
lqMsgData[index+1] |= lqMap[lqCrt];
```

语句之后开中断！这个方法在这一点上是可行的！但你没办法保证后面的程序中不会出现类型上面的情况。当然你也可以去设想各种情况，然后一一处理，但这很累。有一个比较“懒的”的办法：如果你的系统服务涉及到系统定义的全局变量，那么进入这个系统服务函数后首先关中断，离开前再开中断。这样做有一个弊端就是会延长中断响应时间。如果你的系统可以忍受这个弊端，用这种方法，否则就只有斤斤计较，精打细算了。

在我的 lq51 系统中我愿意接受这样的弊端。

```
lqTaskState[lqCrt] &= 0xF0;
lqTaskState[lqCrt] |= index;
index *= 2;
```

lqTaskState[lqCrt] 里面存储当前任务的状态，高 4 位与系统服务有关，从低到高依次是 K_TMO、K_FLG、K_SEM、K_MSG，关于这些的详细定义请参考前面全局变量那一栏。低四位用于存储当前邮箱的索引(index)。为什么要存储这个 index 呢？

在 keil 中局部变量不是在堆栈中分配的，而是为每个函数的局部变量分配固定的地址空间，相当与是全局变量。如果两个函数没有调用关系，那么这两个函数的局部变量可以这同一空间中。比如：

```
void fun1( )
{
    char i;
    函数体;
}
```

```
void fun2( )
{
    char j;
    函数体;
}
```

因为 fun1 与 fun2 没有相互调用，所以可以把 fun1 的局部变量 i，fun2 的局部变量 j 分配到同一个地址空间。也就说它们相当于是一个变量，因为它们不可能同时出现，所以可以这样做。

这样做涉及到一个问题，就是这个函数不能递归调用。说的专业一点就是这个函数是不可重入的[1*]。

现在回到我们的问题，如果我们不保存这个 index，那么到程序执行到 lqSche()之后，当前任务就被切换出去。如果新的任务也执行 lqWaitMsg 那么这个老任务的 index 就会被新任务的 index 覆盖掉，因为它们在同一段地址空间。那么当老任务被切换回来之后，此时的 index 已经变了。所以要保存这个 index。这样就限定了 lq51 系统中，最多只能提供 16 个邮箱服务。

```
if(lqMsgData[index+1] & 0x80){      /*当前邮箱有邮件*/
    lqMsgData[index+1] &= 0x7F;
    EA=1;
    return lqMsgData[index];
}
```

如果当前邮箱有邮件，那么直接返回邮箱中的邮件。

```
lqRdyTbl &= ~lqMap[lqCrt];
lqMsgData[index+1] |= lqMap[lqCrt];
lqTaskState[lqCrt] |= K_MSG;
if(tmo){
    lqTaskState[lqCrt] |= K_TMO;
    lqTaskTimer[lqCrt] = tmo;
}
EA=1;
lqSche();
```

当前邮箱中没有邮件，那么将任务从就绪表中删除，同时将任务添加到邮箱的任务等待列表中再设置当前任务相应的状态位。最后任务切换，执行现行就绪表中最高优先级的任务。

```
EA=0;
index = lqTaskState[lqCrt] & 0x0F;
index *= 2;
if(lqTaskState[lqCrt] & K_MSG){
    lqTaskState[lqCrt] &= ~K_MSG;
    lqTaskState[lqCrt] |= K_TMO;
    lqMsgData[index] = MSG_TMO;
    lqMsgData[index+1] &= ~lqMap[lqCrt];
}
lqMsgData[index+1] &= 0x7F;
EA=1;
return lqMsgData[index];
```

当任务重新被激活后，先加载任务邮箱的索引。任务被激活，只能是两种情况：一、等待的消息已经发到了，二、任务超时返回。如果是等待的消息已经到了，那么任务状态表上的 K_MSG 位一定被清除，否则就是任务等待超时返回。如果是等待超时返回，那么清除状态表上的 K_MSG 标记同时设置超时标记位，设置邮箱中的消息为 MSG_TMO 并把任务从邮箱的等待列表中删除。清除邮箱中有消息的标志位，然后开中断返回。

lqSendMsgISR：定位到 lq51_c.c 中的 lqSendMsgISR 函数。如果是中断服务子程序要发送消息，则调用这个函数。在中断中不能执行任务切换，因为中断是突发事件，必须被优先执行，否则要中断干啥！发送消息后有可能会使更高优先级的任务处于就绪状态。所以这个函数返回值表示是否有更高优先级的任务处于就绪状态。

这个函数首先检查消息邮箱中是否已存在消息，如果有则直接返

回。然后把消息存到消息邮箱中，再检查是否有任务在等待这个消息，如果有则找出最高优先级的任务，再对比当前任务，决定返回值是多少。

lqSendMsg：如果任务要发送消息，这调用这个函数。如果发送消息后使更高优先级的任务处于就绪状态，则执行任务切换。

7、空闲任务。空闲任务在 lq51_a.asm 的 353 行定义：

```
lqIdleTask:
        ORL   PCON,#01H
        SJMP  lqIdleTask
```

当用户任务均被挂起的时候执行空闲任务。

代码解读到此结束，对于其他系统服务（信号量及标志量），对比邮箱事件，不难明白。剩下的多简单了。

四、使用说明

1、将 lq51_a.asm、lq51_c.c、lq51.h 复制到工程的目录下，确保这三个文件在同一个文件夹内。将 lq51_a.asm、lq51_c.c 添加到你的工程中，将 lq51.h 包含到你的文件中。

2、配置 lq51.h

lqTimerTick ：定义 0 号定时器中断发生所需的机器周期数。0 号定时器作为系统时钟。

lqRamTop ：堆栈栈顶位置。8051 系类是 0X7F，8052 系类是 0XFF。

lqMaxID ：最大的 ID 号，这个 ID 号是为空闲任务准备。lq51 是基于优先级调度的，0 号任务具有最高优先级。当前系统只支持 8 个任务，包括空闲任务。实际系统只支持 7 个用户任务，因此这个值最大为 7。

lqFlgMax ：标志事件个数，最多为 16。

lqSemMax ：信号量事件个数，最多为 16。

lqMsgMax ：消息邮箱事件个数，最多为 16。

LQ_DELAY_EN ：任务延迟功能使能，1 为使用该功能，0 为不使用该功能

LQ_FLG_EN ：标志量事件使能

LQ_SEM_EN ：信号量事件使能

LQ_MSG_EN：消息邮箱事件使能

LQ_TASK_TMO_CHK_EN ：任务超时检查功能使能

LQ_CHK_MSG_EN ：检查邮箱是否为空

MSG_TMO ：设置消息邮箱事件超时返回值

3、任务

任务函数必须写成死循环，按下面这样：

```
void  任务函数名(void)  _task_  任务优先级
```

```
{
    变量定义，任务初始化；
    while(1){
        具体任务;
    }
}
```

对于 0 号任务，必须在进入死循环之前执行 lqStart()函数。

4、中断

用户的中断服务子程序必须写成这样：

```
void  函数名( void )  using 0
{
    函数体;
}
```

using 0 这句必须得加上。

再在 lq51.h 中的 ISR_Timer0 附近添加中断服务子程序的函数原型。

在 lq51.h 的 lqISREnter 数组中相应中断号上添加中断服务子程序的函数名，如果当前中断号超过已存在的，不足之处必须补 0，同时在 lq51_a.asm 中添加如下代码：

```
CSEG    AT      中断入口地址
CLR     EA
ORL     lqIntNum,中断号
JMP     lqInterruptISR
```

注意：在中断服务子程序中不能调用 lqWaitFlg、lqWaitSem、lqWaitMsg。如果中断服务子程序要发送相应的系统事件，则调用 lqSendFlgISR、lqSendSemISR、lqSendMsgISR。

5、系统服务

lq51 系统提供 3 中系统服务：标志量、信号量、消息邮箱，每种系统服务最多提供 16 个。

标志量：在 lq51 系统中用一个 0 到(lqFlgMax-1)的值来索引标志量。使用标志量前必须先初始化标志量数组 lqFlgData。标志量相当于一个二值信号量。不同的是，当标志量有效时，所有在等待这个标志量的任务均被激活。有标志量有关的函数有 lqWaitFlg、lqSendFlg、lqSendFlgISR。

信号量：在 lq51 系统中用一个 0 到(lqSemMax-1)的值来索引信号量。使用信号量前必须先初始化信号量数组 lqSemData。信号量用一个 0~255 之间的值存储当前信号量。当任务请求信号量时(lqWaitSem)，如果信号的值不为 0 则直接返回，同时信号量的值减 1。如果信号量的值为 0，则挂起当前任务，运行其他任务。发送信号量时(lqSendSemISR、lqSendSem)，如果有任务在等待这个信号量，则激活最高优先级的任务，否则信号量的值加一。

消息邮箱：在 lq51 系统中用一个 0 到(lqMsgMax-1)的值来索引消息邮箱。使用消息邮箱前必须先初始化消息邮箱数组 lqMsgData。lq51 系统的

邮箱发送的是一个 0~255 之间的整数值，而不是一个指针。当邮箱中有消息时，在向邮箱中发送消息(lqSendMsgISR、lqSendMsg)，则发送失败，原有的消息不会被覆盖。

6、选择操作系统

选择Project->Option for Target ' Target1 '，在target项目栏中的operating中选择 RTX51—Tiny或RTX51---Full。□ □ 必□ 得□ 。□

7、函数功能说明

void lqStart(void) ：初始化函数，这个函数必须在 0 号任务未进入死循环之前被调用。

void lqSche(void) ：任务切换函数，该函数运行就绪表中的最高优先级任务。用户不应该直接调用这个函数。

void lqDelay(unsigned char tmo) ：任务延迟。

unsigned char lqWaitFlg(unsigned char index,unsigned char tmo) ：等待标志量。

char lqSendFlgISR(unsigned char index)：中断服务子程序发送标志量。

void lqSendFlg(unsigned char index)：任务发送标志量。

unsigned char lqWaitSem(unsigned char index,unsigned char tmo)：等待信号量。

char lqSendSemISR(unsigned char index)：中断服务子程序发送信号量。

void lqSendSem(unsigned char index)：任务发送信号量。

unsigned char lqWaitMsg(unsigned char index,unsigned char tmo)：等待消息邮箱。

char lqIsMsgEmpty(unsigned char index)：检查消息邮箱是否为空。

char lqSendMsgISR(unsigned char index,unsigned char Msg)：中断服务子程序向消息邮箱中发送消息。

void lqSendMsg(unsigned char index,unsigned char Msg)：任务向消息邮箱中发送消息。

char lqIsTaskTmo()：检查当前任务是否是超时运行。

五、后记

写这个东西之前曾看过 ucosii（看的是 ucosii 的书，没仔细研究过 ucosii 的源代码）。当时就想自己写一个 OS 试试。最初是想用纯 C 语言编写，这样可以不用修改的任意移植（当时有点嚣张，不好意思）。当时在网上找到这个网页： http://en.wikipedia.org/wiki/Longjmp

我认为用这个网页上介绍的方法使用纯 C 编写是有可能的。因为他用 C 语言实现了我认为必须使用汇编语言才能实现的现场保护及任务切换。他使用了标准 C 里的 setjmp 及 longjmp 实现。当时没考虑效率问题。于是我就按照网页上介绍的方法自己写了一个 OS，我还取了个名字叫做 LQOS(Learn Quickly)。当我

把我的 LQOS 放到 51 单片机上时，我发现他不能运行。查了好久才模模糊糊知道 C51 与标准 C 不一样，而且 keil 中的局部变量不在堆栈中分配，所有函数是不可重入的。后来 LQOS 就放下了。

我认为 lq51 系统有两个特点：一、使用 _task_ 关键字；二、关于中断的处理方法。

一、使用 _task_ 关键字。因为 keil 自己的操作系统 RTX51 也是使用 _task_ 关键字的，所以我认为在 keil 进行代码覆盖分析时，被不同任务调用的函数即使没有相互调用，他们的局部变量也不会相互覆盖。比如：

```
void  task1()  _task_   1
{
    变量定义，初始化;
    while(1){
        fun1();
        其他任务;
    }
}
void fun1()
{
    char i;
    函数体;
}
```

```
void  task2()  _task_   2
{
    变量定义，初始化;
    while(1){
        fun2();
        其他任务;
    }
}
void fun2()
{
    char j;
    函数体;
}
```

虽然 fun1()与 fun2()没有相互调用，但是他们的局部变量(i,j)不会相互覆盖，因为他们是被不同的任务所调用的。如果我们假设 i,j 是可以相互覆盖的，也就是说他们实际上是同一个变量。那么当 task1 在 fun1 这个函数体内发送任务切换，切换到 task2，task2 调用 fun2。那么 fun2 的局部变量就会修改 fun1 的局部变量，因为他们在同一个地址空间内。那么 task2 返回 task1 时，task1 的信息已经被破坏掉了。所以我认为 被不同任务调用的函数即使没有相互调用，他们的局部变量也不会相互覆盖。

以上这一点纯属假设，因为我没找到“证据”。但我任务这种假设是可信的。这样使用 _task_关键字就为我们免去了在 SmallRTOS 中存在的一些麻烦。我们不需要手动的去禁止局部变量的相互覆盖。

二、中断。关于中断的这种处理方法，我前前后后想了好几天。进入中断服务子程序之前必须要保存寄存器变量，而且从中断服务子程序退出之后必须要有任务切换，但在中断服务子程序的过程中不能有任务切换。原先是想借用 ucosii 的方法的，进入中断后先调用一个函数，标记系统已经进入中断，在中断函数的最后再调用一个函数表示中断已经结束，同时进行任务切换。

但是如果原模原样的照搬会发现，在我们的 lq51 系统当中，在中断服务子程序最后调用一个函数表示中断结束进行任务切换时，但对 CPU 来说中断还未结束，因为我们最后调用的那个函数也是中断的一部分。而且进入中断前就保存的局部变量也没出栈。

在写 lq51 的时候曾认真得阅读过 RTX51—Tiny 的源代码，因此 lq51 的 main 函数与 RTX51--Tiny 在最开始的那一部分有点相似。但要使用 keil 的 _task_关键字，好像必须得这么做似的。关于嵌入式实时操作系统的许多概念是从 ucosii 的作者写的那本书上学来的。另外从 SmallRTOS 上也学了点东西。lq51 消息邮箱的功能与 SmallRTOS 消息邮箱的功能就很相似吗！

六、附录

1*：关于函数课可重入性的详细说明，请参考 ucosii

七、后记：

本人是个在校大学生，本专业是自动化。我没做过实际的项目，自我感觉这个东西是整合了 RTX51—Tiny,SmallRTOS，ucosII 的一个东东。

你对这个东西感兴趣的话，你自己想怎么玩就怎么玩。因为 lq51 系统中用到了_task_关键字，不知道这会不会触犯 keil 的专利。

lq51_c.c 文件中 lqFlgData[lqFlgMax]、lqSemData[lqSemMax*2]、lqMsgData[lqMsgMax*2] 这三个数据的类型不一定是 data 也可以是 xdata。

这个系统的名字----lq51，源于人名，一个女生的名字。

最后留一下我的邮箱： adream307@163.com