**实验报告**

专业：电气工程及其自动化

姓名： 严旭铧

学号：\_3220101731

地点：紫金港东三406

课程名称： 微机原理与应用实验 指导老师：\_\_ 胡斯登\_\_\_ \_成绩：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

实验名称：\_\_\_\_\_\_\_\_倒计时定时器\_\_\_\_\_实验类型：\_微机实验\_\_\_\_\_

# 期中实验 倒计时定时器

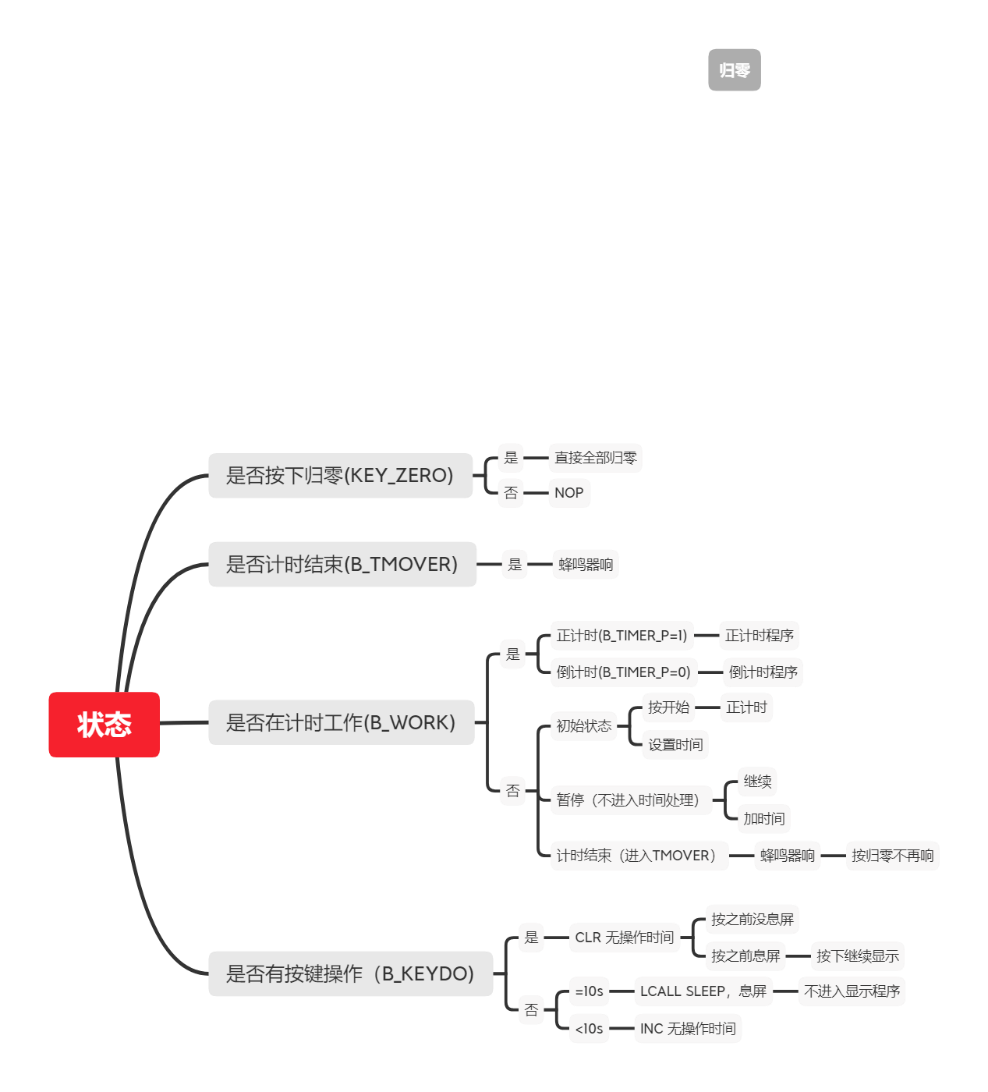
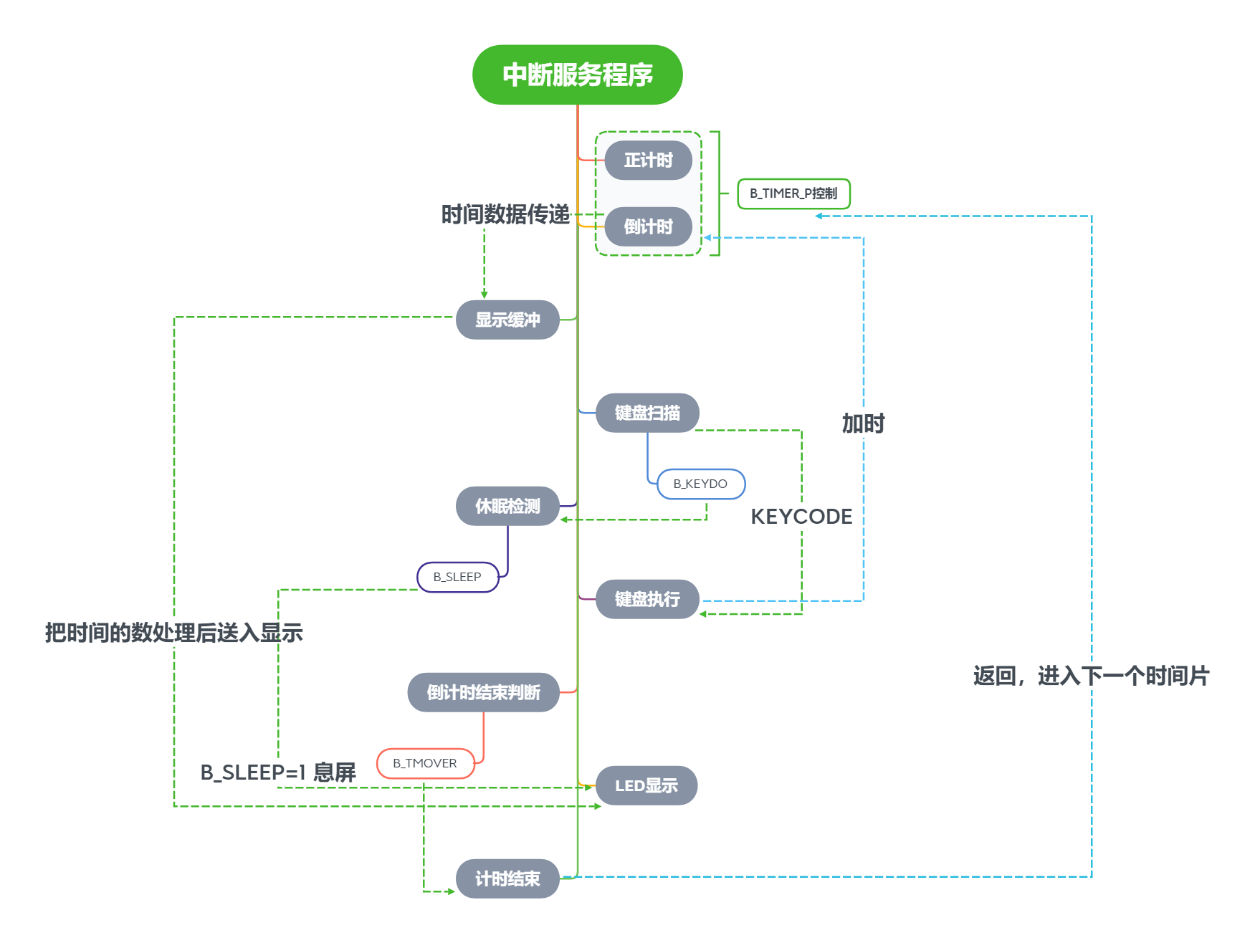
1. **实验目的**
   1. 加深对51单片机定时器及中断系统原理和应用的理解。
   2. 锻炼逆向工程能力。
   3. 综合考察汇编语言和80C51单片机的应用。
2. **实验要求**

复现倒计时定时器的功能。根据自己理解，具体要求及分析如下：

* 1. 有以下按键：
     1. “开始/暂停”：按下开始计时，再按暂停
     2. “归零”：无论是否处于计时状态，按下就归零，时间显示00:00
     3. “10秒”“1分”“10分”：按一次加相应的时间
  2. 开机后，初始为00:00，称为归零状态。
  3. 归零时按开始正计时
  4. 仅暂停/归零时候可以改时间
  5. 10秒无按键按下自动息屏，息屏后按任意按键恢复显示，息屏时其他功能均正常
  6. 计时结束蜂鸣器响，按归零停止蜂鸣并恢复初始状态

1. 设计思路
   1. 使用状态机思想进行编程，引入多个状态变量使得编程可以实现模块化，方便更改。
   2. 时间的处理：将值的计算和显示分开，让时间单纯作为数计算，再将数映射到数码管上。
   3. 分模块：
      1. 初始化模块
         1. 逻辑状态变量（B\_x）初始化：所有状态用B\_开头，表示bool（位）变量，用0/1判断状态。共设置以下6个逻辑状态变量：



* + - 1. 定时器等其他变量初始化
    1. 时间处理模块
       1. 正计时及时间处理（TIMER\_P）
       2. 倒计时及时间处理（TIMER\_N）
          1. 倒计时处理
          2. 倒计时结束（TMOVER）
          3. 倒计时结束状态判断（TIMER\_N\_TF）（全局判断）
       3. 息屏时间计时（TIMER\_SLEEP）
    2. LED显示模块
       1. 缓冲模块，存放时间值
       2. 显示模块
    3. 键盘模块
       1. 键盘扫描（KEY\_SCAN）
       2. 键值映射及处理（DO\_KEY）
  1. 思维导图呈现：

状态变量的一个基本使用方式：在各个子程序前用状态变量的值来判断是否进入该子程序（中断服务程序），如果不进入，则跳转到该子程序的RET处（这里的RET可以另起一个标号，使结构更清晰，方便跳转）。

1. **各模块与功能对照分析**
   1. 初始化模块
      1. 各种EQU和BIT，将意义不那么直白明确的地址数用伪指令“命名”。本次实验我使用的命名规则是：B\_\*表示逻辑状态变量，就是可以采用位寻址的部分，从00H开始，值为0/1；其他功能则以大模块名加”\_”表示。这样的命名方式有助于结构化和模块化的编程。
      2. 定时器初始化：选用定时器0方式1，时间片长度选取为2ms。另外，将正计时、倒计时、和无操作时间计时的2ms及100ms计数器初始化置零。中断程序如下：



其中，定时器2ms的参数设置：

2ms = 2000us, 65536-2000 = 63536 = F830H，因此低8位取30H，高8位取F8H。

* + 1. 主程序（LOOP）：

LOOP:;主

        LCALL   TIMER\_P          ;正计时

        LCALL   TIMER\_N          ;倒计时

        LCALL   DISP\_TIMER       ;时钟放入显示缓冲

        LCALL   KEY\_SCAN         ;键盘扫描

        LCALL   TIMER\_SLEEP     ;休眠时间检测

        LCALL   DO\_KEY           ;根据键码值运行相应操作

        LCALL   TIMER\_N\_TF      ;倒计时结束判断

        LCALL   DISP\_LED         ;LED显示

        LCALL   TMOVER           ;计时结束及蜂鸣器

        JNB      B\_2ms,$

        CLR      B\_2ms

        LJMP    LOOP

需要注意的是，这里要讲究一个LCALL的顺序，并在子程序前设置判断语句。如果判断之后进不去，那么就RET返回，进行下一个LCALL。每个时间片（2ms）循环一次。由此，可以很多操作。这里的DISP\_TIMER，KEY\_SCAN，TIMER\_N\_TF都是全局运行的，相当于每时每刻都在检测当前的状态。

* 1. 时间处理模块
     1. 正计时（TIMER\_P）：默认开机和复位归零后，按下开始键进行正计时，再按暂停取消，再按开始/暂停继续。
        1. 控制用状态变量：B\_TIMER\_P（=1正计时，=0倒计时），该变量由“开始/暂停”分配的按键决定。初始化默认为1，复位时置1，设置时间后归0进行倒计时。
        2. 代码块分析：
           1. 利用状态变量判断是否进入正计时：

        JNB      B\_WORK,TIMER\_P\_RET;B\_WORK=1 -->

        JNB      B\_TIME\_P,TIMER\_P\_RET;正计时

这里，如果定时器正在计时，且需要进行正计时才能往下走进正计时程序；否则，返回主程序。

* + - * 1. 2ms 🡪 100ms 🡪 1s：

   INC     T\_2MS ;;2ms计数加1

        MOV     A,T\_2MS;放入A中方便实用SUBB 或者 CJNE

        CLR     C       ;用SUBB之前一定要先把C清零，否则影响判断

        SUBB    A,#50   ;也可以用CJNE

        ;IF T\_2MS<50

        JC      TIMER\_P\_RET;T\_2MS<50时，小减大C=1，跳转RET,计时继续滚动

        ;IF T\_2MS=50

        MOV     T\_2MS,A;T\_2MS=50时，减完=0，把0重新放回T\_2MS，计数器重置，也可以直接放入#0

;100ms计数加1

        INC     T\_100MS

        MOV     A,T\_100MS

        CLR     C

        SUBB    A,**#10**

        JC      TIMER\_P\_RET

        MOV     T\_100MS,A;与前面同理

;秒计数加1

        INC     SECOND

        MOV     A,SECOND

因为16位最多到65536us，并且前面最小时间片也就是2ms，因此，设定变量计数，2ms次数每次达到50就归零，同时100ms计数器+1，当100ms计数器到10，秒（计数器）（SECOND）+1。所有涉及进位的地方都可以用这种思路。

* + - * 1. 时间进位及非法输入控制  
           与上面同理，1分=60秒，当SECOND=60时，MINUTE+=1，**在当前时间片中，把SECOND清零**。这就是把时间的计算、处理和分析分开的好处，在同一个时间片里面，上一个子程序中已经把60的过程处理好了，输送给显示程序的就是0，这样显示就是连贯的59🡪00.

对于非法输入，其实也就是进制。我设定了分钟的上限为99，当MINUTE达到99时，再加1也就清零。代码与上一步同理，更改SUBB指令后高亮部分即可。

* + - * 1. 返回程序（TIMER\_P\_RET）

这一整段代码中出现了很多CJNE或者JC之类的条件转移指令。以CJNE为例，不满足相等就跳转，相等才往下走，那么不相等时如果要返回主程序循环就比较麻烦。单独加一个返回程序会使得返回变得很简单。

TIMER\_P\_RET:

        RET

* + 1. \*倒计时模块（TIMER\_N）：设置了时间之后按开始进入倒计时。
       1. B\_TIMER\_P=0且 B\_WORK = 1进入倒计时
       2. 代码块分析
          1. 先弄出1s来，和前面一样，不过为了做区分，将2ms计数器命名为T\_2MS\_N，其余同理

TIMER\_N\_MIN:

        DEC     MINUTE

        MOV     A,MINUTE

        MOV     SECOND,#59;没结束，秒59

**CJNE    A,#0,TIMER\_N\_RET**

**JB      B\_TMOVER,TMOVER**

        LJMP    TIMER\_N\_RET

TIMER\_N\_RET:

        RET

* + - * 1. \*倒计时主体：

TIMER\_N\_SEC:

        MOV     A,SECOND

        CLR     C

        SUBB    A,#1

        JC      TIMER\_N\_MIN;减到0跳转减分

        MOV     SECOND,A

        SJMP    TIMER\_N\_RET

左边是 秒计数-1，当秒减到0的时候跳转到右边 分计数-1的程序

难点：00🡪59 or 00🡪00?

TIMER\_N\_TF:

        JNB     B\_WORK,TIMER\_N\_TF\_RET

        JB      B\_TIME\_P,TIMER\_N\_TF\_RET

        MOV     A,MINUTE

        CJNE    A,#0,NOT\_OVER

        MOV     A,SECOND

        CJNE    A,#0,NOT\_OVER

        SETB    B\_TMOVER

        CLR     B\_WORK

TIMER\_N\_TF\_RET:

        RET

NOT\_OVER:

        SJMP    TIMER\_N\_TF\_RET

如果计时没有结束，那么当秒到0再-1之后会变成59；

如果秒到0之后计时结束了，那么就达到计时结束状态，时间为0，蜂鸣器响。

这里我的做法是，只要跳转到减分，先把秒的数值置59，因为显示程序在最后，可以倒计时子程序和显示子程序之间继续对秒的数值进行修改。加了下划线的两行非常重要。CJNE指令让分为0的时候再去判断有没有停止，JB指令则充分体现了引入状态变量的优越性：不需要在这个子程序中去判断时间有没有倒计时结束，而是另起一个专门的判断子程序(TIMER\_N\_TF)返回状态变量B\_TMOVER, 若为1则跳转倒计时结束子程序（TMOVER）.判断的方法就是看分和秒是否全为零，并且先判断分（这里用到了数电中的思想），再判断秒，全为0就把计时结束标志位置1，并且计时运行标志位（B\_WORK）置零。

* + 1. 无操作时间（TIMER\_SLEEP）

JB      B\_SLEEP,TIMER\_S\_RET0

JB      B\_KEYDO,TIMER\_S\_RET0

* + - 1. B\_SLEEP=0 且 B\_KEYDO=0(没有键按下，这里双保险)。但是这里是返回了\_RET0而不是\_RET，下面会讲。
      2. 代码块分析：

TIMER\_S\_RET:

        RET

TIMER\_S\_RET0:

        MOV     T\_100MS\_SLEEP,#0

        MOV     T\_2MS\_SLEEP,#0

        SJMP    TIMER\_S\_RET

* + - * 1. 2ms🡪100ms🡪10s，原理相同，最后一步改成#100即可，并且最后到达10s改成SETB B\_SLEEP
        2. 返回程序：

这里采用了两步返回，得到的效果是，每次息屏/按键之后，这个子程序都能进该返回程序，并且让无操作时间计数归零。特别是按下按键之后。B\_KEYDO是用来表示按键是否按下的状态变量，一旦有按键按下，无操作时间必须归零，否则在第一次亮起之后10秒后，无论你有没有操作都会息屏，这是不符合要求的。

* 1. LED显示模块
     1. 时间缓存和处理：把SECOND和MINUTE中存放的时间数进行分离
        1. 全局，每次都会进入该子程序。
        2. 代码块分析：

      MOV     A,MINUTE

        MOV     B,#10

        DIV     AB

        MOV     @R0,A

        INC     R0

        MOV     @R0,B

        INC     R0               ;;DISPLAY MINUTE

以分钟处理为例，就是把个位和十位通过DIV AB指令进行分离，然后放到不同的地址。R0存放的是地址。

* + 1. LED显示

JB      B\_SLEEP,DISP\_LED\_NO

…………………………………………

DISP\_LED\_NO:

        MOV     A,#0FFH

        MOV     P2,A

        MOV     P0,A

* + - 1. 息屏实现：

这里开始时就判断是否应该处于息屏状态。应该息屏就跳转DISP\_LED\_NO程序，将IO口都放成0FFH（有些数码管用00H），就可以实现息屏，同时其他的功能不受影响。

* 1. 键盘模块
     1. 键盘扫描（KEY\_SCAN）
        1. 全局，每个时间片都要进入
        2. 代码块分析：这里采用了理论课上吴老师给的方案，通过检测上一次取到的键码与新取到键码的一致次数来去抖动。
           1. 行扫描和列扫描

;这是防抖动的程序，只有连续三次扫描得到

的是同一个按键按下时才把它写进键码

        CJNE    A,LASTCODE,GET\_NEW\_KEY

        ;采到的键码若等于上一个键码，把相同的次数与3比较

        MOV     A,N\_SAME

        CJNE    A,#5,GET\_NEXT;<5,继续获取

;=5,确定是新键码

        MOV     KEYCODE,LASTCODE

KEY\_EXIT:

        INC     N\_SAME

        RET

GET\_NEXT:

        JC      KEY\_EXIT

        RET

GET\_NEW\_KEY:

        MOV     LASTCODE,A

        MOV     N\_SAME,#0

        RET

KEY\_SCAN:

        MOV     P1,#0F0H

        NOP

        MOV     A,P1

        ANL     A,#0F0H;1111 0000 P1.4-P1,7，行扫描

        MOV     B,A;暂存保护

        MOV     P1,#0FH

        NOP

        MOV     A,P1

        ANL     A,#0FH;0000 1111 P1.0-1.3，列扫描

        ORL     A,B

        MOV     DPTR,#TAB\_KEY

        MOVC    A,@A+DPTR

* + - * 1. 去抖动，并返回键值（KEYCODE）
    1. \*键盘功能处理（DO\_KEY）
       1. 全局，每个时间片都要进入
       2. 代码块分析：
          1. 有效键检测：当有有效按键按下的时候，将B\_KEYDO置1，将B\_SLEEP和B\_TMOVER置0，并且只有有效按键按下之后，才会进入后续“按下了哪个键”的依次判断。此处会影响息屏。

DO\_KEY:

        MOV     A,KEYCODE

        CJNE    A,#0FFH,KEY\_1;A!=FF --> KEY\_1，这里键盘是低电平有效的，全为FF就是没有有效按键按下的状态

        CLR     B\_KEYDO

        LJMP    KEY\_RET

* + - * 1. 开始、暂停和继续

这里我设置的开始/暂停键位于键盘的0F位置，不是这个键的话就往下去判断。如果是按下了“开始”，那么用JNB判断状态，此时若是在计时状态（B\_WORK=1）那么跳转暂停（PAUSE），否则跳转继续（RESTART）。这里的暂停继续还是用改变B\_WORK值的方法做。它会影响上面的时间处理模块，在倒计时部分，B\_WORK控制了倒计时的暂停和继续。

KEY\_ZERO:

        SETB    B\_KEYDO

        MOV     A,KEYCODE

        CJNE    A,#0EH,KEY\_10S

        CLR     B\_TMOVER

        CLR     B\_WORK;叫停

        SETB    B\_TIME\_P

        MOV     SECOND,#0

        MOV     MINUTE,#0

        LJMP    KEY\_RET

KEY\_1:;开始

        CLR     B\_SLEEP

        SETB    B\_KEYDO

        CLR     B\_TMOVER

        MOV     A,KEYCODE

        CJNE    A,#0FH,KEY\_ZERO;A!=0F --> KEY\_1，否则去归零

        SETB    B\_KEYDO

        CLR     B\_SLEEP

        CLR     B\_TMOVER

        JNB     B\_WORK,RESTART;B\_WORK=0,重启计时；=1，暂停

        SJMP    PAUSE ;=1,暂停

PAUSE:  CLR     B\_WORK

        LJMP    KEY\_RET

RESTART:SETB    B\_WORK

        LJMP    KEY\_RET

* + - * 1. 归零（KEY\_ZERO）：优先级很高，无论何时只要按下就归零，放在0E位置。

此处不仅要将B\_WORK置零使得计时不进行，还要将计时结束后的蜂鸣器叫停，因此要将B\_TMOVER置零。然后才是将时间值统统置零。

KEY\_10S:

        SETB    B\_KEYDO

        CLR     B\_TIME\_P

        MOV     A,KEYCODE

        MOV     B,#60

        CJNE    A,#04H,KEY\_1M

        MOV     A,SECOND

        ADD     A,#10

        DIV     AB;这里考虑进位的，如果秒数超过60，/60，取出商进位和余数放秒

        MOV     SECOND,B

        ADD     A,MINUTE

        MOV     MINUTE,A

        CLR     C

        SUBB    A,#100

        JC      KEY\_RET

        MOV     MINUTE,#0

        LJMP    KEY\_RET

* + - * 1. 加时间：10s（04H），1min（03H），10min（02H）

我的设计是，这里设置过时间后才开始倒计时，因此需要将B\_TIMER\_P置零。同时要考虑非法输入，这里设计成秒为60进制，59🡪00，分钟上限为99,99🡪00.

要注意的是，不仅在加分的时候要用SUBB 或者CJNE使其完成100进制，还要注意在加10s的时候，发生进位也要考虑，即99:50🡪+10s🡪00:00

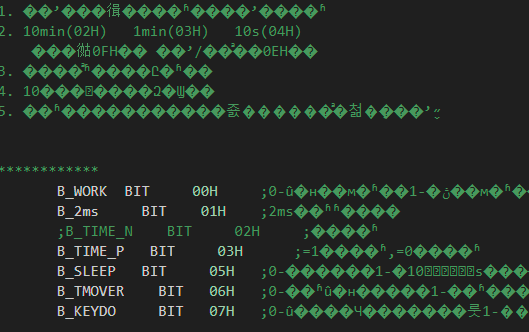
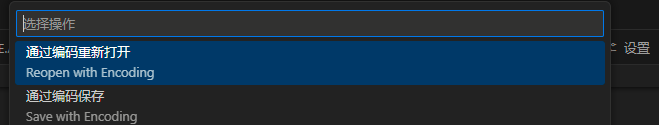
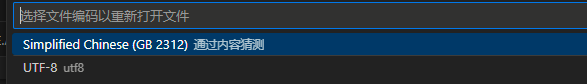
进位的做法和前面正计时一样。这里用了CJNE和SUBB两种方式实现了同样的功能。

* + - * 1. 返回：返回的时候要将键码归为#0FFH的无效码，否则前后会牵扯不清

KEY\_RET:

        MOV     KEYCODE,#0FFH

        RET

1. **实验体会与思考**
   1. 和老师交流过后我对所谓“状态机”有了一定的理解。我发现状态变量真是个好东西，采用位寻址的话，可以用0和1表示状态。在写代码之前，不妨先对照要求和目的，整理出这个系统一共有哪些功能和那些可能的状态，再将功能和状态结合起来，可以用流程图或者思维导图的形式，这样看上去会非常清晰，在编写代码的时候也可以做到模块化和结构化调试。
   2. 在设计蜂鸣器功能的时候，我先是把蜂鸣器单独放在外面直接当做计时结束程序，然后希望在倒计时模块中直接判断是不是计时结束了，但是这样我发现很乱。后面我是在上厕所的时候突然想到了这个解决方案。就是在中端服务程序中放一个实时监测分秒有没有同时到0的程序，然后给系统一个反馈量B\_TMOVER，这样，我在倒计时模块中就不用考虑那么多复杂情况，一个JB或者JNB就可以实现功能。
   3. 如果可以用EQU或者BIT给地址或者立即数“重命名”，一定要试着做，并且要用一致的命名规则，这样在编写较大程序的时候会给程序的调试带来很大的方便。
   4. **说到这里，伟福的编译器看上去太难用了。**更方便一点的可以用VS Code，在拓展商店下载MASM/TASM插件后，ASM文件就有高亮显示了。在VS Code编译器里面，批量改匹配项或者查找都很方便，对眼睛也更友好。但是这里还有一个小的设置要注意，伟福的中文注释采用的是GB 2312编码，如果直接在VS Code里面打开的话，它默认UTF-8，注释会变乱码。这里在右下角找到UTF-8，选择通过编码重新打开或者保存，搜2312，选GB 2312即可。

可以同时打开伟福和VS Code在VS Code里面改完代码之后，ctrl+s保存，回到伟福窗口就可以实时载入的。编码类型一致之后也不会出现乱码问题。这样VS Code用作写代码的工具，伟福用来编译和调试，是一个比较舒适的解决方案。