实验名称：_倒计时定时器__姓名：____严旭铧_____学号：___3220101731_____

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___

浙江大学实验报告

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

一、 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。

二、 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

三、 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
      i. 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：

```
B_WORK  BIT    00H    ;0-没有进行计时；1-在进行计时工作
B_2ms    BIT    01H    ;2ms 定时时间标记
B_TIME_P  BIT    03H      ;=1 正计时,=0 倒计时
B_SLEEP  BIT    05H    ;0-正常；1-无操作满 10s 进入息屏
B_TMOVER   BIT   06H    ;0-计时没有结束；1-计时结束，随后蜂鸣器响
B_KEYDO    BIT   07H    ;0-没有有效按键输入；1-有有效按键
```

      ii. 定时器等其他变量初始化
   (2) 时间处理模块
      i. 正计时及时间处理（TIMER_P）
      ii. 倒计时及时间处理（TIMER_N）
         a) 倒计时处理

b) 倒计时结束（TMOVER）

c) 倒计时结束状态判断（TIMER_N_TF）（全局判断）

iii. 息屏时间计时（TIMER_SLEEP）

(3) LED 显示模块

i. 缓冲模块，存放时间值

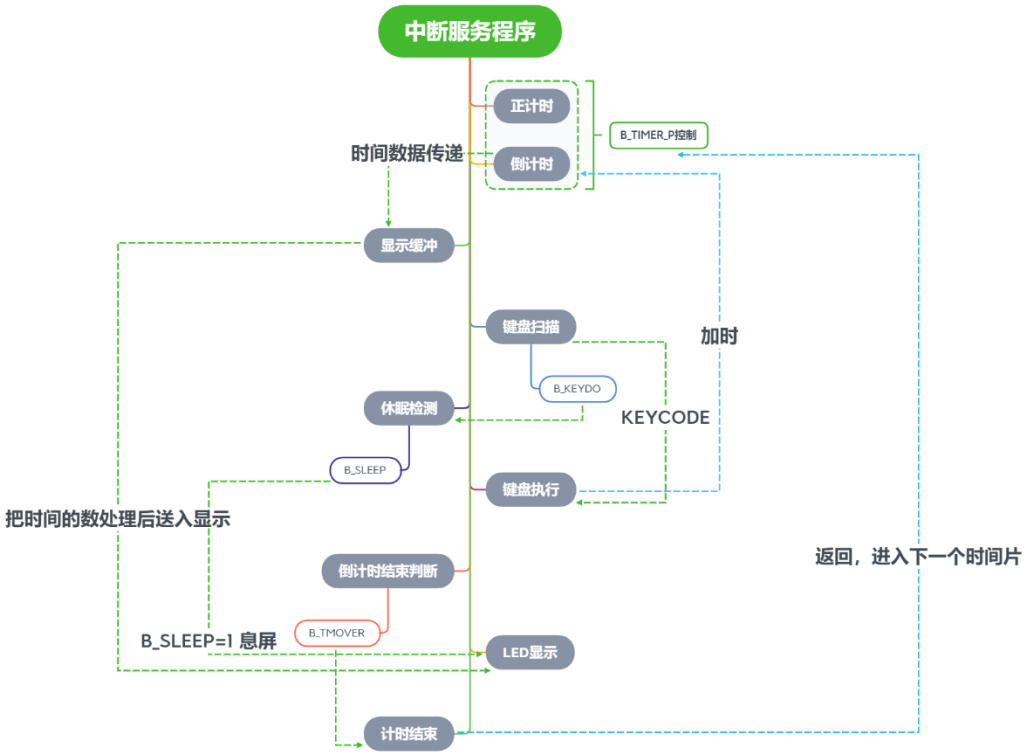
ii. 显示模块

(4) 键盘模块

i. 键盘扫描（KEY_SCAN）

ii. 键值映射及处理（DO_KEY）

4. 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）。

四、 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：

```
TIMER0_INT:
        MOV     TH0,#0F8H
        MOV     TL0,#30H                ;2mS
        SETB      B_2ms                 ;2ms 计时标志
        RETI
```

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。

(3) 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。

2. 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。

i. 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。

ii. 代码块分析：

a) 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。

b) 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。

c) 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。

d) 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
```

(2) *倒计时模块（TIMER_N）：设置了时间之后按开始进入倒计时。

i. B_TIMER_P=0 且 B_WORK = 1 进入倒计时

ii. 代码块分析

a) 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

b) *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
```

```
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
```

i. 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

ii. 难点：00→59 or 00→00?

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。

```
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
```

(3)无操作时间（TIMER_SLEEP）

```
JB      B_SLEEP,TIMER_S_RET0
JB      B_KEYDO,TIMER_S_RET0
```

```
TIMER_S_RET:
        RET
TIMER_S_RET0:
        MOV     T_100MS_SLEEP,#0
        MOV     T_2MS_SLEEP,#0
        SJMP    TIMER_S_RET
```

i. 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。

ii. 代码块分析：

a) 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

b) 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。

3. 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

i. 全局，每次都会进入该子程序。

ii. 代码块分析：

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。

```
MOV     A,MINUTE
MOV     B,#10
DIV     AB
MOV     @R0,A
INC     R0
MOV     @R0,B
INC     R0
```

(2) LED 显示

i. 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。

```
JB      B_SLEEP,DISP_LED_NO
.....
DISP_LED_NO:
        MOV     A,#0FFH
        MOV     P2,A
        MOV     P0,A
```

4. 键盘模块

(1) 键盘扫描（KEY_SCAN）

i. 全局，每个时间片都要进入

ii. 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。

a) 行扫描和列扫描

```
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
```

b) 去抖动，并返回键值（KEYCODE）

```
;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
```

(2) *键盘功能处理（DO_KEY）

i. 全局，每个时间片都要进入

ii. 代码块分析：

a) 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
```

b) 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。

```
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
```

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。

c) 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_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
```

d) 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。

```
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
```

e) 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

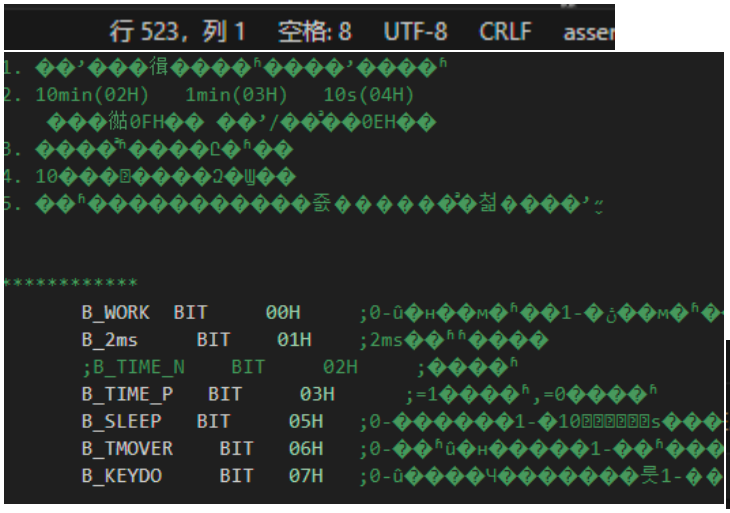
```
KEY_RET:
        MOV     KEYCODE,#0FFH
```

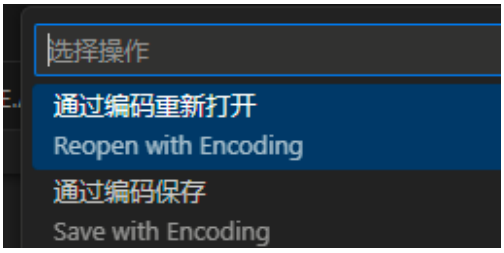
五、 实验体会与思考

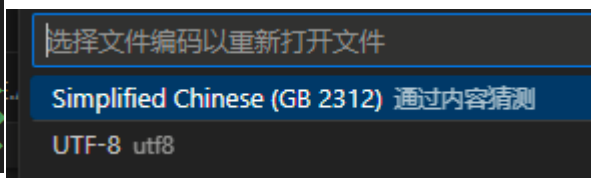
1. 和老师交流过后我对所谓“状态机”有了一定的理解。我发现状态变量真是个好东西，采用位寻址的话，可以用 0 和 1 表示状态。在写代码之前，不妨先对照要求和目的，整理出这个系统一共有哪些功能和那些可能的状态，再将功能和状态结合起来，可以用流程图或者思维导图的形式，这样看上去会非常清晰，在编写代码的时候也可以做到模块化和结构化调试。
2. 在设计蜂鸣器功能的时候，我先是把蜂鸣器单独放在外面直接当做计时结束程序，然后希望在倒计时模块中直接判断是不是计时结束了，但是这样我发现很乱。后面我是在上厕所的时候突

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。