合肥工业大学
计算机与信息学院
课程设计

课　　程：硬件工程师综合训练

专业班级：计算机科学与技术 18-3 班

学　　号：2018211991

姓　　名：余梓俊

一、设计题目及要求：

【课题 5】模拟计算器

设计要求：

1．通过小键盘输入数据和运算符，完成加、减、乘运算。左侧四个数码管用于输入数据和结果数据的显示。

2．按键规定：

⑴ 数字用小键盘 0~9 输入。

⑵ 功能按键设定：

“A”——“+”

“B”——“-”

“C”——“*”

“D”——“括号”

“E”——“=”

“F”——开始运算（包括撤消运算），屏幕显示“0”。

3．运算要求：

⑴ 输入待计算数据（小于四位数），数码管跟随显示。

⑵ 按“+”、“-”、“*”或“括号”时，当前显示内容不变。

⑶ 再输入数据时，数码管跟随显示。

⑷ 按“E”时，显示最终结果数据。若计算结果为奇数，则点亮 1 个红色发光二极管，并持续以 1 秒间隔（硬件实现）闪烁；若计算结果为偶数，则点亮 2 个绿色发光二极管，并持续以 2 秒间隔（硬件实现）闪烁。

⑸ 按“F”键：左侧四个数码管中最右边（对应个位数）的一个显示“0”，其余三个不显示内容。同时熄灭点亮的发光二极管，等待下一次运算的开始。

⑹ 需要考虑运算的优先级问题。

⑺ 可以只考虑正整数运算，不考虑负数和实数运算。括号可以不考虑嵌套情况，但必须能实现算式中存在多组平行括号的计算。

设计说明：

输入数据时，若超出显示范围则不响应超出部分。在计算结果超出显示范围时，则显示“F”。

参考实验：

键盘扫描显示实验;8255 并行口实验(三):控制交通灯;定时/计数器:8253 方波;8259 单级中断控制器实验。

二、设计思想：

1. 准备工作：

（1）通过阅读 8259 中断示例程序及其连线，回顾 8259 芯片初始化编程和中断向量设置的方法，以及 38 译码器与地址总线和各芯片的连接并计算连线后的端口地址。具体为：译码器 abc 口分别与地址总线 a2,a3,a4 相连，译码器每个选中的端口对应四个地址，且 Y0 口对应的最小地址为 0ffe0 H。

（2）通过阅读键盘扫描示例程序，了解读取按键的方法和数码管显示数字的原理。具体为：getkey 示例子程序会将按下的键码处理后保存在 al 中（处理后 012...def 分别对应 00h 01h 02h...，空键对应 20h）；数码管显示子程序 disp 会依次点亮显示 6 个数字，需要不停调用才能看到正常的常量的数字。

（3）通过阅读 8253 并行接口芯片示例程序，掌握了控制向交通灯的方法，通过 8253 向交通灯送 1 为灭，送 0 为亮，且 8253 端口地址为固定值。

2. 协作思想：

笔者先编写整体的代码框架，定义好可能需要的变量名和函数名，编写主程序的伪码，然后使用 GitHub 创建一个代码仓库并提交框架代码。然后进行大致分工，每个人研究透自己负责部分的程序原理，fork 代码仓库后编写好自己的代码，进行代码提交和合并，合并时一起解决合并冲突，并交流各自部分的原理。

其中笔者负责编写程序框架和各部分的连接、取得栈顶符号和当前符号优先级的子程序、对栈进行一次符号运算的子程序，以及将 word 型变量 display_num 转换成一个个数字送个 led_buf 的子程序。

由于汇编代码协作的特殊性，考虑到用栈来实现子程序的传参和返回，不仅工作量大，而且容易出错，我们决定在各个函数之间通过对全局变量的修改来进行协作开发。

3. 算法思想：

主程序不断循环，在其中先后调用 getkey（把键值保存在 current_key 中），handle_key（根据键码相应地调用不同的 handle 子函数，进行栈操作、运算以及 set_led_num），disp（把 led_buf 中存的要展示的数字依次点亮）。

4. 程序细节：

（1）8259 采用边沿触发，开放 IRQ0，屏蔽字为 1111 1110，icw2 为 08h，因此 IRQ0 对应中断向量号为 08h，中断向量地址为 20h 22h。 8253 采用方

式三，分频 T7=19.2KHz，1s 的计数初值应为 19200，2s 相应翻倍。由于方式三自动重装初值，因此中断服务子程序中不需要再次写入计数初值，只需要根据交通灯状态相应开和关即可。

（2）对于如何取得栈顶符号和当前符号优先级，先定义一个 5*5 的 byte 表（详见程序），取得这两个符号后，分别根据这些符号所在的行号和列号将值修改为 m-1 和 n-1（行号减一和列号减一），通过 (m-1)*5 + n-1 的偏移地址，就可以取出相应的优先级。

（3）Handle_key 子程序根据具体按下的键，分别调用不同的 handle 子程序，根据处理逻辑的不同，分为对数字的处理、对加减乘的处理、对括号的处理和对等号、清空键的处理。

（4）其余一些重要函数的算法流程见下文的功能流程部分。

三、功能流程图：

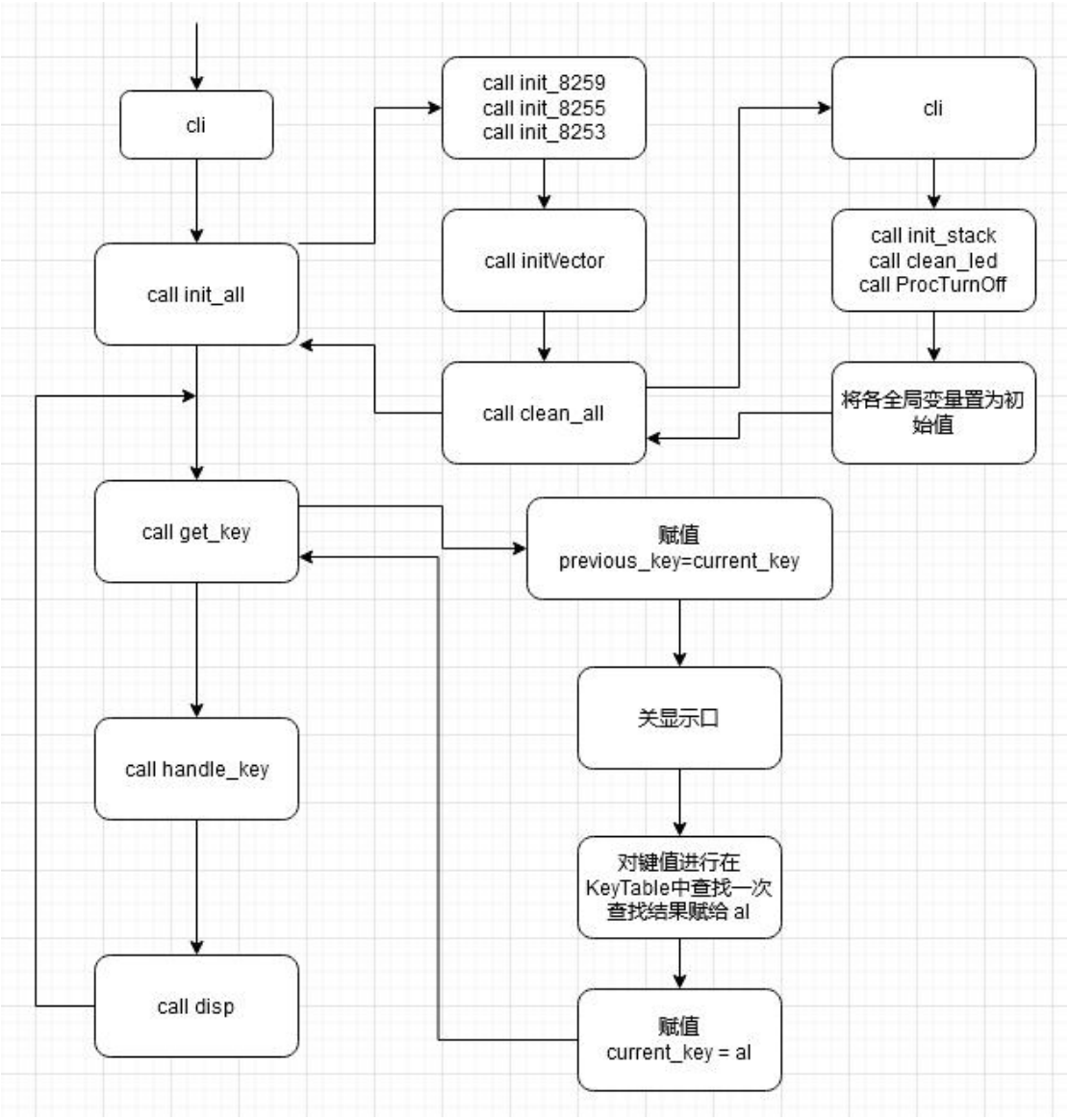
主程序以及部分简单子程序的流程：

说明：

1. Handle_key 会根据 current_key 的具体键值来进一步调用 handle_number、handle_a、handle_e 等等子过程。

2. handle_f 也会调用 clean_all 子程序，调用过程中首先 cli，即不再相应 8253 计时结束发出的中断，以及调用关灯、清数码管、重置栈和全局变量。Handle_f 逻辑非常简单，后文不再给出流程图。

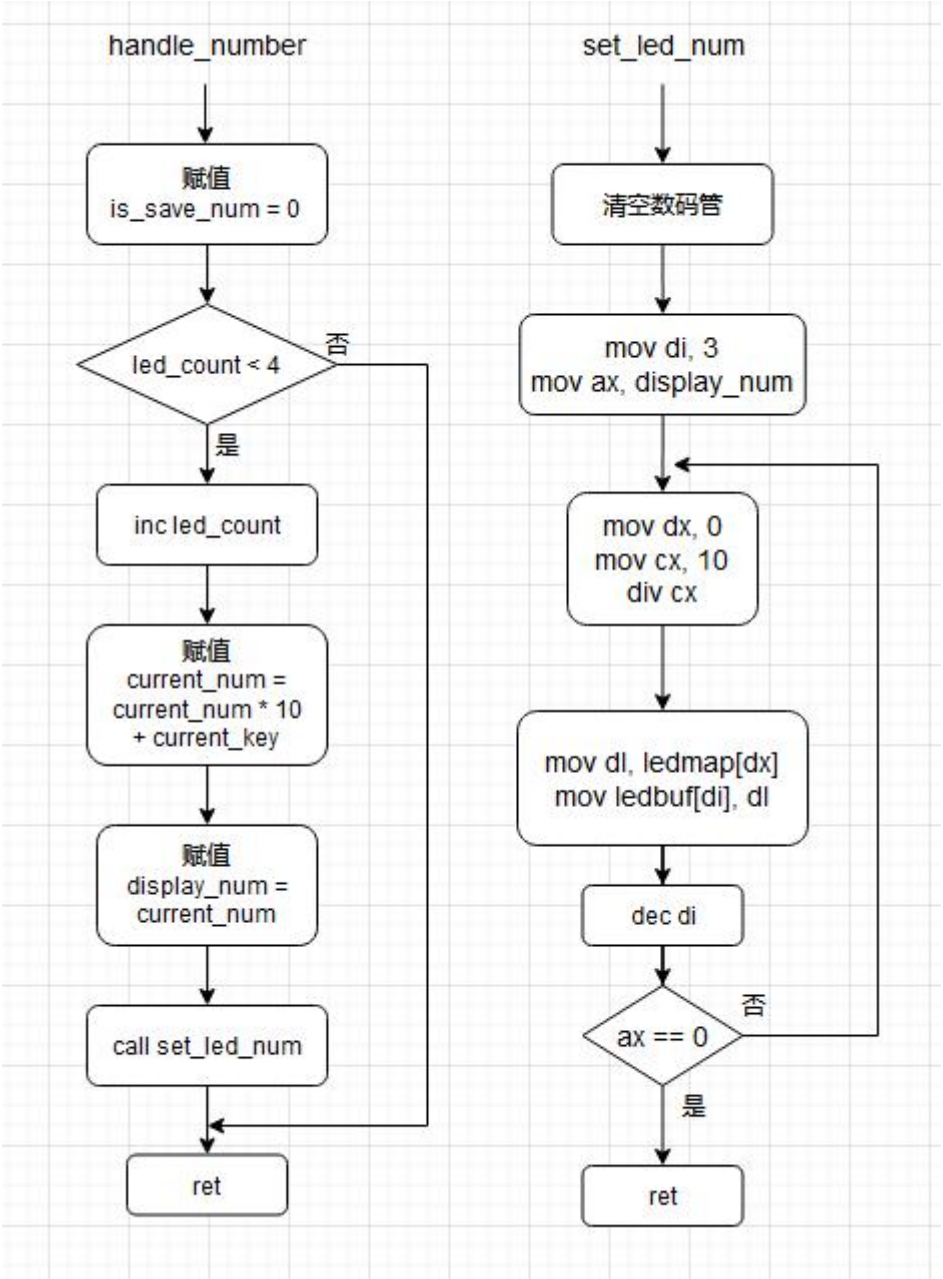
3. 注意图中 get_key 这一示例代码中的子程序被进行了简单地修改，加入了对 current_key 和 previous_key 的赋值操作，previous_key 的作用是防止主程序在循环的过程中，对上一循环的相同键值做出反应。具体来说，当 current_key 和 previous_key 相同，说明当前这一瞬间和前一瞬间的键值相同，也就是按下未松手或一直未按下，此时不应该重复进行按钮的逻辑操作。这一过程在 handle_key 中体现，如果 current_key = previous_key，handle_key 子程序将直接返回。



按下数字键的子程序（handle_number）流程：

说明：

1. is_save_num 变量将在 handle_a 子程序中用到，其含义是 current_num 是否已经保存，按下数字时应设为 0。



按下加减乘的子程序（handle_a）流程：

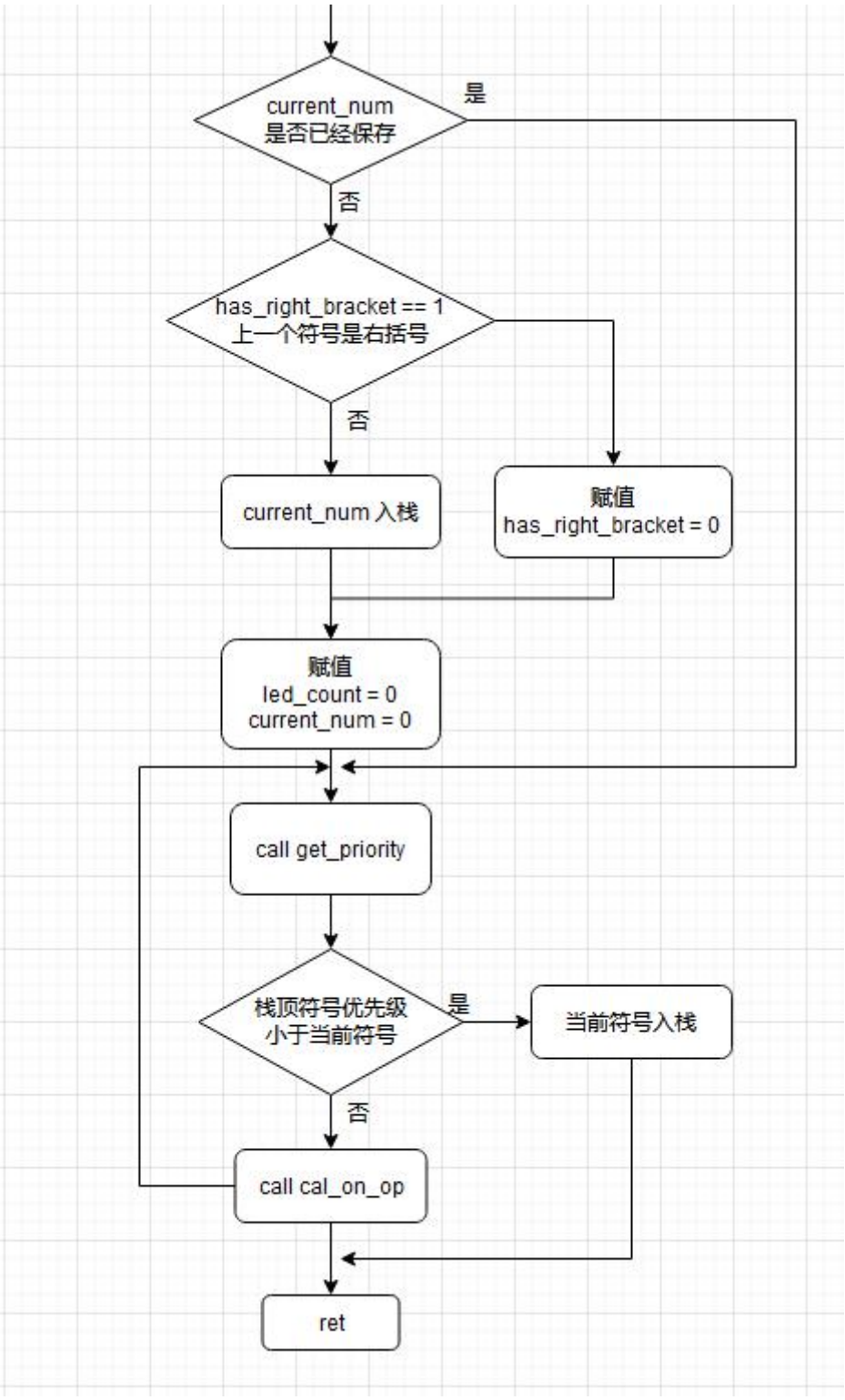
说明：

1. 子程序中还有简单错误处理逻辑（通过 whole_error 全局变量实现），为流程简洁清晰，图中不进行描述。

2. 如果 current_num 已经保存或前一个符号是右括号，那么 current_num 不需要入栈。

3. get_priority 子程序会计算栈顶符号与当前符号的优先级比较结果，结果保存在 priority 全局变量中，计算方法在前文 二.4.(2) 中已经介绍。

4. cal_one_op 子程序将弹出一个符号，把该符号作用于数字栈顶端的两个数字，然后弹出一个数字，并用计算结果覆盖弹出后的栈顶数字。

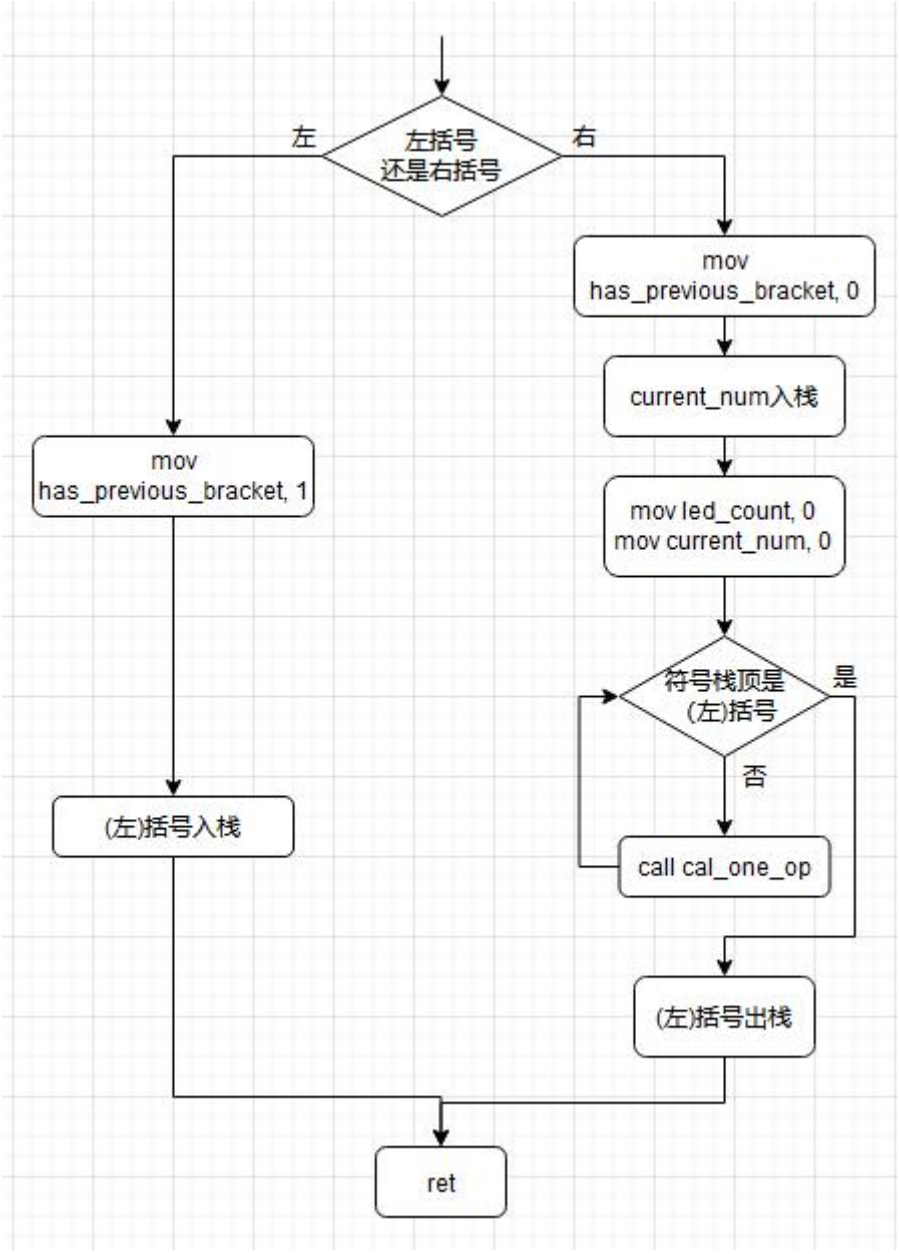


按下括号的子程序（handle_d）流程：

说明：

1. 通过 has_previous_bracket 变量判断当前是左括号还是右括号，如果值为 1，说明当前输入为右括号，否则为左括号。

2. 与 handle_a 子程序的逻辑不同，（合法输入中）左括号前面必是符号，右括号前面必是数字，而加减乘的前面既可能是数字也可能是符号（括号）。



按下等号的子程序（handle_e）流程：

说明：

1. 按下等号的计算逻辑与右括号类似，其余逻辑不多，不再用流程图表示。

2. 首先判断上一个符号是否为右括号，不是的话将 current_num 入

栈，接下来反复调用 cal_one_op 直至符号栈顶为'#'号，然后 sti 开中断，调用 procTurnOn 开红绿灯，调用 procWriteCount 写计数初值。

3. 上述反复计算的过程中以及 cal_one_op 子程序中，都会判断是否出错，比如数字栈内数字不够或者做完最后的运算时仍有多余一个的数字等等，一旦错误，数码管会显示'F'（同溢出的显示相同，包括上述介绍的其他一些子程序中也有对溢出和非法表达式的判断，为简洁起见，没有在流程图写出这些逻辑）。

中断服务子程序流程：

说明：

1. 同样地，因为采用方式 3 自动重新开始计数，中断服务子程序逻辑简单，不再给出流程图。

2. 首先关中断，然后根据 flag（灯是否在亮的标志）来决定调用 procTurnOff 还是 procTurnOn，然后开中断，返回。

3. 由于在按下 e 时会 sti 开中断，而按下 f 调用 clean_all 子程序时会 cli 关中断，因此虽然在按下 f 重置后，8253 依然在不停地计时并出发中断，但 8088 并不会相应。

四、结果讨论：

实验结果：

在小组五人的共同努力下，我们实现了题目所要求的各种功能，并且在最后验收之后，我们还继续补充了输入非法表达式也显示‘F’这一错误处理功能。得益于清晰的整体框架和高可读的代码风格，我们在程序设计的过程中就留有错误处理的空间，补充错误显示也相当轻松。

实验收获：

与隔壁班的同课题小组相比，我们在实验的前中期进度非常缓慢，但是得益于我们清晰的分工、协作工具 git 的使用以及模块化的代码设计，使得我们在中后期的速度非常快。

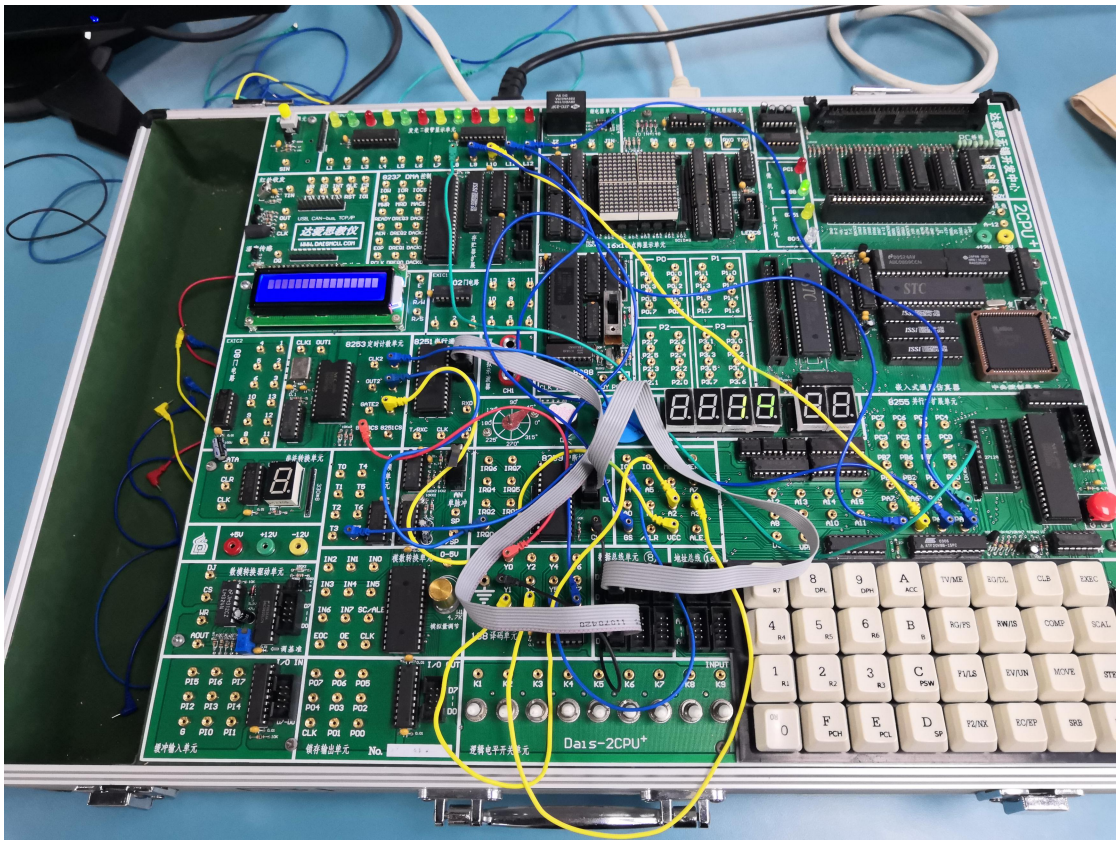
实验中也遇到了不少问题，既有代码的逻辑问题、硬件的原理问题，也有硬件本身的问题。

比如在实验初期，由于我们定义了大量的全局变量，我们的代码地址与数据的地址产生了意料之外的冲突，我们花费了大量时间才找到了问题的原

因，最后通过将全局变量定义在代码段的开头，然后用一条 jmp 语句跳过这些放在代码段的数据，才成功的解决了问题。

再比如，我们在原来的实验箱进行计算时，结果时对时错，我们一度认为是代码逻辑存在漏洞，但最后发现换了箱子之后，运算结果就变得稳定了。

诸如此类的问题还有许多，但欣慰的是我们最后解决了所有的问题，并在此过程中大大加深了对微机原理的理解，比如译码器、地址总线、接口芯片三者的连接方法与对应的关系，中断程序和中断向量的关系等等，同时也加深了小组同学之间的合作默契，结识了隔壁班级同课题小组的同学，收获满满。



附录：实验代码：（完整的源程序）

```
code    segment
        assume cs:code

org  1200h
start:

    jmp true_start
    ; 中断控制器 8259
    ; 8259 只处理来自 8253 的计时中断
    port59_0    equ 0ffe4h
    port59_1    equ 0ffe5h
    icw1        equ 13H          ; 边沿触发
    icw2        equ 08h          ; 中断类型号 08H 09H ...
    icw4        equ 09h          ; 全嵌套,缓冲从片,非自动 EOI,8086/88 模式 为
什么要用缓冲单片?
    ocw1open    equ 0feh         ; IRQ0，类型号为 08h
    vectorOffset EQU 20H         ; 中断向量的地址 08H*4=20H
    vectorSeg   EQU 22H          ; 中断向量的 CS 段地址在中断向量表中的地址，
值为 0

    ; 并行接口芯片 8255
    ; 8255 向 led 灯输出 led 状态
    port55_a    equ 0ffd8H
    port55_ctrl equ 0ffdBH

    ; 计数定时芯片 8253
    port53_0    equ 0ffe0H
    port53_ctrl equ 0ffe3H       ; 控制口
    count53_second1  equ 19200       ; 1s 计数次
数 T7=19.2KHz Tn=4.9152MHz/2^n
    count53_second2  equ 38400       ; 2s 计数次数


    ledbuf                  db 6 dup(?)
    led_count               db 0
    previous_key            db 20h
    current_key             db 20h
    has_previous_bracket    db 0
    has_right_bracket       db 0
    same_as_pre             db 0
```

```
    operator_stack          db '#', 100 dup(?)      ; si
    operand_stack           db 0ffh, 0ffh, 100 dup(?)   ; di

    priority                db 0    ; 0 栈顶<下一个; 1 =; 2 >
    is_save_num             db 0    ; 当按下一个运算符时，current_num 是否
已经保存，为了处理连输运算符号的错误情况
    current_num             dw 0
    display_num             dw 0    ; 按下符号后，会把 current_num 清零，但
不把 display_num 清零
    has_input_e             db 0    ; 统计是否已经按下过 e 键
    result                  dw 0    ; 总的计算结果
    overflow                db 0
    whole_error             db 0

    ; # ( +- *    优先级从小到大

    ;   # ( + - *
    ; # f 0 0 0 0
    ; ( f f 0 0 0
    ; + 2 2 1 1 0
    ; - 2 2 1 1 0
    ; * 2 2 2 2 1
    priority_table  db  0ffh, 0, 0, 0, 0
                    db  0ffh, 0ffh, 0, 0, 0
                    db  2, 2, 1, 1, 0
                    db  2, 2, 1, 1, 0
                    db  2, 2, 2, 2, 1


    OUTSEG  equ  0ffdch             ;段控制口
    OUTBIT  equ  0ffddh             ;位控制口/键扫口
    IN_KEY  equ  0ffdeh             ;键盘读入口

    LightOnGreen EQU 0edh ;绿灯  1110 1101   按照题目例子中给的接线方式接线
    LightOnRed EQU 0feh ;红灯    1111 1110
    lightOff EQU 0FFH; 关灯      1111 1111
    flag    db 0                ;灯是否在亮


true_start:
        cli
        call init_all
main:
        call get_key
        cmp current_key, 20h
```

```
        je handle
        and  al,0fh
    handle:
        call handle_key
        ;call set_led_num
        call disp
        jmp main
; end



init_all proc
        call init8259
        call init8255
        call init8253
        call initVector
        call clean_all
        ret
init_all endp


init8259 proc
        push ax
        push dx
        mov dx, port59_0
        mov al, icw1
        out dx, al
        mov dx, port59_1
        mov al, icw2
        mov dx, port59_1
        out dx, al
        mov al, icw4
        out dx, al
        mov al, ocw1open
        out dx, al
        pop dx
        pop ax
        ret
init8259 endp


init8255 proc
        push ax
        push dx
        mov dx, port55_ctrl
```

```
        mov al, 88H             ;8255A 控制字 88H，使 AB 端口均为输出口，C 口
高位输入，低位输出，且全部工作在方式 0 下
        out dx, al
        mov al, lightOff
        mov dx, port55_a
        out dx, al
        pop dx
        pop ax
        ret
init8255 endp


init8253 proc
        push dx
        push ax
        mov dx, port53_ctrl
        mov al, 36H             ; 计数器 0，先低 8 位，再高 8 位，方式 3，二进制
计数
        out dx, al
        pop ax
        pop dx
        ret
init8253 endp


init_stack proc
        mov si, 0
        mov di, 0
        ret
init_stack endp


initVector proc
        cli
        Push bx
        Push ax

        Mov ax , offset flash   ;中断向量表的初始化
        Mov bx , vectorOffset
        Mov [bx] , ax

        mov bx,vectorSeg        ;中断向量的段地址对应的中断向量表的地址
        mov ax,0000H
        mov [bx],ax
```

```
        Pop ax
        Pop bx
        sti
        Ret
initVector endp


clean_all proc
        cli
        call init_stack
        call clean_led
        call ProcTurnOff
        mov previous_key, 20h
        mov current_key, 20h
        mov led_count, 0
        mov has_previous_bracket, 0
        mov has_right_bracket, 0
        mov same_as_pre, 0
        mov current_num, 0
        mov display_num, 0
        mov result, 0
        mov overflow, 0
        mov is_save_num, 0
        mov whole_error, 0
        mov has_input_e, 0
        ret
clean_all endp


clean_led proc
        mov  LedBuf+0,0ffh
        mov  LedBuf+1,0ffh
        mov  LedBuf+2,0ffh
        mov  LedBuf+3,0c0h
        mov  LedBuf+4,0ffh
        mov  LedBuf+5,0ffh
        ret
clean_led endp


;---------------中断服务程序---------------------
flash proc
        cli ;关中断
        test flag,1 ;判断当前灯是否亮
        Jz turnOn       ;不亮则开灯
```

```
        ;TurnOff
        call ProcTurnOff    ;亮灯则关上
        Jmp flashOK
    turnOn:
        call ProcTurnOn

    flashOK:
        ; call ProcWriteCount;重新计数
        mov dx,port59_0
        mov al,20h  ;0010 0000 普通 EOI 方式 OCW2
        out dx,al
        STI     ;开中断
        IRET
flash endp


ProcTurnOn proc
        push dx
        push ax

        Mov dx, Port55_A
        test result,1h      ;判断是否是奇数
        jz green        ;是偶数则亮绿灯
        mov al, LightOnRed
        jmp rgOk
    green:
        mov al, LightOnGreen
    rgOk:
        Out dx, al
        mov flag,1

        pop ax
        pop dx

        ret
ProcTurnOn endp

ProcTurnOff proc
        push dx
        push ax

        Mov dx, Port55_A
        Mov al, lightOff
        Out dx, al
        mov flag,0
```

```
        pop ax
        pop dx

        ret
ProcTurnOff endp


ProcWriteCount proc
        mov dx, port53_0  ;第一个计数器通道的端口地址
        test result,1h       ;判断 result 是否为奇数
        jz second2
        mov ax,count53_second1  ;如果是奇数，则写入计数初值 1s
        jmp countSetOK
    second2:
        mov ax,count53_second2
    countSetOK:
        out dx,al       ;先写低 8 位，再读写高八位，方式 3，二进制计数
        mov al,ah
        out dx,al
        ret
ProcWriteCount endp


get_key proc                    ;键扫子程序
    ; store key in current_key
        push ax
        push bx
        push cx
        push dx

        mov al, current_key     ;上一次扫描的符号
        mov previous_key, al

        mov  al,0ffh            ;关显示口
        mov  dx,OUTSEG
        out  dx,al
        mov  bl,0
        mov  ah,0feh
        mov  cx,8
    key1:
        mov  al,ah
        mov  dx,OUTBIT
        out  dx,al
        shl  al,1
```

```
    mov  ah,al
    nop
    nop
    nop
    nop
    nop
    nop
    mov  dx,IN_KEY
    in   al,dx
    not  al
    nop
    nop
    and  al,0fh
    jnz  key2
    inc  bl
    loop key1
nkey:
    mov  al,20h
    mov current_key, al
    pop dx
    pop cx
    pop bx
    pop ax
    ret
key2:
    test al,1
    je   key3
    mov  al,0
    jmp  key6
key3:
    test al,2
    je   key4
    mov  al,8
    jmp  key6
key4:
    test al,4
    je   key5
    mov  al,10h
    jmp  key6
key5:
    test al,8
    je   nkey
    mov  al,18h
key6:
```

```
        add  al,bl
        cmp  al,10h
        jnc  fkey
        mov  bx,offset KeyTable
        xlat
    fkey:
        mov current_key, al
        pop dx
        pop cx
        pop bx
        pop ax
        ret
get_key endp


handle_key proc
        push ax
        call is_same_as_pre
        cmp same_as_pre, 1
        jne handle_key_continue
        pop ax
        ret
    handle_key_continue:
        mov al, current_key
        cmp al, 10
        jnb handle_key_a
        call handle_number
        pop ax
        ret
    handle_key_a:
        cmp al, 0ah
        jne handle_key_b
        call handle_a
        pop ax
        ret
    handle_key_b:
        cmp al, 0bh
        jne handle_key_c
        call handle_b
        pop ax
        ret
    handle_key_c:
        cmp al, 0ch
        jne handle_key_d
        call handle_c
```

```
        pop ax
        ret
    handle_key_d:
        cmp al, 0dh
        jne handle_key_e
        call handle_d
        pop ax
        ret
    handle_key_e:
        cmp al, 0eh
        jne handle_key_f
        call handle_e
        pop ax
        ret
    handle_key_f:
        cmp al, 0fh
        jne key_error
        call handle_f
        jmp handle_key_f_ret
        key_error:
        call handle_error
        handle_key_f_ret:
        pop ax
        ret
handle_key endp

is_same_as_pre proc
    ;给 same_as_pre 赋值
        push ax
        mov al, current_key
        cmp al, previous_key
        je is_same
        mov same_as_pre, 0
        jmp return
    is_same:
        mov same_as_pre, 1
    return:
        pop ax
        ret
is_same_as_pre endp



handle_number proc
    ; 如果 led_count < 4
```

```
    ;   current_num = current_num * 10 + current_key
    ;   led_count += 1
    ; 当输入数字以外的符号的时候需要把 led_count 清空
        push ax
        push bx
        push dx
        mov is_save_num, 0          ; 输入新的数字时，设置成当前数字还未保
存
        cmp led_count, 4
        jae handle_number_ret
        mov ax, current_num
        mov bx, 10
        mul bx
        mov bl, current_key
        mov bh, 0
        add ax, bx
        mov current_num, ax         ;current_num = current_num * 10 +
current_key
        inc led_count
        push ax
        mov ax, current_num
        mov display_num, ax
        pop ax
        call set_led_num
    handle_number_ret:
        pop dx
        pop bx
        pop ax
        ret
handle_number endp

handle_error proc
    ;处理 get_key 得到的字符不是数字和符号的情况，包含 current_key=20h
        cmp current_key, 20h
        je handle_error_ret
        ; 处理其它的符号
    handle_error_ret:
        ret
handle_error endp

handle_a proc
    ; 处理输入的是加减乘的情况

    ; 如果数字已经保存或刚输入过右括号
```

```
    ;    则不把数字压入栈
    ; 否则，数字入栈

    ; 然后计算

        cmp is_save_num, 0
        jne calculate_a
        mov is_save_num, 1

        ; 数字入栈
        cmp has_right_bracket, 1
        je number_not_push
        inc di
        inc di
        push ax
        mov ax, current_num
        mov operand_stack[di], ah           ;将 current_num 入栈
        mov operand_stack[di + 1], al
        pop ax
        jmp next_a

    number_not_push:
        mov has_right_bracket, 0

    next_a:
        mov led_count, 0
        mov current_num, 0                  ;按下运算符时，数字输入结束，将
当前的数字清空
    calculate_a:
        cmp whole_error, 1
        je a_ret                            ;当前面的式子已经计算出错的时
候后面的式子不需要计算了
        call get_priority
        cmp priority, 0
        je push_a                           ;当前符号优先级大于栈顶符号，直
接入栈
        call cal_one_op                     ;否则计算一次
        jmp calculate_a
    push_a:
        inc si
        push ax
        mov al, current_key
        mov operator_stack[si], al          ;将当前运算符入栈
        pop ax
```

```
    a_ret:
        ret
handle_a endp

handle_b proc
        call handle_a
        ret
handle_b endp

handle_c proc
        call handle_a
        ret
handle_c endp

handle_d proc
        ;处理输入符号是括号的情况
        ;输入是左括号，直接入栈
        ;输入是右括号，反复计算直到栈顶是左括号
        cmp has_previous_bracket, 0
        je no_previous
        mov has_previous_bracket, 0
        mov has_right_bracket, 1

        inc di
        inc di
        push ax
        mov ax, current_num
        mov operand_stack[di], ah          ;将 current_num 入栈
        mov operand_stack[di + 1], al
        pop ax
        mov led_count, 0
        mov current_num, 0                    ;按下运算符时，数字输入结束，
将当前的数字清空

    cal_between_bracket:
        cmp operator_stack[si], 0dh
        je is_left_bracket
        call cal_one_op
        jmp cal_between_bracket
    is_left_bracket:
        dec si
        jmp ret_d

    no_previous:
```

```
        mov has_previous_bracket, 1
        inc si
        push ax
        mov al, current_key
        mov operator_stack[si], al
        pop ax
    ret_d:
        ret
handle_d endp

handle_e proc
        ;处理输入符号是'='的情况
        ;第一次按下'='时，反复计算直到栈顶是'#'号
        push ax

        cmp has_input_e, 0                ;为了解决按下第二次'e'键后显示第二个操作数的问题，判断已经输入过'e'后再次输入'e'时不进行任何操作
        jne handle_e_ret
        mov has_input_e, 1

        cmp has_right_bracket,1
        je num_not_push_e
        inc di
        inc di
        push ax
        mov ax, current_num
        mov operand_stack[di], ah         ;将 current_num 入栈
        mov operand_stack[di + 1], al
        pop ax
    num_not_push_e:
        mov has_right_bracket, 0

    cal_e:
        cmp whole_error, 1
        je ret_e
        cmp operator_stack[si], '#'
        je ret_e
        call cal_one_op
        jmp cal_e
    ret_e:
        cmp whole_error, 1
        je show_error
        cmp di, 2
```

```
        ja show_error                       ;计算完成后，运算符栈内剩余数字大
于一个时计算结果错误
        mov ah, operand_stack[di]
        mov al, operand_stack[di + 1]
        mov display_num, ax
        mov result, ax

        call set_led_num
        sti
        call ProcTurnOn
        call ProcWriteCount
        jmp handle_e_ret
    show_error:
        mov  LedBuf+0,0ffh                  ;计算出现错误时结果显示'F'
        mov  LedBuf+1,0ffh
        mov  LedBuf+2,0ffh
        mov  LedBuf+3,8eh
    handle_e_ret:
        pop ax
        ret
handle_e endp

handle_f proc
        ;处理按下'F'的情况
        call clean_all
        ret
handle_f endp


cal_one_op proc
        push ax
        push bx
        push dx
        cmp si, 1
        jb cal_error
        cmp di, 4
        jb cal_error
        mov ah, operand_stack[di - 2]
        mov al, operand_stack[di - 1]
        mov bh, operand_stack[di]
        mov bl, operand_stack[di + 1]
        mov dl, operator_stack[si]

        cmp dl, 0ah                  ; +
        jne cal_not_plus
```

```
        add ax, bx
        cmp ax,9999
        ja cal_overflow
        jmp cal_ret
    cal_not_plus:
        cmp dl, 0bh                 ; -
        jne cal_not_minus
        cmp ax,bx
        jb cal_overflow         ; 减法得负也为 overflow
        sub ax, bx
        jmp cal_ret
    cal_not_minus:
        cmp dl, 0ch                 ; *
        jne cal_error               ; 不是 + - * 为 error
        mul bx
        cmp dx, 0
        ja cal_overflow            ; 乘法溢出为 overflow
        cmp ax,9999
        ja cal_overflow
        jmp cal_ret
    cal_error:
        mov whole_error, 1
        jmp cal_ret
    cal_overflow:
        mov overflow, 1
    cal_ret:
        dec di
        dec di
        dec si
        mov operand_stack[di], ah
        mov operand_stack[di + 1], al
        pop dx
        pop bx
        pop ax
        ret
cal_one_op endp


get_priority proc
        push ax
        push bx
        push dx
        mov al, operator_stack[si]
        cmp al, '#'
        jne top_not_pound
```

```
        mov al, 0
        jmp curr_operator
        top_not_pound:
        cmp al, 0dh
        jne top_not_bracket
        mov al, 1
        jmp curr_operator
        top_not_bracket:
        sub al, 0ah
        add al, 2

        curr_operator:
        mov dl, current_key
        cmp dl, 0dh
        jne curr_operator_not_pound
        mov dl, 1
        jmp find_in_table
        curr_operator_not_pound:
        sub dl, 0ah
        add dl, 2

        find_in_table:
        mov dh, 5           ; 5 x 5 的优先表
        mul dh
        add al, dl
        mov ah, 0
        mov bx, ax
        mov dl, priority_table[bx]
        mov priority, dl
        jmp get_priority_ret
    get_priority_err:
        mov whole_error, 1
    get_priority_ret:
        pop dx
        pop bx
        pop ax
        ret
get_priority endp


set_led_num proc
    ; 在 handle_number 里调用时
    ; 此时 led_count = 已输入的数字位数
    ; led_count - 1 = 已显示的数字位数
        push ax
```

```
        push bx
        push cx
        push dx
        push di
        mov  LedBuf+0,0ffh
        mov  LedBuf+1,0ffh
        mov  LedBuf+2,0ffh
        mov  LedBuf+3,0ffh
        mov di, 3
        mov ax, display_num

        cmp overflow, 1
        jne ax_not_zero

        overFshow:
        mov LedBuf+3,08eh       ; 8eh : 'F'
        JMP set_led_num_ret

    ax_not_zero:

        mov bx, offset ledmap
        mov dx, 0               ; dx 为被除数的高位，需要置为 0
        mov cx, 10              ; cx 做除数
        div cx
        add bx, dx              ; 除完后，dx 为余数，ax 为商，把余数加到 ledmap
的偏移地址上
        mov dl, [bx]            ; bx 为段码的地址，dl 就是要显示的段码

        mov bx, offset ledbuf
        add bx, di
        mov [bx], dl            ; 等价于 mov ledbuf+di, dl
        dec di

        cmp ax, 0
        jne ax_not_zero

    set_led_num_ret:
        pop di
        pop dx
        pop cx
        pop bx
        pop ax
        ret
set_led_num endp
```

```
disp proc
        mov  bx,offset LEDBuf
        mov  cl,6               ;共 6 个八段管
        mov  ah,00100000b       ;从左边开始显示
    DLoop:
        mov  dx,OUTBIT
        mov  al,0
        out  dx,al              ;关所有八段管
        mov  al,[bx]
        mov  dx,OUTSEG
        out  dx,al

        mov  dx,OUTBIT
        mov  al,ah
        out  dx,al              ;显示一位八段管

        push ax
        mov  ah,1
        call Delay
        pop  ax

        shr  ah,1
        inc  bx
        dec  cl
        jnz  DLoop

        mov  dx,OUTBIT
        mov  al,0
        out  dx,al              ;关所有八段管
        ret
disp endp

delay proc                          ;延时子程序
        push  cx
        mov   cx,256
        loop  $
        pop   cx
        ret
delay endp

    ;八段管显示码
    LedMap  db   0c0h,0f9h,0a4h,0b0h,099h,092h,082h,0f8h
```

```
            db   080h,090h,088h,083h,0c6h,0a1h,086h,08eh
    ;键码定义
    KeyTable db   07h,04h,08h,05h,09h,06h,0ah,0bh
            db   01h,00h,02h,0fh,03h,0eh,0ch,0dh

code    ends
        end start
```