一、 实验目的

1、熟练掌握循环和分支程序设计。特别是分支判断的相关用法要熟练掌握。
2、熟练掌握使用 call 和 ret 来实现子模块的设计，能够使用 push 和 pop 在保存中间的数据。
3、熟练掌握系统中断程序的调用。
4、注意字节类型和子类型的区别，在程序设计时应充分考虑数据类型对程序设计的影响。

二、 实验设备

操作系统：个人 PC Windows10 系统
模拟器环境：emu 8086

三、 实验内容和要求

1、设有 10 个学生成绩分别是 76, 69，84，73，88，99，63，100 和 80。试编写一个子程序统计 60-69 分，70-79 分，80-89 分，90-99 分和 100 分的人数，并分别放到 S6，S7，S8，S9，S10 单元中

2、编制程序计算 S=1+2・3+3・4+4・5+……+N（N+1）+……直到 N（N+1）大于 200 为止，并将结果由屏幕上显示出来。要求计算和部分用子程序实现。

3、假设三组数据的数据个数分别在 CNT1、CNT2、CNT3 单元中。

请计算三组字数据中正数、负数和零的个数，并分别存入 PCOUNT、MCOUNT、ZCOUNT 单元。

（tips: 自行定义数组的首地址及数组元素的值，其中数组元素的值建议采用十进制）

子程序功能：统计一组字数据中正数、负数、零的个数。

主程序功能：三次调用子程序。

四、 实验步骤

实验 1

设计思路
存储逻辑：数组的一个元素为成绩的个数，之后的元素为成绩。

在统计学生成绩的个数中，可以设计一个子程序，用于统计学生成绩的等级（X/10-6），然后由由于S6~S10是一个连续的空间，因此S6+(X/10-6)便是其应当存放的位置。

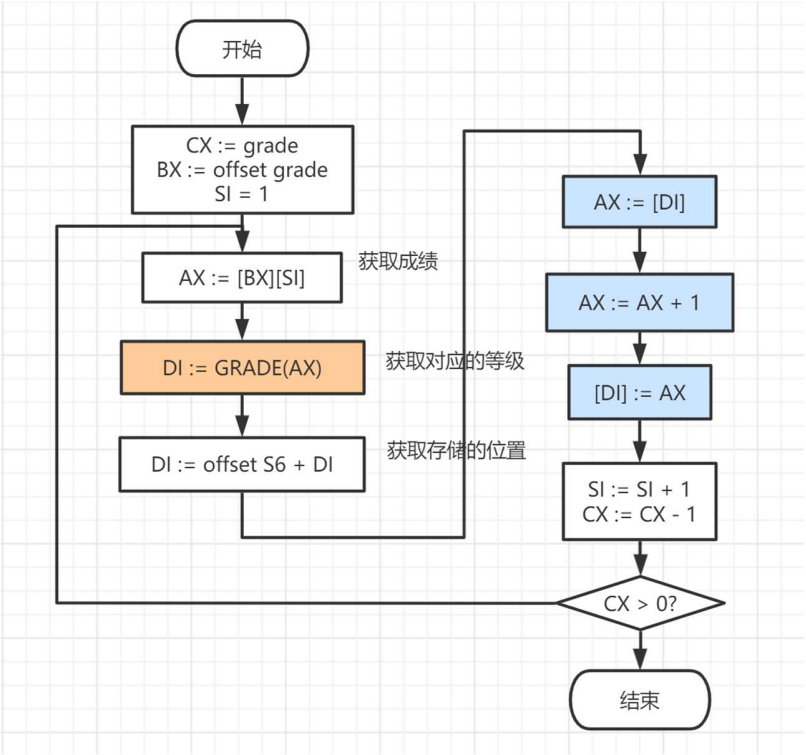

图 8-1 实验 1 流程图

实验代码

```
; multi-segment executable file template.

data segment
    ; add your data here!
    pkey db "press any key...$"
    grade_store db 9,76,69,84,73,88,99,63,100,80
    s6 db 0
    s7 db 0
    s8 db 0
    s9 db 0
    s10 db 0
ends

stack segment
    dw   128  dup(0)
ends

code segment
```

```
start:
; set segment registers:
    mov ax, data
    mov ds, ax
    mov es, ax

    lea bx, grade_store
    mov cl, [bx]
    mov ch, 0 ; cx = 成绩个数
    mov si, 1 ; [bx][si]当前的成绩
loop_grade:
    mov al, [bx][si]
    mov ah, 0
    inc si
    call grade_level
    lea di, s6
    add di, ax ; di = 存储的位置
    mov dl, [di]
    inc dl
    mov [di], dl
    loop loop_grade

    lea dx, pkey
    mov ah, 9
    int 21h        ; output string at ds:dx

    ; wait for any key....
    mov ah, 1
    int 21h

    mov ax, 4c00h ; exit to operating system.
    int 21h
; input ax = 当前的成绩; output ax = 成绩的等级,60~69=0，70~79=1，依次类推
grade_level:
    pushf
    push dx

    mov dl, 10
    div dl ; al= 6..10
    sub al, 6 ;al = 0..5
    cbw

    pop dx
    popf
    ret
```

```
ends

end start ; set entry point and stop the assembler.
```

实验 2

设计思路

将其分为两个模块，一个是计算和的模块，另外一个是输出数字的模块（已经在之前实验中给出）。

其中计算和的模块，需要用到乘法和循环逻辑。

其中在计算和的模块中，AX 存储每次存储的结果，DX 存储和的临时结果，每次循环时，根据 AX 的值判断是否退出循环。

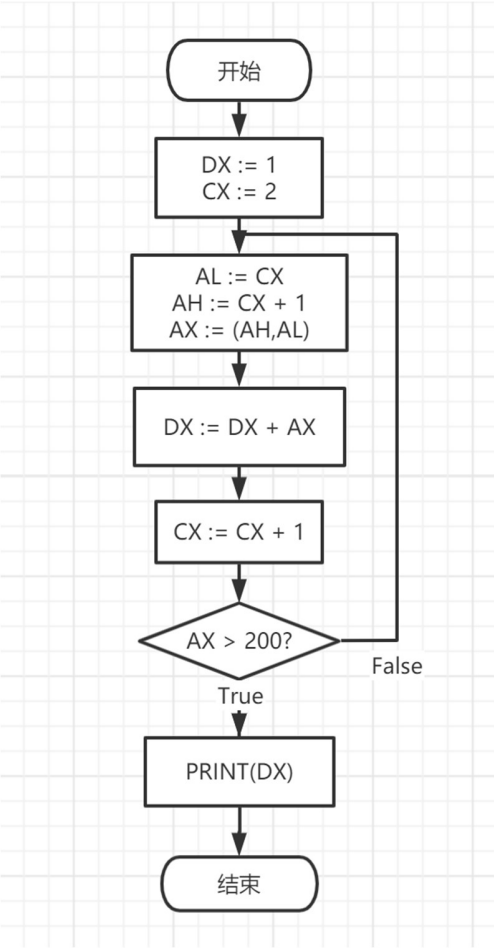

图 8-2 实验流程图

实验代码

```
; multi-segment executable file template.

data segment
    ; add your data here!
```

```
    pkey db "press any key...$"
ends

stack segment
    dw   128  dup(0)
ends

code segment
start:
; set segment registers:
    mov ax, data
    mov ds, ax
    mov es, ax

    call calculate
    call print_number

    lea dx, pkey
    mov ah, 9
    int 21h        ; output string at ds:dx

    ; wait for any key....
    mov ah, 1
    int 21h

    mov ax, 4c00h ; exit to operating system.
    int 21h

calculate:
    pushf
    push dx
    push cx
    mov dx, 1
    mov cx, 2
add_calculate:
    mov al, cl
    mov ah, al
    inc ah ; ah = n + 1, al = n
    mul ah ; ax = ah * al
    add dx, ax
    inc cx
    cmp ax, 200
    jna add_calculate
    mov ax, dx
    pop cx
```

```
    pop dx
    popf
    ret
; 子模块，输入(AX)，输出为屏幕。
print_number:
    ; 保存数据
    pushf
    push bx
    push cx
    push dx

    mov cx, 0
    mov bx, 10

loop_div_number:
    cmp ax, 0
    jz branch_show_number ; 如果 ax!=0 继续执行取数

    mov dx, 0 ; 扩展无符号数
    div bx ; 除 10
    inc cx
    push dx ; 将中间的数字压入堆栈
    jmp loop_div_number ; 无条件循环，必须使用 jmp

branch_show_number:
    cmp cx, 0
    jz print_number_0

loop_print_number:
    pop dx
    add dl, 30h
    mov ah, 2
    int 21h ; 输出堆栈中栈顶的数字
    loop loop_print_number
    jmp print_number_out
print_number_0:
    mov dl, 30H
    mov ah, 2
    int 21h ; 输出字符'0'

print_number_out:
    ; 恢复数据
    pop dx
    pop cx
    pop bx
```

```
    popf
    ret
ends

end start ; set entry point and stop the assembler.
```

实验 3

设计思路

子程序的功能输入当前统计的数组(存入 BX)和目标单元的首地址(存入 DI)，然后每次将数组的首地址赋给 BX，调用子程序。

子程序的功能是依次迭代数组中的元素，判断，并将目标单元中的数据+1。子程序的功能和实验 1 的流程相似。

实验代码

```
; multi-segment executable file template.

data segment
    ; add your data here!
    pkey db "press any key...$"
    cnt1 dw 8,7,6,-2,5,0,0,3,9
    cnt2 dw 9,1,-5,-7,8,11,9,9,1,0
    cnt3 dw 4,3,0,0,-9
    pcount db 0
    mcount db 0
    zcount db 0
    array dw 3,cnt1,cnt2,cnt3
ends

stack segment
    dw   128  dup(0)
ends

code segment
start:
; set segment registers:
    mov ax, data
    mov ds, ax
    mov es, ax

    mov cx, array
    mov si, 2
    lea di, pcount
loop_array:
    mov bx, array[si]
```

```
    add si, 2
    call scan_array
    loop loop_array


    lea dx, pkey
    mov ah, 9
    int 21h        ; output string at ds:dx

    ; wait for any key....
    mov ah, 1
    int 21h

    mov ax, 4c00h ; exit to operating system.
    int 21h

; 输入 bx = 数组的首地址，di = 输出区的首地址[p,m,z]
scan_array:
pushf
push dx
push cx
push ax
push si
    mov cx, [bx]
    mov si, 2
loop_scan:
    mov ax, [bx][si]
    push bx ; bx 暂存
    push di ; di 暂存
    mov bx, di
    cmp ax, 0
    jng scan_branch_ng
    ; > 0
    mov di, 0
    jmp scan_branch_out
scan_branch_ng:
    jl scan_branch_l
    ; = 0
    mov di, 2
    jmp scan_branch_out
scan_branch_l:
    ; < 0
    mov di, 1
scan_branch_out:
    ; [bx][di]对应的输出区
```

```
    mov dx, [bx][di]
    inc dx
    mov [bx][di], dx
    pop di
    pop bx
    add si, 2
    loop loop_scan
pop si
pop ax
pop cx
pop dx
popf
ret
ends

end start ; set entry point and stop the assembler.
```

五、 实验结果分析

实验 1

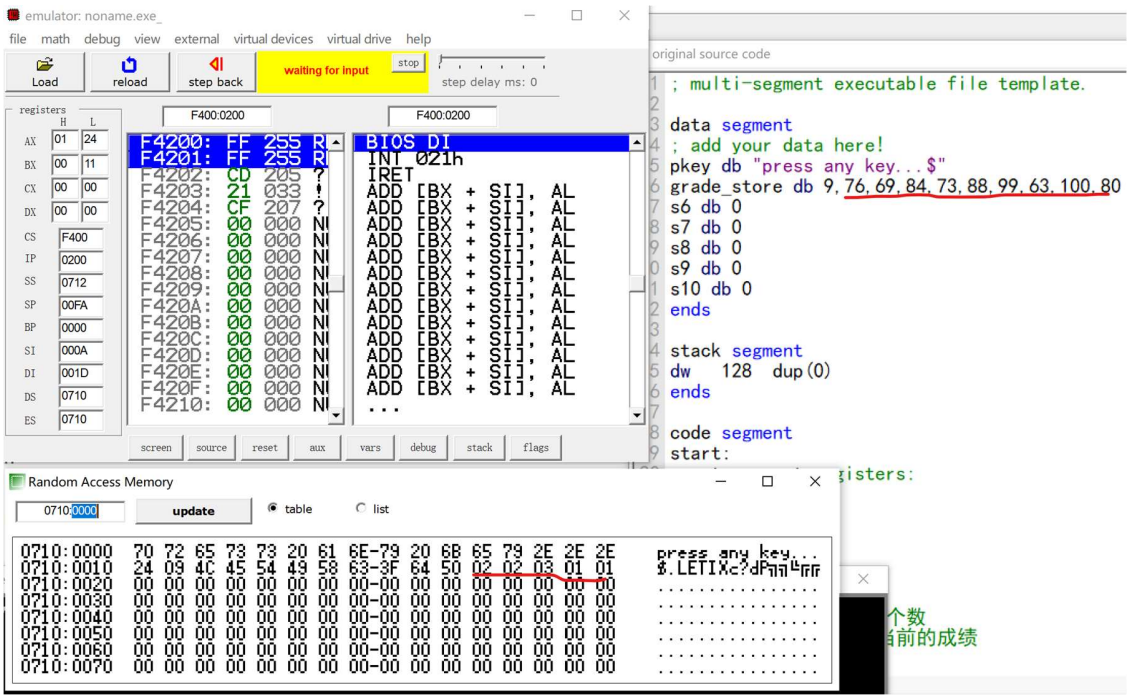

图 8-3 实验 1 统计各个等级的成绩的个数的实验结果

如图所示，结果为(2, 2, 3, 1, 1)，与给出的数据一致。

实验 2

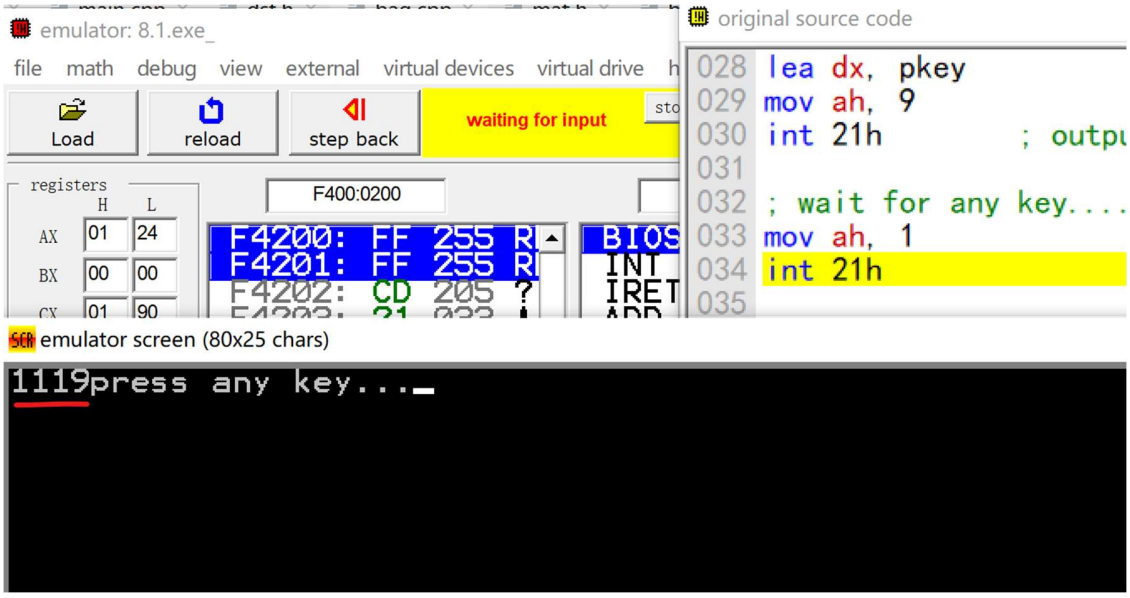

图 8-4 实验 2 的实验结果

实验 3

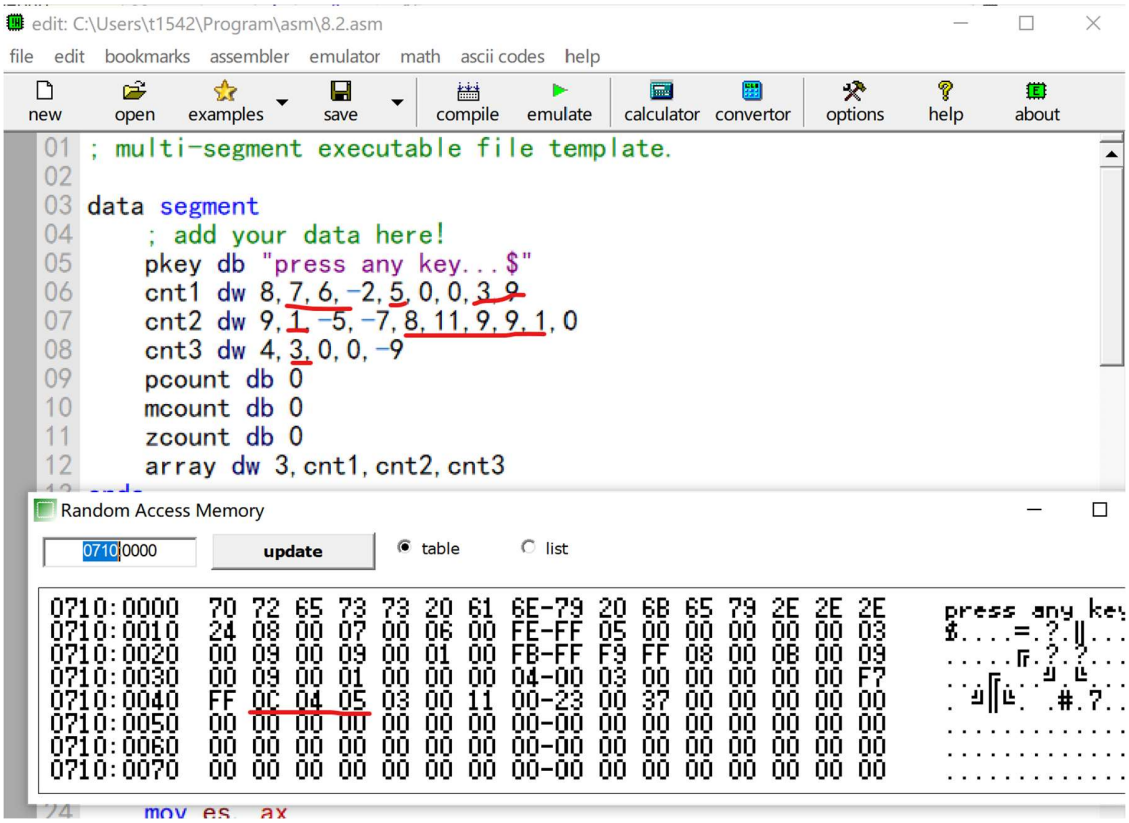

图 8-5 实验 3 的实验结果

如图所示，正数有 12 个，负数有 4 个，零有 5 个，与预期一致。

六、 结果讨论

1、此次实验的重点在于思考如何设计子程序，是否有设计子程序的必要，以及如何设计子程序让子程序的适应性（或者）通用性更强。
2、子程序的传参方式一般借助于寄存器和堆栈，也可以使用地址表来传递，但是因为地址表来传递参数会影响子程序的扩展能力，因此并不是很推荐。
3、在子程序中，经常出现 push 和 pop 的操作来存储中间的数据，注意 push 和 pop 必须是成对且嵌套的，一般来说，如果涉及到 psw 的寄存器的更改也会使用 pushf 和 popf 来存储中间的操作。