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、能够使用 call 和 ret 来实现子模块的设计。能够使用 push 和 pop 来保存中间数据。
3、熟练掌握系统功能的调用，特别是关于输入字符，输出字符，输出字符串的功能。
4、能够熟练掌握数组和串的相关语法。

实验设备

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

二、 实验内容和要求

1.编写程序，将 20 个数据的数组分成两组，正数数组 P 和负数数组 N，并分别显示两个数组的个数。
2.求出首地址为 DATA 的 20 个字数组的最小偶数,并把它存放在 AX 中。

三、 实验步骤

实验 1:

实验内容：编写程序，将 20 个数据的数组分成两组，正数数组 P 和负数数组 N，并分别显示两个数组的个数。
实验思路：从数组的两端开始（设两个指针 i,j），直到 i>j，否则一直执行下列动作：i 向后扫描找到一个负数，j 向前扫描找到一个正数。如果找到满足条件的两个数，则交换两个数，同时 i++,j--，继续重复此动作。这种方式不需要使用辅助数组。最后 i 将会执行第一个负数（即第二部分的开头），通过此可以计算出正数的个数。
本实验中，我们使用 BX 来存储数组的首地址，SI 存储前指针，DI 存储后指针。AL 来存储正数的个数。
此实验将会显示有符号数的（10 进制）模块，此模块在实验 5 中已经给出，在这里不再赘述。
流程图：

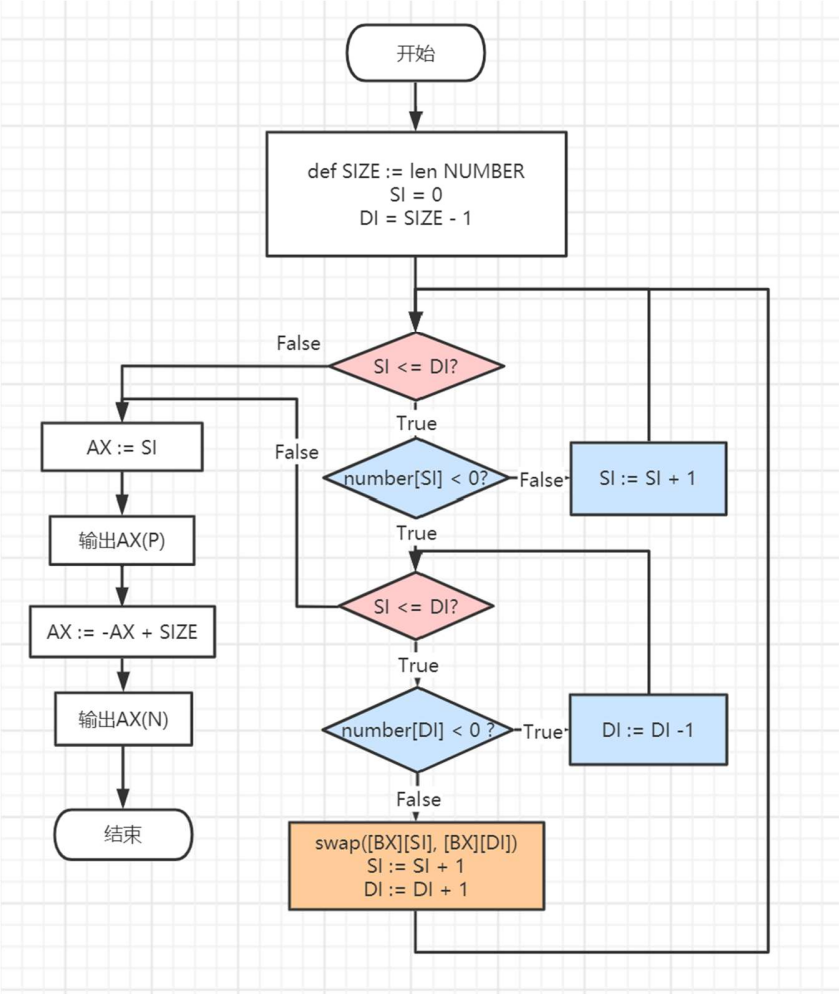

图 6-1 实验 1（划分数组为正数和负数两个部分的实验代码）的流程图

其中程色部分为进行对话的关键代码，红色是判断循环是否结束的标志。蓝色部分是进行迭代寻找的过程。

实验代码：

```
; multi-segment executable file template.

data segment
    ; add your data here!
    pkey db "press any key...$"
    numbers db -4,6,7,-8,12,14,17,-9,-8,20,18,-3,-5,7,27,32,84,-7,-2,6
    number_len equ $-numbers

ends

stack segment
    dw   128  dup(0)
```

```
ends

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

    mov si, 0
    mov di, number_len - 1 ; 分别指向第一个和最后一个元素
judge_1:
    cmp si, di
    ja print_p_count ; si > di 则直接跳到输出
    cmp numbers[si], 0
    jng judge_2 ; numbers[si] <= 0 则进入下一层判断
    inc si
    jmp judge_1
judge_2:
    cmp si, di
    ja print_p_count ; si > di 则直接跳到输出
    cmp numbers[di], 0
    jg do_swap ; numbers[di] > 0 则进入交换
    dec di
    jmp judge_2
do_swap:
    mov al, numbers[si] ; al(1) <- numbers[si]
    push ax
    mov al, numbers[di] ; al(2) <- numbers[di]
    mov numbers[si], al ; numbers[si] <- al(2)
    pop ax
    mov numbers[di], al ; numbers[di] <- al(1)
    inc si
    dec di
    jmp judge_1

print_p_count:
    mov ax, si
    call print_number ; 输出正数的个数
    push ax
    mov dl, 20h
    mov ah, 2
    int 21h ; 输出一个空格
    pop ax
    xor ax, 0ffffh
```

```
    inc ax
    add ax, number_len
    call print_number ; 输出负数的个数

    mov dl, 0dh
    mov ah, 2
    int 21h
    mov dl, 0ah
    mov ah, 2
    int 21h

    mov cx, number_len
    lea si, numbers

loop_print:
    lodsb
    cbw ; 将字节扩展至字
    call print_number_flag ; 输出当前的数字
    mov dl, 20h
    mov ah, 2
    int 21h
    loop loop_print

    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

; 子模块，有符号数以 10 进制显示，输入(AX,数字)，输出(Console)。
print_number_flag:
    push ax
    push dx ; 暂存 dx
    cmp ax, 0
    jge call_print_number

    push ax ; 暂存 ax
    mov dl, 2dh
    mov ah, 2
    int 21h ; 输出一个'-'(0x2d)字符
```

```
    ; 转化为相应的正数
    pop ax ; 恢复 ax
    xor ax, 0ffffh ; 所有位取反
    inc ax ; +1
call_print_number:
    call print_number
    pop dx
    pop ax
    ret

; 子模块，无符号数以 10 进制显示，输入(AX,数字)，输出(Console)。
print_number:
    ; 保存数据
    push ax
    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
    pop ax
    ret
ends

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

实验结果：

实验数据：-4,6,7,-8,12,14,17,-9,-8,20,18,-3,-5,7,27,32,84,-7,-2,6

运行后的数据：6,6,7,84,12,14,17,32,27,20,18,7,-5,-3,-8,-9,-8,-7,-2,-4

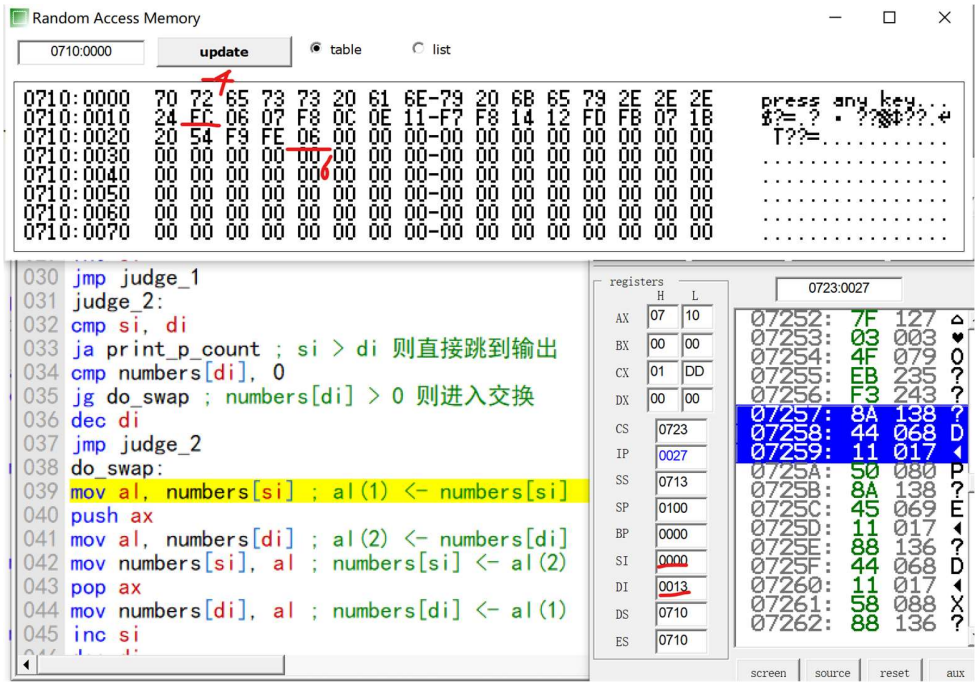

图 6-2 实验 1 第一次交换前的状态

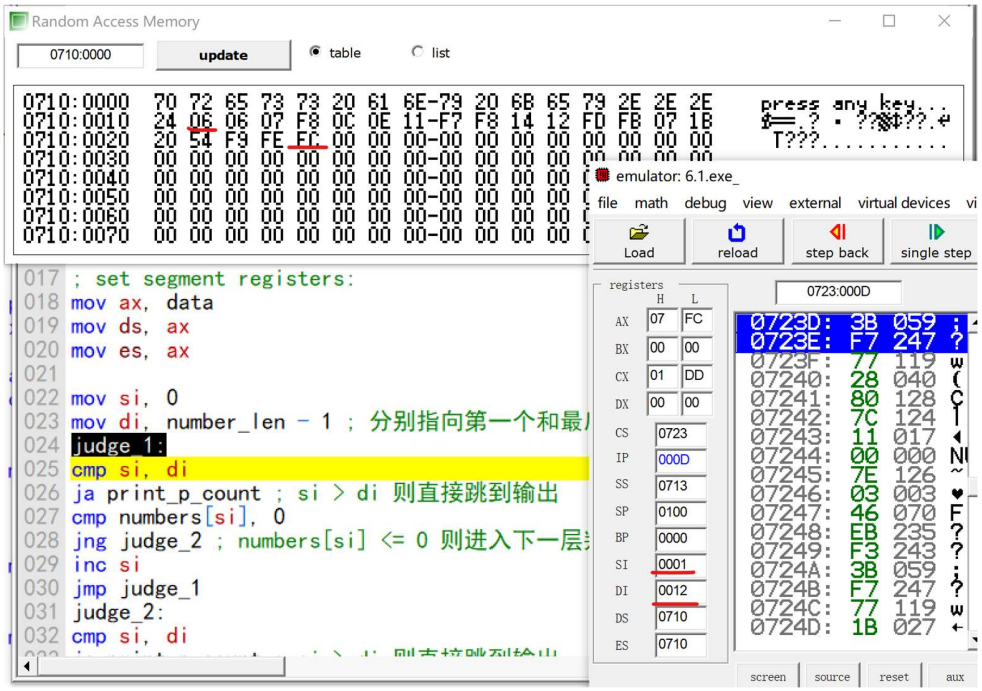

图 6-3 实验 1 第一次交换后的状态

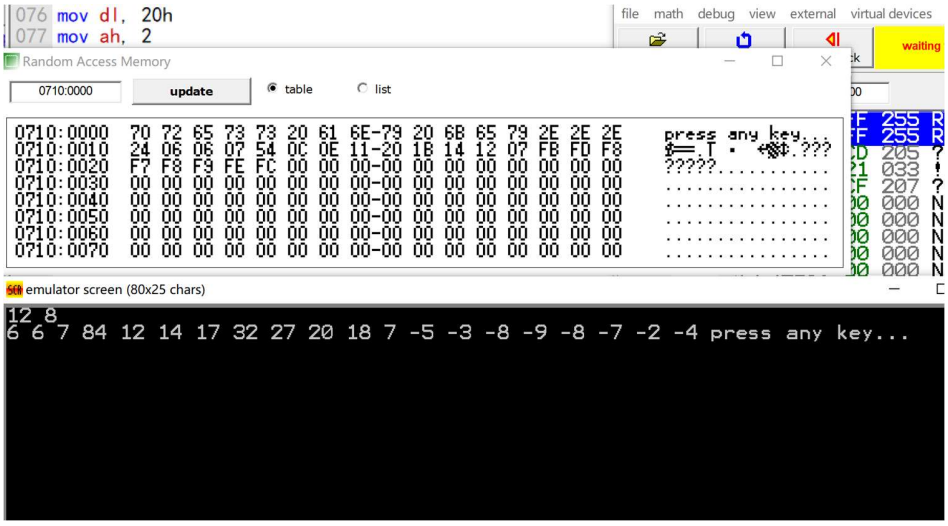

图 6-4 实验 1 运行后的结果，发现内存的空间和输出的结果和预期的一致

实验 2：

实验题目：求出首地址为 DATA 的 20 个字数组的最小偶数（假设为有符号数），并把它存放在 AX 中。

实验思路：这题和实验 5 求最大数字的代码框架大致相同，需要注意的是，因为这里限定了最小偶数这一条件，可以在 AX 中先放一个奇数。循环迭代，如果该数为偶数且满足以下条件，则用概述替代 AX：①AX 为奇数。②AX 为偶数且当前数小于 AX。

流程图：

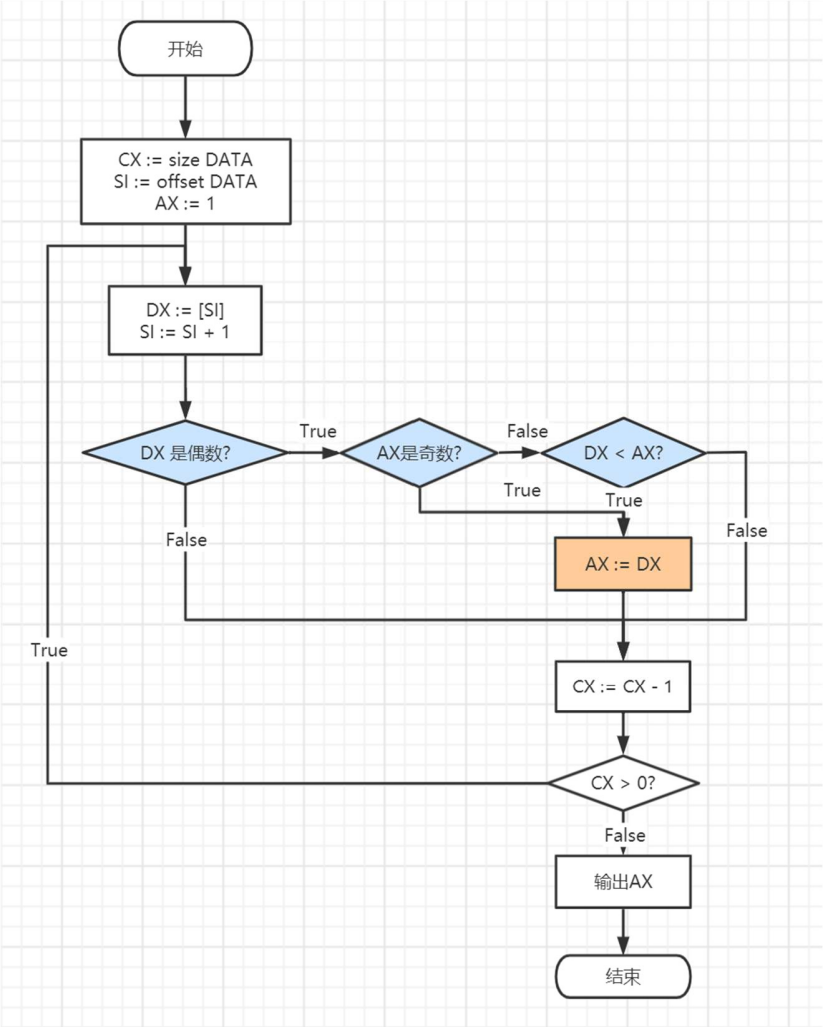

图 6-5 实验 2（求一个数组的最小偶数）的流程图

其中淡蓝色的为关键的判断代码，程色为核心语句。

实验代码：

```
; multi-segment executable file template.

data segment
    ; add your data here!
    pkey db "press any key...$"
    datas dw -4,6,7,-8,12,14,17,-9,-8,20,18,-3,-5,7,27,32,84,-7,-2,6
    data_len equ ($-numbers)/2

ends

stack segment
    dw   128  dup(0)
ends
```

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

    ; 初始化
    lea si, datas
    mov cx, data_len
    mov ax, 1

loop_select:
    mov dx, [si]
    add si, 2 ; 读取一个数到 dx
    test dx, 1
    jnz out_select ; dx 是奇数则跳出当前循环
    test ax, 1
    jnz out_cmp ; ax 是奇数则直接赋值
    cmp dx, ax
    jnl out_select ; dx >= ax 跳出当前循环
out_cmp:
    mov ax, dx
out_select:
    loop loop_select

    call print_number_flag


    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

; 子模块，有符号数以 10 进制显示，输入(AX,数字)，输出(Console)。
print_number_flag:
    ;;; print_number_flag 的代码省略
```

```
ends

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

实验结果：

实验数据：-4,6,7,-8,12,14,17,-9,-8,20,18,-3,-5,7,27,32,84,-7,-2,6

预期结果：-8

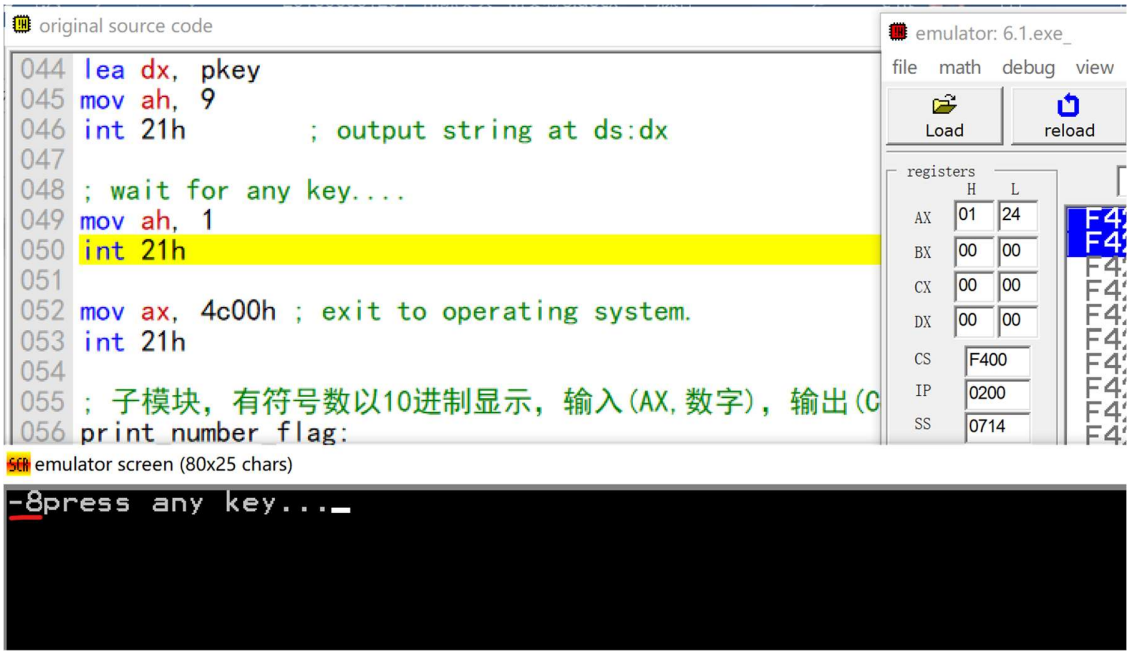

图 6-5 实验 2（求一个数组的最小偶数）的实验结果

四、 实验结果分析

实验结果已经在实验步骤中给出。

五、 结果讨论

1、一般来说，常用的数据类型有 db,dw,dd 类型，其每个元素的宽度分别为 1,2,4。在进行位移操作时，应当特别注意位移的长度，一般使用+-type [name]来提高代码的扩展性，不过由于 emu8086 不支持 type 伪指令，因此此时需要特别注意数据的宽度，否则可能出现到意想不到的情况。

2、来进行数据交换的过程中，既可以使用两个寄存器来存储中间变量，也可以只使用一个寄存器来存储中间变量，不过此时需要借助于堆栈，在实验 1 中，交换[si]和[di]两个元素的代码如下，使用了 push ax,pop ax 来存储中间变量，从而能够实现两个数据的交换。

```
do_swap:
    mov al, numbers[si] ; al(1) <- numbers[si]
    push ax
```

```
    mov al, numbers[di] ; al(2) <- numbers[di]
    mov numbers[si], al ; numbers[si] <- al(2)
    pop ax
    mov numbers[di], al ; numbers[di] <- al(1)
    inc si
    dec di
```