学院：**信息学院**　学期：**2022 年秋季学期**　课程名称：**汇编语言程序设计**　指导教师：**周丽华**

专业：**计算机科学与技术**　年级：**2020**　姓名：**胡诚皓**　学号：**20201060330**

实验六 排序和检索程序设计

一、实验目的

1. 进一步掌握多重循环程序的结构。
2. 进一步掌握多重循环程序的设计、编写及调试方法。

二、实验内容

1. 复习教材中有关多重循环程序设计的相关内容。
2. 复习排序和检索算法。
3. 编写程序 1：将内存中的 10 个无符号数（长度为 2 字节）由大到小排序。
4. 编写程序 2：在上述已排好序的数据区里查找某一个数。若找到，显示‘Y’，否则显示‘N’。

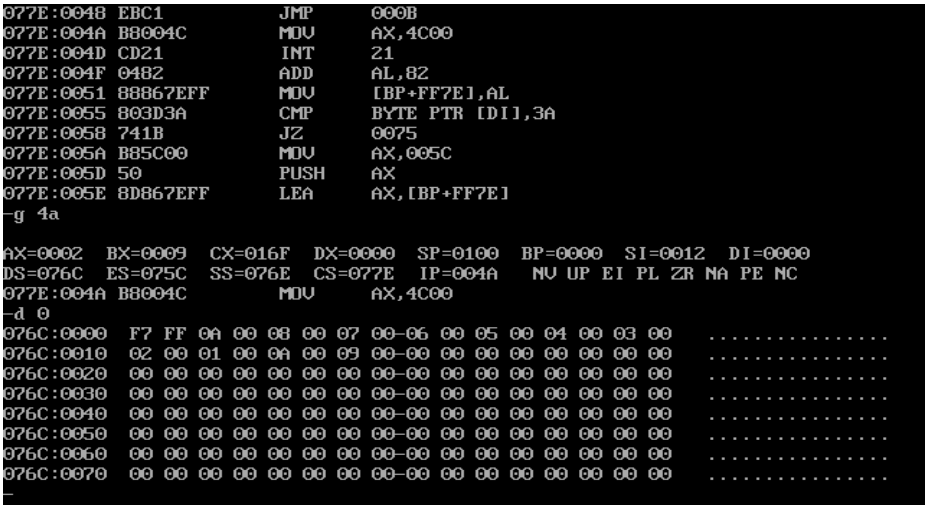
三、实验结果（截图）

1. 程序 1

此程序使用冒泡排序完成无符号数从大到小的排序，直接在原来 10 个数存放的空间中完成排序。为了验证完成的是否为无符号数的排列，特意在待排列的数据中加入了“-9”，它应该是最大的。

待排序序列为“5，10，2，3，4，6，8，7，-9，1”，位于 076C:0000-076C:0013 的内存单元处，共 20 个字节。

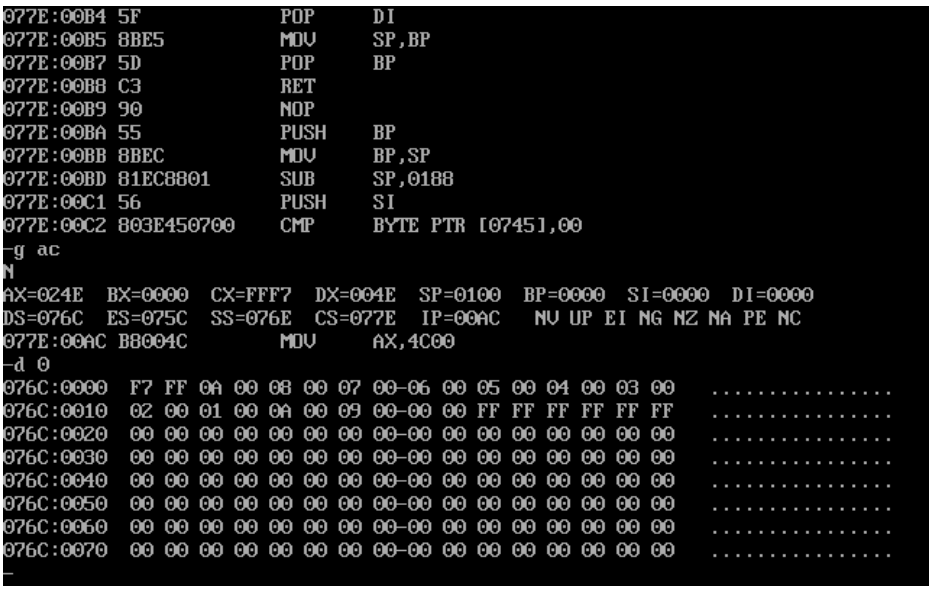
从下图中可以看到排序的结果为“-9，10，8，7，6，5，4，3，2，1”



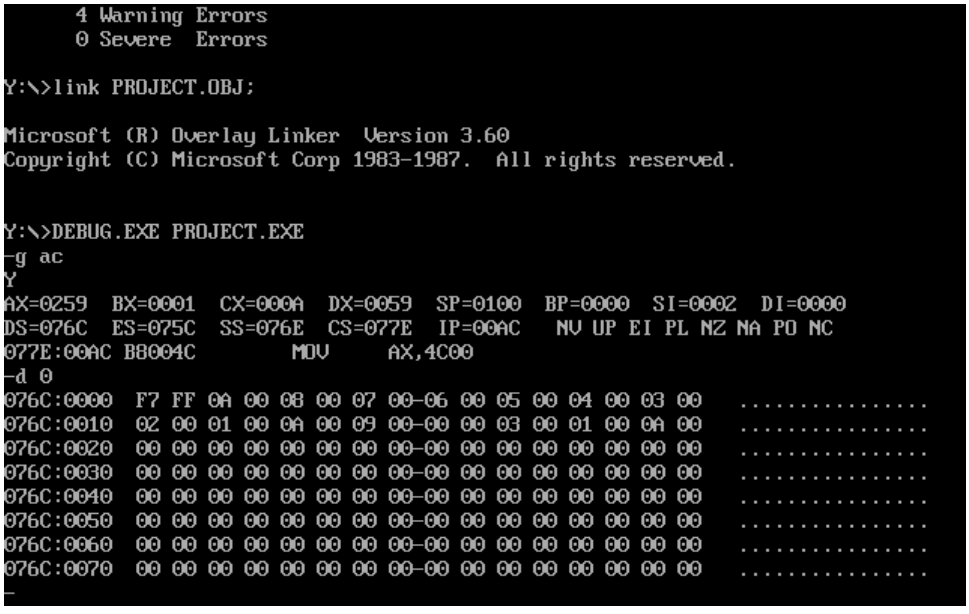
2. 程序 2

先使用与程序 1 相同的代码对序列进行从大到小的排列。之后使用了二分查找的方法进行查找，若查找成功，会将其在数组中的下标存在 index（076C:001C）中并输出 Y；若查找失败，index 中的内容为初始的-1，即 FFFF，并输出 N。

下图为查找序列中不存在的-1 的结果，输出 N，下标记录为 FFFF



下图为查找序列中第二个数 10 的结果，输出 Y，下标记录为 0001



四、实验报告要求（习题）

1. 程序 1——排序

这段汇编代码相当于如下的高级语言代码

```
for (int i = 0; i < 10; i++) {
```

```
    for (int j = 9; j > i; j--) {
           ax = buf[j-1];
        if (ax < buf[j]) {
            buf[j-1] = buf[j];
            buf[j] = ax;
        }
    }
}
```

```
DATA SEGMENT
    ; 待排序序列
    ARRAY DW 5, 10, 2, 3, 4, 6, 8, 7, -9, 1
    ; 两个循环变量
    I DW 0
    J DW 9
DATA ENDS

STACK SEGMENT STACK
    DW 128 DUP(0)
STACK ENDS

CODE SEGMENT
    ASSUME cs:CODE, ds:DATA, ss:STACK
START:
    ; 设置段寄存器
    mov ax, DATA
    mov ds, ax
    ; 设置循环变量 i 的初值
    ; for i=0
    mov I, 0
Iloop:
    ; 外层循环
    ; for i<10
    cmp I, 10
    jae EXIT
    ; 设置循环变量 j 的初值
    ; for j=9
```

```
    mov J, 9
Jloop:
    ; 内层循环
    ; for j>i
    mov bx, I
    cmp J, bx
    jbe Iinc
    ; ax = array[j-1]
    mov si, J
    shl si, 1
    mov ax, ARRAY[si-2]
    ; 判断是否要交换相邻元素
    ; if (ax < array[j])
    cmp ax, ARRAY[si]
    jae Jdec
    ; 交换 array[j-1]和 array[j]
    mov bx, ARRAY[si]
    mov ARRAY[si-2], bx
    mov ARRAY[si], ax
Jdec:
    ; j 循环一遍后自减 1
    ; for j--
    dec J
    jmp Jloop
Iinc:
    ; i 循环一遍后自增 1
    ; for i++
    inc I
    jmp Iloop
EXIT:  ; 退出代码
    mov ax, 4C00H
    int 21H
CODE ENDS
    END START
```

2. 程序 2——查找目标数

相当于如下的高级语言代码

```
int mid, left, right, tmp, index;
```

```
left = 0, right = 9, index = -1;

// 排序代码同上，不再重复
while (left <= right) {
    mid = (left+right)/2;
    tmp = buf[mid];
    if (target == tmp) {
        index = mid;
        break;
    } else if (target < tmp) {
        left = mid + 1;
    } else {
        right = mid - 1;
    }
}
if (index >= 0) {
    printf("Y");
} else {
    printf("N");
}
```

```
DATA SEGMENT
    ARRAY DW 5, 10, 2, 3, 4, 6, 8, 7, -9, 1
    ; 两个循环变量
    I DW 0
    J DW 9
    ; 用于二分查找时的左右标记
    LEFT DW 0
    RIGHT DW 9
    ; 若找到了目标数，将其数组下标存在 index 中
    INDEX DW -1
    ; 待查找的目标数
    TARGET DW -1
DATA ENDS

STACK SEGMENT STACK
    DW 128 DUP(0)
```

```
STACK ENDS

CODE SEGMENT
    ASSUME cs:CODE, ds:DATA, ss:STACK
START:
    ; 设置段寄存器
    mov ax, DATA
    mov ds, ax
; 和程序 1 一样的代码，对序列进行排序
    ; for i=0
    mov I, 0
Iloop:
    ; for i<10
    cmp I, 10
    jae FIND
    ; for j=9
    mov J, 9
Jloop:
    ; for j>i
    mov bx, I
    cmp J, bx
    jbe Iinc
    ; ax = array[j-1]
    mov si, J
    shl si, 1
    mov ax, ARRAY[si-2]
    ; if (ax < array[j])
    cmp ax, ARRAY[si]
    jae Jdec
    ; 交换 array[j-1]和 array[j]
    mov bx, ARRAY[si]
    mov ARRAY[si-2], bx
    mov ARRAY[si], ax
Jdec:
    ; for j--
    dec J
    jmp Jloop
```

```
Iinc:
    ; for i++
    inc I
    jmp Iloop

; 二分查找从这里开始
FIND:
    ; 初始化左右标志（下标）
    mov LEFT, 0
    mov RIGHT, 9
    ; cx 中存放中间位置值
    mov cx, 0
AGAIN:
    ; while 循环体
    ; while (left <= right)
    mov bx, LEFT
    cmp bx, RIGHT
    jg END
    ; 计算中间位置（下标）
    ; bx=(left+right)/2
    mov bx, LEFT
    add bx, RIGHT
    shr bx, 1
    ; 乘 2 得到字节位置
    mov si, bx
    shl si, 1
    ; 得到中间位置的值
    ; cx=array[bx]
    mov cx, ARRAY[si]
EQ: ; 找到了目标值
    ; if (target==cx)
    cmp TARGET, cx
    jne LT
    ; 把下标记录在 INDEX 中
    mov INDEX, bx
    ; break
    jmp END
```

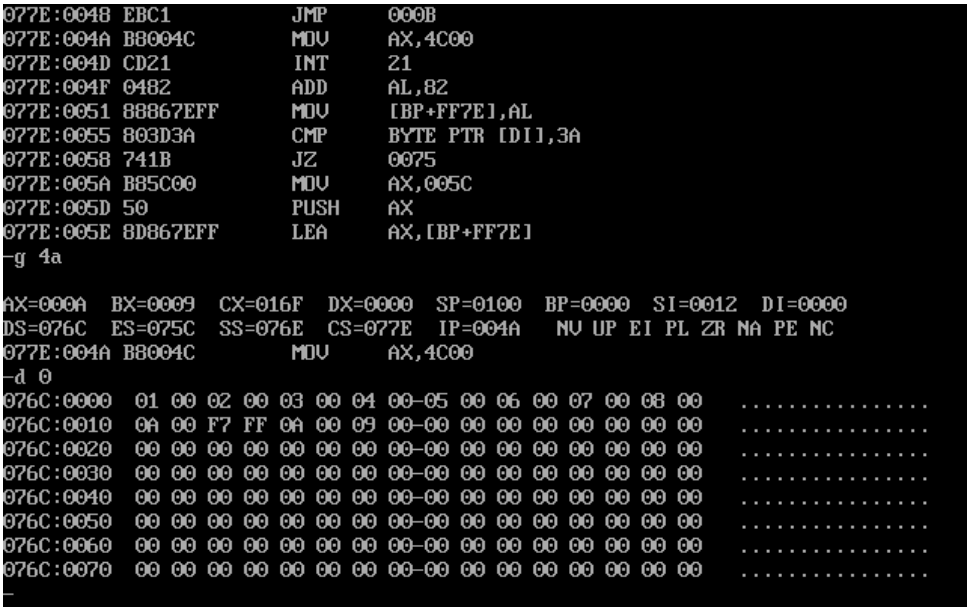
```
LT: ; 目标值比中间值小，向右找
    ; else if (target<cx)
    cmp TARGET, cx
    jae GT
    ; left=bx+1
    inc bx
    mov LEFT, bx
    jmp AGAIN
GT: ; 目标值比中间值大，向左找
    ; else
    ; right=bx-1
    dec bx
    mov RIGHT, bx
    jmp AGAIN

END: ; 判断是否找到了，即查看 INDEX 是否为负
    mov ax, INDEX
    AND ax, ax
    js NO
YES: ; 找到了
    mov dl, 'Y'
    mov ah, 2
    int 21H
    jmp EXIT
NO: ; 没找到
    mov dl, 'N'
    mov ah, 2
    int 21H
EXIT: ; 退出代码
    mov ax, 4C00H
    int 21H
CODE ENDS
END START
```

3. 修改程序 1 实现从小到大排序

事实上要将从大到小排序改为从小到大，只需要将代码中所有判断部分反向即可。汇编代码中就是将用于判断相邻数大小的条件跳转指令替换为含义相反的

指令。具体的来说，就是将程序 1 中的“jae Jdec”改为“jbe Jdec”即可。修改完后重新汇编链接得到如下的运行结果，确实是从小到大排列的。



4. 遇到的问题及如何解决

本次实验用汇编完成了最基本的排序和查找算法，整体编写过程与上个实验类似，先用高级程序语言编写，之后再将其“翻译”为汇编语言。在实验过程中，由于需要处理的是 2 字节的无符号数，需要尤其注意偏移地址的处理，数组中的偏移地址相当于是下标*2，左移一位即可。

五、个人体会与总结

通过本次实验，发现了有符号数和无符号数的比较处理的不同之处。无论是有符号数还是无符号数，“比较”本身都是由 CMP 指令完成的，只不过需要区分有符号数和无符号数两种情况下，判断大小使用到的标志位情况不同。总结如下表所示。

无符号数比较

大小关系	ZF	CF
目的操作数<源操作数	0	1
目的操作数>源操作数	0	0
目的操作数=源操作数	1	0

有符号数比较

大小关系	标志位情况
目的操作数<源操作数	SF≠OF
目的操作数>源操作数	SF=OF

目的操作数=源操作数	ZF=1

对于无符号数，使用的条件跳转指令有 je、jne、ja、jb、jae、jbe，分别表示等于、不等于、大于、小于、大于等于、小于等于；对于有符号数，使用的条件跳转指令有 je、jne、jg、jl、jge、jle，分别表示等于、不等于、大于、小于、大于等于、小于等于。