

廈門大學



《汇编语言》实验报告

(六)

姓 名 宋浩元

学 号 37220232203808

学 院 信息学院

专 业 软件工程

2024 年 11 月

1 实验目的

- (1) 熟悉 16 位机 8086 的串操作指令；
- (2) 掌握子程序的程序设计方法，体会不同参数传递方式的区别；
- (3) 进一步熟练运用各种程序设计基本结构。

2 实验环境

Windows11 环境下的 masm 与 DOSBOX；

3 实验内容

- (1) 下述代码段是计算 $n!$

$$n! = \begin{cases} n * (n - 1)! & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$$

子程序代码段采用了递归和嵌套结构， n 存放在 AL 中， $n!$ 存放在 BX 中。请结合给出的部分代码完成程序，并绘制出程序调用示意图和堆栈变化示意图。

```

01
02 ;主程序
03 MAIN: MOV AX,3 ; 设n=3
04 CALL FACT
05 XI: MOV BX,DX
06 HLT
07
08 ;阶乘子程序
09 ;入口参数: AL中存放n
10 ;出口参数: DX中存放n!
11 ;所用寄存器: CX
12
13 FACT PROC
14 CMP AL,0
15 JNE IIA
16 MOV DL,1
17 RET ;(1)
18 IIA: PUSH AX
19 DEC AL
20 CALL FACT
21 X2: POP CX
22 CALL MULT
23 X3: MOV DX,AX
24 RET ;(2)
25 FACT ENDP
26 ;无符号字节数乘法子程序
27 ;入口参数: CL, DL中各为一乘数
28 ;出口参数: AX中为乘积
29
30 MULT PROC
31 MOV AL,DL
32 MUL CL
33 RET ;(3)
34 MULT ENDP
35

```

(2) 设有一个数组存放学生成绩 (0~100)，编写一个子程序，统计 0~59 分，60~69 分，70~79 分，80~89 分，90~100 分的人数，并分别存放在 scoreE、scoreD、scoreC、scoreB 以及 scoreA 单元中。
要求：补充主程序，数值初始值自定义用于验证子程序，参数传递方式自选。

(3) 有 10 个字节的数据表 array，表内元素已按从小到大的顺序排列好。现给定一元素 w，试编制子程序，实现在表内查找给定元素 w 的任务，若表内已有此元素，则显示“Y”；否则，按顺序将此元素插入表中适当的位置。

- 1) 数据表需要自行进行初始化，数值或字符类型均可；
- 2) 主程序和子程序之间采用变量方式进行传参。
- 3) 若主程序和子程序采用堆栈方式进行传参，原始代码将如何修改？

(4) 编写程序, 判断主存 DS:0 开始的缓冲区 buffer 中是否有字符串 “DEBUG” 若没有，显示 “no finding!”，若有，则返回首字母 D 的下标。(注: 需针对不同情况，初始化 buffer 内的字符串。如: 没有，有 1 个或者有多个的情况不强制要求，但若完成可加 10 分)

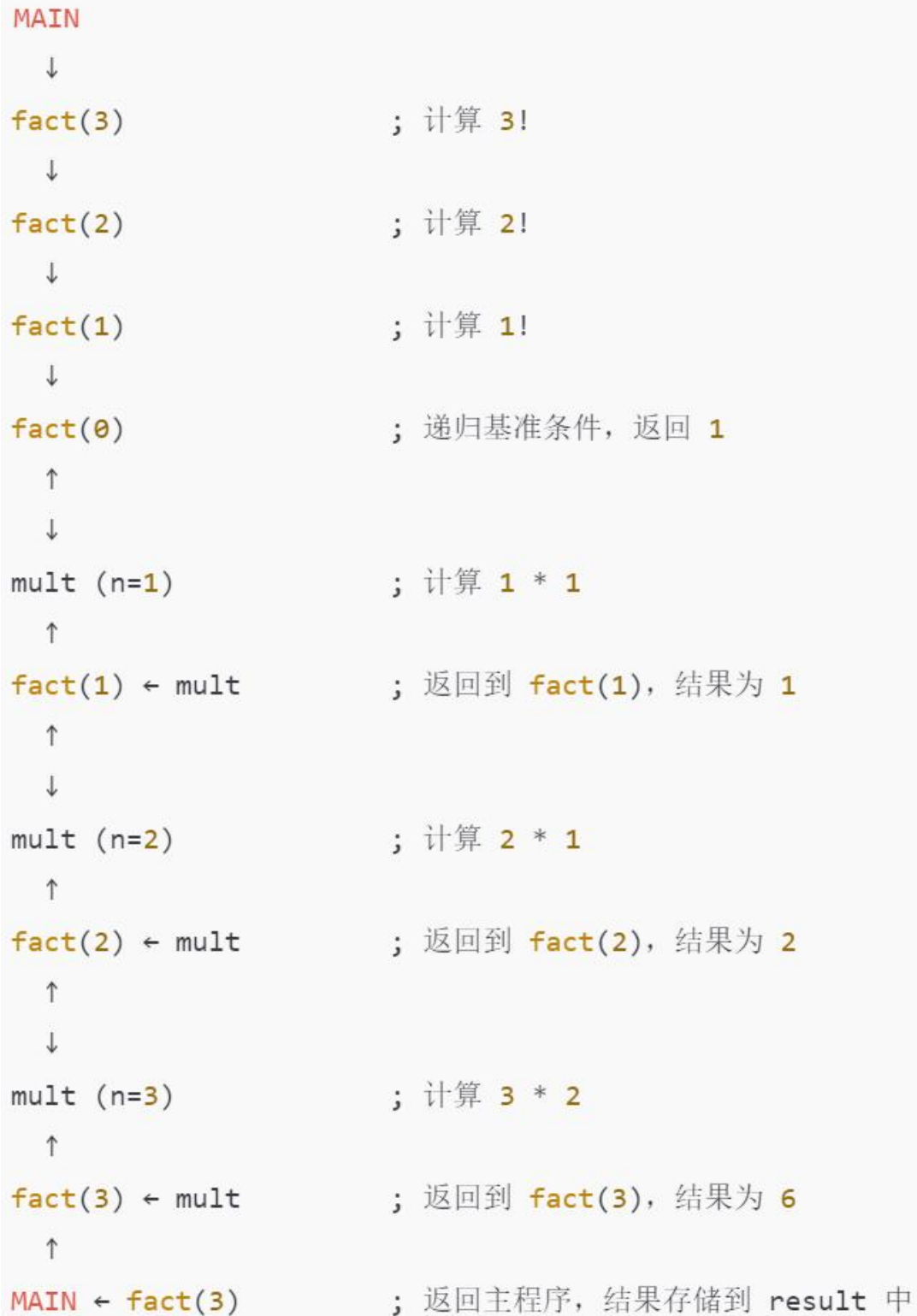
(5) 编写一个递归子程序，计算指数函数 x^n 的值。(仿照例 4.17)

4 实验具体实现

(1) 汇编程序如下

```
1  data segment
2      org 3000h
3      result dw 0000h
4      org 3100h
5      n db 3
6  data ends
7
8  code segment
9      assume cs:code
10 MAIN:
11     mov ax,data
12     mov ds,ax
13
14     mov si,3100h
15     mov ax,[si]
16     call fact
17     mov si,offset result
18     mov [si],dx
19
20     mov ax,4c00h
21     int 21h
22 ;阶乘子程序
23 ;入口参数:  al中存放n
24 ;出口参数:  bx中存放n!
25 fact proc near
26     cmp ax,0
27     jne IIA
28     mov bx,1
29     ret
30 IIA:
31     push ax
32     dec ax
33     call fact
34 X2:
35     pop cx
36     call mult
37 X3:
38     mov bx,ax
39     ret
40 fact endp
41 ;无符号字节数乘法子程序
42 ;入口参数: cx,bx中的各为一个乘数
43 ;出口参数: ax中的为乘积
44 mult proc near
45     mov ax,bx
46     mul cx
47     ret
48 mult endp
49 code ends
50 end MAIN
51
52
```

程序调用示意图：



堆栈变化示意图：

初始堆栈：空

调用 fact(3) → 堆栈：[3]	； 3 被压入堆栈，AX = 3
调用 fact(2) → 堆栈：[3, 2]	； 2 被压入堆栈，AX = 2
调用 fact(1) → 堆栈：[3, 2, 1]	； 1 被压入堆栈，AX = 1
调用 fact(0) → 堆栈：[3, 2, 1, 0]	； 0 被压入堆栈，AX = 0
返回 fact(0) → 恢复 AX = 0, BX = 1	； 恢复 AX = 0, 返回 1
恢复 fact(1) → 堆栈：[3, 2, 1]	； 恢复 AX = 1
调用 mult → 执行乘法，BX = 1 * 1 = 1	； BX 中存储结果
返回 fact(1) → 堆栈：[3, 2]	； 恢复 AX = 1, 返回 BX = 1
恢复 fact(2) → 堆栈：[3, 2]	； 恢复 AX = 2
调用 mult → 执行乘法，BX = 2 * 1 = 2	； BX 中存储结果
返回 fact(2) → 堆栈：[3]	； 恢复 AX = 2, 返回 BX = 2
恢复 fact(3) → 堆栈：[3]	； 恢复 AX = 3
调用 mult → 执行乘法，BX = 3 * 2 = 6	； BX 中存储结果
返回 fact(3) → 堆栈：空	； 返回结果 6

(2)

```

1  data segment
2      ; 定义数组存放学生成绩
3      scores db 95, 67, 85, 45, 74, 88, 56, 92, 67, 99
4      ; 定义存放统计结果的单元
5      scoreA db 0      ; 90~100分
6      scoreB db 0      ; 80~89分
7      scoreC db 0      ; 70~79分
8      scoreD db 0      ; 60~69分
9      scoreE db 0      ; 0~59分
10 data ends
11
12 code segment
13     assume cs:code, ds:data
14 start:
15     ; 初始化数据段
16     mov ax, data
17     mov ds, ax
18
19     ; 初始化计数器为0
20     mov byte ptr scoreA, 0
21     mov byte ptr scoreB, 0
22     mov byte ptr scoreC, 0
23     mov byte ptr scoreD, 0
24     mov byte ptr scoreE, 0
25
26     ; 设置数组指针指向成绩数组的起始位置
27     lea si, scores
28
29     ; 设置数组长度 (在本例中为10)
30     mov cx, 10
31
32     ; 调用统计子程序
33     call countScores
34
35     ; 退出程序
36     mov ax, 4C00h
37     int 21h
38
39 ; 统计成绩分布的子程序
40 ; 入口参数: 学生成绩数组的地址, 长度 (cx)
41 ; 入口: si - 数组指针, cx - 数组长度
42 countScores proc near
43     ; 遍历数组中的成绩
44     s:
45         ; 读取当前成绩
46         mov al, [si]
47
48         ; 判断成绩的区间
49         ; 0~59分区
50         cmp al, 60
51         jl scoreE_inc
52         ; 60~69分区
53         cmp al, 70
54         jl scoreD_inc
55         ; 70~79分区
56         cmp al, 80
57         jl scoreC_inc
58         ; 80~89分区
59         cmp al, 90
60         jl scoreB_inc
61         ; 90~100分区
62         scoreA_inc:
63             inc byte ptr scoreA
64             jmp next
65
66         scoreB_inc:
67             inc byte ptr scoreB
68             jmp next
69
70         scoreC_inc:
71             inc byte ptr scoreC
72             jmp next
73
74         scoreD_inc:
75             inc byte ptr scoreD
76             jmp next
77
78         scoreE_inc:
79             inc byte ptr scoreE
80
81     next:
82         ; 移动到下一个成绩
83         inc si
84         loop s
85     ret
86 countScores endp
87
88 code ends
89 end start
90

```

运行结果为：3 2 1 2 2 与预期相符。

```
-t
AX=0763 BX=0000 CX=0001 DX=0000 SP=FFFE BP=0000 SI=000A DI=0000
DS=0770 ES=0760 SS=076F CS=0771 IP=005E  NU UP EI PL NZ NA PE NC
0771:005E E2CD          LOOP    002D
-t
AX=0763 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=000A DI=0000
DS=0770 ES=0760 SS=076F CS=0771 IP=0060  NU UP EI PL NZ NA PE NC
0771:0060 C3          RET
-t
AX=0763 BX=0000 CX=0000 DX=0000 SP=0000 BP=0000 SI=000A DI=0000
DS=0770 ES=0760 SS=076F CS=0771 IP=0028  NU UP EI PL NZ NA PE NC
0771:0028 B8004C      MOV     AX,4C00
-d 0000
0770:0000 5F 43 55 2D 4A 58 38 5C-43 63 03 02 01 02 02 00  _CU-JXB\Cc.....
```

(3)

变量方式进行传参


```

1  DATAS SEGMENT
2      W DB 14H
3      ARRAY DB 1h,2h,3h,5h,6h,8h,9h,10h,15h,20h
4  DATAS ENDS
5
6  STACK SEGMENT
7
8  STACK ENDS
9
10 CODES SEGMENT
11     ASSUME CS:CODES,DS:DATAS,SS:STACK
12 START:
13 MAIN:
14     MOV AX,DATAS
15     MOV DS,AX
16     CALL FACT
17     MOV AH,4CH
18     INT 21H
19 FACT PROC
20     MOV CX,10;设置循环次数
21     MOV BX,0
22     MOV AX,-1
23 Looop:
24     INC AX
25     MOV SI,AX
26     MOV BL,[ARRAY+SI]
27     CMP BL,W
28     JE case1;找到跳转
29     JG case2;大于说明不存在
30     LOOP Looop
31     JMP case2
32 case1:
33     MOV AH,02
34     MOV DL,'Y'
35     INT 21H
36     RET
37 case2:
38     push AX
39     ADD AX, CX
40 lop:;通过循环将数组元素后挪
41     SUB AX,1h
42     MOV SI,AX
43     MOV BL,[ARRAY+SI]
44     MOV [ARRAY+SI+1],BL
45     LOOP lop
46     MOV BL,W
47     pop SI
48     MOV [ARRAY+SI],BL;存入
49     RET
50 FACT ENDP
51 CODES ENDS
52     END START
53
54
55
56
57

```

成功查找

```
01 DATAS SEGMENT
02     W DB 1H
03     ARRAY DB 1h,2h,3h,5h,6h,8h,9h,10h,15h,20h
04 DATAS ENDS
05
06 STACK SEGMENT
07     DOSBox 0.74, Cpu speed: 3000 cycles, Frames...
08 STACY
09
10 CODE
11     Press any key to continue
12 STAR
```

成功插入

```
DATAS SEGMENT
W DB 14H
ARRAY DB 1h,2h,3h,5h,6h,8h,9h,10h,15h,20h
DATAS ENDS

STACK SEGMENT

STACK ENDS

CODES SEGMENT
    ASSUME CS:CODES,DS:DATAS,SS:STACK
START:
MAIN:
    MOV AX,DATAS
    MOV DS,AX
    CALL FACT
    MOV AH,4CH
    INT 21H
FACT PROC
    MOV CX,10;设置循环次数
    MOV BX,0
    MOV AX,-1
```

```
AX=0000 BX=0014 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0770 ES=0760 SS=076F CS=0771 IP=0046 NU UP EI PL NZ NA PO NC
0771:0046 8B9C0100 MOV [SI+0001],BL DS:0009=
-t
AX=0000 BX=0014 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0770 ES=0760 SS=076F CS=0771 IP=004A NU UP EI PL NZ NA PO NC
0771:004A C3 RET
-t
AX=0000 BX=0014 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0770 ES=0760 SS=076F CS=0771 IP=0000 NU UP EI PL NZ NA PO NC
0771:0000 B44C MOV AH,4C
-t
0770:0000 14 01 02 03 05 06 00 09-10 14 15 20 00 00 00 00 .....
0770:0010 B8 70 07 8E D8 E8 04 00-B4 4C CD 21 B9 0A 00 BB .p.....L.!..
0770:0020 00 00 BB FF FF 40 BB F0-BA 9C 01 00 3A 1E 00 00 .....@.....:..
0770:0030 74 06 7F 0B E2 EF EB 07-B4 02 B2 59 CD 21 C3 50 t.....Y.!...
0770:0040 03 C1 83 E8 01 8B F0 8A-9C 01 00 88 9C 02 00 E2 .....^.....
0770:0050 F1 8A 1E 00 00 5E 88 9C-01 00 C3 00 00 00 00 00 .....
0770:0060 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
0770:0070 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
```

堆栈方式进行传参

```

1  .model small
2  .stack
3  .data
4      array db 2,4,6,8,10,12,14,16,18,20
5      target db 4
6  .code
7  .startup
8
9  pop ax
10 pop bx
11 pop cx
12 pop dx
13 pop si
14 call insert
15 pop si
16 pop dx
17 pop cx
18 pop bx
19 pop ax
20 .exit 0
21
22 ;使用变量传参
23 insert proc
24     mov cx,lengthof array
25     mov dl,target
26     mov bx,offset array
27     mov si,0
28     .while si<cx
29         mov al,[bx+si]
30         .if al==dl ;找到对应元素，直接打印'Y'返回
31             mov dl,'Y'
32             mov ah,2
33             int 21h
34             mov dl,10
35             mov ah,2
36             int 21h
37             ret
38         .endif
39         .if al>dl ;元素大于target，直接插入
40             .while si<=cx
41                 xchg dl,[bx+si]
42                 inc si
43             .endw
44             ret
45         .endif
46         ;当前元素小于target，查看下一个元素
47         inc si
48     .endw
49     .if si==cx ;若target大于所有数组中的数
50         mov [bx+si],dl ;将target放在末尾
51         ret
52     .endif
53 insert endp
54 end
55
56
57

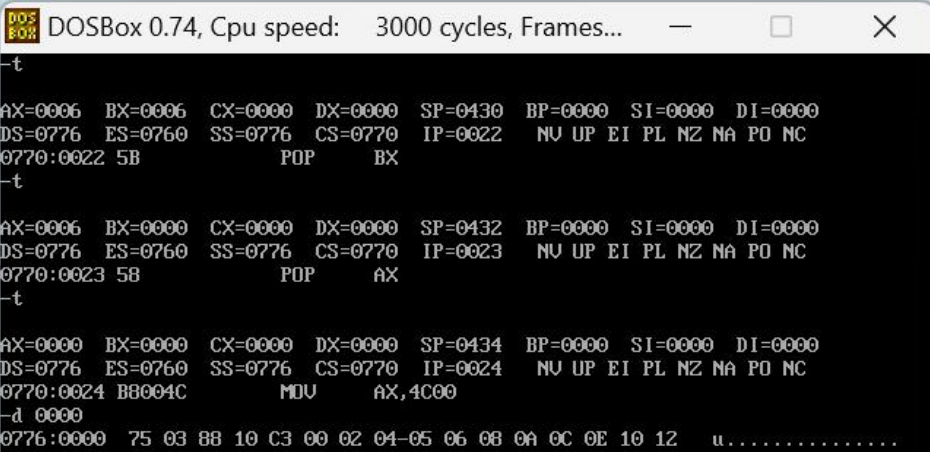
```

成功插入

```

1 .model small
2 .stack
3 .data
4     array db 2,4,6,8,10,12,14,16,18,20
5     target db 5
6 .code
7 .startup
8
9 pop ax
10 pop bx
11 pop cx
12 pop dx
13 pop si
14 call insert
15 pop si
16 pop dx
17 pop cx
18 pop bx
19 pop ax

```

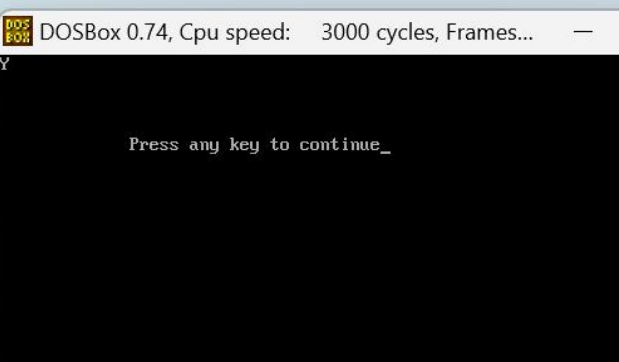


成功查找

```

01 .model small
02 .stack
03 .data
04     array db 2,4,6,8,10,12,14,16,18,20
05     target db 4
06 .code
07 .startup
08
09 pop ax
10 pop bx
11 pop cx
12 pop dx
13 pop si
14 call insert
15 pop si
16 pop dx
17 pop cx
18 pop bx
19 pop ax

```



(4)



```
1  DATAS SEGMENT
2      BUFFER DB 'DEBUGDEBUGDE'
3      STRING1 DB 'NO FINDING!','$'
4      STRING2 DW 0000H,'$'
5  DATAS ENDS
6
7  STACK SEGMENT
8
9  STACK ENDS
10
11 CODES SEGMENT
12     ASSUME CS:CODES,DS:DATAS,SS:STACK
13 START:
14 MAIN:
15     MOV AX,DATAS
16     MOV DS,AX
17     MOV CX,12;循环次数为buffer长度
18     MOV SI,-1
19 loop:
20     MOV AX,0
21     INC SI;每次前进1字节
22     CALL FACT
23     CMP AX,1; 存在跳转
24     JE NEXT
25     LOOP loop
26     MOV AH,09H;不存在输出字符串
27     MOV DX,OFFSET STRING1
28     INT 21H
29     JMP OUT1
30 NEXT:
31     MOV AX,SI
32     MOV STRING2,AX;存入'D'的下标
33     MOV AH,09H
34     MOV DX,OFFSET STRING2
35     INT 21H
36 OUT1:
37     MOV AH,4CH
38     INT 21H
39 FACT PROC
40     MOV BL,[BUFFER+SI];检查字符是否相同
41     CMP BL,'D';相同继续检查下一个，不同退出
42     JNE case
43     MOV BL,[BUFFER+SI+1];同上
44     CMP BL,'E'
45     JNE case
46     MOV BL,[BUFFER+SI+2];同上
47     CMP BL,'B'
48     JNE case
49     MOV BL,[BUFFER+SI+3];同上
50     CMP BL,'U'
51     JNE case
52     MOV BL,[BUFFER+SI+4];同上
53     CMP BL,'G'
54     JNE case
55     MOV AX,1;无中途退出，存在
56     RET
57 case:
58     MOV AX,0
59     RET
60 FACT ENDP
61 CODES ENDS
62     END START
63
64
```

```
01 DATAS SEGMENT
02     BUFFER DB 'abbabbabbabb'
03     STRING1 DB 'NO FINDING!','$'
04     STRING2 DW 0000H,'$'
05 DATAS ENDS
06
07 STACK SEGMENT
08
09 STACK ENDS
10
11 CODES SEGMENT
12     ASSUME CS:CODES,DS:DATAS,SS:STACK
13 START:
14 MAIN:
15     MOV AX,DATAS
16     MOV DS,AX
```

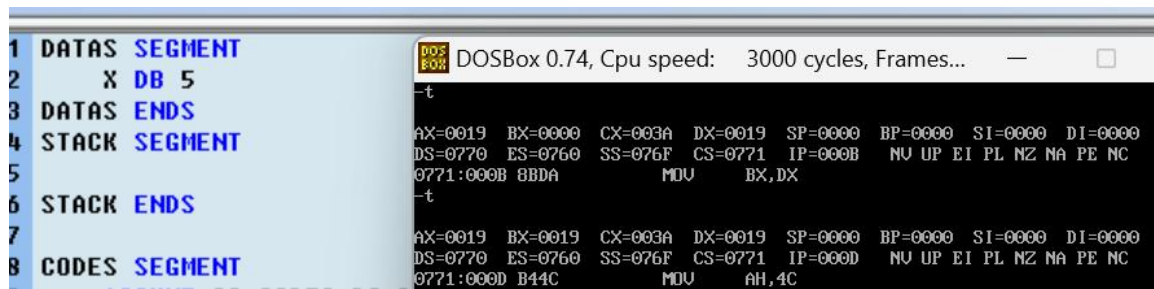


(5)



```
1  DATAS SEGMENT
2      X DB 5
3  DATAS ENDS
4  STACK SEGMENT
5
6  STACK ENDS
7
8  CODES SEGMENT
9      ASSUME CS:CODES,DS:DATAS,SS:STACK
10 START:
11 MAIN:
12     MOV AX,DATAS
13     MOV DS,AX
14     MOV AX,2; 计算5的平方
15     CALL FACT
16     MOV BX,DX
17
18     MOV AH,4CH
19     INT 21H
20 FACT PROC
21     CMP AL,0; AL来计数
22     JNE IIA
23     MOV DL,1
24     RET
25 IIA:
26     DEC AL
27     CALL FACT
28 X2:
29     CALL MULT
30 X3:
31     MOV DX,AX
32     RET
33 FACT ENDP
34
35 MULT PROC
36     MOV AL,DL; 乘X
37     MUL X
38     RET
39 MULT ENDP
40 CODES ENDS
41     END START
42
```

计算 5 的平方为 19h



The screenshot shows a DOSBox window with a title bar "DOSBox 0.74, Cpu speed: 3000 cycles, Frames...". The left pane displays assembly code with line numbers 1 through 8. The right pane shows the current state of the 8086 registers and flags.

```
1 DATAS SEGMENT
2     X DB 5
3 DATAS ENDS
4 STACK SEGMENT
5
6 STACK ENDS
7
8 CODES SEGMENT
```

Registers and flags display:

```
-t
AX=0019 BX=0000 CX=003A DX=0019 SP=0000 BP=0000 SI=0000 DI=0000
DS=0770 ES=0760 SS=076F CS=0771 IP=000B  NU UP EI PL NZ NA PE NC
0771:000B 8BDA      MOV     BX,DX
-t
AX=0019 BX=0019 CX=003A DX=0019 SP=0000 BP=0000 SI=0000 DI=0000
DS=0770 ES=0760 SS=076F CS=0771 IP=000D  NU UP EI PL NZ NA PE NC
0771:000D B44C      MOV     AH,4C
```

5 实验分析与总结

- (1) 对子程序及其书写更熟悉
- (2) 掌握基本汇编递归写法
- (3) 能够掌握基本的汇编程序写法
- (4) 熟悉 16 位机 8086 的串操作指令
- (5) 掌握子程序的程序设计方法，体会不同参数传递方式的区别