Experiment 3 Arithmetic and bit string processing program

Purpose:

Master the design of programs such as multi-digit arithmetic operations, shift operations, and string operations, learn to use basic programming methods such as branching and looping, and use Debug proficiently.

Experiment content:

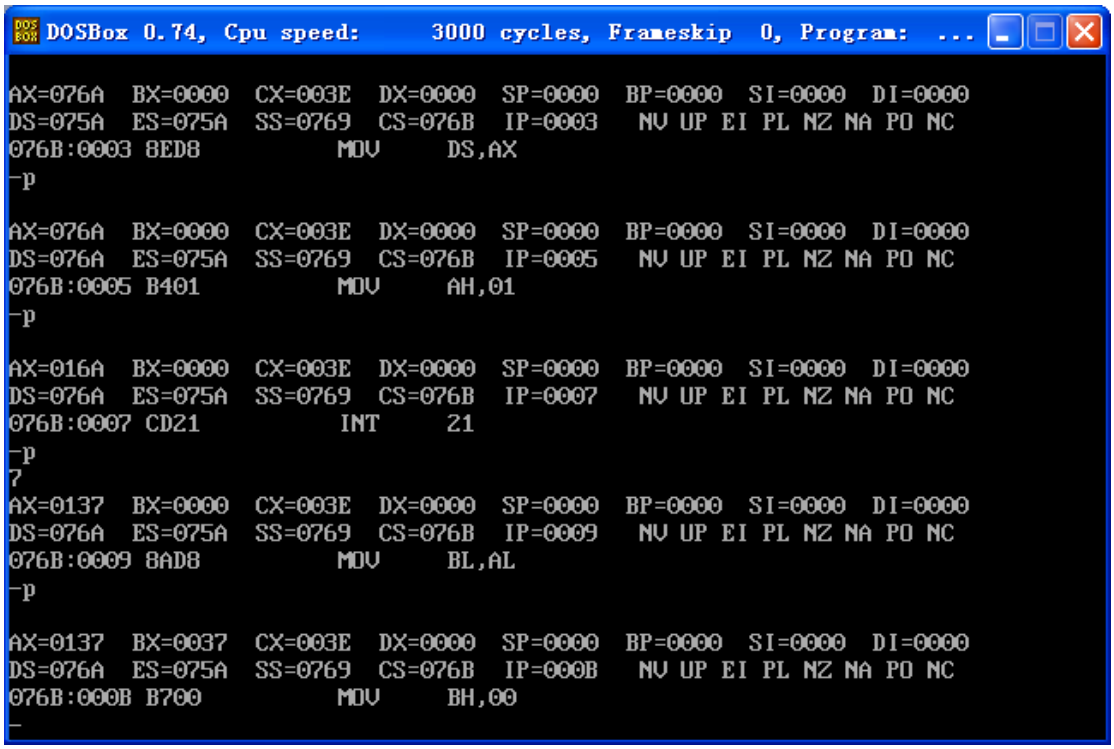
【1】 Store 16 ASCII codes of hexadecimal numbers in advance in the data segment, and the first address is ASC. Input a hexadecimal number from the keyboard to BX, use the ASC[BX] addressing mode to find the ASCII code of the corresponding digit, and take it out for display.

code:

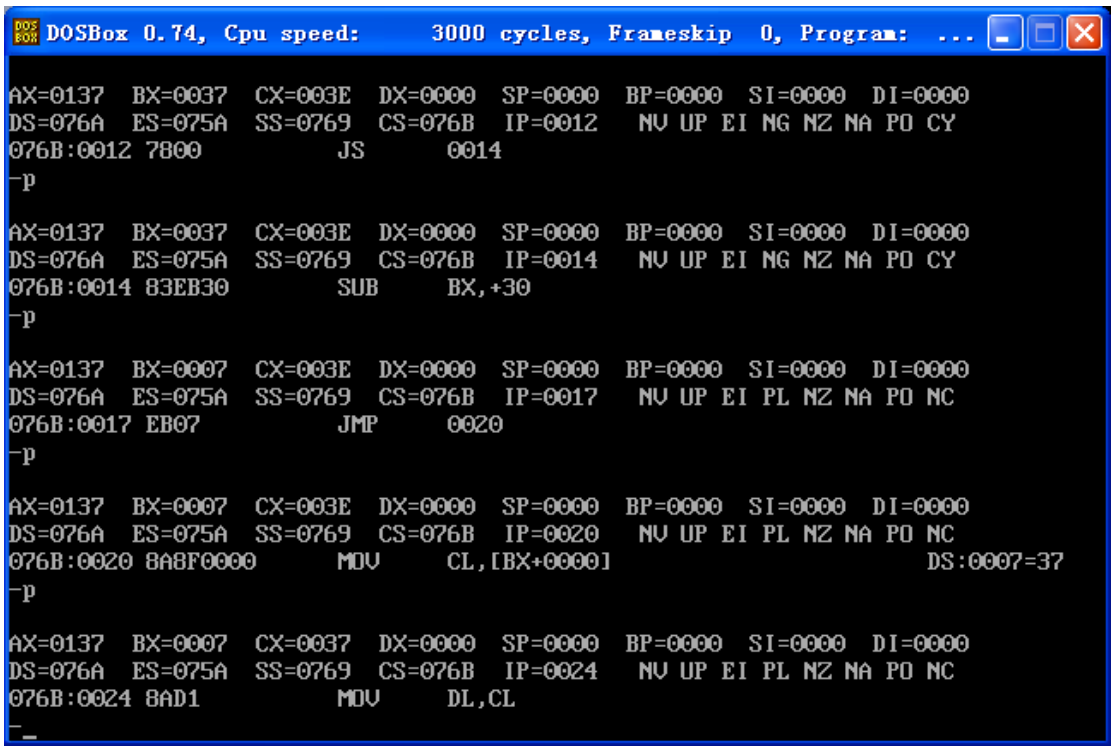
```
DATA SEGMENT

    ASC DB '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'

DATAS ENDS



CODES SEGMENT

    ASSUME CS:CODES,DS:DATAS

START:

    MOV AX,DATAS

    MOV DS,AX

    MOV AH,1H

    INT 21H



    MOV BL,AL

    MOV BH,0

    CMP BX,40H
```

```
    JNS ALP
    JS NUM

    NUM:SUB BX,30H
    JMP OVER
ALP: SUB BX,37H
JMP OVER

OVER: MOV CL,ASC[BX]
    MOV DL,CL
    MOV AH,2
int 21h

    MOV AH,4CH
INT 21H
CODES ENDS
    END START
```

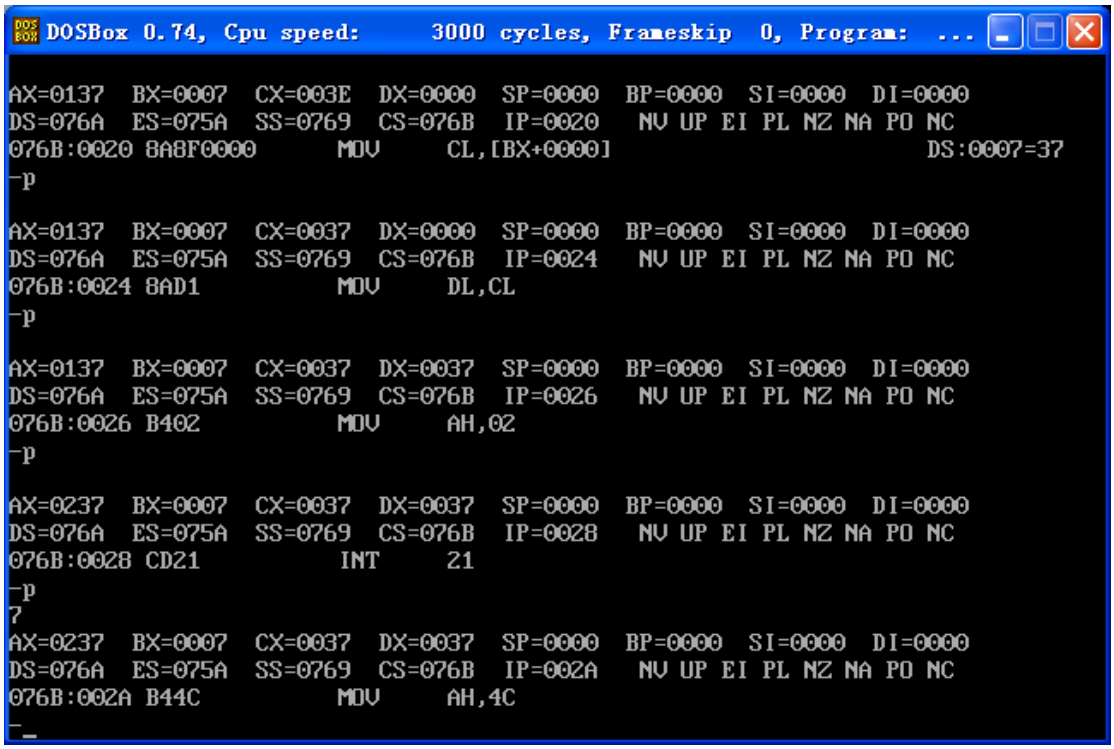
(As shown in the figure below, input the hexadecimal number 7 to BX from the keyboard)



As shown in the figure below, use the ASC [bx] addressing mode to find the ASCII code of the corresponding digit



As shown in the figure below, take out the found ASCII code and display it



【2】 Program with 16-bit instructions to handle 32-bit addition, subtraction, multiplication, and division.

Require:

（1） All variables are defined as word type, where negative numbers are expected. Some variables can also use registers, which are temporarily given under Debug. The program must be executed under Debug in order to verify the results.

（2） Track the program and record the ZF, SF, CF, OF flags after each instruction is executed. Answer the reason why the ZF, SF, CF, OF flags are set after each instruction is executed.

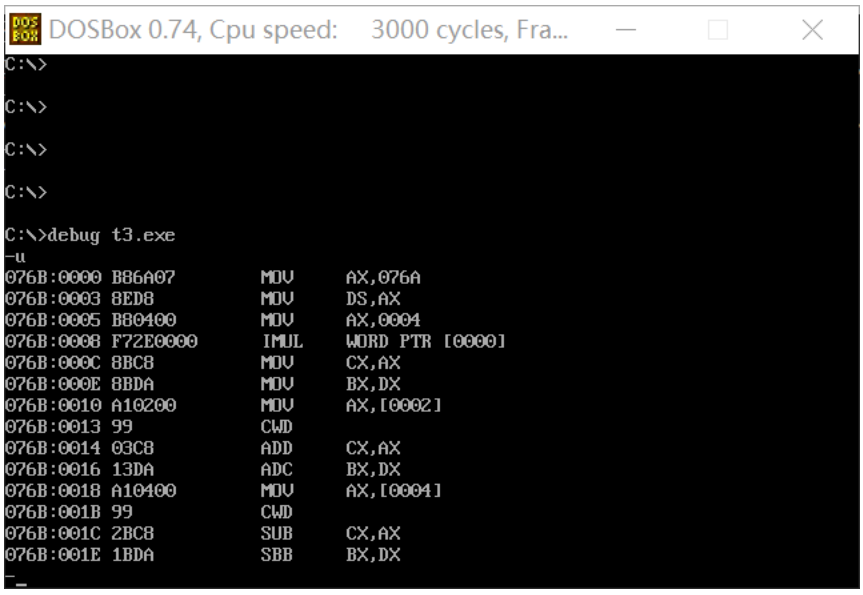
The code of this question intends to calculate (4 *X+YZ)/ 7 the value of

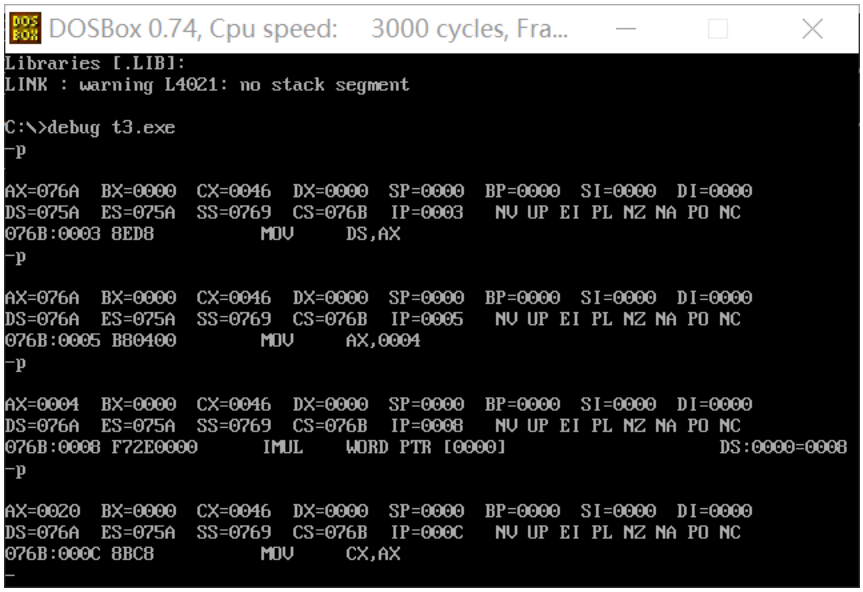
code:

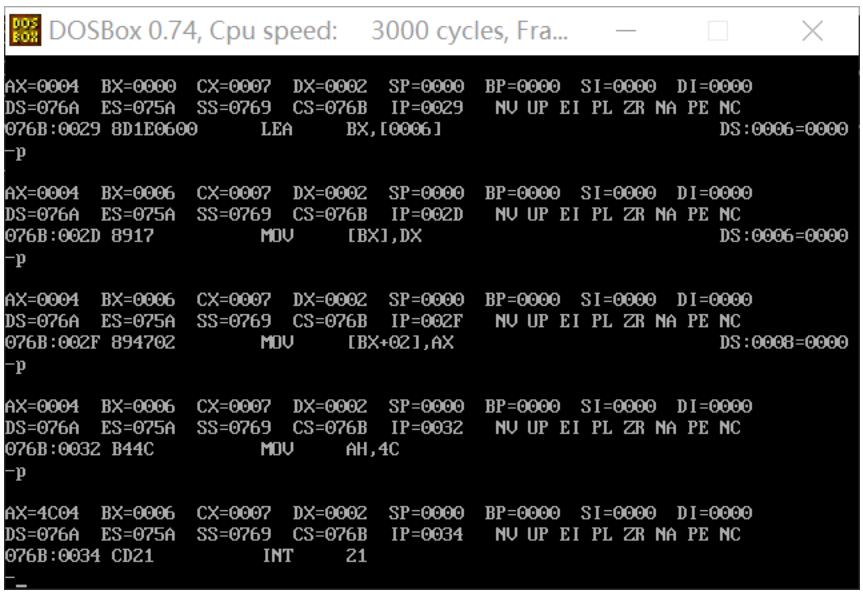
```
data segments
x dw 8;
y dw 4
z dw 6
v dd ?
data ends
code segments
assume cs:code,ds:data
start:
mov ax,data
mov ds,ax
mov ax,4
```

```
imul x
mov cx,ax
mov bx,dx
mov ax,y
cwd
add cx,ax
adc bx,dx
mov ax,z
cwd
sub cx,ax
sbb bx,dx
mov ax,cx
mov dx,bx
mov cx,7
idiv cx
lea bx,v
mov [bx],dx
mov [bx+2],ax
mov ah,4ch
int 21h
code ends
end start
```

实验结果：







As shown in the figure, this code handles the formula of (4*8+4-6)/7, and the answer should be quotient 4 and remainder 2.

As shown in the screenshot, after Debug, AL=04, DL=02, it is in compliance.

During the above process,

ZF is the zero flag. If ZR is displayed, the operation result is zero; if NZ is displayed, the operation result is non-zero.

SF is a symbol flag. Displaying NG indicates that the highest bit of the operation result is '1'.

CF is the carry flag. Displaying CY means that there is a carry in the addition operation of two unsigned numbers, or there is a borrow in the subtraction operation, and the high bits need to be supplemented; displaying NC means that there is no carry or borrow.

OF is the overflow flag. When OV is displayed, it means that the operation result of two signed numbers exceeds the range that can be expressed, and the result is wrong; if NV is displayed, it means that there is no overflow, and the result is correct.

【3】 The data section has the following definition:

BUFF DB 'ABCD$EFGHIJK$ '
STR1 DB 12 DUP(?)
LEN DB?

Write a program with string instructions to complete the following operations:

(1) Put all ' * ' symbols on the string STR1.

（2） Transfer the character string in BUFF to STR1 from left to right.

（3） Transfer the character string in BUFF to STR1 from right to left.

（4） Compare whether the two strings of BUFF and STR1 are equal, if they are equal, DX=1, otherwise DX=0.

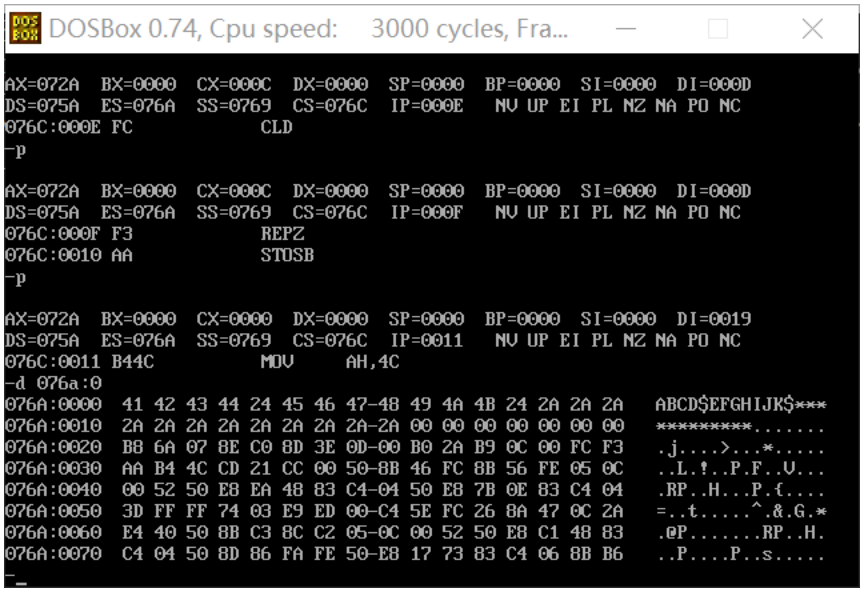
（5） Find whether there is a character $ in BUFF, and count the number of occurrences of the character $ into the BX register.

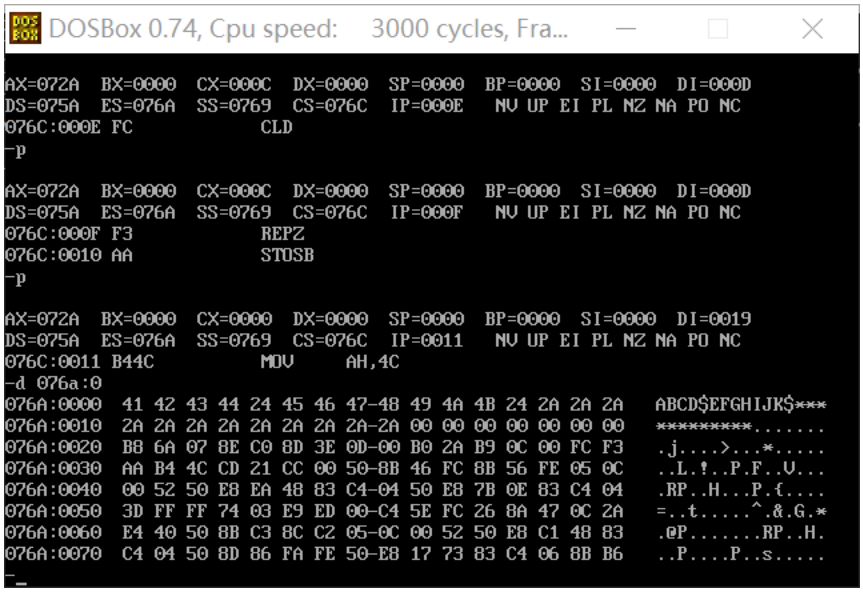
code:

(1)

```
DATA SEGMENT
BUFF DB 'ABCD$EFGHIJK$'
STR1 DB 12 DUP(?)
     LEN    DB   ?
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE,DS:DATA

START:
```

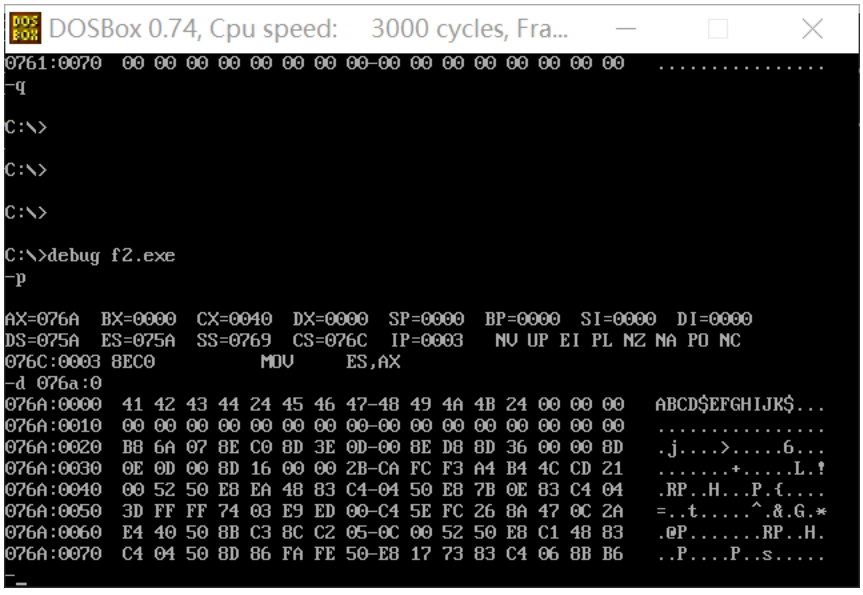
```
    mov ax,data
    mov es,ax
    lea di,STR1
    mov al,2AH ;2ah '*'
    mov cx,12
    cld
    rep stosb

MOV AH,4CH
INT 21H

CODE ENDS
END START
```

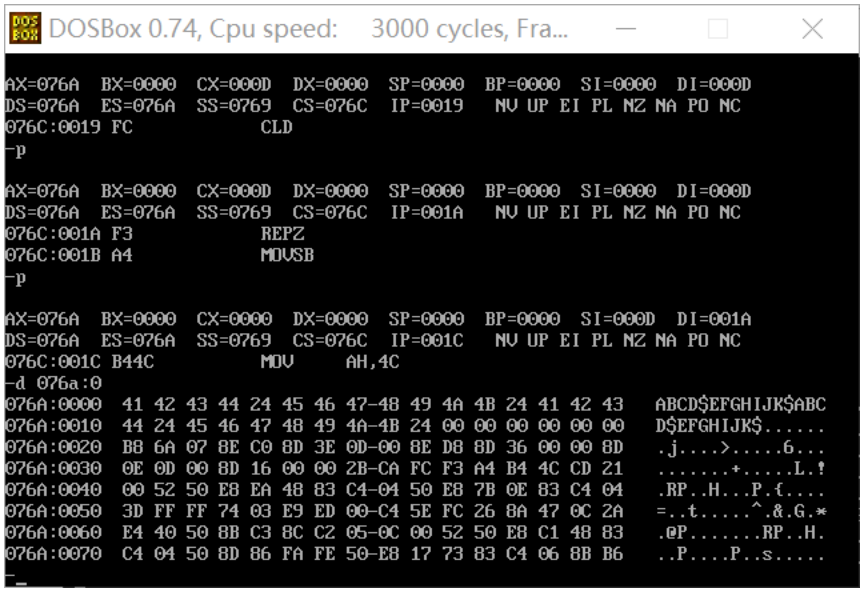




（2）

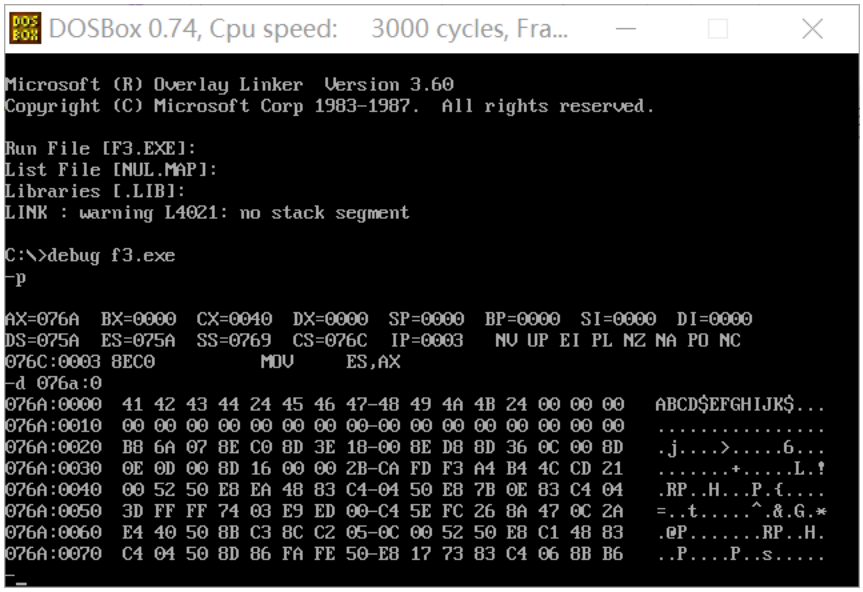
```
DATA SEGMENT
BUFF DB 'ABCD$EFGHIJK$'
     STR1    DB    12    DUP(?)
     LEN     DB    ?
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE,DS:DATA

START:
     mov ax,data
     mov es,ax
     lea di,STR1
     mov al,2AH
     mov cx,12
     cld
     rep stosb

MOV AH,4CH
INT 21H

CODE ENDS
END START
```





（3）

```
DATA SEGMENT
BUFF DB 'ABCD$EFGHIJK$'
     STR1   DB   12   DUP(?)
     LEN    DB   ?
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE,DS:DATA

START:

MOV AX,DATA

MOV ES,AX
LEA DI,LEN-1
MOV DS,AX
LEA SI,STR1-1
MOV CX,STR1-BUFF

STD
REP MOVSB

MOV AH,4CH
INT 21H

CODE ENDS
END START
```



（4）

```
DATA SEGMENT
BUFF DB 'ABCD$EFGHIJK$'
     STR1   DB   12   DUP(?)
     LEN    DB   ?
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE,DS:DATA

START:
     mov ax,data
     mov es,ax
     lea di,STR1
     mov ds,ax
     lea si,buff
     lea cx,STR1
     lea dx,buff
     sub cx,dx
     cld
     repe cmpsb
     jz yes
     mov dx,0
     yes:
          mov dx,1

MOV AH,4CH
INT 21H
```