子程序设计

一、实验目的：

1.掌握子程序设计方法，能合理划分子程序。

2.掌握汇编子程序的定义、调用、返回、参数传递等有关问题的实现，以及运行过程中的堆栈、标志位变化情况。

二、实验内容：

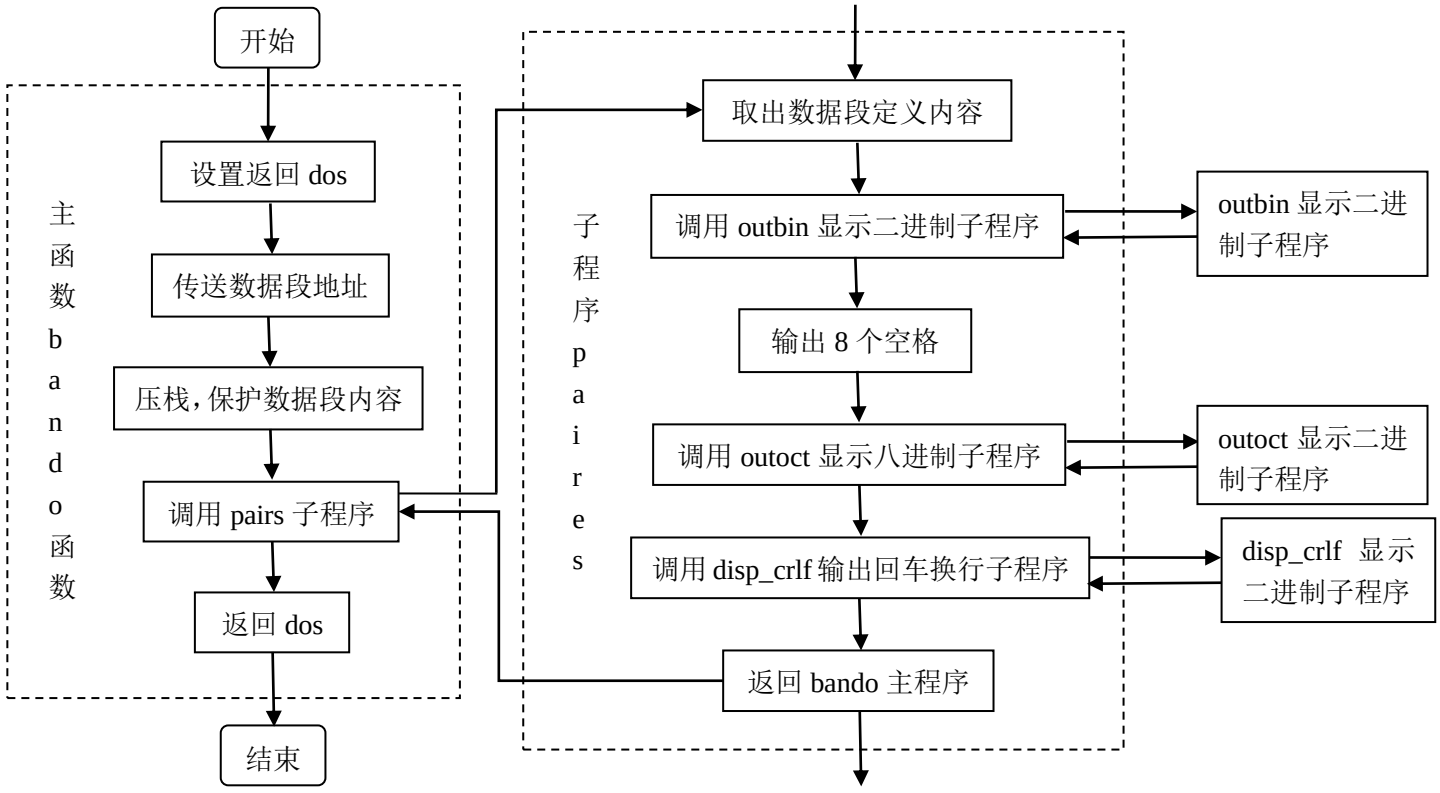
1.编写子程序嵌套结构的程序，把整数分别用二进制和八进制形式显示出来。

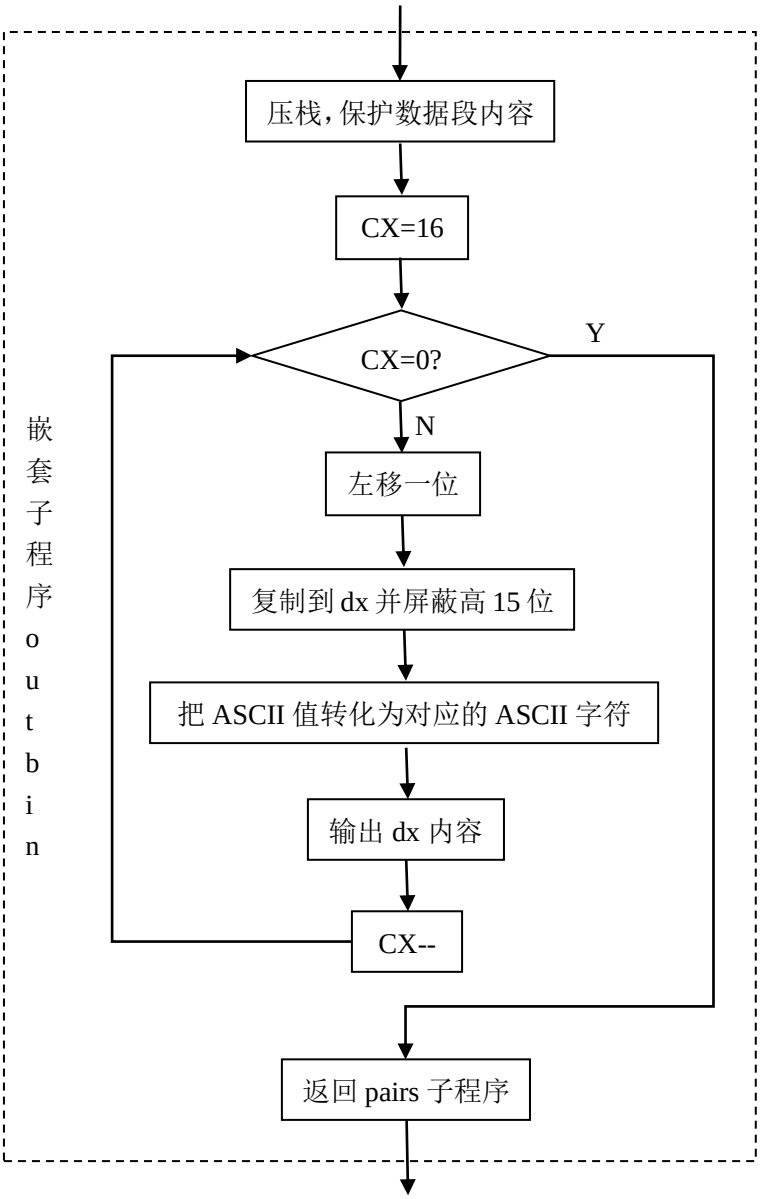
主程序 BANDO:把整数变量 VAL1 存入堆栈，并调用子程序 PAIRS;

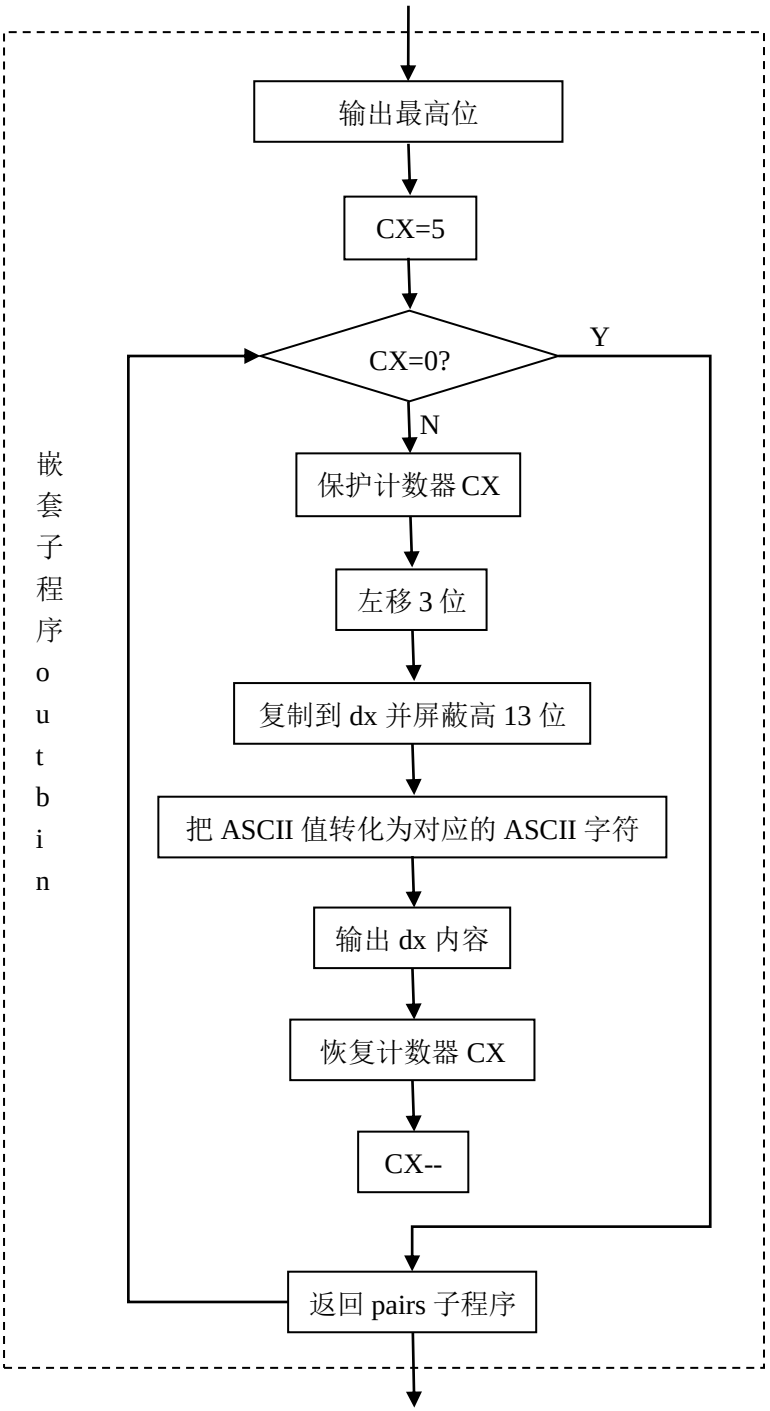
子程序 PAIRS:从堆栈中取出 VAL1，调用二进制显示子程序 OUTBIN，显示出与其等效的二进制数；输出 8 个空格；调用八进制显示子程序 OUTOCT 显示与其等效的八进制数；调用输出回车及换行符子程序

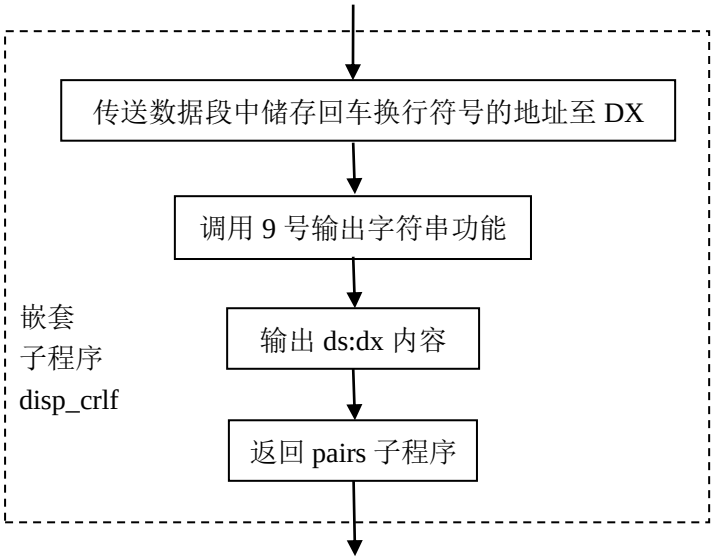
三、实验主要步骤：

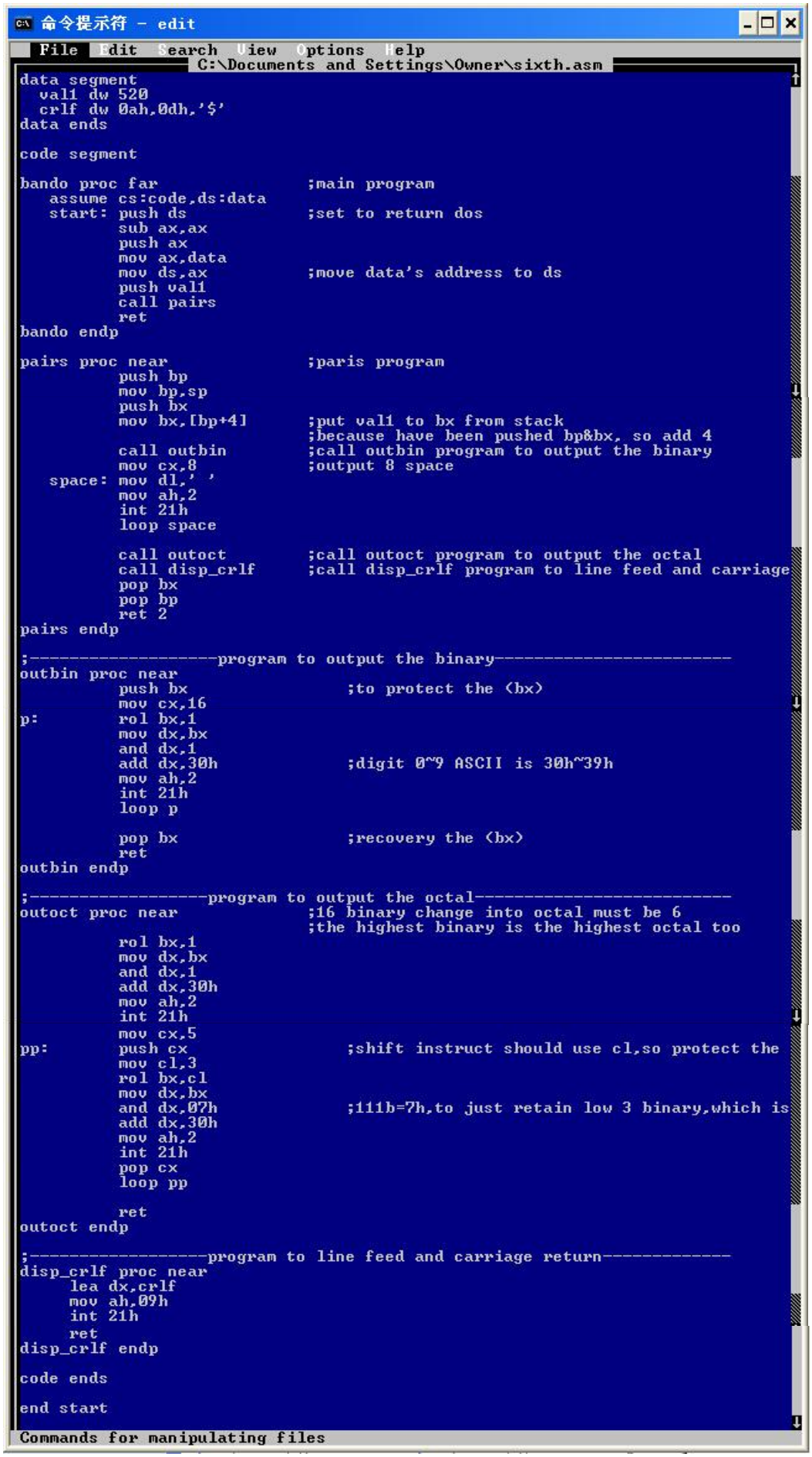
1.流程图（边幅问题，无法整合到同一页）




压栈，保护数据段内容
CX=16
CX=0?
Y
N
左移一位
复制到 dx 并屏蔽高 15 位
把 ASCII 值转化为对应的 ASCII 字符
输出 dx 内容
CX--
返回 pairs 子程序
嵌套子程序 outbin


输出最高位
CX=5
CX=0?
Y
N
保护计数器 CX
左移 3 位
复制到 dx 并屏蔽高 13 位
把 ASCII 值转化为对应的 ASCII 字符
输出 dx 内容
恢复计数器 CX
CX--
返回 pairs 子程序
嵌套子程序 outbin


传送数据段中储存回车换行符号的地址至 DX
调用 9 号输出字符串功能
输出 ds:dx 内容
返回 pairs 子程序
嵌套
子程序
disp_crlf


命令提示符 - edit
File Edit Search View Options Help
C:\Documents and Settings\Owner\sixth.asm
data segment
val1 dw 520
crlf dw 0ah,0dh,'$'
data ends
code segment
bando proc far ;main program
assume cs:code,ds:data
start: push ds ;set to return dos
sub ax,ax
push ax
mov ax,data
mov ds,ax ;move data's address to ds
push val1
call pairs
ret
bando endp
pairs proc near ;paris program
push bp
mov bp,sp
push bx
mov bx,[bp+4] ;put val1 to bx from stack
;because have been pushed bp&bx, so add 4
call outbin ;call outbin program to output the binary
mov cx,8 ;output 8 space
space: mov dl,' '
mov ah,2
int 21h
loop space
call outoct ;call outoct program to output the octal
call disp_crlf ;call disp_crlf program to line feed and carriage
pop bx
pop bp
ret 2
pairs endp
;--------------------program to output the binary-------------------------
outbin proc near
push bx ;to protect the (bx)
mov cx,16
p: rol bx,1
mov dx,bx
and dx,1
add dx,30h ;digit 0~9 ASCII is 30h~39h
mov ah,2
int 21h
loop p
pop bx ;recovery the (bx)
ret
outbin endp
;--------------------program to output the octal---------------------------
outoct proc near ;16 binary change into octal must be 6
;the highest binary is the highest octal too
rol bx,1
mov dx,bx
and dx,1
add dx,30h
mov ah,2
int 21h
mov cx,5
pp: push cx ;shift instruct should use cl,so protect the
mov cl,3
rol bx,cl
mov dx,bx
and dx,07h ;111b=7h,to just retain low 3 binary,which is
add dx,30h
mov ah,2
int 21h
pop cx
loop pp
ret
outoct endp
;--------------------program to line feed and carriage return--------------
disp_crlf proc near
lea dx,crlf
mov ah,09h
int 21h
ret
disp_crlf endp
code ends
end start
Commands for manipulating files

2.编译连接执行

```
C:\DOCUME~1\Owner>masm
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988.  All rights reserved.

Source filename [.ASM]: sixth
Object filename [sixth.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  49176 + 445299 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors
```

Masm 编译

```
C:\DOCUME~1\Owner>link

Microsoft (R) Overlay Linker  Version 3.64
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

Object Modules [.OBJ]: sixth
Run File [SIXTH.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment
```

Link 连接

执行 vall=520D，其二进制为 0000001000001000，八进制为 001010，结果正确。

```
C:\DOCUME~1\Owner>sixth
0000001000001000        001010
```

执行 vall=1314D，其二进制为 0000010100100010，八进制为 002442，结果正确。

执行

```
C:\DOCUME~1\Owner>sixth
0000010100100010        002442
```

执行 vall=1566H，其二进制为 0001010101100110，八进制为 012546，结果正确。

```
C:\DOCUME~1\Owner>sixth
0001010101100110        012546
```

3.Debug 调试

```
C:\DOCUME~1\Owner>debug sixth.exe
Microsoft (R) Symbolic Debug Utility  Version 4.00
Copyright (C) Microsoft Corp 1984, 1985.  All rights reserved.

Processor is [80286]
-u
0E81:0000 1E            PUSH    DS
0E81:0001 2BC0          SUB     AX,AX
0E81:0003 50            PUSH    AX
0E81:0004 B8800E        MOV     AX,0E80
0E81:0007 8ED8          MOV     DS,AX
0E81:0009 FF360000      PUSH    [0000]
0E81:000D E80100        CALL    0011
0E81:0010 CB            RETF
-u
0E81:0011 55            PUSH    BP
0E81:0012 8BEC          MOV     BP,SP
0E81:0014 53            PUSH    BX
0E81:0015 8B5E04        MOV     BX,[BP+04]
0E81:0018 E81600        CALL    0031
0E81:001B B90800        MOV     CX,0008
0E81:001E B220          MOV     DL,20                   ;' '
0E81:0020 B402          MOV     AH,02
```

反汇编

```
-u
0E81:0022 CD21          INT     21
0E81:0024 E2F8          LOOP    001E
0E81:0026 E81E00        CALL    0047
0E81:0029 E84100        CALL    006D
0E81:002C 5B            POP     BX
0E81:002D 5D            POP     BP
0E81:002E C20200        RET     0002
0E81:0031 53            PUSH    BX
-u
0E81:0032 B91000        MOV     CX,0010
0E81:0035 D1C3          ROL     BX,1
0E81:0037 8BD3          MOV     DX,BX
0E81:0039 83E201        AND     DX,+01
0E81:003C 83C230        ADD     DX,+30
0E81:003F B402          MOV     AH,02
0E81:0041 CD21          INT     21
0E81:0043 E2F0          LOOP    0035
-u
0E81:0045 5B            POP     BX
0E81:0046 C3            RET
0E81:0047 D1C3          ROL     BX,1
0E81:0049 8BD3          MOV     DX,BX
0E81:004B 83E201        AND     DX,+01
0E81:004E 83C230        ADD     DX,+30
0E81:0051 B402          MOV     AH,02
0E81:0053 CD21          INT     21
-u
0E81:0055 B90500        MOV     CX,0005
0E81:0058 51            PUSH    CX
0E81:0059 B103          MOV     CL,03
0E81:005B D3C3          ROL     BX,CL
0E81:005D 8BD3          MOV     DX,BX
0E81:005F 83E207        AND     DX,+07
0E81:0062 83C230        ADD     DX,+30
0E81:0065 B402          MOV     AH,02
-u
0E81:0067 CD21          INT     21
0E81:0069 59            POP     CX
0E81:006A E2EC          LOOP    0058
0E81:006C C3            RET
0E81:006D 8D160200      LEA     DX,[0002]
0E81:0071 B409          MOV     AH,09
0E81:0073 CD21          INT     21
0E81:0075 C3            RET
```

反汇编

以 vall=520D 为例查看执行不同语句后各寄存器的状态

```
-g 0009
AX=0E80  BX=0000  CX=0086  DX=0000  SP=FFFC  BP=0000  SI=0000  DI=0000
DS=0E80  ES=0E70  SS=0E80  CS=0E81  IP=0009   NV UP EI PL ZR NA PE NC
0E81:0009 FF360000      PUSH    [0000]                          DS:0000=0208
-d ds:0 l10
0E80:0000  08 02 0A 00 0D 00 24 00-00 00 00 00 00 00 00 00  ......$.........
```

①其中 520D=0208H
②接着 000A 是“换行”的 ASCII
③000D 是“回车”的 ASCII
④0024 是“$”的 ASCII，是字符串的结束标

```
-g 0055
0000001000001000        0AX=0230  BX=0410  CX=0000  DX=0030  SP=FFF2  BP=FFF6  S
I=0000  DI=0000
DS=0E80  ES=0E70  SS=0E80  CS=0E81  IP=0055   NV UP EI PL NZ NA PE NC
0E81:0055 B90500        MOV     CX,0005
-t
AX=0230  BX=0410  CX=0005  DX=0030  SP=FFF2  BP=FFF6  SI=0000  DI=0000
DS=0E80  ES=0E70  SS=0E80  CS=0E81  IP=0058   NV UP EI PL NZ NA PE NC
0E81:0058 51            PUSH    CX
-t
AX=0230  BX=0410  CX=0005  DX=0030  SP=FFF0  BP=FFF6  SI=0000  DI=0000
DS=0E80  ES=0E70  SS=0E80  CS=0E81  IP=0059   NV UP EI PL NZ NA PE NC
0E81:0059 B103          MOV     CL,03
-t
AX=0230  BX=0410  CX=0003  DX=0030  SP=FFF0  BP=FFF6  SI=0000  DI=0000
DS=0E80  ES=0E70  SS=0E80  CS=0E81  IP=005B   NV UP EI PL NZ NA PE NC
0E81:005B D3C3          ROL     BX,CL
-t
AX=0230  BX=2080  CX=0003  DX=0030  SP=FFF0  BP=FFF6  SI=0000  DI=0000
DS=0E80  ES=0E70  SS=0E80  CS=0E81  IP=005D   NV UP EI PL NZ NA PE NC
0E81:005D 8BD3          MOV     DX,BX
```

520D 对应八进制的最高位是 0，经过处理后，此时 DX=0030，对应的 ASCII 符号是’0’。

520D 对应八进制的第三高位是 1，经过处理后，此时 DX=0031，对应的 ASCII 符号是'1'。

四、实验结果与分析：

通过实验，进一步熟知和掌握子程序的定义、嵌套和编写。

在八进制显示小程序中，循环计数器 CX 和移位计数器 CL 冲突，此时可以使用 PUSH 和 POP 指令对 CX 进行保护。具体是进入循环体后立刻把 CX 压栈，在结束本次结束循环前让 CX 出栈，使得循环计数器回复，正常控制循环次数。

同时在子程序的编写时要注意保护数据段的内容。例如我们把 vall 中的数字放到 BX 中，那么要注意在子程序结束前把 BX 恢复到原来的内容，不然可能会影响下一个子程序对他的使用。

还有一点是 MOV 和 LEA 指令的不同，同样是双目操作，mov 指令的操作是(DST)←(SRC)而 lea 指令的操作是(REG)←SRC，把 DST 和 SRC 看成是地址，那么 mov 后 DST 地址的内容是 SRC 地址的内容，而 lea 后 REG 的内容是 SRC 这个地址。所以 lea 叫有效地址传送指令。在处理字符串时，由于一般是用到指针型寄存器，一般用 lea 指令。所以在输出回车换行的子程序中用 lea 指令进行传送，把定义数据段内容看成字符串处理（因为定义时就把字符串结束符'$'定义了）。