****

计算机原理与汇编语言程序设计实验报告

|  |  |
| --- | --- |
| 题 目： | 汇编语言程序设计 |
| 学生姓名： | 张子洋 |
| 指导教师： | 雷向东 |
| 学 院： | 计算机学院 |
| 专业班级： | 计科2203班 |

本科生院制

2023年12月

目录

[第一章 实验目的与实验内容 1](#_Toc122039355)

[1.1 实验目的 1](#_Toc122039356)

[1.2 实验内容 1](#_Toc122039357)

[第二章 实验一 2](#_Toc122039359)

[2.1 程序功能 2](#_Toc122039360)

[2.2 程序分析 2](#_Toc122039361)

[2.3 程序的具体实现 3](#_Toc122039362)

[第三章 实验二 1](#_Toc122039363)0

[3.1 程序功能 1](#_Toc122039364)0

[3.2 程序分析 1](#_Toc122039365)0

[3.3 程序的具体实现 1](#_Toc122039366)1

[第四章 实验三 1](#_Toc122039367)8

[4.1 程序功能 1](#_Toc122039368)8

[4.2 程序分析 1](#_Toc122039369)8

[4.3 程序的具体实现 1](#_Toc122039370)9

[第五章 总结 2](#_Toc122039371)6

[5.1 经验和感想 2](#_Toc122039372)6

[参考文献 2](#_Toc122039374)6

1. 实验目的与实验内容
   1. 实验目的

通过实验熟悉汇编语言的操作指令和内存管理，了解斐波那契，冒泡排序和素数算法的实现。

* 1. 实验内容

1. 输出斐波那契数列前五十项；
2. 输出一千万以内的所有素数；
3. 采用冒泡排序对200个有正有负的数进行排序。
4. 实验一

2.1 程序功能

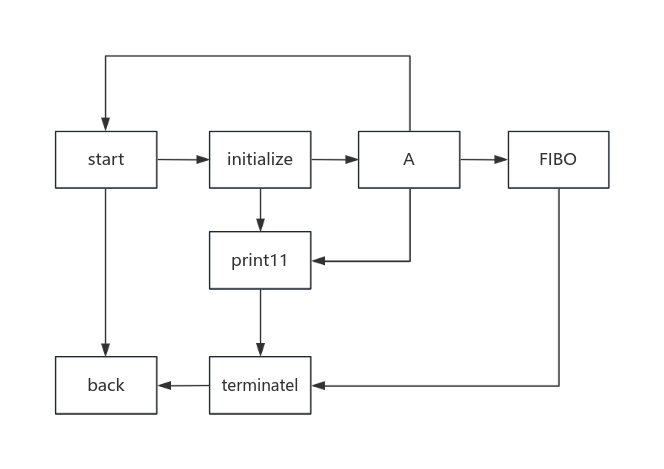
程序生成斐波那契数列并逐位打印它们。它使用自定义堆栈来处理 64 位结果并使用子例程进行打印，利用 DOS 中断调用进行输出。斐波那契数列以迭代方式计算，并打印前两位数字作为循环的起点。当满足特定条件时程序终止。

2.2 程序分析

数据段：定义一个 200 字节的空间 (DW 200 DUP(0)) 来存储 64 位数字（用于斐波那契计算）。定义 30 字节堆栈空间 (DW 30 DUP(0))。

代码段（CODES）：初始化段和寄存器。定义主 START 标签。使用 32 位指令。

斐波那契计算（FIBO）：使用循环计算斐波那契数列。使用自定义堆栈进行临时存储。每次迭代后打印结果。打印子程序（打印）：打印存储在 EDX:EAX 中的 64 位结果。单独处理结果的每个数字，将其转换为 ASCII 并打印。使用 INT 21H 进行 DOS 函数调用。对于每个小数位重复除法和数字提取过程。初始化和循环（START）：初始化数据和堆栈段。启动标记为 A 的循环。调用 FIBO 子例程，然后打印结果。打印子程序（PRINT11）：打印斐波那契数列的前两位数字。



2.3 程序的具体实现

DATAS SEGMENT

DW 200 DUP(0)

;数字64位,4\*50=200

DATAS ENDS ;

STACKS SEGMENT

DW 30 DUP(0)

STACKS ENDS

;完整段的定义

CODES SEGMENT

ASSUME CS:CODES,DS:DATAS,SS:STACKS

START:

.386

MOV AX,DATAS

MOV DS,AX

MOV AX,STACKS

MOV SS,AX

;地址初始化

MOV SI,0

MOV DI,0

MOV EAX,1;数列第1个值

MOV EDX,0 ;数列第0个值

JMP PRINT11

A:

CALL FIBO

FIBO:

CMP DI,392

JE BACK

CMP DI,0

JE SET1

SET1BACK:

CMP DI,8

JE SET2

SET2BACK:

ADD DI,8

MOV EAX,DS:[DI-8]

MOV EDX,DS:[DI-4]

MOV ECX,DS:[DI-12]

MOV EBX,DS:[DI-16]

ADD EAX,EBX

ADC EDX,ECX ;加上进位标志，且将进位标志置0

MOV DS:[DI],EAX

MOV DS:[DI+4],EDX

CALL PRINT

PRINTBACK:

CALL FIBO

SET1:

MOV DS:[0],EAX

MOV DS:[4],EDX;双字占四个字节

ADD DI,8

JMP SET1BACK

SET2:

MOV DS:[8],EAX

MOV DS:[12],EDX

JMP SET2BACK

PRINT:

PUSH EAX

PUSH EBX

PUSH ECX

PUSH EDX

CMP EDX,2 ;只存在两次进位，因为第一次进位后，舍弃超出范围的二进制数，后面到了50位才再次超出

JNB P1 ;得到的EDX为12不可以直接字符输出，所以直接输出最高位，再给EAX赋值

CMP EDX,0;当产生第一次进位时,因为此时EAX储存的值要<1000000000,所以要重新判断

JA P2

CMP EAX,1000000000

JNB P2

CMP EAX,100000000

JNB P3

CMP EAX,10000000

JNB P4

CMP EAX,1000000

JNB P5

CMP EAX,100000

JNB P6

CMP EAX,10000

JNB P7

CMP EAX,1000

JNB P8

CMP EAX,100

JNB P9

CMP EAX,10

JNB P10

CMP EAX,1

JNB P11

P1:

MOV DL,1

ADD DL,30H

MOV AH,2H

INT 21H

MOV EAX,10011010001001110101010101100001B

MOV EDX,0

P2:

MOV EBX,1000000000

DIV EBX

PUSH EDX

MOV EDX,EAX

ADD DL,30H

MOV AH,2H

INT 21H

POP EAX

MOV EDX,0

P3:

MOV EBX,100000000

DIV EBX;除法为64位/32位的除法，所以扩充EDX

PUSH EDX

MOV EDX,EAX

ADD DL,30H

MOV AH,2H

INT 21H

POP EAX

MOV EDX,0

P4:

MOV EBX,10000000

DIV EBX

PUSH EDX

MOV EDX,EAX

ADD DL,30H

MOV AH,2H

INT 21H

POP EAX

MOV EDX,0

P5:

MOV EBX,1000000

DIV EBX

PUSH EDX

MOV EDX,EAX

ADD DL,30H

MOV AH,2H

INT 21H

POP EAX

MOV EDX,0

P6:

MOV EBX,100000

DIV EBX

PUSH EDX

MOV EDX,EAX

ADD DL,30H

MOV AH,2H

INT 21H

POP EAX

MOV EDX,0

P7:

MOV EBX,10000

DIV EBX

PUSH EDX

MOV EDX,EAX

ADD DL,30H

MOV AH,2H

INT 21H

POP EAX

MOV EDX,0

P8:

MOV EBX,1000

DIV EBX

PUSH EDX

MOV EDX,EAX

ADD DL,30H

MOV AH,2H

INT 21H

POP EAX

MOV EDX,0

P9:

MOV EBX,100

DIV EBX

PUSH EDX

MOV EDX,EAX

ADD DL,30H

MOV AH,2H

INT 21H

POP EAX

MOV EDX,0

P10:

MOV EBX,10

DIV EBX

PUSH EDX

MOV EDX,EAX

ADD DL,30H

MOV AH,2H

INT 21H

POP EAX

MOV EDX,0

P11:

MOV EBX,1

DIV EBX

PUSH EDX

MOV EDX,EAX ;DX是EDX的低16位

ADD DL,30H;PUSH只是把寄存器的内容送入堆栈

MOV AH,2H

INT 21H

POP EAX;POP把栈顶的元素送入相应的寄存器

MOV EDX,0

MOV DL,32

MOV AH,2H

INT 21H

POP EDX

POP ECX

POP EBX

POP EAX

JMP PRINTBACK

;PRINT11

PRINT11:

PUSH EDX

PUSH EAX

;输出第一位

MOV DL,1

ADD DL,30H

MOV AH,2H

INT 21H

;输出空格

MOV DL,32

MOV AH,2H

INT 21H

;输出第二位

MOV DL,1

ADD DL,30H

MOV AH,2H

INT 21H

;输出空格

MOV DL,32

MOV AH,2H

INT 21H

POP EAX

POP EDX

JMP A

BACK:

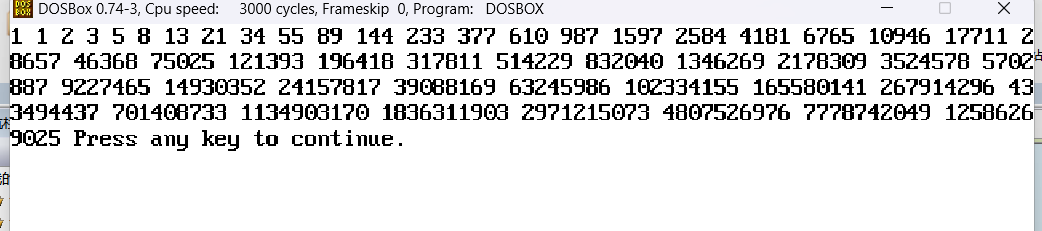
MOV AX,4C00H

INT 21H

CODES ENDS

END START

运行结果：



1. 实验二

[3.1 程序功能](#_Toc122039360)

运行程序后在dos窗口中输出一千万以内的所有质数，每一项以空格隔开。该代码遵循一个结构，在该结构中初始化堆栈、迭代值、检查素数并显示素数。

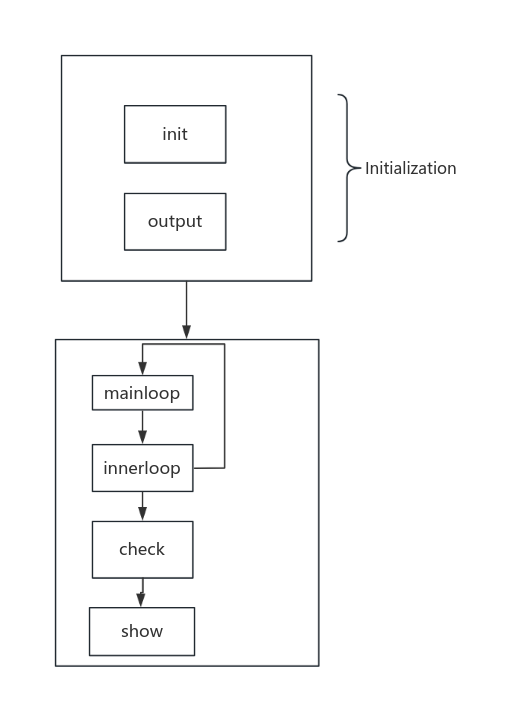
[3.2 程序分析](#_Toc122039361)

细分：零段 (zero)： 包含用于初始化的单个零。值段 (value)： 存储要检查素数的当前值。堆栈段（stack）：充当程序的堆栈。

代码段 (code)：初始化 (init)： 设置程序的堆栈。获取下一个值 (get\_next)： 计算下一个值进行检查，将当前值增加 2。素数检查 (check)： 确定当前值是否为素数。实现简单的试除算法。显示素数 (show)： 显示当前素数。显示输出 (dout)： 以四位数字的形式输出数字，处理前导零。

主循环 (main)：调用check来确定某个值是否为素数。如果是质数，则调用show 显示质数。继续循环。

关键点：该程序使用堆栈和段进行组织。素性检查是通过试除到数字的平方根来完成的。输出格式确保每个数字都以四位数字显示，并处理前导零。该程序打印它找到的质数，质数之间有一个空格。



[3.3 程序的具体实现](#_Toc122039362)

assume cs:code, ss:stack

zero segment

db 0

zero ends

value segment ;存当前要检查是否为质数的值

dw 3, 0

value ends

stack segment ;栈

dw 40H dup(0)

stack ends

code segment

main:

call init ;初始化栈

mov dl, '2' ;先输出 2 和空格

mov ah, 2

int 21h

mov dl, ' '

mov ah, 2

int 21h

mov cx, 1000 ;二重循环来枚举 3~1e7 的值

main\_s1:

push cx

mov cx, 10000

main\_s2:

call check ;检查当前枚举到的 value 是否为质数

push cx

mov cx, ax

jcxz main\_s3

call show

main\_s3:

pop cx

call get\_next ;value += 2

call get\_next

sub cx, 1

loop main\_s2

pop cx

loop main\_s1

mov ax, 4c00h

int 21h

init: ;初始化栈

mov ax, stack

mov ss, ax

mov sp, 40H

Ret

get\_next: ;计算下一项

push dx

push si

push ax

push bx

push cx

push es

push ds

mov ax, value

mov es, ax

mov ax, es:[0] ;ax 为低位

mov bx, es:[2] ;bx 为高位

add ax, 1

cmp ax, 10000

jb get\_next\_done

mov ax, 0 ;低位超过 10000 则进位

add bx, 1

get\_next\_done:

mov es:[0], ax

mov es:[2], bx

pop ds

pop es

pop cx

pop bx

pop ax

pop si

pop dx

ret

check:

; mov ax, 1

; ret

push cx

push dx

push si

push bx

push es

mov ax, value

mov es, ax

mov ax, es:[0]

mov bx, es:[2]

mov dx, 2

mov cx, 9999 ;确定试除的上界

cmp bx, 1

jae check\_s1

cmp ax, 3

jbe check\_is\_prime

mov cx, ax

sub cx, 1

check\_s1:

mov ax, value

mov es, ax

mov ax, es:[0]

mov bx, es:[2]

check\_s2: ;使用当前 cx 的值对 value 进行试除，减法模拟求余运算

cmp ax, dx

ja check\_minus

jb check\_s3

cmp bx, 0

je check\_is\_not\_prime ;恰好整除则非质数

check\_s3:

cmp bx, 1 ;向高位借位，借不到则无法整除，检查下一个 cx

jb check\_s4

sub bx, 1

add ax, 10000

check\_minus:

sub ax, dx ;做减法模拟求余运算

jmp check\_s2

check\_s4:

sub cx, 1

inc dx

cmp cx, 1

ja check\_s1

check\_is\_prime:

mov ax, 1

jmp check\_done

check\_is\_not\_prime:

mov ax, 0

check\_done:

pop es

pop bx

pop si

pop dx

pop cx

ret

show: ;输出字符串

push cx

push dx

push si

push ax

push bx

push es

mov ax, zero

mov ds, ax

mov ax, 0

mov ds:[0], ax

mov ax, value

mov es, ax

mov bx, es:[2] ;先输出高位

mov cx, 1000

call dout

mov cx, 100

call dout

mov cx, 10

call dout

mov cx, 1

call dout

mov bx, es:[0] ;再输出低位

mov cx, 1000

call dout

mov cx, 100

call dout

mov cx, 10

call dout

mov cx, 1

call dout

mov dl, ' '

mov ah, 2

int 21h ;输出一个空格

pop es

pop bx

pop ax

pop si

pop dx

pop cx

ret

dout:

push dx

push si

push ax

push es

mov dx,0 ;dx清0,除cx时，被除数为dx,ax

mov ax,bx ;将bx值（第一次为输入的数，随后为余数）赋值给ax

div cx ;(dx,ax)，实际为ax（dx==0）除以cx（cx值在调用程序前设置，作为参数传递进来）

xchg ax,dx ;ax与dx交换内容。交换后：ax中为余数，dx中为商

mov bx,ax ;将ax值（余数）赋予bx（进入下一轮运算）

;如果用户前面输入65535，那么在第一轮除以10000后，dx中值为6，bx中值为5535

cmp dl,0

jne outanum ;如果dx中值不为0，则直接输出相应的数值

mov ax, zero

mov ds, ax

mov ax, 0

cmp ds:[0], ax ;如果dx中值为0,那么判断是前面无意义的0，还是中间有意义的0。

;如305，那么如果不进行次判断将输入00305。通过此位可以不输出前面两个0，但是输出中间0。

je con ;如果是前面无意义的0 ，则不输出

outanum:

mov ax, 1

mov ds:[0], ax ;如果输出了一个大于0的数字，则置标志位为1，使得其后所有0都会被输出

add dl, 30h ;dl中数值加上30h，变成对应的ASCII码。

mov ah, 2

int 21h ;输出该数字

con:

pop es

pop ax

pop si

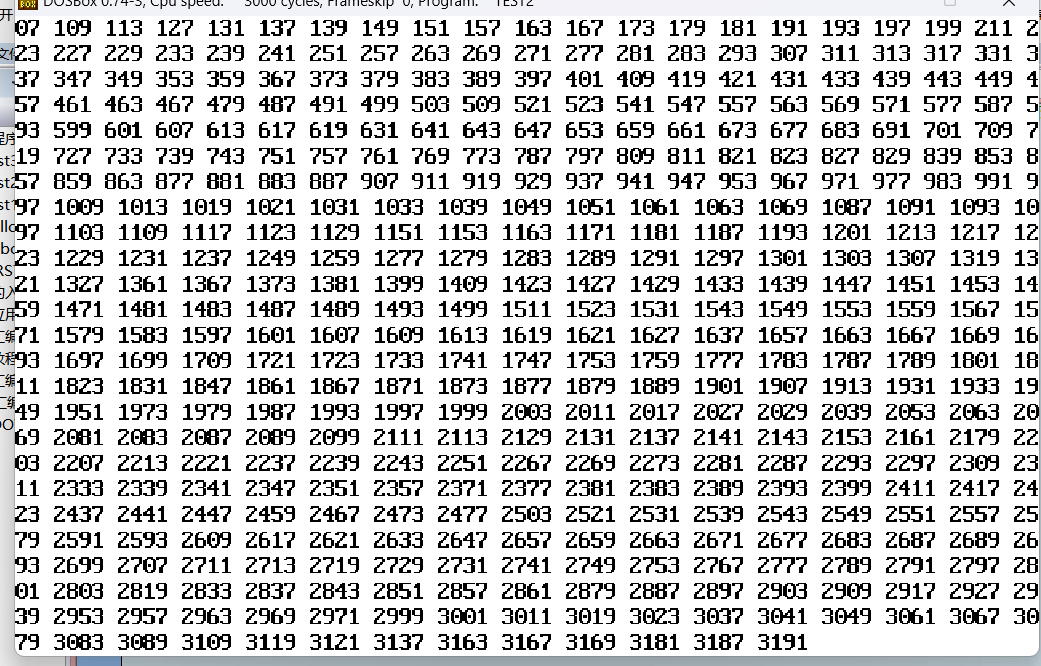
pop dx

ret

code ends

End

运行结果：



1. 实验三

[4.1 程序功能](#_Toc122039360)

采用冒泡排序对200个有正有负的数进行排序。代码将整数序列读取为字符串，将它们转换为整数，然后对整数数组执行简单的冒泡排序。最后，它打印排序后的整数。

[4.2 程序分析](#_Toc122039361)

数据段（DATA）：定义用于输入字符串的缓冲区 (buf)、用于存储整数的数组 (array) 和变量 (number, symbol) 用于跟踪数组大小和整数的符号。

代码段（CODE）：以 main 标签开头。将整数序列读取为字符串，将其转换为整数，并将它们存储在array中。实现一个简单的冒泡排序算法来按升序对整数进行排序。打印排序后的整数，包括每个整数之间的换行符。

程序（read 和 print\_int）：read 过程将字符串转换为整数。print\_int 过程打印一个整数。该程序读取整数序列，直到遇到换行符。它使用read过程将输入字符串转换为整数，并考虑负数的可能性。整数存储在数组中 (array)。然后，程序对数组执行简单的冒泡排序，以升序排列整数。最后，它打印排序后的整数，每个整数后跟一个换行符。

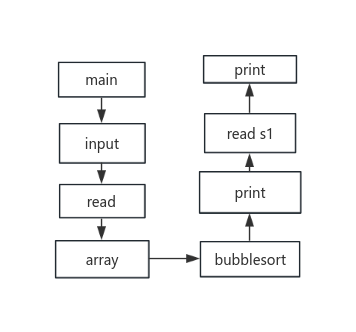
数据段（DATA）：buf：存储输入字符串的缓冲区。它用于接收用户输入。array：存储整数的数组。number：用于跟踪数组中整数数量的变量。symbol：用于确定整数符号（正或负）的变量。

输入部分（input）：使用 DOS 中断 21h (AH=0Ah) 接受整数序列作为字符串。将输入缓冲区中的换行符转换为 DOS 样式的换行符 (CR-LF)。

处理部分（read 和 print\_int 过程）：read过程将字符串转换为整数，考虑负数的可能性。print\_int过程打印一个整数，包括处理负数。

排序部分 (main)：使用简单的冒泡排序算法对数组 (array) 中的整数按升序进行排序。外循环 (L1) 迭代每个元素，内循环 (L2) 根据需要比较和交换元素。排序后，排序后的整数会以换行符打印。

关键点：程序使用缓冲区 (buf) 来存储用户输入，并使用数组 (array) 来存储整数。整数转换和排序是使用汇编语言实现的。负整数在输入和打印期间处理。该程序在新行上打印每个排序的整数。



[4.3 程序的具体实现](#_Toc122039362)

.model large

assume ds:data, cs:code

data segment

buf db ?

db ?

db 10000 dup(?)

array dw 500 dup(?)

number dw 1 dup(?)

symbol dw 1 dup(?)

data ends

code segment

main:

mov [number], 0

input:

lea dx, buf ; 输入字符串，到buf中

mov ah, 0ah

int 21h

mov dl, 13 ; 打印换行

mov ah, 02h

int 21h

mov dl, 0ah

mov ah, 02h

int 21h

mov bl, [buf + 2]

cmp bl, 13

jz input\_end

call read ; 调用ReadUInt，该函数将buf转换成整数，存放到ax中

mov si, [number]

shl si, 1 ; si = si \* 2，一个数字占2个字节，因此乘以2

mov [array + si], ax ; 将读取的数字存放到数组中

shr si, 1

inc si

mov [number], si

jmp input

input\_end:

push sp

mov si, 0 ; 冒泡排序，i=si,j=di

mov di, 0

L1:

cmp si, [number]

jge L1\_END

mov di, si ; j = i + 1

inc di

L2:

cmp di, [number]

jge L2\_END

shl si, 1

shl di, 1

mov ax, [array+si]

mov bx, [array+di]

shr si, 1

shr di, 1

cmp ax, bx

jg exchange

inc di

jmp L2

exchange:

shl si, 1

shl di, 1

mov [array+di], ax

mov [array+si], bx

shr si, 1

shr di, 1

inc di

jmp L2

L2\_END:

inc si

jmp L1

L1\_END:

mov bx, 0

L3:

cmp bx, [number]

jge L3\_END

mov si, bx

shl si, 1

mov ax, [array+si]

call print\_int

push bx

push si

mov dl, 13 ; 打印换行

mov ah, 02h

int 21h

mov dl, 0ah

mov ah, 02h

int 21h

pop si

pop bx

inc bx

jmp L3

L3\_END:

pop sp

mov ah, 4ch

int 21h

read proc

pushf

push bx

push cx

push dx

push si

push di

mov [symbol], 0

mov cx, 0

mov cl, [buf + 1] ; 获取字符串长度

lea si, [buf + 2] ; 获取字符串地址

mov ax, 0

mov di, 0

add si, di

mov bl, [si]

cmp bl, '-'

jne read\_s1

mov [symbol], 1

sub cl, 1

inc di

read\_s1:

lea si, [buf + 2]

add si, di ; 获取当前处理的字符地址

inc di

mov bl, [si] ; 获取字符

sub bl, 48 ; 字符减去'0'

mov dl, 10

mul dl ; 乘以10

mov bh, 0

add ax, bx ; 加上数字

loop read\_s1

mov bx, [symbol]

cmp bx, 1

jne read\_s1\_end

neg ax

read\_s1\_end:

pop di

pop si

pop dx

pop cx

pop bx

popf

ret

read endp

print\_int proc

pushf ; save eflags and register

push bx

push cx

push dx

mov [symbol], 0

cmp ax, 0

jz print\_zero

js print\_minus

jmp skip\_print\_zero\_and\_minus

print\_zero:

mov dl, '0'

mov ah, 02h

int 21h

jmp print\_int\_end2

print\_minus:

neg ax

mov [symbol], 1

skip\_print\_zero\_and\_minus:

mov cx, 0

mov bx, 10

print\_int\_loop1:

mov dx, 0

div bx

add dl, 30h

push dx

inc cx

cmp ax, 0

jne print\_int\_loop1

cmp [symbol], 1

jnz print\_int\_loop2

mov dl, '-'

mov ah, 02h

int 21h

print\_int\_loop2:

pop dx

mov ah, 02h

int 21h

loop print\_int\_loop2

print\_int\_end2:

mov dl, 13 ; 打印换行

mov ah, 02h

int 21h

pop dx

pop cx

pop bx

popf

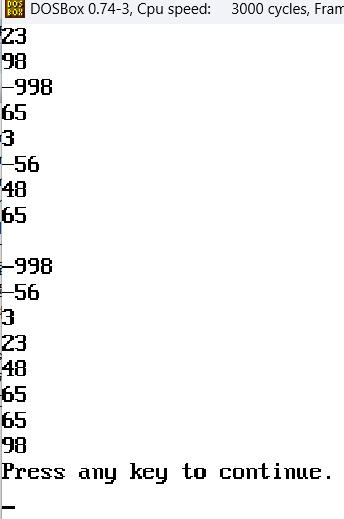
ret

print\_int endp

code ends

end main

运行结果：



1. 经验总结和感想

5.1经验和感想

学习汇编语言代码让我深入理解了底层计算机操作的细节，同时也展现了在汇编层面如何进行基本的输入输出、循环、条件判断等操作。 汇编语言是一种低级语言，直接操作计算机硬件，这使得我更深刻地理解了计算机的底层工作原理，包括寄存器、栈、内存等。

输入输出操作涉及到从键盘获取输入、将整数转换为字符串输出，需要仔细处理各种细节。而通过循环和条件判断的使用，实现了基本的控制流程，对理解高级语言中的循环和条件语句提供了一个良好的基础。代码中还使用了栈来进行一些操作，包括子过程的调用和返回地址的保存。这加深了我对栈的理解，也让我更加明白函数调用的底层原理。

学习汇编语言时，调试变得尤为重要。我通过单步执行代码，观察寄存器和内存的变化，逐步分析程序执行过程。这有助于培养对程序执行流程的逻辑思考和调试技能。

总体而言，学习汇编语言是一次深入了解计算机底层原理的体验，虽然代码显得冗长而琐碎，但从中获得的底层控制和运算经验对于理解计算机体系结构和编程语言设计具有重要意义。

参考文献

[1] 雷向东. 汇编语言程序设计. 中南大学出版社, 2019

[2] 唐朔飞. 计算机组成原理（第2版）.高等教育出版社, 2010