

1. 实验目的

- (1) 掌握汇编语言子程序设计方法；
- (2) 熟悉汇编语言主程序和子程序不同的参数传递方法；
- (3) 了解 C 语言和汇编语言混合编程方法以及主、子程序之间参数传递的机制。

2. 实验内容

任务 1.1：立即数寻址与间接寻址

立即数寻址与间接寻址

描述

现在有一个输出的字符，已知该字符的有效地址为buffer，利用立即数寻址和间接寻址的方法，给该需要输出的字符赋值为字母a对应的ascii码。

输入

无

输出

a

输入样例 1

无

输出样例 1

a

提示

字母a对应的ascii码值为0x61，将该值赋值给eax，在将eax 赋值给有效地址为buffer的存储单元。

任务 1.2：实现简单的数学公式

实现简单的数学公式

描述

利用汇编实现实现 $((X-Y)*Z+50)/V$ ，并把对应的结果存放在eax ,与edx中。

假设此时，x, y, z, v都已经输入并保存。可以直接通过 [NUM] 调用。

输入

1. x=4, y=2, z=2, v=2

2. x=9, y=9, z=2, v=3

输出

1. eax=27, edx=0

2. eax=16, edx=2

输入样例 1

1. x=4, y=2, z=2, v=2

2. x=9, y=9, z=2, v=3

输出样例 1

1. eax=27, edx=0

2. eax=16, edx=2

任务 2.1：显示"hello,everyone!"

显示"hello,everyone!"

描述

已知msg可以表示字符串"hello,world!"的首地址，现在修改对应数组内数据，输出hello,everyone!

输入

无

输出

hello,everyone!

输入样例 1

无

输出样例 1

hello,world!

提示

用msg的地址和立即数来实现寄存器相对寻址。

可以用eax来修改，注意eax为32位可以存放4个ascii码值，比如：

mov [msg + imm], 'abcd'

任务 2.2：熟练应用变量定义和标号属性

熟练应用变量定义和标号属性

描述

读题变量定义，利用标号属性修改变量中的内容，使用循环，最终显示HUST70th!!!!

输入

无

输出

最后6个!用loop指令实现

输入样例 1

1

输出样例 1

HUST70th!!!!

提示

首先修改v_word中的h改变为H再将70th赋值给target 的前4位，再使用loop循环，连续赋值6个！

任务 3.1：去除字符串中的数字和空格

去除字符串中的数字和空格

描述

去除字符串中的数字和空格，处理后的字符串保存在原有的字符串空间中，同时将新的字符串长度输出到edx寄存器中

示例1:

字符串string=" ABC123 "，处理后string的存储内容变更为"ABC"，edx=3

示例2:

字符串string=" ABC123 abBC"，处理后string的存储内容变更为"ABCabBC"，edx=7

输入

无

输出

string=新的字符串

edx=字符串长度

输入样例 1

无

输出样例 1

string修改为"ABC"，edx赋值3

任务 3.2：计算字符串的出现次数

计算字符串的出现次数

描述

计算字符串的出现次数

给出两个字符串，string与cmpstr，计算字符串cmpstr在string中的出现次数

其中cmpstr中的字符为"ABC"，string中的字符为任意字符

字符串比较过程大小写不敏感，即字符串"ABC"与字符串"aBC"认为匹配

示例1:

string="ABC abc", cmpstr="ABC", cmpstr在string中出现2次

示例2:

string="ABC aBAbC", cmpstr="ABC", cmpstr在string中出现2次

输入

无

输出

字符串的出现次数，存放在寄存器eax中

任务 4.1：汇编子程序计算斐波那契数列(用寄存器传递参数)

汇编子程序计算斐波那契数列(用寄存器传递参数)

描述

斐波那契数列（Fibonacci sequence），又称黄金分割数列，因数学家莱昂纳多·斐波那契（Leonardo Fibonacci）以兔子繁殖为例子而引入，故又称为“兔子数列”，指的是这样一个数列：1、1、2、3、5、8、.....

斐波那契数列以如下被以递推的方法定义： $F(0)=0$ ， $F(1)=1$ ， $F(n)=F(n-1)+F(n-2)$ ($n \geq 2$ ， $n \in \mathbb{N}^*$)

请编写用寄存器eax传递入口和出口参数的形式递归子程序feibo1来计算斐波那契数列。

先在vscode环境下完成MASM版本，之后，提交部分代码到oj上进行验证。

输入

输入要获得的数列项数编号n，已被保存到eax寄存器中

输出

输出对应斐波那契数列对应第n项的值，需要保存到eax寄存器中

任务 4.2：汇编子程序计算斐波那契数列(用栈传递参数)

汇编子程序计算斐波那契数列(用栈传递参数)

描述

斐波那契数列（Fibonacci sequence），又称黄金分割数列，因数学家莱昂纳多·斐波那契（Leonardo Fibonacci）以兔子繁殖为例子而引入，故又称为“兔子数列”，指的是这样一个数列：1、1、2、3、5、8.....

在数学上，斐波那契数列可被递归公式定义为： $F(0)=0$ ， $F(1)=1$ ， $F(n)=F(n-1)+F(n-2)$ ($n \geq 2$ ， $n \in \mathbb{N}^*$)

请编写用栈传递入口和出口参数的形式的递归子程序feibo1来输出斐波那契数列。

先在vscode环境下完成MASM版本，之后，提交部分代码到oj上进行验证。

输入

输入要获得的数列项数编号n（已经被保存到栈中）

输出

输出对应斐波那契数列对应第n项的值需要压入栈中

任务 4.3：汇编子程序计算斐波那契数列(用公共变量传递参数)

汇编子程序计算斐波那契数列(用公共变量传递参数)

描述

斐波那契数列（Fibonacci sequence），又称黄金分割数列，因数学家莱昂纳多·斐波那契（Leonardo Fibonacci）以兔子繁殖为例子而引入，故又称为“兔子数列”，指的是这样一个数列：1、1、2、3、5、8、.....

斐波那契数列计算方法定义为： $F(0)=0$, $F(1)=1$, $F(n)=F(n-1)+F(n-2)$ ($n \geq 2$, $n \in \mathbb{N}^+$)

请编写用N作为入口参数和result作为出口参数的形式子程序feibo1来计算斐波那契数列。

先在vscode环境下完成MASM版本，之后，提交部分代码到oj上进行验证。

输入

输入要获得的数列项数编号n，已经被保存在公共变量N中

输出

输出对应斐波那契数列对应第n项的值，需要保存到公共变量result中

3. 实验要求

- (1) 任务 1、2、3 需要首先在 oj 平台上提交通过，在实验报告中需要填写可 masm6.x 以上版本编译通过简化段定义的完整汇编语言程序包括输入输出，**不只是子程序代码片段**。
- (2) 理解思考任务 1、2、3 中不同参数传递方式对汇编子程序编写的区别，掌握 dosbox 下调试汇编程序的方法。
- (3) 任务 4 中在不同的 C 语言开发环境中实现与汇编语言程序的混合编程，其操作方法有可能是不同的。请大家选择自己熟悉的 C 语言开发环境并查找相关的资料完成本实验。在实验报告中，详细地描述采用的开发环境及其实现方法。
- (4) 观察 C 语言编译器中对各种符号的命名规则（指编译器内部可以识别的命名规则，比如，符号名前面是否加下划线“_”等），主、子程序之间参数传递的机制，通过栈传递参数后堆栈空间回收的方法。
- (5) 对混合编程形成的执行程序，用调试工具观察由 C 语言形成的程序代码与由汇编语言形成的程序代码之间的相互关系，包括段、偏移的值，汇编指令访问 C 的变量时是如何翻译的等。
- (6) 通过本次实验，希望大家掌握汇编子程序的设计方法以及不同的编程语言是可以协同解决一个问题的，而且可以利用不同语言的特点来更好地解决问题；利用汇编语言的知识，能够更好地理解高级语言的内部处理原理与策略。

4. 实验过程

任务 1.1



The screenshot shows a debugger window with two panes. The left pane displays assembly code for a program named '1.1.asm'. The right pane shows the state of the CPU registers and memory.

Assembly Code (Left Pane):

```
1 .model small
2 .stack
3 .data
4 buffer dw ?
5 .code
6 .startup
7     mov ax,61h
8     mov [buffer],ax
9
10    mov cx, buffer ; 参数二: 要显示的字符串
11    mov dx, 2 ; 参数三: 字符串长度
12    mov bx, 1 ; 参数一: 文件描述符(stdout)
13    mov ax, 4 ; 系统调用号(sys_write)
14    int 80h ; 调用内核功能
15    ret
16 .exit 0
17 end
```

Registers and Memory (Right Pane):

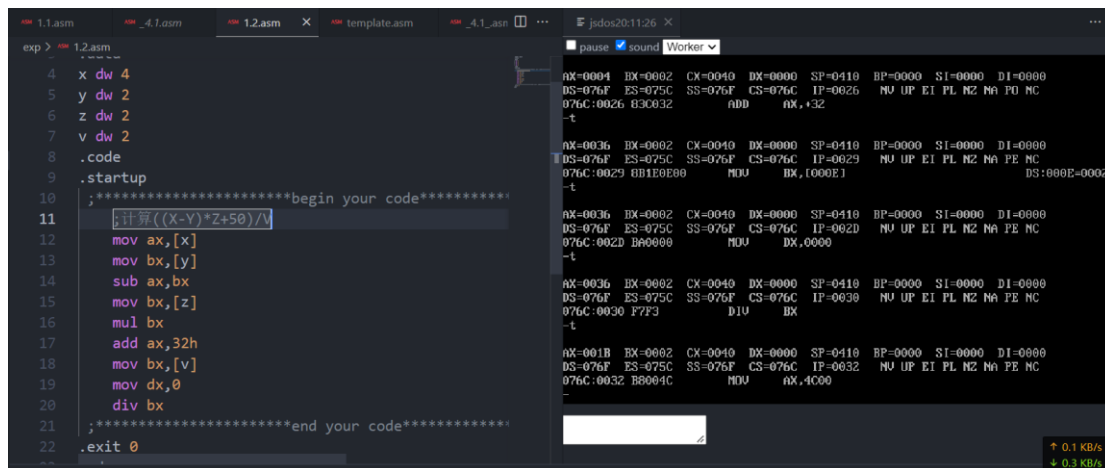
The right pane shows the state of the CPU registers and memory. The registers are listed in a table with their values and segment registers. The memory pane shows the contents of memory locations.

Register	Value
AX	0000
ECX	0010
EDX	0034
ESI	076F
EDI	0400
EIP	0000
ESI	0000
EDI	0000

非常简单, 将 61h 移入 ax, 再将 ax 移入 buffer 即可

```
.model small
.stack
.data
buffer dw ?
.code
.startup
    mov ax,61h
    mov [buffer],ax
    mov cx, buffer ; 参数二: 要显示的字符串
    mov dx, 2 ; 参数三: 字符串长度
    mov bx, 1 ; 参数一: 文件描述符(stdout)
    mov ax, 4 ; 系统调用号(sys_write)
    int 80h ; 调用内核功能
    ret
.exit 0
end
```

任务 1.2



如图可见, 结果(dx.ax)为 1BH, 即 27

.model small

.stack

.data

x dw 4

y dw 2

z dw 2

v dw 2

.code

.startup

;*****begin your code*****

;计算((X-Y)*Z+50)/V

mov ax,[x]

mov bx,[y]

sub ax,bx

mov bx,[z]

mul bx

add ax,32h

mov bx,[v]

mov dx,0

div bx

;*****end your code*****

.exit 0

end

任务 2.1


```

jsdos20:39:44 X
[ ] pause [x] sound Worker v
-t
AX=0000 BX=0010 CX=0060 DX=0771 SP=0410 BP=0000 SI=0000 DI=000F
DS=0771 ES=075C SS=0771 CS=076C IP=004A  NU UP EI PL NZ NA PE NC
076C:004A B8004C      MOV     AX,4C00
-t
AX=4C00 BX=0010 CX=0060 DX=0771 SP=0410 BP=0000 SI=0000 DI=000F
DS=0771 ES=075C SS=0771 CS=076C IP=004D  NU UP EI PL NZ NA PE NC
076C:004D CD21      INT     21
-t
AX=4C00 BX=0010 CX=0060 DX=0771 SP=040A BP=0000 SI=0000 DI=000F
DS=0771 ES=075C SS=0771 CS=F000 IP=1480  NU UP DI PL NZ NA PE NC
F000:1480 FB      STI
-d 0000
0771:0000 68 65 6C 6C 6F 2C 65 76-65 72 79 6F 6E 65 21 00  hello,everyone!.
0771:0010 B1 03 F6 F1 59 C1 E0 02-89 26 76 13 8C 16 74 13  ....Y....&v...t.
0771:0020 2E 8E 16 00 00 8B 26 8C-1F 81 2E 8C 1F 00 01 50  ....&.....P
0771:0030 EA F0 01 58 00 58 8E 16-74 13 8B 26 76 13 81 06  ...X.X..t..&v...
0771:0040 8C 1F 00 01 55 8B EC 81-66 0A 00 03 09 46 0A 5D  ....U...f....F.l
0771:0050 A1 7A 13 1F 07 55 8B EC-F7 46 06 00 02 5D 74 01  .z...U...F...lt.
0771:0060 FB CF 58 FF 36 74 13 FF-36 76 13 6A FF FF 36 78  ..X.6t..6v..j..6x
0771:0070 13 0E 68 6C 16 93 FF 36-78 13 8E 06 1C 00 26 FF  ..hl...6x.....&.

```

用 mov 修改数据即可

```
.model small
```

```
.stack
```

```
.data
```

```
msg db 'hello,world!',3 dup(?), '$'
```

```
.code
```

```
.startup
```

```
    mov di, offset msg
```

```
    add di, 6h
```

```
    mov [di], 'e'
```

```
    inc di
```

```
    mov [di], 'v'
```

```
    inc di
```

```
    mov [di], 'e'
```

```
    inc di
```

```
    mov [di], 'r'
```

```
    inc di
```

```
    mov [di], 'y'
```

```
    inc di
```

```
    mov [di], 'o'
```

```
    inc di
```

```
    mov [di], 'n'
```

```
    inc di
```

```
    mov [di], 'e'
```

```
    inc di
```

```

    mov [di], '!'
.exit 0
end

```

任务 2.2

```

-t
AX=4C00 BX=0020 CX=0000 DX=0770 SP=0420 BP=0000 SI=0000 DI=0012
DS=0770 ES=075C SS=0770 CS=076C IP=0041  NV UP EI PL NZ NA PE NC
076C:0041 CD21          INT     21
-d 0006
0770:0000                53 54-37 30 74 68 21 21 21 21          ST70th!!!!
0770:0010  21 21 BC A3 78 13 8C C0-87 46 04 5D 2D D3 12 51  ??..x....F.l-..Q
0770:0020  B1 03 F6 F1 59 C1 E0 02-89 26 76 13 8C 16 74 13  ....Y....&v...t.
0770:0030  2E 8E 16 00 00 8B 26 8C-1F 81 2E 8C 1F 00 01 50  .....&.....P
0770:0040  EA F0 01 58 00 58 8E 16-74 13 8B 26 76 13 81 06  ...X.X..t..&v...
0770:0050  8C 1F 00 01 55 8B EC 81-66 0A 00 03 09 46 0A 5D  ....U...f....F.l
0770:0060  A1 7A 13 1F 07 55 8B EC-F7 46 06 00 02 5D 74 01  .z...U...F...lt.
0770:0070  FB CF 58 FF 36 74 13 FF-36 76 13 6A FF FF 36 78  ..X.6t..6v.j..6x
0770:0080  13 0E 68 6C 16 93          ..hl..
-d 0000
0770:0000  4C CD 21 00 48 55 53 54-37 30 74 68 21 21 21 21  L.!.HUST70th!!!!
0770:0010  21 21 BC A3 78 13 8C C0-87 46 04 5D 2D D3 12 51  ??..x....F.l-..Q
0770:0020  B1 03 F6 F1 59 C1 E0 02-89 26 76 13 8C 16 74 13  ....Y....&v...t.
0770:0030  2E 8E 16 00 00 8B 26 8C-1F 81 2E 8C 1F 00 01 50  .....&.....P
0770:0040  EA F0 01 58 00 58 8E 16-74 13 8B 26 76 13 81 06  ...X.X..t..&v...
0770:0050  8C 1F 00 01 55 8B EC 81-66 0A 00 03 09 46 0A 5D  ....U...f....F.l
0770:0060  A1 7A 13 1F 07 55 8B EC-F7 46 06 00 02 5D 74 01  .z...U...F...lt.
0770:0070  FB CF 58 FF 36 74 13 FF-36 76 13 6A FF FF 36 78  ..X.6t..6v.j..6x

```

mov 赋值即可

```

.model small
.stack
.data
v_word db 'hUST'
target db 10 dup(0)
.code
.startup
    mov di, offset v_word
    sub [di], 20h
    mov di, offset target
    mov [di], '7'
    inc di
    mov [di], '0'
    inc di
    mov [di], 't'
    inc di
    mov [di], 'h'
    inc di

```

```

    mov cx, 6
    mov al, '!'
lp:
    mov [di], al
    inc di
    loop lp
.exit 0
end

```

任务 3.1

The screenshot shows a debugger window with the following assembly code and memory dump:

```

-t
AX=0241 BX=0020 CX=0000 DX=0006 SP=0420 BP=0000 SI=0000 DI=0005
DS=0770 ES=075C SS=0770 CS=076C IP=0041  MOV  UP EI PL NZ NA PE NC
076C:0041 B8004C      MOV     AX,4C00
-t
AX=4C00 BX=0020 CX=0000 DX=0006 SP=0420 BP=0000 SI=0000 DI=0005
DS=0770 ES=075C SS=0770 CS=076C IP=0044  MOV  UP EI PL NZ NA PE NC
076C:0044 CD21      INT     21
-t
AX=4C00 BX=0020 CX=0000 DX=0006 SP=041A BP=0000 SI=0000 DI=0005
DS=0770 ES=075C SS=0770 CS=F000 IP=1480  MOV  UP DI PL NZ NA PE NC
F000:1480 FB      STI
-d 0000
0770:0000 00 B8 00 4C CD 21 41 42-43 61 62 42 43 20 41 42 ...L.tABCabBC AB
0770:0010 43 31 32 33 20 20 20 20-20 20 61 62 42 43 24 51 C123 abBC$Q
0770:0020 B1 03 F6 F1 59 C1 E0 02-09 26 76 13 8C 16 74 13 ....Y....&v...t.
0770:0030 2E 8E 16 00 00 8B 26 8C-1F 81 2E 8C 1F 00 01 50 .....8.....P
0770:0040 EA F0 01 58 00 58 8E 16-74 13 8B 26 76 13 81 06 ...X.X..t..&v...
0770:0050 8C 1F 00 01 55 8B EC 81-66 0A 00 03 09 46 0A 5D ....U...f....F.l
0770:0060 A1 7A 13 1F 07 55 8B EC-F7 46 06 00 02 5D 74 01 .z...U...F...lt.
0770:0070 FB CF 58 FF 36 74 13 FF-36 76 13 6A FF FF 36 78 ..X.6t..6v.j..6x

```

思路:遍历 string, 如果遇到大小写字母就入栈, 计数+1, 遇到'\$'结束, 最后再根据计数逐个出栈对 string 赋值, 实现去空格和数字的功能

```

.model small
.stack
.data
string db "    ABC123    abBC"
    db '$'
length equ $-string
.code
.startup
    mov dx, 0
    mov cx, length
    mov di, offset string
eliminate:
    mov ax, 0

```

```

mov al, [di]
inc di
cmp al, 40h
js lp
inc dx
push ax
lp: loop eliminate
    mov cx, dx
    mov di, offset string
    add di, dx
    dec di
restore:
    pop ax
    mov [di], al
    dec di
    loop restore
mov ah, 02h
mov dx, offset string
.exit 0
end

```

任务 3.2

The screenshot shows a DOSBox emulator window with the title 'jsdos20:50:12'. The interface includes a 'pause' button, a 'sound' checkbox (checked), and a 'Worker' dropdown menu. The main display area shows assembly code and register values. The registers are: DS=0772, ES=075C, SS=0772, CS=076C, IP=005D, NU UP EI PL ZR NA PE NC. The code is: 076C:005D 40 INC AX. The next line shows: AX=0002 BX=0043 CX=0001 DX=0043 SP=0420 BP=0000 SI=000F DI=000C. The code continues: DS=0772 ES=075C SS=0772 CS=076C IP=005E NU UP EI PL NZ NA PO NC. The next line shows: 076C:005E 47 INC DI. The code continues: AX=0002 BX=0043 CX=0001 DX=0043 SP=0420 BP=0000 SI=000F DI=000D. The code continues: DS=0772 ES=075C SS=0772 CS=076C IP=005F NU UP EI PL NZ NA PO NC. The next line shows: 076C:005F E2C8 LOOP 0029. The code continues: -t. The next line shows: AX=0002 BX=0043 CX=0000 DX=0043 SP=0420 BP=0000 SI=000F DI=000D. The code continues: DS=0772 ES=075C SS=0772 CS=076C IP=0061 NU UP EI PL NZ NA PO NC. The next line shows: 076C:0061 B8004C MOV AX, 4C00. The code continues: -d 0000. The next line shows: 0772:0000 C8 B8 00 4C CD 21 41 42-43 20 61 42 41 62 43 41 ...L.!ABC aBAbCA. The code continues: 0772:0010 42 43 16 00 00 8B 26 BC-1F 81 2E 8C 1F 00 01 50 BC....&.....P. The code continues: 0772:0020 EA F0 01 58 00 58 8E 16-74 13 8B 26 76 13 81 06 ...X.X..t..&v... The code continues: 0772:0030 8C 1F 00 01 55 8B EC 81-66 0A 00 03 09 46 0A 5DU...f....F.. The code continues: 0772:0040 A1 7A 13 1F 07 55 8B EC-F7 46 06 00 02 5D 74 01 .z...U...F...lt. The code continues: 0772:0050 FB CF 58 FF 36 74 13 FF-36 76 13 6A FF FF 36 78 ..X.6t..6v..j..6x The code continues: 0772:0060 13 0E 68 6C 16 93 FF 36-78 13 8E 06 1C 00 26 FF ..hl...6x.....&. The code continues: 0772:0070 77 02 26 FF 37 93 A1 7A-13 1E 07 CF 2E 8E 1E 00 w.&.7..z.....

如图:string="ABC aBAbC", cmpstr="ABC", cmpstr 在 string 中出现 2 次, 即 ax=2

思路:先比较第一个字符, 如果成功则跳转到比较第二个的代码, 第三个同理, 否则向前移动, 比较下一个字符

```
.model small
.stack
.data
string db "ABC aBAbC"
slength equ $-string
cmpstr db 'ABC'
cmpslength equ $-cmpstr
.code
.startup
mov cx, slength
sub cx, 2
mov si, offset cmpstr
mov di, offset string
xor ax, ax
xor bx, bx
xor dx, dx
com:
mov bl, [di]
cmp bl, 60h
js upper1
sub bl, 20h
upper1:
mov dl, ds:[si]
cmp bl, dl
jnz fal
mov bl, ds:[di+1]
cmp bl, 60h
js upper2
sub bl, 20h
upper2:
mov dl, ds:[si+1]
cmp bl, dl
jnz fal
mov bl, ds:[di+2]
cmp bl, 60h
js upper3
sub bl, 20h
upper3:
mov dl, ds:[si+2]
cmp bl, dl
jnz fal
inc ax
```

```

fal:
inc di
loop com
.exit 0
end

```

任务 4.1

```

jsdos20:55:44
[ ] pause [x] sound Worker v
AX=0001 BX=0010 CX=0004 DX=0002 SP=040A BP=0000 SI=0000 DI=0000
DS=0770 ES=075C SS=0770 CS=076C IP=0017  NU UP EI NG NZ AC PE CY
076C:0017 03C2      ADD     AX,DX
-t
AX=0003 BX=0010 CX=0004 DX=0002 SP=040A BP=0000 SI=0000 DI=0000
DS=0770 ES=075C SS=0770 CS=076C IP=0019  NU UP EI PL NZ NA PE NC
076C:0019 5A      POP     DX
-t
AX=0003 BX=0010 CX=0004 DX=0770 SP=040C BP=0000 SI=0000 DI=0000
DS=0770 ES=075C SS=0770 CS=076C IP=001A  NU UP EI PL NZ NA PE NC
076C:001A 59      POP     CX
-t
AX=0003 BX=0010 CX=0042 DX=0770 SP=040E BP=0000 SI=0000 DI=0000
DS=0770 ES=075C SS=0770 CS=076C IP=001B  NU UP EI PL NZ NA PE NC
076C:001B C3      RET
-t
AX=0003 BX=0010 CX=0042 DX=0770 SP=0410 BP=0000 SI=0000 DI=0000
DS=0770 ES=075C SS=0770 CS=076C IP=003D  NU UP EI PL NZ NA PE NC
076C:003D B8004C  MOV     AX,4C00
-

```

寄存器传参, 这是第 4 项的结果, ax=3

```

.model small
.stack
.data
.code
;寄存器传参
feibo1 proc
;比较特殊情况
cmp ax, 3
jl LessThanThree
;计算
push cx
push dx
mov cx, ax
dec ax

```

```

call feibo1 ;前一项
mov dx, ax
mov ax, cx
sub ax, 2
call feibo1 ;前第二项
add ax, dx
pop dx
pop cx
ret
;小于 3 直接输出 1
LessThanThree:
mov ax, 1
ret
feibo1 endp
.startup
mov ax, 4
call feibo1

.exit 0
end

```

任务 4.2

```

... jsdos21:00:59 X
[ ] pause [x] sound Worker v
AX=0005 BX=0010 CX=0055 DX=0771 SP=040A BP=0000 SI=0000 DI=0000
DS=0771 ES=075C SS=0771 CS=076C IP=002F  MV UP EI PL NZ NA PE NC
076C:002F 5B          POP     AX
-t
AX=0005 BX=0010 CX=0055 DX=0771 SP=040C BP=0000 SI=0000 DI=0000
DS=0771 ES=075C SS=0771 CS=076C IP=0030  MV UP EI PL NZ NA PE NC
076C:0030 C3          RET
-t
AX=0005 BX=0010 CX=0055 DX=0771 SP=040E BP=0000 SI=0000 DI=0000
DS=0771 ES=075C SS=0771 CS=076C IP=004F  MV UP EI PL NZ NA PE NC
076C:004F 5B          POP     AX
-t
AX=0005 BX=0010 CX=0055 DX=0771 SP=0410 BP=0000 SI=0000 DI=0000
DS=0771 ES=075C SS=0771 CS=076C IP=0050  MV UP EI PL NZ NA PE NC
076C:0050 B8004C      MOV     AX,4C00
-t
AX=4C00 BX=0010 CX=0055 DX=0771 SP=0410 BP=0000 SI=0000 DI=0000
DS=0771 ES=075C SS=0771 CS=076C IP=0053  MV UP EI PL NZ NA PE NC
076C:0053 CD21      INT     21
-t

```

↑ 0.4 KB/s
↓ 3.5 KB/s

堆栈传参, 关键需要找对参数相对 sp 的偏移位置即可, 记得+IP, 此处是计算第 5 项的结

果, ax 从栈中 pop 得到 5

.model small

.stack

.data

.code

;栈传参

feibo1 proc

;比较特殊情况

push ax

push bp

push cx

push dx

mov bp, sp

mov ax, [bp+10]

cmp ax, 3

jl LessThanThree

;计算

mov cx, ax

dec ax

push ax

call feibo1 ;前一项

pop ax

mov dx, ax

mov ax, cx

sub ax, 2

push ax

call feibo1 ;前第二项

pop ax

add ax, dx

jmp done

;小于 3 直接输出 1

LessThanThree:

mov ax, 1

done:

mov [bp+10], ax

pop dx

pop cx

pop bp

pop ax

ret

feibo1 endp

.startup


```

mov ax, 5
push ax
call feibo1
pop ax

```

```

.exit 0
end

```

任务 4.3

```

AX=0000 BX=0010 CX=005A DX=0771 SP=040E BP=0000 SI=0000 DI=0000
DS=0771 ES=075C SS=0771 CS=076C IP=0033  MV UP EI PL NZ NA PE NC
076C:0033 C3          RET
-t

AX=0000 BX=0010 CX=005A DX=0771 SP=0410 BP=0000 SI=0000 DI=0000
DS=0771 ES=075C SS=0771 CS=076C IP=004E  MV UP EI PL NZ NA PE NC
076C:004E A10800      MOV     AX,[0008]          DS:0008=0003
-t

AX=0003 BX=0010 CX=005A DX=0771 SP=0410 BP=0000 SI=0000 DI=0000
DS=0771 ES=075C SS=0771 CS=076C IP=0051  MV UP EI PL NZ NA PE NC
076C:0051 B8004C      MOV     AX,4C00
-d 0008
0771:0000                03 00 04 5D 2D D3 12 51          ...]-..Q
0771:0010 B1 03 F6 F1 59 C1 E0 02-89 26 76 13 BC 16 74 13  ....Y....&v...t.
0771:0020 2E 8E 16 00 00 8B 26 8C-1F 81 2E 8C 1F 00 01 50  ....&.....P
0771:0030 EA F0 01 58 00 58 BE 16-74 13 8B 26 76 13 81 06  ...X.X..t..&v...
0771:0040 8C 1F 00 01 55 8B EC 81-66 0A 00 03 09 46 0A 5D  ....U...f....F..l
0771:0050 A1 7A 13 1F 07 55 8B EC-F7 46 06 00 02 5D 74 01  .z...U...F...lt.
0771:0060 FB CF 58 FF 36 74 13 FF-36 76 13 6A FF FF 36 78  ..X.6t..6v..j..6x
0771:0070 13 0E 68 6C 16 93 FF 36-78 13 8E 06 1C 00 26 FF  ..hl...6x....&.
0771:0080 77 02 26 FF 37 93 A1 7A                w.&.7..z

```

公共变量传参, 如图为 $n=4$ 的结果, $result=3$

```

.model small
.stack
.data
N dw 4
result dw ?
.code
;公共变量
feibo1 proc
push ax
push cx
push dx
mov ax, N

```

```

;比较特殊情况
cmp ax, 3
jl LessThanThree
;计算
mov cx, ax
dec ax
mov N, ax
call feibo1 ;前一项的值
mov dx, result
mov ax, cx
sub ax, 2
mov N, ax
call feibo1 ;前第二项的值
mov ax, result
add ax, dx
jmp done
;小于 3 直接输出 1
LessThanThree:
mov ax, 1
done:
mov result, ax
pop dx
pop cx
pop ax
ret
feibo1 endp

.startup
call feibo1
mov ax, result

.exit 0
end

```

5. 总结与体会

在 oj 上无法调试, 做汇编实验十分考验对知识的掌握运用能力, 以及对内存的熟悉程度, 同时 oj 环境是 32 位, 也考验了我们对所学知识的迁移类比能力, 最后三题斐波那契递归程序比较有难度, 需要综合所学才能做出来.