**华 中 科 技 大 学**

**网络空间安全学院**

《汇编语言程序设计实验》

实验报告

姓 名

班 级

学 号

联系方式

分 数

评 分 人

2025年 1月 14日

**报告要求**

**1. 报告不可以抄袭，发现雷同者记为0分。**

**2. 报告中不可以只粘贴大段代码，应是文字与图、表结合的，需要说明流程的时候，也应该用流程图或者伪代码来说明；如果发现有大段代码粘贴者，报告需重写。**

**3. 报告格式严格按照要求规范，并作为评分标准。**

目 录

[1 实验概述 1](#_Toc187844993)

[1.1 实验名称 1](#_Toc187844994)

[1.1 实验目的 1](#_Toc187844995)

[1.2 实验环境 1](#_Toc187844996)

[1.3 实验内容 1](#_Toc187844997)

[实验说明 1](#_Toc187844998)

[1.4 实验要求 2](#_Toc187844999)

[2 实验过程 3](#_Toc187845000)

[2.1 程序结构设计 3](#_Toc187845001)

[2.2 详细设计 5](#_Toc187845002)

[2.3 代码实现 6](#_Toc187845003)

[2.4 任务4 16](#_Toc187845004)

[3 测试与分析 18](#_Toc187845005)

[3.1 测试及结果说明 18](#_Toc187845006)

[3.2 遇到的问题及解决方法 20](#_Toc187845007)

[3.3 设计方案存在的不足 21](#_Toc187845008)

[4 实验总结 22](#_Toc187845009)

[4.1 实验感想 22](#_Toc187845010)

[4.2 意见和建议 22](#_Toc187845011)

# 实验概述

## 实验名称

汇编语言程序设计实验。

## 实验目的

1. 熟悉分支、循环程序的结构及控制方法，掌握分支、循环程序的调试方法；
2. 加深对转移指令及一些常用的汇编指令的理解；
3. 熟悉递归子程序编写基本格式和参数传递方式。
4. 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。

## 实验环境

操作系统：Windows/DOSBox

编程语言：Intel 8086 16位汇编

## 实验内容

## 实验说明

设计汇编递归子程序程序来实现如下数列计算，输入两个正整数num1和num2，输出num1和num2的最大公因数。数列以如下被以递推的方法定义：

当num2 >0 时 GCD(num1, num2) = GCD(num2, num1%num2) ；

当 num2=0 时，GCD(num1,num2)=num1

**任务1：****子程序计算递推公式(用寄存器传递参数)**

请编写vscode环境下完整的MASM可编译通过的汇编程序，其中需要使用**寄存器ax和bx传递入口参数和ax为出口参数**的形式递归子程序来计算最大公因数。

输入通过dos系统调用获取，输出需要通过dos系统调用**输出显示为16进制数。**

**任务2：****子程序计算递推公式(用栈传递参数)**

请编写vscode环境下完整的MASM可编译通过的汇编程序，其中需要使用**堆栈传递入口和出口参数的形式**的递归子程序来计算最大公因数。

输入通过dos系统调用获取，输出需要通过dos系统调用**输出显示为10进制数。**

**任务3：****子程序计算递推公式(用公共变量传递参数)**

请编写vscode环境下完整的MASM可编译通过的汇编程序，其中需要**使用公共变量num1和num2作为入口和出口参数**的形式子程序来计算最大公因数。

输入通过dos系统调用获取，输出需要通过dos系统调用**输出显示为8进制数。**

**任务4：子程序计算递推公式(混合编程)**

请用汇编语言编写实现GCD递推公式的子程序，对入口和出口参数形式不做要求，但需要用C语言函数来获取输入、调用汇编递推子程序，并且用C语言显示子程序返回的结果。

## 实验要求

（1）任务1-3采用Intel 16位汇编语言编写，任务4使用C语言和32位汇编语言编写；

（2）提交实验设计报告和源代码；实验设计报告应按照实验报告模板，核心内容须包括程序流程图，**源代码必须加注释**。

（3）实验设计报告按照文档规范撰写。

（4）任务1、2、3需要首先在线上平台上提交通过，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，**不只是上机提交的子程序代码片段。**

（5）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。

（6）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。请大家选择白己熟悉的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。

（7）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。

# 实验过程

## 程序结构设计

### **用寄存器传递参数**

1.子程序功能说明

Read 子程序：从键盘读取一个整数，并将其转换为二进制形式存储在 ax 寄存器中。

实现细节：通过循环读取键盘输入的字符，判断是否为数字字符。如果是，则将其转换为二进制形式并累加到 bx 寄存器中。最后将 bx 的值赋给 ax 并返回。

GCD 子程序：使用欧几里得算法计算两个整数的最大公约数，并将结果存储在 num1 中。

输入：num1 和 num2 中存储要计算的两个整数。输出：num1 中存储计算得到的最大公约数。

print\_num 子程序：将 dx 寄存器中的整数以十进制形式输出到屏幕。

实现细节：通过循环将 dx 中的整数逐位转换为 ASCII 码并压入栈中，然后依次弹出栈中的字符并通过 int 21h 的 02h 功能输出到屏幕。

2. 输入数据完整的处理流程

初始化数据段：程序开始时，将数据段寄存器 ds 初始化为指向数据段。

读取第一个整数：调用 Read 子程序，从键盘读取第一个整数，并将其存储在 num1 中。

读取第二个整数：再次调用 Read 子程序，从键盘读取第二个整数，并将其存储在 num2 中。

计算最大公约数：调用 GCD 子程序，使用欧几里得算法计算 num1 和 num2 的最大公约数，并将结果存储在 num1 中。

输出结果：将 num1 的值赋给 dx，调用 print\_num 子程序，将 dx 中的整数以十进制形式输出到屏幕。

程序结束：使用 int 21h 的 4Ch 功能，返回操作系统。

### 用栈传递参数

**1.子程序功能说明**

**Read 子程序**：从键盘读取一个整数并将其转换为二进制形式。

**实现方式**：通过循环不断读取键盘输入字符（ah = 1 及 int 21h），判断字符是否在 '0' 到 '9' 之间。如果是，则将其转换为数值并通过移位和加法操作累加到 bx 中。最后将 bx 的值赋给 ax 并返回。

**GCD 子程序**：使用欧几里得算法计算两个整数的最大公约数。

**输入**通过栈传递的两个整数。**输出**计算得到的最大公约数在栈顶位置（调用结束后通过栈平衡恢复现场）。

**print\_num 子程序**：将 ax 寄存器中的整数以十进制形式输出到屏幕。

**实现方式**：通过循环将 ax 不断除以 10 取余数，将余数加上 '0' 转换为字符后压入栈，直到 ax 为 0。然后通过循环从栈中弹出字符，使用 ah = 02h 及 int 21h 将字符输出到屏幕。

### 用公共变量传递参数

**1.子程序功能说明**

**Read 子程序**：从键盘读取一个整数，并将其转换为二进制形式存储在 ax 寄存器中。

**实现细节**：使用 int 21h 的 1 号功能调用逐字符读取键盘输入。判断每个字符是否为数字字符（'0' 到 '9'），若是，则将其转换为数值并通过移位和加法操作累加到 bx 寄存器中。例如，shl bx, 1 左移 bx 一位，相当于乘以 2，再通过 mov dx, bx 和 shl bx, 1 以及 shl bx, 1 实现 bx 乘以 8，然后 add bx, dx 相当于 bx 乘以 10，最后 add bx, ax 将当前字符对应的数值加到 bx 中。读取完成后，将 bx 的值赋给 ax 并恢复寄存器现场。

**GCD 子程序**：使用欧几里得算法计算 num1 和 num2 两个整数的最大公约数，并将结果存储在 num1 中。

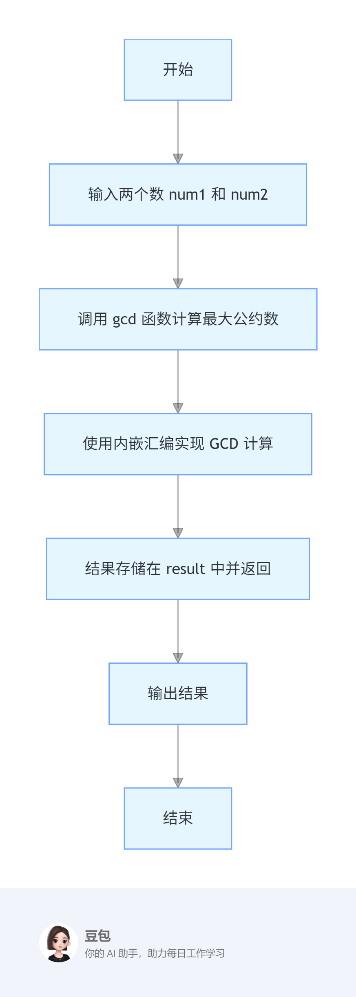
**输入**存储在 num1 和 num2 中的两个整数。**输出**计算得到的最大公约数存储在 num1 中。

**print\_num 子程序**：将 dx 寄存器中的整数以八进制形式输出到屏幕。

**实现细节**：通过循环将 dx 中的整数逐位转换为八进制对应的 ASCII 码字符。每次循环将 dx 除以 8（mov bx, 8 和 div bx），将余数加上 '0' 转换为 ASCII 码字符后压入栈（add dx, '0' 和 push dx），直到 dx 为 0。然后通过另一个循环从栈中弹出字符，并使用 int 21h 的 2 号功能调用输出到屏幕。

### 混合编程

1.功能模块框图



2.子程序功能说明

gcd 函数：使用内嵌汇编语言实现的计算两个整数最大公约数（GCD）的函数，采用递归的欧几里得算法。输入num1，num2，输出result。

## 详细设计

关于GCD算法，所有程序都使用了欧几里得算法（辗转相除法）计算最大公约数：如果b=0，则a就是最大公约数；否则，GCD(a,b) = GCD(b, a mod b)。

### 寄存器实现

实现细节：通过比较 num2 是否为零，如果为零则返回 num1 作为最大公约数。否则，用 num1 除以 num2，将余数赋给 num2，将 num2 的值赋给 num1，然后递归调用 GCD 子程序。

数据结构：纯寄存器操作

ax存储第一个数，bx存储第二个数；递归调用时直接使用寄存器传递参数

### 栈实现

**实现方式**：首先通过 bp 寄存器指向栈顶，获取栈中传递的两个整数。然后比较第二个数（bx）是否为零，如果为零则直接返回第一个数（此时第一个数即为最大公约数）。否则，用第一个数（ax）除以第二个数（bx），将余数（在 dx 中）和除数（bx）压入栈，递归调用 GCD 子程序。递归结束后，通过 ret 4 平衡栈，恢复调用前的栈状态。

数据结构：使用栈进行参数传递

参数通过栈传递和恢复；更接近现代编程中的函数调用方式

### 公共变量实现

**实现细节**：首先检查 num2 是否为 0，如果是，则直接返回，此时 num1 即为最大公约数（je zero）。否则，用 num1 除以 num2（div num2），将 num2 的值赋给 num1，余数赋给 num2，然后递归调用 GCD 子程序。

数据结构：使用内存变量存储数据

num1和num2在数据段中定义；通过内存变量在递归调用间传递数据

### 内联汇编实现

实现细节：首先将 num1 存储在 eax 寄存器，将 num2 存储在 ebx 寄存器。进入 gcd\_loop 标签：比较 ebx 和 0，如果 ebx 为 0，跳转到 gcd\_done。清空 edx 寄存器，将 eax 除以 ebx，商存储在 eax 中，余数存储在 edx 中。将 ebx 的值赋给 eax，将 edx 的值赋给 ebx，实现 num1 = num2 和 num2 = num1 % num2 的操作。递归调用 gcd\_loop。最终结果存储在 eax 中，将 eax 的值存储到 result 变量。

数据结构：C语言函数框架下的寄存器操作

使用32位寄存器(eax, ebx, edx)；参数通过C函数参数传入；结果通过C变量返回

## 代码实现

### 任务1

.model

.stack

.data

num1 dw ? ; 输入的第一个整数 (a)

num2 dw ? ; 输入的第二个整数 (b)

.code

main:

mov ax, @data

mov ds, ax ; 初始化数据段

call read

mov num1, ax

call read

mov num2, ax

; 加载输入的两个整数

mov ax, num1 ; a -> ax

mov bx, num2 ; b -> bx

call GCD ; 调用 GCD 子程序

;;//mov dl, al ; 将结果保存到dx

call print\_num

; 程序结束

mov ah, 4Ch

int 21h

GCD proc

cmp bx,0 ; 检查除数是否为0

je zero ; 如果为0，说明上一个除数就是最大公约数

xor dx,dx ; 清空dx寄存器，为div准备32位被除数(dx:ax)

div bx ; ax除以bx，商在ax中，余数在dx中

mov ax, bx ; ax = 原来的除数(b)

mov bx, dx ; bx = 余数(a mod b)

call GCD ; 递归调用：GCD(b, a mod b)

zero: ; 递归基准情况

ret ; 此时ax中包含最大公约数

GCD endp

print\_num proc

xor cx,cx

store: mov bx,16

xor dx,dx

div bx

cmp dl,9 ; 比较余数和9

jbe digit ; 如果小于等于9，直接加'0'

add dl,7 ; 如果大于9，需要额外加7来得到'A'-'F'

digit: add dl,'0' ; 转换为ASCII

push dx

inc cx

cmp ax,0

jne store

mov ah,02h

output:

pop dx

int 21h

loop output

ret

print\_num endp

Read proc

push bx

push cx

push dx

xor bx,bx

xor cx,cx

read1: mov ah,1

int 21h

cmp al,'0'

jb read4

cmp al,'9'

ja read4

sub al,30h

shl bx,1

mov dx,bx

shl bx,1

shl bx,1

add bx,dx

mov ah,0

add bx,ax

jmp read1

read4: mov ax,bx

pop dx

pop cx

pop bx

ret

Read endp

end main

### 任务2

.model small

.stack

.code

main:

mov ax, @data

mov ds, ax ; 初始化数据段

call read

push ax

call read

push ax

call GCD ; 调用 GCD 子程序

call print\_num

; 程序结束

mov ah, 4Ch

int 21h

GCD proc

mov bp,sp ; 设置栈基址指针

mov ax, [bp+4] ; 获取第一个参数(a)，栈中较高地址

mov bx, [bp+2] ; 获取第二个参数(b)，栈中较低地址

cmp bx,0 ; 检查b是否为0

je zero ; b=0时跳转

xor dx,dx ; 清空dx，准备除法

div bx ; ax/bx，商在ax，余数在dx

push bx ; 将b压入栈作为下一次递归的第一个参数

push dx ; 将余数(a mod b)压入栈作为第二个参数

call GCD ; 递归调用：GCD(b, a mod b)

zero:

ret 4 ; 返回并清理栈上的两个参数(2个word = 4字节)

GCD endp

print\_num proc

;;//ax存要十进制表示的数，ax/10余数+‘0’入栈，直到ax为0.dx存余数ax存商

xor cx,cx

;mov ax, dx

store: mov bx,10

xor dx,dx

div bx

add dx, '0'

push dx

inc cx

cmp ax,0

jne store

output:

mov ah, 02h

pop dx

int 21h

loop output

ret

print\_num endp

Read proc

push bx

push cx

push dx

xor bx,bx

xor cx,cx

read1: mov ah,1

int 21h

cmp al,'0'

jb read4

cmp al,'9'

ja read4

sub al,30h

shl bx,1

mov dx,bx

shl bx,1

shl bx,1

add bx,dx

mov ah,0

add bx,ax

jmp read1

read4: mov ax,bx

pop dx

pop cx

pop bx

ret

Read endp

end main

### 任务3

.model small

.stack

.data

num1 dw ? ; 输入的第一个整数 (a)

num2 dw ? ; 输入的第二个整数 (b)

.code

main:

mov ax, @data

mov ds, ax ; 初始化数据段

call read

mov num1, ax

call read

mov num2, ax

call GCD ; 调用 GCD 子程序

mov dx, num1 ; 将结果保存到dx

call print\_num

; 程序结束

mov ah, 4Ch

int 21h

GCD proc

cmp num2,0 ; 检查num2(b)是否为0

je zero ; b=0时跳转

xor dx,dx ; 清空dx准备除法

mov ax, num1 ; ax = num1(a)

div num2 ; ax除以num2，商在ax，余数在dx

mov cx, num2 ; 临时保存num2的值

mov num1, cx ; num1 = 原来的num2(b)

mov num2, dx ; num2 = 余数(a mod b)

call GCD ; 递归调用：GCD(b, a mod b)

zero:

ret ; num1中包含最大公约数

GCD endp

print\_num proc

xor cx,cx

mov ax, dx

store: mov bx,8

xor dx,dx

div bx

add dx, '0' ; 转换为ASCII码

push dx

inc cx

cmp ax,0

jne store

mov ah, 02h

output:

pop dx

int 21h

loop output

ret

print\_num endp

Read proc

push bx

push cx

push dx

xor bx,bx

xor cx,cx

read1: mov ah,1

int 21h

cmp al,'0'

jb read4

cmp al,'9'

ja read4

sub al,30h

shl bx,1

mov dx,bx

shl bx,1

shl bx,1

add bx,dx

mov ah,0

add bx,ax

jmp read1

read4: mov ax,bx

pop dx

pop cx

pop bx

ret

Read endp

end main

## 任务4

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

// GCD 函数，使用内嵌汇编实现（递归版）

int gcd(int num1, int num2) {

int result;

\_\_asm {

mov eax, num1 ; eax = 第一个数(a)

mov ebx, num2 ; ebx = 第二个数(b)

gcd\_loop: ; 递归的循环标签

cmp ebx, 0 ; 比较b是否为0

je gcd\_done ; 如果b=0，跳转到完成

xor edx, edx ; 清空edx，准备32位除法

div ebx ; eax/ebx，商在eax，余数在edx

mov eax, ebx ; eax = 原来的除数(b)

mov ebx, edx ; ebx = 余数(a mod b)

call gcd\_loop ; 递归调用：GCD(b, a mod b)

gcd\_done: ; 计算完成的标签

mov result, eax ; 保存结果到C变量

}

return result; ; 返回最大公约数

}

int main() {

int num1, num2, result;

printf("Enter two numbers: ");

scanf("%d %d", &num1, &num2);

// 调用汇编程序

result = gcd(num1, num2);

printf("The GCD of %d and %d is %d\n", num1, num2, result);

return 0;

}

# 测试与分析

## 测试及结果说明

### 测试环境

硬件环境：

CPU: x86架构处理器

内存: ≥ 4MB

操作系统支持16位DOS程序执行

软件环境：

操作系统: Windows 11（通过DOSBox模拟）

汇编器: MASM 6.11

链接器: LINK 5.60

C编译器: VSCODE（用于4.c）

DOSBox 0.74-3版本

### 测试及结果

### 任务1-3测试及结果



图1 任务1测试结果

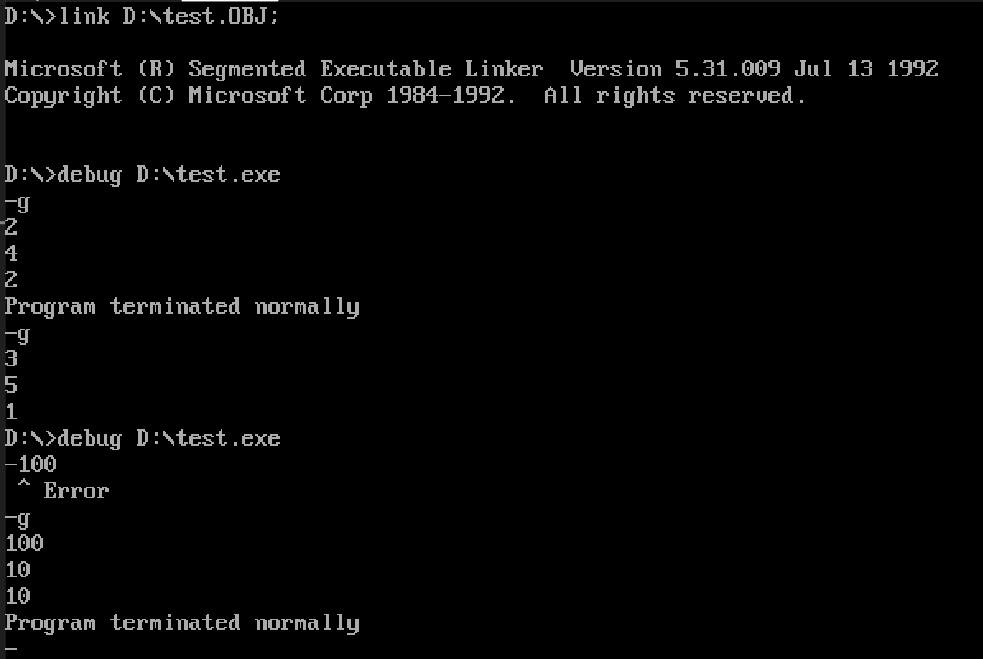


图2 任务2测试结果

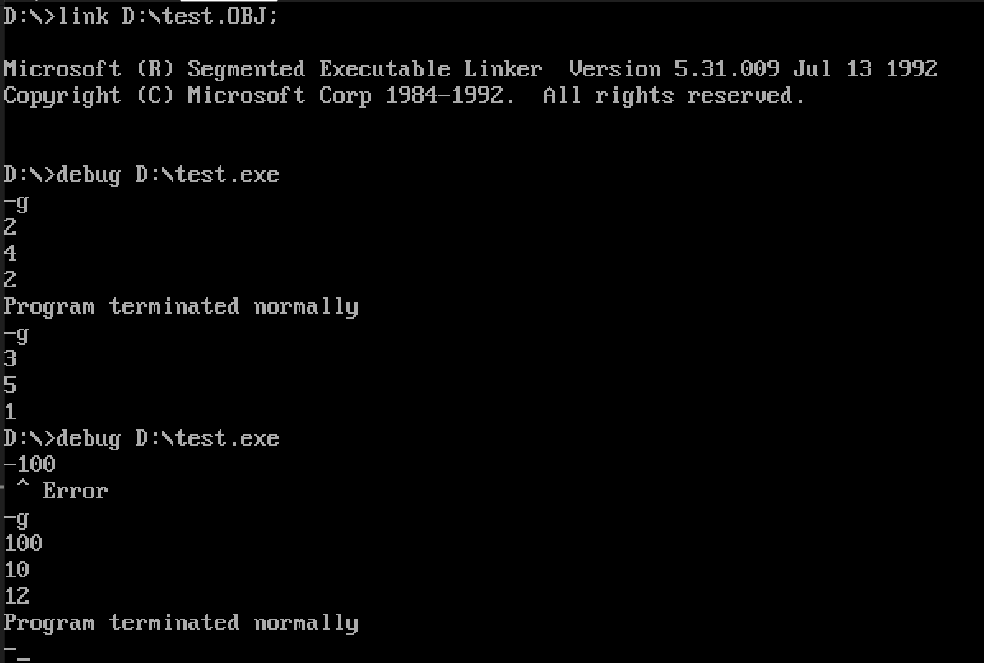


图3任务3测试结果

#### 任务4测试及结果

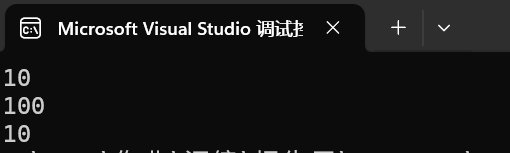
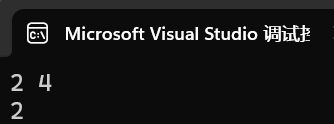
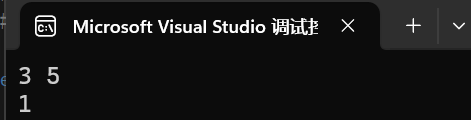


图4-6 任务4测试及结果

## 遇到的问题及解决方法

在完成这次汇编语言实验的过程中，我遇到了一些值得记录的问题。通过查阅资料和请教老师同学，我最终找到了解决方案，这些经验对我来说非常宝贵。

首先是编译环境的配置问题。由于我的是Windows系统直接运行MASM无法编译16位的汇编代码。经过查找资料，我了解到需要使用DOSBox模拟器来解决这个问题。我安装了DOSBox，并将MASM、LINK等工具复制到指定目录下，然后编写了一个批处理文件来自动挂载。这样不仅解决了编译问题，还大大提高了开发效率。

另一个困扰我的是递归实现导致的栈溢出问题。在测试较大的数据时（比如计算1989和1590的最大公约数），程序偶尔会出现栈溢出的情况。起初我以为是算法问题，后来发现是栈空间声明不足导致的。我通过增加栈空间声明从.stack 100h改为.stack 200h解决了这个问题。不过，这个问题也让我思考到，在实际应用中，使用循环可能比递归更可靠，因此我还尝试了用循环来重写GCD算法。

最后一个问题是输出格式的混乱。由于三个汇编版本分别使用了十六进制、十进制和八进制输出，在调试过程中经常会混淆。为了解决这个问题，我在输出函数中添加了进制标识符，在数字前后添加括号和进制说明。这个小改动大大提高了程序的可读性，也方便了调试过程。

## 设计方案存在的不足

首先是输入验证的缺失。目前的程序对用户输入的数据缺乏有效的检查机制，这可能导致程序在处理非法输入时出现意外情况。例如，当输入超出16位整数范围或输入非数字字符时，程序可能会产生错误结果或直接崩溃。为了提高程序的健壮性，我们应该在read过程中添加输入合法性检查，包括数值范围验证和字符类型判断，并在发现非法输入时给出适当的错误提示。

其次是程序的可维护性问题。在当前的实现中，三个汇编版本分别使用了不同的进制输出方式，这种不统一的设计增加了代码维护的难度。虽然在实验中这种差异可以帮助我们理解不同的输出方式，但在实际应用中，最好能够统一输出格式，或者通过参数配置来灵活选择输出进制。此外，代码中的注释虽然基本完整，但缺乏对算法复杂度和空间使用的分析说明，这对后期的代码维护和优化造成了一定困难。

# 实验总结

## 实验感想

通过这次汇编语言最大公约数算法的实验，我深入理解了汇编语言程序设计的精髓。在实现相同算法的不同版本过程中，我体会到了汇编语言的灵活性和底层控制能力。特别是在处理数据存储方式时，分别使用了寄存器、栈和内存变量三种不同方式，这让我对计算机系统的数据访问和管理有了更深的认识。

在调试过程中遇到的问题，如栈溢出、进制转换等，都促使我去思考和解决实际编程中的技术难题。这些经历不仅锻炼了我的问题解决能力，也加深了我对计算机底层工作原理的理解。通过对比不同实现方式的优劣，我学会了如何在效率和可维护性之间做出权衡。

最让我感触的是，汇编语言虽然编程较为复杂，但它能够让我们直观地了解程序是如何与硬件交互的。这次实验不仅提升了我的编程技能，更重要的是培养了我严谨的编程思维和解决问题的能力。这些经验对我今后深入学习计算机科学奠定了良好的基础。

## 意见和建议

汇编课本对c语言和汇编语言混合编程的内容不够准确和详细，我在编写代码中遇到很多运行调试的问题，希望能够在课程中给出更加细节的实操注意事项。

|  |
| --- |
| 原创性声明 |
| 本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  已阅读并同意以下内容。  判定为不合格的一些情形：  （1）请人代做或冒名顶替者；  （2）替人做且不听劝告者；  （3）实验报告内容抄袭或雷同者；  （4）实验报告内容与实际实验内容不一致者；  （5）实验代码抄袭者。  **作者签名：** |