

北京理工大学

汇编与接口实验报告

大数相乘实验

学 院：	计算机学院
专 业：	计算机科学与技术
学生姓名：	郑子帆
学 号：	1120200822
指导教师：	张全新

2022 年 12 月 12 日

原创性声明

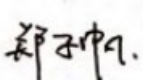
本人郑重声明：所呈交的毕业设计（论文），是本人在指导老师的指导下独立进行研究所取得的成果。除文中已经注明引用的内容外，本文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。

特此申明。

本人签名：  日期：2022年12月12日

关于使用授权的声明

本人完全了解北京理工大学有关保管、使用毕业设计（论文）的规定，其中包括：①学校有权保管、并向有关部门送交本毕业设计（论文）的原件与复印件；②学校可以采用影印、缩印或其它复制手段复制并保存本毕业设计（论文）；③学校可允许本毕业设计（论文）被查阅或借阅；④学校可以学术交流为目的，复制赠送和交换本毕业设计（论文）；⑤学校可以公布本毕业设计（论文）的全部或部分内容。

本人签名：  日期：2022年12月12日

指导老师签名： 日期： 年 月 日

汇编语言与接口设计——大数相乘实验报告

摘 要

自从 1946 年第一台计算机问世，将近 80 年的计算机发展历程中，人类和机器相互沟通的语言从机器语言到汇编语言，再到高级语言。虽然如今大学生、业界工程师更多地使用的是高级语言，但是仍然不可忽视更底层的、更接近硬件结构的汇编语言。在《汇编语言与接口技术》这门课，系统地学习了 `masm` 的语法和程序结构。在此基础上，本实验在基于 `win32` 的 `masm` 汇编环境下完成了对于两个 200 位以内的大数的乘法。本实验旨在通过上机实践掌握 `win32` 汇编程序的基本结构，掌握基本的汇编指令，用汇编实现程序的分支和循环等结构；熟练使用 `Visual Studio` 进行汇编程序的编写与调试；深入理解汇编数据类型之间的差异，学习调用 C 语言函数，综合提高汇编编程能力，同时从汇编的角度加强对 C 语言优化的理解。通过测试和分析，可以得到结论：200 位大数乘法能够突破 `int`、`long long` 等类型对于大数的限制，扩充了大数相乘的算法，具有很普遍的使用价值。

关键词：汇编程序；`MASM`；大数乘法

The Subject of Undergraduate Graduation Project (Thesis) of Beijing Institute of Technology

Abstract

Since the first computer came out in 1946, in the nearly 80 years of computer development, the language that humans and machines communicate with each other has changed from machine language to assembly language, and then to high-level language. Although today's college students and engineers use the high-level language more, the assembly language, which is at the lower level and is closer to the hardware structure, cannot be ignored. The course *Assembly Language and Interface Technology* systematically teach the syntax and program structure of masm programming. On this foundation, this experiment completed the multiplication of two large numbers within 200 bits in the win32 based masm assembly environment. This experiment aims to master the basic structure of win32 assembly program and the basic assembly instructions, and use assembly to realize the branch and loop structure of the program; Proficient in compiling and debugging assemblers with Visual Studio; Deeply understand the differences between assembly data types, learn to call C language functions, comprehensively improve assembly programming ability, and strengthen the understanding of C language optimization from the perspective of assembly. Through testing and analysis, we can draw a conclusion that the 200-bit big integer multiplication can break through the limitations of int, long long and other types on large numbers, expanding the algorithm of large number multiplication, and has a very common use value in the real world.

Key Words: assembly program; MASM; multiplication of big integers

目录

摘 要 I

Abstract II

第 1 章 背景与需求说明..... 1

 1.1 背景介绍与目标 1

 1.2 需求设计 1

第 2 章 总体功能框架设计..... 2

第 3 章 详细设计方案..... 4

 3.1 实验环境及配置 4

 3.2 输入模块详细设计 4

 3.2.1 引导用户输入 4

 3.2.2 输入检查 4

 3.3 计算模块详细设计 5

 3.4 输出模块详细设计 5

第 4 章 关键算法与代码..... 6

 4.1 输入模块 6

 4.1.1 GetLength 函数..... 6

 4.1.2 CheckValid 函数..... 7

 4.2 计算模块 8

 4.2.1 char2num 函数..... 8

 4.2.2 MulSim 函数..... 10

 4.2.3 CarryNum 函数..... 12

 4.3 输出模块 13

 4.3.1 pdSign 函数..... 13

 4.3.2 num2char 函数..... 14

 4.4 主函数 15

第 5 章 系统测试..... 18

 5.1 运行测试及截图 18

 5.2 边界值测试 18

第 6 章 人员分工..... 20

结 论 21

第1章 背景与需求说明

1.1 背景介绍与目标

乘法作为基本运算之一，在生活中有着非常广泛和频繁的应用。但是在实际生活中，普通的计算器和计算机中内置的计算器通常无法计算位数较大的数字的乘法。例如高级语言中，C 语言所给出的 `long long` 类型最多只有 64 个二进制位，gcc 所提供的 `int128` 类型也才只能有 128 个二进制位来保存一个整数。如果我们计算两个超过约 40 位的十进制数，那么计算机将会报错，产生越界的情况；当位数越来越多时，甚至连存储都成了一个问题。

因此对于位数大的正数而言，我们需要找到一个合适的数据结构存放大数，和一个合适的算法进行乘法计算并得到正确的结果。当今的高级语言中，python 的独特执行机理使得其能够实现大位数整数的运算，本实验的设计目标即能够达到相同或者近似的效果。

1.2 需求设计

程序运行后，用户分别输入两个整数 A 和 B（可以是位数 ≤ 200 的整数），程序内部进行如下操作：

1. 进行输入检查，判断用户所输入的 A 和 B 是否是位数 ≤ 200 的整数。具体地，判断 A 和 B 中除了首位为负号“-”之外，其他位均限制在 0-9 中，如果不满足条件则报错并退出程序。
2. 进行高精度乘法模拟，计算 A 与 B 的乘积
3. 判断 A 与 B 的乘积的正负号
4. 输出 A 与 B 的乘积结果

第2章 总体功能框架设计

为了满足上述需求，现需要设计相应的数据结构和算法。对于数据结构，由于有比较充足的空间，本实验考虑采用数组的形式形式数的存放和乘法过程的模拟。具体地，输入时采用字符数组，将其转变为整数数组后模拟乘法竖式，最后在转换回字符数组作为最终的结果。

算法部分，构思总体功能框架分为输入模块、计算模块、输出模块，三个模块串行执行构成了大数相乘计算器的整体框架。

输入模块 完成用户对两个大数 A 和 B 的输入，并进行一系列处理判断是否还需要继续往下执行程序，该模块的主要功能如下：

1. 引导用户完成大数 A 和 B 的输入
2. 提取 A、B 是否有“-”
3. 去除 A、B 中的前导零
4. 判断 A、B 的输入是否都合法，如果不合法则报告错误并结束程序

计算模块 模拟高精度乘法的过程，并进行有关字符串和整数数组的相互转换，该模块的主要功能如下：

1. 计算 A、B 的有效数字段的长度，有效数字段即去除 A、B 的负号（如果有）和前导零后剩余的数字串
2. 将输入时的字符数组转换成整数数组并翻转存放
3. 模拟乘法竖式
4. 结合乘法竖式完成进位

输出模块 将计算模块得到的结果进行转换，并结合 A、B 是否是负数输出乘积 C，该模块的主要功能如下：

1. 将计算模块中得到的结果的整数数组转换回字符数组并再次反转存放
2. 判断结果 C 的正负性
3. 输出结果 C

具体的设计方案思维导图可参考下图。



图 2-1 设计方案思维导图

第3章 详细设计方案

3.1 实验环境及配置

在实验开始之前，需要进行环境配置。本次实验的电脑配置如下：

- 硬件配置：Intel(R)Core(TM)i7-10750H CPU @ 60GHz，6 个内核
- 独立显卡：NVIDIA GeForce RTX 2080s
- 操作系统：Microsoft Windows 10 家庭中文版
- 代码编辑器：Microsoft Visual Studio 2019 Community

在下载安装 MASM32，并在 VS 中创建空项目后，默认配置无法编译执行汇编程序，需要进行一系列配置操作，通过在 CSDN 平台上的搜索，进行了下述配置环境操作（截图省略）：

1. 右键右侧资源管理器中项目图标，依次点击生成依赖项、生成自定义、masm
2. 设置 asm 文件属性，右键文件，点击属性，选择 Microsoft Macro Assembler
3. 配置 MASM32 库目录，右键项目图标，依次点击属性、VC++目录，将本机中 masm32 安装目录下的 include 文件的路径导入到“包含目录”中
4. 设置平台工具集，在“配置属性”的“常规”栏中设置平台工具集为 Visual Studio 2019，并更改到合适的目标平台版本号上

3.2 输入模块详细设计

3.2.1 引导用户输入

程序中导入 `msvcrt.inc` 可以使得程序中调用一些微软的 C 标准库的函数，在 C 函数前加 `"crt_"` 即可。本实验用到了 `crt_printf` 和 `crt_scanf` 分别用于实现打印对用户的引导标语和让用户输入两个整数。

3.2.2 输入检查

GetLength 函数 该函数实现了对于输入的两个字符数组分别求出其去除前导零和负号（如果有）的长度

CheckValid 函数 该函数检查了输入字符数组中是否有非法的情况，主要为出现除 0-9 外的字符（负号除外）、是否越界（超过 200 位）

3.3 计算模块详细设计

char2num 函数 模拟乘法前的处理，该函数实现了将 A、B 的有效数字段从字符数组转换到整数数组

MulSim 函数 该函数实现了两个整数模拟竖式乘法的过程，此时并不用考虑进位，有特定函数完成进位

CarryNume 函数 该函数实现了将 MulSim 函数得到的整数数组进行进位操作，保证每一位上的数字都在 0-9 范围内，并判断最高位是否进位，如果产生进位则乘积的位数 $\text{LengthC}=\text{LengthA}+\text{LengthB}$ ，否则 $\text{LengthC}=\text{LengthA}+\text{LengthB}-1$ ，此时得到了结果 C 的整数数组表示

3.4 输出模块详细设计

pdSign 函数 结合输入数 A、B 的正负性判断结果 C 的正负性

num2char 函数 模拟乘法竖式后的处理，该函数实现了将结果 C 的有效数字段从整数数组转换到字符数组

第4章 关键算法与代码

由于报告篇幅的限制, 本章仅展示部分关键代码, 完成代码详见 BigNumMul.asm 文件。

4.1 输入模块

4.1.1 GetLength 函数

该函数的功能等同于 C 标准库中的 strlen 函数, 即对字符数组进行正序遍历; 同时判断第一位是否是负号, 如果是则打上标记, 伪代码如下:

Algorithm 1: GetLength

Input: An char array pointer *chAB*, len pointer *len*, negative flag pointer *isNeg*.

Output: None

```

1:   $i \leftarrow 0$ 
2:  while  $arr[i] \text{ not } '\0'$  do
3:      if  $arr[i] == '-'$  then
4:           $*isNeg \leftarrow 1$ 
5:      else
6:           $*len \leftarrow *len + 1$ 
7:      end if
8:       $i \leftarrow i + 1$ 
9:  end while
10: return None

```

代码 4-1 GetLength 函数伪代码

汇编代码如下:

```

;-----
; @Function Name: GetLength
; @Argument: chAB: ptr char 表示 A/B 字符串, len: ptr dword 表示 A/B 的
;            长度指针, isNeg: ptr dword 表示 A/B 是否为负
; @Description: 此函数完成了对于输入的数的位数的计算, 并提取出了负号
;-----
GetLength proc uses eax ebx ecx edx esi edi, chAB: ptr char, len:
ptr dword, isNeg: ptr dword
    mov edi, chAB          ; 字符串指针
    mov esi, len            ; 长度

```

```

    mov edx, isNeg
    movzx ebx, byte ptr [edi]    ; 高位补 0
G1:
    cmp ebx, 00H                ; 判断结束符 '\0'
    je G2
    cmp ebx, 2DH                ; 如果有负号
    jne G3
    mov dword ptr [edx], 1
    jmp G4
G3:
    add dword ptr [esi], 1
G4:
    add edi, 1
    movzx ebx, byte ptr [edi]    ; 移到下一位
    jmp G1
G2:
    ret
GetLength endp

```

代码 4-2 GetLength 函数汇编代码

4.1.2 CheckValid 函数

该函数判断了输入的数 A 和 B 是否合法，如果不合法则 InvalidFlag 置 1，汇编代码如下：

```

;-----
; @Function Name: CheckValid
; @Argument: numAB: ptr dword 表示数字数组的指针, en: ptr dword 表示长度
; @Description: 判断输入的数是否合法
;-----
CheckValid proc uses eax ebx ecx edx esi edi, numAB: ptr dword,
len: dword
    mov edi, numAB
    mov esi, TYPE dword
    mov ecx, 0
V1:
    cmp ecx, len

```

```

jnl V2
mov eax, dword ptr [edi]
.if eax < 0
    mov InvalidFlag, 1
    jmp V2
.elseif eax > 9
    mov InvalidFlag, 1
    jmp V2
.endif
add edi, esi
inc ecx
jmp V1
V2:
    ret
CheckValid end

```

代码 4-3 CheckValid 函数汇编代码

4.2 计算模块

4.2.1 char2num 函数

该函数实现了有效数字段的字符数组向整数数组的转换，具体实现即一个正序遍历，将每个 byte 类型的 ASCII 编码减去 30H，用 dword 类型保存结果。这样的原理是字符'0'-'9'的 ASCII 编码分别为 30H-39H。另外，这里使用 dword 类型保存时采用了倒序存储，这样在字符数组中下标 0 变成了 dword 类型数组中的 $length-1$ ，1 变成了 $length-2$ ，如下图。

index	0	1	...	$len-2$	$len-1$
type byte	3	9	...	5	4
type dword	4	5	...	9	3

图 4-1 char2num 坐标变换示例

函数的汇编代码如下：

```

;-----
; @Function Name: char2num

```

```
; @Argument: charAB: ptr char 表示字符串, numAB: ptr dword 表示数字数组, len: dword 表示数位
; @Description: 将字符串转换为数字, 并反转过来存放
;-----
char2num proc uses eax ebx ecx edx esi edi, chAB: ptr char, numAB:
ptr dword, len: dword
    mov edi, chAB
    mov esi, TYPE byte
    mov ecx, 0
C1:
    cmp ecx, len
    jnl C2
    movzx ebx, byte ptr [edi]
    sub ebx, 30H          ; 0~9 字符的 ascii 码从 48 开始
    push ebx              ; 将当前字符压入栈
    add edi, esi
    add ecx, 1
    jmp C1

C2:
    ; 逐一弹栈完成转换
    mov edi, numAB
    mov esi, TYPE dword
    mov ecx, 0
C3:
    cmp ecx, len
    jnl C4
    pop ebx
    mov dword ptr [edi], ebx
    inc ecx
    add edi, esi
    jmp C3

C4:
    ret
char2num endp
```

代码 4-4 char2num 函数汇编代码

4.2.2 MulSim 函数

此函数为本程序的**重点函数**，这个函数使用上面的有效数字段转化成的整数数组 numA 和 numB 进行了乘法竖式的模拟，暂不考虑进位，将结果放入 numC 中。具体地，剖析乘法竖式的实质，即两个乘数中某一位数进行个位数的乘法，不考虑进位的情况下，只对积中的某一位产生贡献。如果按我们上述所构造的小端存储的 numA 和 numB 数组，并设定 numC 也为小端存储，探究其中的某两个坐标下的两位相乘会对 numC 中的那个下标中的数产生影响。下图展现了一个简单的例子，通过观察可以发现 numA 中下标*i*的数和 numB 中下标*j*的数相乘会对 numC 中下标*i + j*的数位产生贡献。

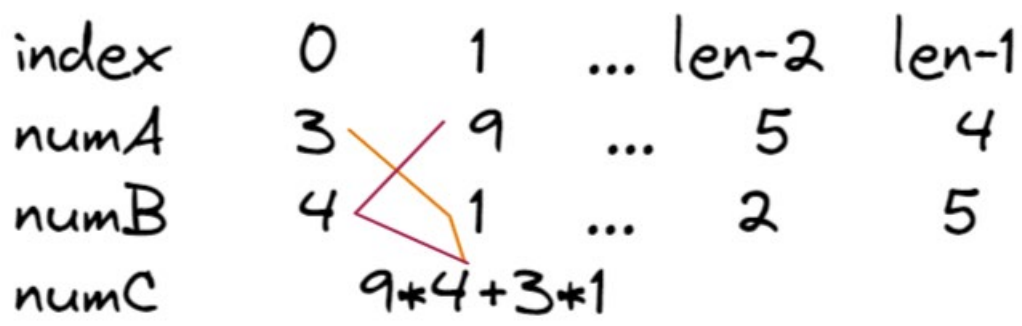


图 4-2 MulSim 原理示例

故可设计算法如下：

Algorithm 2: MulSim

Input: numA, numB
Output: numC

```
1: i ← 0
2: j ← 0
3: while i < LengthA do
4:   while j < LengthB do
5:     numC[i+j] = numA[i] * numB[j]
6:     j ← j+1
7:   end while
8:   i ← i+1
9: end while
10: return None
```

代码 4-5 MulSim 函数伪代码

汇编代码如下：

```
;-----  
; @Function Name: MulSim  
; @Argument:  
; @Description: 模拟 numA 和 numB 的乘法竖式，并将结果存在 numC 中  
;-----  
MulSim proc uses eax ebx ecx edx esi edi  
    mov eax, LengthA  
    add eax, LengthB    ; 计算初始 result 的长度 (A 长度+B 长度)  
    dec eax  
    mov LengthC, eax  
  
    mov edi, offset numA    ; numA 的指针头  
    mov esi, offset numB    ; numB 的指针头  
    mov edx, offset numC    ; numC 的指针头  
    mov eax, 0  
M1:  
    cmp eax, LengthA  
    jnl M2  
    mov ebx, 0    ; numB 的数组当前位置  
M3:  
    cmp ebx, LengthB  
    jnl M4  
    mov ecx, dword ptr [edi + 4*eax]  
    imul ecx, dword ptr [esi + 4*ebx]  
    add eax, ebx    ; 得到当前 numC 的下标  
    add dword ptr [edx + 4*eax], ecx  
    sub eax, ebx    ; 还原 eax  
    inc ebx  
    jmp M3  
M4:  
    inc eax  
    jmp M1  
M2:  
    ret  
MulSim endp
```

代码 4-5 MulSim 函数汇编代码

4.2.3 CarryNum 函数

该函数将上面得到的 NumC 数组逐位处理进位，因为是小端存储，所以只要正序遍历一遍统计进位情况即可。在最后要判断最高位是否产生进位，如果产生则 LengthC 需要加 1。汇编代码如下：

```
;-----  
; @Function Name: CarryNum  
; @Argument:  
; @Description: 模拟 numC 的进位情况  
;-----  
CarryNum proc uses eax ebx ecx edx esi edi  
    LOCAL CarryFlag: dword  
    mov CarryFlag, 0          ; 记录是否产生进位  
    mov edi, offset numC  
    mov ecx, 0  
C1:  
    cmp ecx, LengthC  
    jnl C2  
    cmp dword ptr [edi + 4*ecx], 9  
    jng C3  
    mov eax, dword ptr [edi + 4*ecx]  
    mov edx, 0  
    mov ebx, 10  
    div ebx                   ; 除的结果在 eax, 余数在 edx  
    mov dword ptr [edi + 4*ecx], edx    ; 余数留  
    add dword ptr [edi + 4*ecx + 4], eax    ; 除数进到下一位  
    mov CarryFlag, 1  
    jmp C4  
C3:  
    mov CarryFlag, 0  
C4:  
    inc ecx  
    jmp C1  
C2:  
    .if CarryFlag == 1      ; 如果有进位, LengthC+1  
        inc LengthC
```

```
.endif
ret
CarryNum endp
```

代码 4-6 CarryNum 函数汇编代码

4.3 输出模块

4.3.1 pdSign 函数

该函数用于结合数 A 和数 B 的正负性判断乘积 C 的正负性，如果为负数则在输出时需要多输出一个负号 "-", 具体正负性对应关系如下：

A 的正负性	B 的正负性	C 的正负性
正	正	正
正	负	负
负	正	负
负	负	正

表 4-1 结果 C 的正负性判断

汇编代码如下：

```
;-----
; @Function Name: pdSign
; @Argument:
; @Description: 判断结果的正负号（正数则 IsNeg 为 0，否则为 1）
;-----
pdSign proc uses eax ebx ecx edx esi edi
    mov eax, IsNegA
    mov ebx, IsNegB

    cmp eax, 1
    jne P1
    cmp ebx, 1
    jne P1
    mov IsNegC, 0    ; A B 均为负数
    jmp last
P1:
    cmp eax, 0
    jne P2
```

```

    cmp ebx, 0
    jne P2
    mov IsNegC, 0    ; A B 均为正数
    jmp last
P2:
    mov IsNegC, 1    ; 一正一负
last:
    ret
pdSign endp

```

代码 4-7 pdSign 函数汇编代码

4.3.2 num2char 函数

该函数原理与 char2num 类似，是对偶函数，用于将结果 C 的 dword 类型小端存储转换成 byte 类型的大端存储，方便输出，代码如下：

```

;-----
; @Function Name: num2char
; @Argument: numAB: ptr dword 表示数字数组, charAB: ptr char 表示字符串, len: dword 表示数位
; @Description: 将数字转换为字符串, 并反转过来, 此时反转完为正
;-----
num2char proc uses eax ebx ecx edx esi edi, chAB: ptr char, numAB: ptr dword, len: dword
    mov edi, numAB
    mov esi, TYPE dword
    mov ecx, 0
N1:
    cmp ecx, len
    jnl N2
    mov eax, dword ptr [edi]
    add eax, 30H          ; 0~9 字符的 ascii 码从 48 开始
    push eax              ; 将当前字符压入栈
    add edi, esi
    add ecx, 1
    jmp N1
N2:

```

```

; 逐一弹栈完成转换
mov edi, chAB
mov esi, TYPE byte
mov ecx, 0
N3:
    cmp ecx, len
    jnl N4
    pop eax
    mov byte ptr [edi], al
    inc ecx
    add edi, esi
    jmp N3
N4:
    mov byte ptr [edi], 00H
    ret
num2char endp

```

代码 4-8 num2char 函数汇编代码

4.4 主函数

主函数完成了对于以上三个模块的汇总和其中的函数的调用，代码如下：

```

;-----
; @Function Name: main
; @Argument:
; @Description: 主函数
;-----
main proc
    invoke crt_printf, addr BigNumMulmsg      ; 输出开头提示标语
    invoke crt_puts, addr NullStr

    invoke crt_printf, addr InputAmsg        ; 提示输入第一个数 A

    invoke crt_scanf, addr InputFmt, addr chA ; 输入数 A（字符形式）

    invoke crt_printf, addr InputBmsg        ; 提示输入第二个数 B
    invoke crt_scanf, addr InputFmt, addr chB ; 输入数 B（字符形式）
main endp

```

```
    invoke GetLength, addr chA, addr LengthA, addr IsNegA    ; 取 A
    ; 的长度, 0 为 A 的标识符
    invoke GetLength, addr chB, addr LengthB, addr IsNegB    ; 取 B
    ; 的长度, 1 为 B 的标识符

M7:
    invoke pdSign      ; 判断 result 的符号
    cmp IsNegA, 0
    jne M1
    invoke char2num, addr chA, addr numA, LengthA; 完成字符串 A 到数
    ; 字的转换 (正数)
    jmp M2

M1:
    invoke char2num, addr chA+1, addr numA, LengthA; 完成字符串 A 到
    ; 数字的转换 (负数)

M2:
    cmp IsNegB, 0
    jne M3
    invoke char2num, addr chB, addr numB, LengthB; 完成字符串 B 到数
    ; 字的转换 (正数)
    jmp M8

M3:
    invoke char2num, addr chB+1, addr numB, LengthB; 完成字符串 B 到
    ; 数字的转换 (负数)

M8:
    ; 判断输入是否合法
    invoke CheckValid, addr numA, LengthA    ; 判断 A 是否输入合法
    invoke CheckValid, addr numB, LengthB    ; 判断 B 是否输入合法

    cmp InvalidFlag, 1
    jne M4      ; 如果合法, 模拟高精度乘法
    invoke crt_printf, addr ErrorMessage    ; 不合法, 显示错误信息
    jmp M6

M4:
```

```
invoke MulSim      ; 模拟高精度乘法
invoke CarryNum    ; 模拟进位

invoke num2char, addr chC, addr numC, LengthC; 将 numC 转成 chC

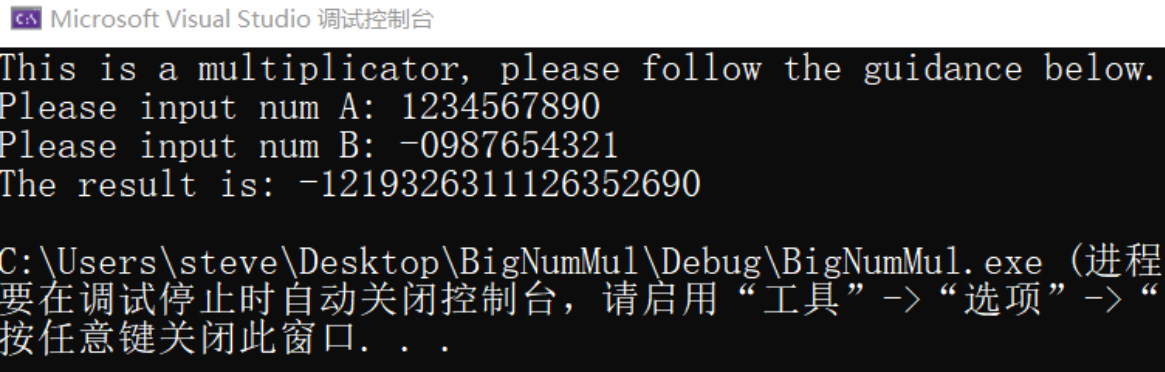
cmp IsNegC, 0
jne M5
invoke crt_printf, addr Outputmsg, addr chC ; 输出正数结果
jmp M6
M5:
    invoke crt_printf, addr OutputmsgNeg, addr chC ; 输出负数结果
M6:
    ret
main endp
end main
```

代码 4-9 主函数代码

第5章 系统测试

5.1 运行测试及截图

编译运行代码，可以实现两个大整数的乘法，运行结果截图如下：



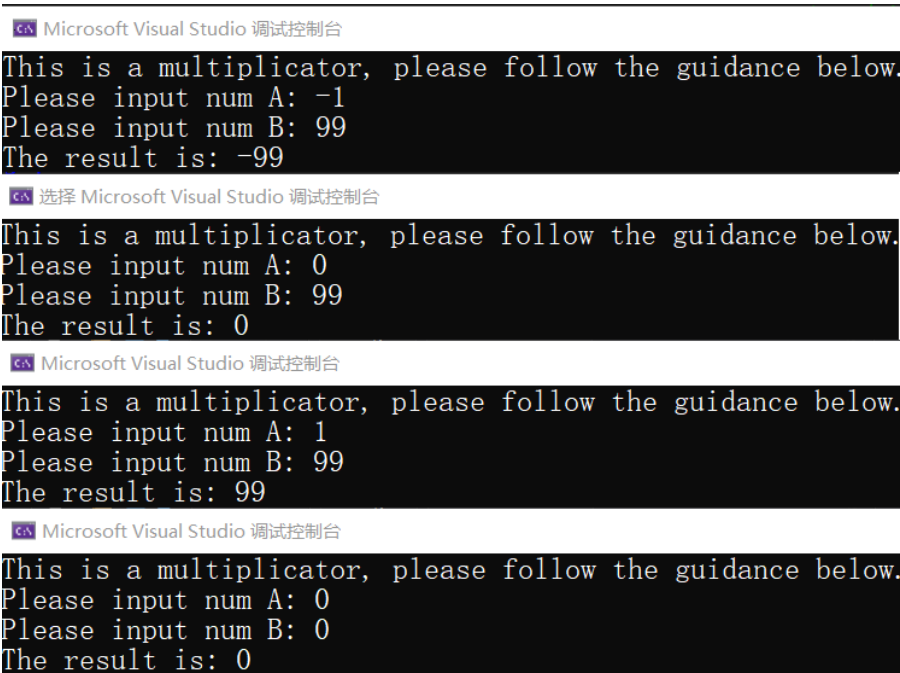
```
Microsoft Visual Studio 调试控制台
This is a multiplicator, please follow the guidance below.
Please input num A: 1234567890
Please input num B: -0987654321
The result is: -1219326311126352690

C:\Users\steve\Desktop\BigNumMul\Debug\BigNumMul.exe (进程
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“
按任意键关闭此窗口. . .
```

图 5-1 运行结果测试

5.2 边界值测试

结合软件测试相关测试技术，本实验采取了边界值测试方法，测试了一些边界值相乘的数据，具体的取值有-99、-1、0、1、99，和错误输入测试，部分结果截图如下：



```
Microsoft Visual Studio 调试控制台
This is a multiplicator, please follow the guidance below.
Please input num A: -1
Please input num B: 99
The result is: -99

选择 Microsoft Visual Studio 调试控制台
This is a multiplicator, please follow the guidance below.
Please input num A: 0
Please input num B: 99
The result is: 0

Microsoft Visual Studio 调试控制台
This is a multiplicator, please follow the guidance below.
Please input num A: 1
Please input num B: 99
The result is: 99

Microsoft Visual Studio 调试控制台
This is a multiplicator, please follow the guidance below.
Please input num A: 0
Please input num B: 0
The result is: 0
```

图 5-2 边界值测试部分结果展示

北京理工大学汇编语言与接口技术——大数相乘实验报告

C:\Users\steve\Desktop\BigNumMul.exe

```
This is a multiplier, please follow the guidance below.  
Please input num A: -546156  
Please input num B: -sdfc-51  
Error! Your input if not two numbers, please input two numbers using 0~9!
```

图 5-3 错误测试用例结果展示

由于数不大，可以很容易判断出程序运行结果均正确，但是对于 100 位乃至 200 位的大数而言，正确性较难判断，故本实验采用了 python（版本为 3.9.2）进行大数乘法的对比，其中一组的结果对比如下图：

```
Microsoft Visual Studio 调试控制台  
This is a multiplier, please follow the guidance below.  
Please input num A: 98089375293405720839679512356794024378534985341515641651841656845689165489144  
Please input num B: 51698652335648856143189615341865324568965132481653479845615342135986565187893346054  
The result is: 5071088511114766873531413361944015158499536912210378262306989398781181683107745837315750419673853813631949736251991113199117900870446135953029799730207772237776  
  
Python 3.9 (64-bit)  
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>> num1=98089375293405720839679512356794024378534985341515641651841656845689165489144  
>>> num2=51698652335648856143189615341865324568965132481653479845615342135986565187893346054  
>>> num1*num2  
5071088511114766873531413361944015158499536912210378262306989398781181683107745837315750419673853813631949736251991113199117900870446135953029799730207772237776  
>>>
```

图 5-4 大数乘法大整数测试对比

综合上面的多组测试用例和多种测试方法，可以发现本实验所实现的代码在正确性上没有任何的问题，效果理想。

第6章 人员分工

本实验为本人独立完成，即一人完成了算法设计、流程构思、代码编写、系统测试、报告撰写，贡献度 100%。

结 论

本实验在masm汇编环境下编写了两个大整数相乘的代码，从上面的系统测试中可以看出实验中所实现的代码具有运算结果的正确性和准确性，并在边界值上都有着准确的处理，完全实现了本文开篇所提出的需求。

另外，本实验中所编写的代码采用了简单的数据结构和算法完成了所有的功能，其优势在于代码编写简单、逻辑结构清晰、运行执行速度快，与平常我们在计算机上所使用的计算器的运算速度并没有明显的差别，并且还扩充了乘法中乘数的位数，具有很强的实用性和推广性。

最后，总结一些关于我在这次实验中的心得体会。本次实验的实验内容整体难度并不大，但是由于之前对于汇编编程零基础，如果不参考网上的代码，中间的调试过程还是比较痛苦的，比如我从始至终都无法正确调用 `crt_strlen` 函数进行 `byte` 数组长度的计算，无奈之下只能自己写一个函数；在调试过程中，曾尝试用 `crt_printf` 进行中间结果的输出（就像在 C 语言中输出中间结果调试那样），但是后来发现这些语句反而会引起程序出现更多的错误，最后通过查找资料发现，`crt_printf` 会对一些寄存器进行值上的改变（如 `ecx`），而我的函数中也用了这些寄存器，故导致错误，解决方案即使用 `push`、`pop` 指令。通过遇到和解决这些问题也让我对于汇编语言有了更深的理解和认识，让我对计算机的底层实现的理解更加深入了，也为之后的学习打下了扎实的基础。

感谢老师一个学期以来的传道受业与解惑。