

实验名称： 汇编个人实验报告

学 院： 睿信书院

专 业： 计算机科学与技术

班 级： 07111803

姓 名： 宋尚儒

指导教师： 李元章

# 目录

1 实验目的 .....	3
2 大数相乘 .....	3
2.1 实验步骤 .....	3
2.1.1 输入输出大整数字符串 .....	4
2.1.2 大整数字符串与数组的转换 .....	4
2.1.3 大整数数组乘法 .....	4
2.2 实验结果 .....	5
3 多重循环 .....	6
3.1 实验步骤 .....	6
3.1.1 C 语言多重循环程序 .....	6
3.1.2 反汇编代码分析 .....	7
3.1.3 汇编重写程序 .....	8
3.2 实验结果 .....	9
4 文件比较 .....	10
4.1 实验步骤 .....	10
4.1.1 Windows 窗口程序 .....	10
4.1.2 文件比较功能 .....	10
4.2 实验结果 .....	11
5 总结 .....	12

## 1 实验目的

1. 大数相乘。要求实现两个十进制大整数的相乘(100 位以上),输出乘法运算的结果。
2. C 语言编写多重循环程序(大于 3 重),查看其反汇编码,分析各条语句功能(分析情况需要写入实验报告),并采用汇编语言重写相同功能程序。
3. Windows 界面风格实现两个文本文件内容的比对。若两文件内容一样,输出相应提示;若两文件不一样,输出对应的行号。

## 2 大数相乘

大数相乘。要求实现两个十进制大整数的相乘(100 位以上),输出乘法运算的结果。

### 2.1 实验步骤

该程序需要在控制台界面获取输入的两个十进制大整数,并输出相乘结果,普通二进制乘法无法满足要求,故需要设置数组数据结构,保存大整数的每一位,并针对数组结构的大整数设置乘法。所以设置关键数据结构为:

```
input_str1 byte 200 dup(0)
input_str2 byte 200 dup(0)
output_str byte 200 dup(0)
input_num1 dword 200 dup(0)
input_num2 dword 200 dup(0)
output_num dword 200 dup(0)
input_len1 dword 0
input_len2 dword 0
output_len dword 0
```

并将整体程序功能分解如下

- (1) 输入输出大整数字符串
- (2) 大整数字符串与数组的转换

### (3) 大整数数组乘法

#### 2.1.1 输入输出大整数字符串

该部分需要在主函数中实现，通过引用库函数`scanf`获取两个大整数的字符串。此后引用库函数`strlen`获取两个字符串的长度，保存到两个输入长度的全局变量中。然后依次转换两个字符串到全局数组结构中，并调用数组乘法功能，计算获得结果数组，并将结果大整数数组转换为字符串并保存到全局变量中，最后调用库函数`printf`输出大整数字符串。

#### 2.1.2 大整数字符串与数组的转换

该功能由两个自定义函数`str\_to\_num`和`num\_to\_str`组成，分别可以实现将字符串转换为数组和将数组转换为字符串，其输入均为三个参数，分别代表字符串、数组结构、长度，逻辑结构类似，均需要循环遍历输入的结构，并将每一个函数输入元素压入栈中，结束后依次取出栈中的元素并将其赋值给输出结构。这是因为字符串中低位在字符串后部，而计算过程是从低位开始的，所以不妨将低位保存在数组结构的前部，实现两种结构的逆序转换，方便之后的计算。对于每一个元素，需要实现字符和双字无符号数的转换，相对比较简单。

#### 2.1.3 大整数数组乘法

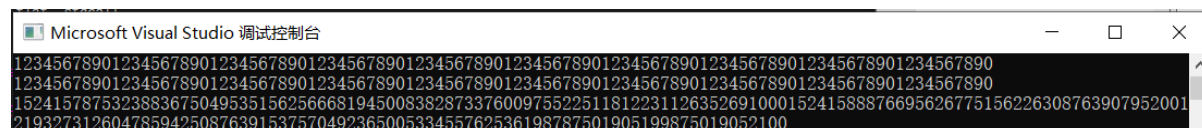
该功能由自定义函数`big\_num\_mul`实现，可以实现两个输入大整数数组结构的相乘并保存到输出数组中。

总体而言，其实现的结构为二重循环，即以输入数组1的每一位乘以输入数组2的每一位，这一过程从数字的低位遍历到高位，并且对于每一次的乘法运算结果，需要将其加上上一位保留的累计进位结果，然后进行除10操作，将余数保存到对应的输出位中，并将商累计到下一位的进位结果中，方便下一次乘法运算的进行。遍历完成后即可得到输出的大整数数组结构。

在最后还需要计算输出大整数的长度，这里需要进行一次特判，通常情况下长度为 $len1 * len2$ ，但如果对应位置处的大整数数组元素如果为零，则说明长度还需要减一，这对零乘零的情况也有效。

## 2.2 实验结果

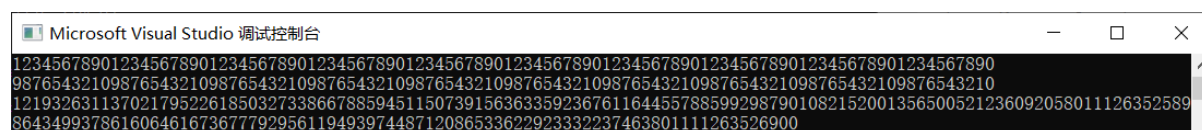
以下为两组大整数乘法的产生结果，均为 100 位大整数相乘，结果已由 Python 进行乘法验证，计算结果均正确。



Microsoft Visual Studio 调试控制台

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
152415787532388367504953515625666819450083828733760097552251181223112635269100015241588876695626775156226308763907952001
2193273126047859425087639153757049236500533455762536198787501905199875019052100
```

图 2-1：大数乘法示例 1



Microsoft Visual Studio 调试控制台

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
987654321098765432109876543210987654321098765432109876543210987654321098765432109876543210
121932631137021795226185032733866788594511507391563633592367611644557885992987901082152001356500521236092058011126352589
86434993786160646167367779295611949397448712086533622923332237463801111263526900
```

图 2-2：大数乘法示例 2

## 3 多重循环

C 语言编写多重循环程序 (大于 3 重), 查看其反汇编码, 分析各条语句功能 (分析情况需要写入实验报告), 并采用汇编语言重写相同功能程序。

### 3.1 实验步骤

首先需要编写并编译 C 语言多重循环程序, 并根据反汇编结果进行分析

#### 3.1.1 C 语言多重循环程序

```
#include<stdio.h>
int i, j, k, l;
int n = 10;

int main()
{
    for (i = 0; i < n; ++i)
    {
        for (j = 0; j < n; ++j)
        {
            for (k = 0; k < n; ++k)
            {
                for (l = 0; l < n; ++l)
                {
                    if (k + j + i + l == n)
                        printf("%d %d %d %d\n", i, j, k, l);
                }
            }
        }
    }
    return 0;
}
```

### 3.1.2 反汇编代码分析

使用 IDA 工具分析主函数对应的反汇编源码，部分分析结果以注释形式附加在汇编代码后。

函数开始段：

.text:00401350	push	ebp	;前栈底指针入栈
.text:00401351	mov	ebp, esp	;更新栈底指针
.text:00401353	push	ebx	;ebx 入栈
.text:00401354	and	esp, 0FFFFFF0h	;栈初始化
.text:00401357	sub	esp, 20h	;预留局部变量空间
.text:0040135A	call	__main	;编译器的初始化函数

共有 4 个循环体，9 个转移标志，这里只展示部分

循环体 1 对应代码块

;for (i = 0; i < n; ++i)			
.text:0040135F	mov	ds:_i, 0	;给 i 赋值为 0
.text:00401369 loc_401369:			
.text:00401369	mov	edx, ds:_i	;edx 赋值为 i
.text:0040136F	mov	eax, _n	;eax 赋值为 n
.text:00401374	cmp	edx, eax	;比较 edx 和 eax
.text:00401376	jge	loc_40146B	;大于等于则跳到目标
;.....			
.text:00401459 loc_401459:			
.text:00401459	mov	eax, ds:_i	;eax 赋值为 n
.text:0040145E	add	eax, 1	;eax 自增 1
.text:00401461	mov	ds:_i, eax	;i 赋值为 eax
.text:00401466	jmp	loc_401369	;无条件跳转到循环开始
.text:0040146B loc_40146B:			;循环结束

循环体 4 对应代码块

;for (l = 0; l < n; ++l)			
{			
; if (k + j + i + l == n)			
; printf("%d %d %d %d\n",i,j,k,l);			
;}			
.text:004013B6	mov	ds:_l, 0	;l 赋值为 0
.text:004013C0 loc_4013C0:			
.text:004013C0	mov	edx, ds:_l	;edx 赋值为 l
.text:004013C6	mov	eax, n	;eax 赋值为 n

.text:004013CB	cmp	edx, eax	;比较 l 和 n
.text:004013CD	jge	short loc_401435	;大于等于循环结束
.text:004013CF	mov	edx, ds:_k	;edx 赋值 k
.text:004013D5	mov	eax, ds:_j	;eax 赋值 j
.text:004013DA	add	edx, eax	;edx 自加 eax 值
.text:004013DC	mov	eax, ds:_i	;eax 赋值 i
.text:004013E1	add	edx, eax	;edx 自加 eax 值
.text:004013E3	mov	eax, ds:_l	;eax 赋值 l
.text:004013E8	add	edx, eax	;edx 获得累加和
.text:004013EA	mov	eax, _n	
.text:004013EF	cmp	edx, eax	;比较累加和与 n
.text:004013F1	jnz	short loc_401426	;不等于进入下一轮
.text:004013F3	mov	ebx, ds:_l	;开始参数传递
.text:004013F9	mov	ecx, ds:_k	
.text:004013FF	mov	edx, ds:_j	
.text:00401405	mov	eax, ds:_i	
.text:0040140A	mov	[esp+10h], ebx	;参数从右到左入栈
.text:0040140E	mov	[esp+0Ch], ecx	;不是传统方式
.text:00401412	mov	[esp+8], edx	
.text:00401416	mov	[esp+4], eax	
.text:0040141A	mov	dword ptr [esp], offset Format	
.text:00401421	call	_printf	;调用输出函数
.text:00401426 loc_401426:			
.text:00401426	mov	eax, ds:_l	;实际上是 l 自增 1
.text:0040142B	add	eax, 1	
.text:0040142E	mov	ds:_l, eax	
.text:00401433	jmp	short loc_4013C0	;跳转到循环开始

### 3.1.3 汇编重写程序

只需要注意九个转移标志之间的跳转关系即可完成编写，并且省略了部分原汇编码中的冗余操作，具体代码可以查看源文件，这里不再重复说明。



## 3.2 实验结果

将源 C 语言程序运行结果与重写汇编程序的结果进行对比，输出完全一致（字体大小存在差别）。

```
D:\资料\2021年春\汇编语言与技术接口\lab\moo.exe
6 0 2 2
6 0 3 1
6 0 4 0
6 1 0 3
6 1 1 2
6 1 2 1
6 1 3 0
6 2 0 2
6 2 1 1
6 2 2 0
6 3 0 1
6 3 1 0
6 4 0 0
7 0 0 3
7 0 1 2
7 0 2 1
7 0 3 0
7 1 0 2
7 1 1 1
7 1 2 0
7 2 0 1
7 2 1 0
7 3 0 0
8 0 0 2
8 0 1 1
8 0 2 0
8 1 0 1
8 1 1 0
8 2 0 0
9 0 0 1
9 0 1 0
9 1 0 0
Process returned 0 (0x0) execution time : 0.319 s
Press any key to continue.
```

图 3-1：C 程序运行结果

```
Microsoft Visual Studio 调试控制台
6 2 1 1
6 2 2 0
6 3 0 1
6 3 1 0
6 4 0 0
7 0 0 3
7 0 1 2
7 0 2 1
7 0 3 0
7 1 0 2
7 1 1 1
7 1 2 0
7 2 0 1
7 2 1 0
7 3 0 0
8 0 0 2
8 0 1 1
8 0 2 0
8 1 0 1
8 1 1 0
8 2 0 0
9 0 0 1
9 0 1 0
9 1 0 0
D:\资料\2021年春\汇编语言与技术接口\lab\Project2\Debug\multiple_iteration.exe (进程 16176) 已退出, 返回代码为: 0。
若要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...
```

图 3-2：汇编程序运行结果

## 4 文件比较

Windows 界面风格实现两个文本文件内容的比对。若两文件内容一样，输出相应提示；若两文件不一样，输出对应的行号。

### 4.1 实验步骤

与之前不同的是，这次需要实现 Windows 界面风格的程序，因此除了文件比较逻辑之外，还需要实现带有输入输出的 Windows 窗口。

#### 4.1.1 Windows 窗口程序

窗口主要元素应该包括

- (1) 两个输入文件路径的文本框
- (2) 一个比较按钮
- (3) 结果提示窗口，可以用 MessageBox 实现

如此，首先需要在主函数中实现窗口的初始化并将其显示，然后就可以进入对消息获取和处理的无限循环。

窗口初始化在自定义函数`\_WinMain`中实现，具体需要完成的工作为

- (1) 创建两个文本框
- (2) 创建按钮，设置响应的信号为 x（在源码中设置为 15）
- (3) 设置对信号 x 的响应操作，具体来说就是获取两个文本框内容保存到文件路径字符串中，然后调用自定义函数`MyCompareFile`进行文件比较，文件比较产生的不一致行数和行号结果会保存在全局变量中，调用 MessageBox 对不同情况的结果进行输出，如果行数为 0 则输出文件一致提示，否则输出不一致行号。

#### 4.1.2 文件比较功能

该程序需要实现文本文件的逐行比较，所以需要逐行处理两个文件，对于每行可能存在四种情况

- (1) 文件 1 和文件 2 均未到达最后一行，需要引用库函数`strcmp`对两个文件行的

字符串对比，如果不一致则保存结果。

- (2) 文件 1 到达最后一行，文件 2 未到达最后一行，保存此时文件 2 的行数
- (3) 文件 2 到达最后一行，文件 1 未到达最后一行，保存此时文件 1 的行数
- (4) 文件 1 和文件 2 均到达最后一行，结束比较

对于如何读取文件的每一行，使用自定义函数`ReadLine`实现，该函数包含两个参数，分别是文件的句柄和对应缓冲区的指针，返回值为读取的行字符串的长度，保存在`eax`中。这里使用了库函数`ReadFile`，每次读取一个字节的内容到缓冲区中，对于该字节存在以下情况

- (1) ‘\n’，说明该行数据读取完毕，停止读取操作
- (2) 空字节，说明该文件读取完毕，停止读取操作
- (3) 其他，继续读取

如此可以实现文件的逐行读取与比较

## 4.2 实验结果

进行两组文件比较以展示结果，注意这里使用的是相对路径，如需自行测试可考虑使用完整路径。

组 1:

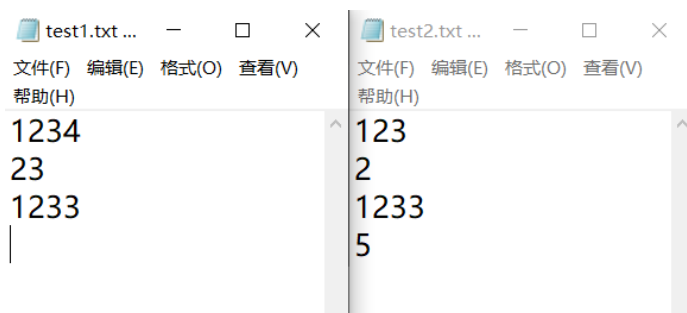


图 4-1：组 1 原文本文件



图 4-2：组 1 文本比较结果

组 2:

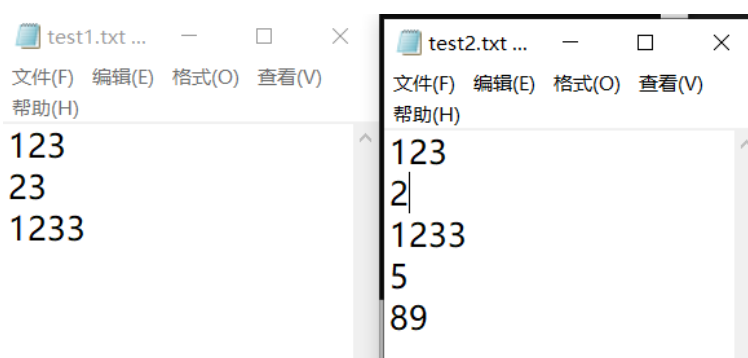


图 4-3: 组 2 原文本文件



图 4-4: 组 2 文本比较结果

完全符合要求。

## 5 总结

在这次汇编上机作业中,我首先完成了大数相乘,了解了各类基础汇编指令的使用,然后万柳个多重循环程序分析,了解了编译器是如何处理多重循环的,最后完成 Windows 界面文本内容比较,参考了网络上大量文档,理解了 windows 界面编程。