北京理工大学

本科生毕业设计（论文）开题报告

学　　院： 计算机学院

专　　业： 计算机科学与技术

班　　级： 0711802

姓　　名： 田佳析

指导教师： 李元章

校外指导教师： 无

二○二一年五月二十一日

目录

1. 实验介绍

在五个编程实验中，我选择了大数相乘、文件比较和计算器实验，通过大数相乘实验，我学会使用汇编语言实现控制台的功能；通过文件比较，我学会了使用 win32 api 函数构建窗口；结合前面两个实验了解到的指示，我实现了一个功能较为全面的窗口计算器程序。

2. 大数相乘

2.1 实验要求

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。

2.2 实验内容

大数乘法在读入数据后，使用 fft 变换计算两个大数的乘积。

2.3 实验过程

整个程序过程为，将输入的数字存储转化为浮点数存储在数组中，接下来对输入的数进行 fft 变换，得到点值表示的数后两数相乘，进行 fft 逆变换，最后对逆变换得到的结果进行处理并输出。

2.3.1 输入处理

对于输入的数字，需要经过翻转数字，将数字每一位转换位复数形式以便利用 fft 的多项式快速乘法计算乘积，然后需要计算乘积达到的最大位数，将复数数组拓展到大于该位数的 2 的幂次位，便于 fft 二分计算。

由于乘积结果位数可能大于乘数，所以为了便于位数扩展，将低位数放在数组前面，高位数放在数组后面，这时需要翻转数字，首先需要获取输入数字长度：

```
1  mov esi, str_p
2  mov ecx, 0
3  main_loop:
4      mov eax, [esi]
```

```
    cmp eax, 0
    jz loop_end
    inc ecx
    inc esi
    jmp main_loop
loop_end:
```

遍历输入的数组，如果识别到末尾 0，ecx 保存数组长度，跳出循环进行数组的翻转：

```
mov esi, str_p
mov ebx, 0
loop1:
    mov al, [esi+ebx]
    mov dl, [esi+ecx-1]
    mov [esi+ecx-1], al
    mov [esi+ebx], dl
    inc ebx
    dec ecx
    cmp ecx, ebx
    jle loop1_end
    jmp loop1
loop1_end:
    ret
```

上面的代码中，ecx 保存数组末尾下标，ebx 保存数组开始下标，交换 ecx 和 ebx 下标的数据，当 ecx 小于等于 ebx 时跳出循环。

之后需要将输入的数字转换为复数形式，调用 changenum 函数进行数字的转换，我建立了一个结构体用于储存数据，定义如下：

```
cp STRUCT
  x dword ? ;实部，为 float 类型
  y dword ? ;虚部，为 float 类型
cp ENDS
```

由于 changenum 函数代码过长，所以采用伪代码说明：

Algorithm 1 数字转换为复数数组

Input: 数字的首地址，复数数组的首地址
Output: 复数数组的长度

procedure ASM PROCEDURE
    将数字首地址赋给 esi, 复数数组首地址赋给 edi .
    初始化临时变量 n，表示数组长度.
arr_loop:
    利用 esi 遍历数字.
    if [esi] == 0 **then**
        goto *done*.
    end if
    if [esi] == '-' **then**
        goto *negtive*.
    end if
    调用 int_to_float 函数，将整数转换为浮点数表示.
    将结果赋给 [edi] 的实部.
    inc esi.
    inc n.
    add edi, 8.
    goto *arr_loop*.
negtive:
    符号标志取反
    inc esi
    goto *arr_loop*.
done:
    mov eax, n.
    ret.
end procedure

上面代码中使用了 int_to_float 函数，因为计算 fft 需要浮点运算，需要以 float 类型存储数据，在刚开始学的时候，不知道可以利用浮点寄存器 fild 指令直接加载整数，然后一个 fstp 转换为浮点数，只能根据浮点数定义，参考网上代码写了个转换函数，代码如下：

```
1  mov edx, int_num
2  xor eax, eax
3  test edx, edx ; 判断 edx 是否为0
4  jz done
5  jns pos ;正数
6  ; 下面是负数的处理
```

```
or eax, 80000000h ;如果判断得到负数，符号位置1
neg edx ;补码取负数
pos:
    bsr ecx, edx ;从最高位向最低位搜索，读取第一个1
    sub ecx, 23 ;由于只能保存23位有效数字，
    ;得把第一个1后23位移到浮点数有效位上，计算偏移量
    ror edx, cl
    ;x-23如果是负数的话，相当于需要左移 x-23 ，也即右移 32n+x-23，
    ;取低 8 位的负数补码相当于 256+(x-23)
    ;256+(x-23)，由于 32 整除 256，所以右移256+(x-23)相当于左移x-23
    and edx, 007fffffh ; 前9位置0
    or eax, edx ; 得到有效部分的数
    add ecx, 150 ; 计算指数部分大小，由于指数为移码，
    ;所以加上127再加上之前减去的23
    shl ecx, 23 ; 左移23位，移到浮点数的指数部分
    or eax, ecx
done:
    ret
```

在转换完复数数组后，需要对复数数组位数进行拓展，调用 get_maxn 函数计算要拓展到的位数：

```
mov ecx, nn1
add ecx, nn2
mov eax, 1
main_loop:
    cmp eax, ecx
    jge done
    sal eax, 1
    jmp main_loop
done:
    ret
```

上面代码中，nn1 为第一个数字的位数，nn2 为第二个数字的位数，将两个相加，得到乘积的最大位数，然后将 1 赋给 eax，不断乘 2 直到 eax 大于乘积的最大位数，这

样就把位数拓展到了 2 的幂次。

2.3.2 fft 变换

采用伪代码说明算法过程：

Algorithm 2 fft

Input: 数组的长度 num_len, 数组首地址 num_p, 傅里叶逆变换标志 inv
procedure ASM PROCEDURE
if num_len == 1 **then**
goto *done*.
end if
计算数组 mid = num_len ÷ 2, 将数组分为两部分进行递归计算.
首先保存在 temp 数组里.
遍历元素组，将偶数部分放到 temp 数组前半部分，奇数部分放到 temp 数组后半部分.
然后将 temp 数组复制回原数组.
调用两次 fft 函数计算前半部分和后半部分.
计算当前数组的 fft 变换结果：
edx 作为遍历下标，遍历 0→mid.
main_loop:
首先计算单位复根, 如果 inv 为 1，单位根变为共轭复数。
根据公式 (1)、公式 (2) 计算得到 fft 的值，保存在 temp 数组中.
inc edx
cmp edx, mid
jz main_loop_break **goto** *main_loop*.
将 temp 数组复制回原数组.
done:
ret.
end procedure

$$A(\omega_n^k) = A_1(\omega_{\frac{n}{2}}^k) + \omega_n^k A_1(\omega_{\frac{n}{2}}^k) \tag{1}$$

$$A(\omega_n^{k+\frac{n}{2}}) = A_1(\omega_{\frac{n}{2}}^k) - \omega_n^k A_1(\omega_{\frac{n}{2}}^k) \tag{2}$$

其中涉及到 cp_mul 函数，代码如下：

```
1 LOCAL temp:cp
2 fld cp1.x
3 fmul cp2.x
4 fld cp1.y
5 fmul cp2.y
```

```
6   fsubp  st(1),st(0)
7   fstp  temp.x
8
9   fld  cp1.x
10  fmul  cp2.y
11  fld  cp1.y
12  fmul  cp2.x
13  faddp  st(1),st(0)
14  fstp  temp.y
15
16  fld  temp.x
17  fld  temp.y
18  ret
```

调用函数后，结果保存在 fpu 中，外部将结果从 fpu 弹出就能得到函数结果。

在得到 fft 变换函数后，首先对两个输入的数字进行 fft 变换，在复数相乘后，进行 ifft 得到乘积。

2.3.3 复数数组相乘

在得到两个数字的点值表示后，将对应下标复数相乘，得到两个数的乘积，由于代码过长，还是简述一下做法，esi 保存数组 1 地址，edi 保存数组 2 地址，遍历两个数组，调用 cp_mul 函数得到结果，将结果保存在 esi 中。

2.3.4 输出处理

由于结果得到的是浮点数组，首先需要把它转换为整数数组，然后进行进位处理，最后去除前导零。需要三个循环，将函数分为三个部分介绍。

浮点数组转换为整数数组：

由于代码过长但运行逻辑简单，所以只说明循环的运行过程。将浮点数组长度赋给 ecx 进行 loop 循环，遍历得到的浮点数组，利用 fld 加载浮点数，fistp 将浮点数转换为整数储存在 ans 数组中，当 ecx 为 0 时跳出循环。

进位处理：

```
1   mov esi, offset ans
```

```
2    mov ebx, 10
3    mov ecx, num_len
4    loop3:
5        xor edx,edx
6        mov eax, [esi]
7        div ebx
8        mov [esi], edx
9        add [esi+4], eax
10       add esi, 4
11       loop loop3
```

去除前导零：

由于将乘数位数拓展到了 2 的幂次，所以可能存在前导零

然后是输出结果，输出的时候判断一下是否是负数，是负数先输出负号，然后从数组末尾向开始输出数字即可。

2.4 实验结果

计算两个 990 位数相乘结果如下：

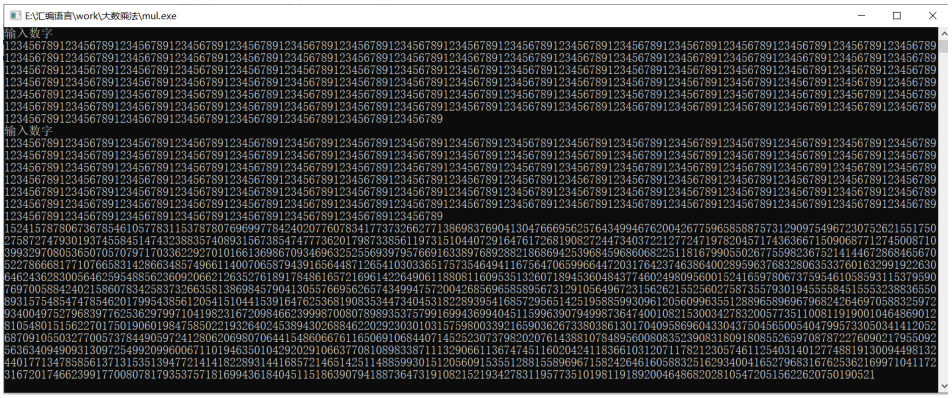


图 **2-1** 相乘结果

在汇编文件中，由于当定义的数组长度超过 1e5 时，构建程序的耗时过长，所以定义的复数数组大小为 8200，比 2 的 13 次方 8192 大一点，这样意味着输入数字最长为 4096 位，经测试，输入 4000 位数字能够得到正确结果。

2.5 心得体会

通过完成这次实验，我了解汇编语言的大致用法，为后面的实验打下了基础。

3. Windows 界面文件比较

3.1 实验要求

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。

3.2 实验内容

通过调用 win32 api 函数实现窗口界面。创建两个富文本编辑窗口；通过两个按钮将文本内容加载到文本编辑窗口中，同时，用户还可以自行编辑文本中的内容；在按下开始比较的按钮后比较两个窗口中的文本。

在本次实验中，由于大部分采用 win32 函数，为了保持整体的一致性，循环和条件采用.while 和.if 伪指令实现。

3.3 实验过程

3.3.1 主窗口

在使用 win32 汇编创建窗口的时候，首先需要创建并注册一个窗口类，然后根据窗口类创建窗口，创建窗口类代码如下：

```
1  invoke RtlZeroMemory, addr st_window_class, sizeof st_window_class
2  invoke LoadCursor,0,IDC_ARROW
3  mov    st_window_class.hCursor,eax
4  push h_instance
5  pop st_window_class.hInstance
6  mov    st_window_class.cbSize,sizeof WNDCLASSEX
7  mov    st_window_class.style,CS_HREDRAW or CS_VREDRAW
8  mov    st_window_class.lpfnWndProc,offset _proc_main_window
9  mov    st_window_class.hbrBackground,COLOR_BTNFACE+1
```

```
mov    st_window_class.lpszClassName,offset str_class_name
invoke RegisterClassEx, addr st_window_class
```

通过上面的代码，设置了窗口的光标、回调函数、背景颜色和类名，然后使用 win32 api 函数 CreateWindowEx 函数创建窗口：

```
invoke CreateWindowEx, WS_EX_OVERLAPPEDWINDOW, \
    offset str_class_name, \
    offset str_caption_main, \
    WS_OVERLAPPEDWINDOW xor WS_SIZEBOX, \
    100, 100, 1100, 700, NULL, NULL, h_instance, NULL
mov h_main_window, eax
```

通过上面的代码，创建了一个以 str_caption_main 字符串为窗口名的主窗口，主窗口的窗口句柄储存在 h_main_window 变量中。

3.3.2 富文本控件

在创建主窗口成功后，系统向主窗口发送 WM_CREATE 消息，在回调函数中处理 WM_CREATE 消息，进行窗口界面的初始化，在初始化函数中，创建两个富文本窗口，文本比较、文本加载按钮。创建的富文本窗口代码如下：

```
invoke CreateWindowEx, WS_EX_CLIENTEDGE, \
    offset str_edit_class_name, NULL, \
    WS_CHILD or WS_VISIBLE or WS_VSCROLL or \
    WS_HSCROLL or ES_MULTILINE, \
    0, 0, 545, 600, h_main_window, 0, \
    h_instance, NULL
mov h_window_edit1, eax

invoke CreateWindowEx, WS_EX_CLIENTEDGE,\
    offset str_edit_class_name, NULL, \
    WS_CHILD or WS_VISIBLE or WS_VSCROLL or \
    WS_HSCROLL or ES_MULTILINE, \
    545, 0, 545, 600, h_main_window, 1, \
    h_instance, NULL
```

```
15  mov h_window_edit2, eax
```

上面的代码创建了两个控件 id 分别为 0，1 的富文本编辑控件，控件句柄分别储存在 h_window_edit1，和 h_window_edit2 变量中。

3.3.3 文本比较按钮

首先是创建按钮代码：

```
1  invoke          CreateWindowEx,NULL,\
2       offset szButton,offset szButtonText,\
3       WS_CHILD or WS_VISIBLE,\
4       450,605,200,30,\
5       h_main_window,5,h_instance,NULL
```

win32 实现了默认的按钮类，所以在创建的时候直接使用这个类就行，文件比较控件 id 为 5。

下面是实现文件比较功能，当按下文件比较按钮的时候，系统会向按钮的父类窗口发送 WM_COMMAND 消息，在主窗口的回调函数中处理 WM_COMMAND 消息，wParam 高 16 位储存控件类型，可以通过 wParam 高 16 位判断消息是否来自按钮，wParam 低 16 位存储控件 id，所以可以通过 id 判断哪个按钮被按下，消息处理代码如下：

```
1  mov eax, wParam
2  mov ecx, wParam
3  shr eax, 16
4  .if ax == BN_CLICKED
5      .if cx == 5
6          call _text_compare
7      .elseif cx == 3
8          mov eax, h_window_edit1
9          mov h_forward_edit,eax
10         call _load_file
11     .elseif cx == 4
12         mov eax, h_window_edit2
13         mov h_forward_edit,eax
```

```
14            call _load_file
15        .endif
16    .endif
```

由于主窗口中存在三个按钮，所以需要判断消息来自哪个按钮，当文本比较按钮被按下，调用 _text_compare 函数进行文本比较，_text_compare 代码过长，所以说明函数运行原理。

_text_compare 函数实现了对于同一行不同文本进行高亮处理。

首先，获取两个窗口文本的行数，便于一行一行遍历比较，代码如下：

```
1  invoke SendMessage, h_window_edit1, EM_GETLINECOUNT, 0, 0
2  mov line1_cnt,eax
3  invoke SendMessage, h_window_edit2, EM_GETLINECOUNT, 0, 0
4  mov line2_cnt,eax
5  mov ebx, 0
```

通过 SendMessage 函数，向对应窗口发送消息，行数存在返回值中。接下来按照 line1_cnt 进行遍历，获取行文本方法如下：

```
1  invoke SendMessage, h_window_edit1, EM_LINEINDEX, ebx,0
2  mov char1_pos, eax
3  invoke SendMessage, h_window_edit1, EM_LINELENGTH, char1_pos, 0
4  add eax, char1_pos
5  invoke SendMessage, h_window_edit1, EM_SETSEL, char1_pos, eax
6  invoke SendMessage, h_window_edit1, \
7    EM_GETSELTEXT,0, offset str_buffer1
8  mov esi, offset str_buffer1
9  mov byte ptr [esi+eax], 0
```

上面代码中，第一个 SendMessage 输入当前行号，返回当前行第一个字符在整个窗口文本中的位置；第二个 SendMessage 输入字符位置，获得字符所在行的行长度；第三个 SendMessage 通过输入第一个字符位置和最后一个字符位置设置选中区域；第四个 SendMessage 获取选中区域文本存放到 str_buffer1 中，返回值为文本长度。选中当前行，不仅可以获取文本，也为后面行文本高亮做了前置工作。

同样方式获取另一个窗口当前行文本后，进行比较，如果出现不同，需要对文本进行高亮处理，方法如下：

```
mov st_cf.crBackColor, 0000FF00h
invoke SendMessage, h_window_edit1, EM_SETCHARFORMAT,\
SCF_SELECTION,addr st_cf
mov st_cf.crBackColor, 000000FFh
invoke SendMessage, h_window_edit2, EM_SETCHARFORMAT,\
SCF_SELECTION,addr st_cf
```

在设置了字体结构体的背景色后，向文本编辑器发送 EM_SETCHARFORMAT 消息，SCF_SELECTION 表示改变的是选中区域。这样就实现了高亮当前行的效果。

在遍历完所有行后，如果没有出现不同的行，就显示文本相同的提示。

```
invoke MessageBox, h_main_window, offset str_same,\
 offset str_caption_edit, MB_OK
```

3.3.4 文件加载按钮

在主窗口接收到加载文件按钮被按下后，调用文件加载函数，在文件加载函数中，首先创建一个打开文件的对话框:

```
invoke          RtlZeroMemory,addr st_of,sizeof st_of
mov    st_of.lStructSize,sizeof st_of
push h_main_window
pop    st_of.hwndOwner
mov    st_of.lpstrFilter,offset str_filter
mov    st_of.lpstrFile,offset str_file_name
mov    st_of.nMaxFile,MAX_PATH
mov    st_of.Flags,OFN_FILEMUSTEXIST or OFN_PATHMUSTEXIST
mov    st_of.lpstrDefExt,offset str_default_ext
invoke          GetOpenFileName,addr st_of
```

GetOpenFileName 函数将用户选择的文件文件名保存在 str_file_name 中，接下来，根据文件名创建文件句柄，将文件内容写入到文本编辑窗口中，代码如下:

```
invoke CreateFile,addr str_file_name,\
    GENERIC_READ or GENERIC_WRITE,\
    FILE_SHARE_READ or FILE_SHARE_WRITE,0,OPEN_EXISTING,\
```

```
        FILE_ATTRIBUTE_NORMAL,0
    mov     h_file ,eax
    mov     st_es.dwCookie ,TRUE
    mov     st_es.pfnCallback ,offset  _ProcStream
    invoke          SendMessage ,h_forward_edit ,EM_STREAMIN,\
    SF_TEXT,addr  st_es
```

在得到文件名后，通过 CreateFile 打开文件，得到文件句柄后，创建一个 EDITSTREAM 结构体，dwCookie 决定流入还是流出，pfnCallback 是回调函数，用于 SendMessage 的 EM_STREAMIN 消息创造数据流写入编辑窗口中。

3.4 实验结果

下面是比较文本中存在差异的结果：

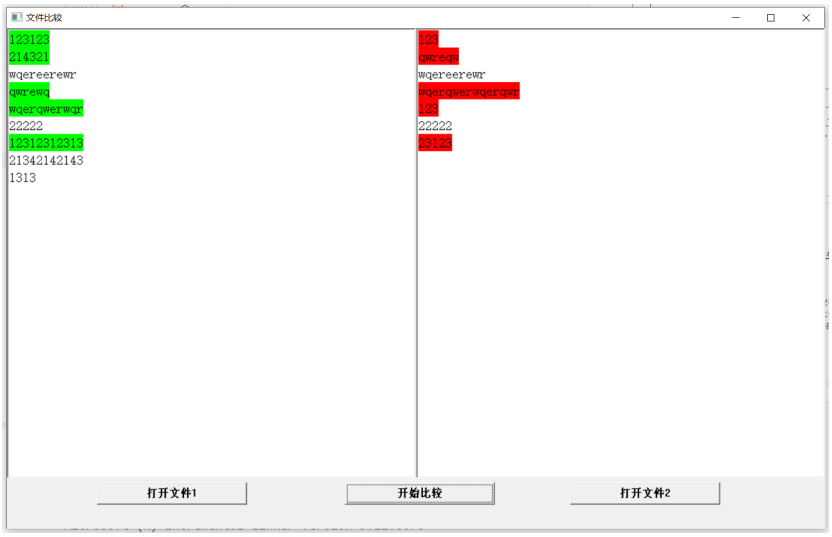


图 3-2 文本存在差异

下面是比较文本相同的结果：

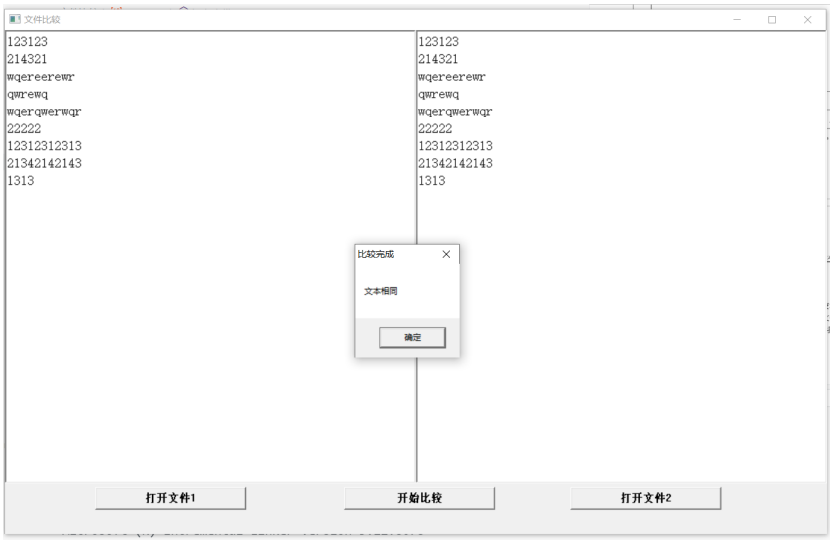


图 **3-3** 文本相同

3.5 可改进的地方

还存在两个未实现的地方：

1) 显示行号

本来想在窗口的最左侧显示行号，但要是显示在文本窗口中就会出现 vim 那样复制的时候连着行号一起复制的问题。

实际的解决办法应该是使用 gdi 绘制显示行号的区域，但当时没有学习 gdi，而又赶着实现计算器，所以没有实现这个功能。

2) 滚动条同步移动

常见的文本比较程序还有一个功能就是同步移动滚动条，但由于我的程序是两个窗口显示文本，所以只能滚动一边的窗口，一旦文本过长看着十分别扭。

解决办法应该是实现两个文本窗口的子类化，每个窗口对接收到鼠标滚动的消息进行处理，让另一个窗口也同步移动，但由于实现起来过于复杂，我最终还是放弃了实现这个功能。

3.6 心得体会

通过完成这次实验，我了解了 windows 窗口运行的过程，学会了使用部分 win32 api 函数，为实现计算器界面打下了基础。

4. Windows 界面计算器

4.1 实验要求

结合 Windows 界面编程，实现完善的计算器功能，支持浮点运算和三角函数等功能。

4.2 实验内容

利用 win32 api 绘制窗口，用浮点寄存器计算，编写函数将中缀表达式转换为后缀表达式得到结果。

4.3 实验过程

4.3.1 界面绘制

界面最主要部件就是按钮，windows 界面的过程在实验二中已经说明，这里不在赘述，这里只说明回调函数在识别到按钮被按下时的识别操作，代码如下：

```
if eax == WM_COMMAND
  mov eax, wParam
  mov ecx, wParam
  shr eax, 16

  .if ax == BN_CLICKED
      call _check_btn
  .endif
```

判断消息类型在实验二已经说明，在识别到按钮被按下后，调用 _check_btn 函数判断哪个按钮被按下进行对应的处理，_check_btn 函数代码如下：

```
.if cx >= 1 && cx <= 26
    mov esi, offset map
    xor eax,eax
    mov ax, cx
    mov edi ,offset id_express
```

```
6       mov ecx, id_len
7       mov [edi+4*ecx], eax
8       mov esi, [esi+4*eax]
9       invoke _input, esi
10      inc id_len
11  .elseif cx == 27
12      call _back
13  .elseif cx == 28
14      mov esi ,offset str_express
15      mov byte ptr [esi], 0
16      mov express_len, 0
17      mov id_len, 0
18      invoke SetWindowText, h_express, esi
19      invoke SetWindowText, h_ans, NULL
20  .elseif cx == 29
21      call _cal
22  .endif
```

按钮编号 1-26 为操作符和数字按钮，27 为回退按钮，28 为清屏按钮，29 为计算结果。对于操作符和操作数构成的表达式的保存，我将操作符和操作数映射为不同的 id，在识别到 1-26 id 时，将值插入 id_express 中，但由于还得把表达式显示在窗口上，所以通过 map 一个映射数组得到操作符和数字的字符串形式，调用 _input 函数将表达式显示在窗口上。

由于 _input 函数不是很重要，只简要说说它的运行过程，首先得到了要插入的字符串后，将字符串插入到 str_express 字符数组的末尾，然后用 SetWindowText 函数将字符数组显示到窗口上。

接下来说明回退功能的实现，也挺简单的，如果 id_express 长度不为 0 ，取出 id_express 末尾 id ，然后根据 map 数组和 id 得到要删除字符串，计算长度，str_express 对应长度位置置 0 ，调用 SetWindowText 函数刷新窗口即可。

清屏功能直接修改 id_express 和 str_express 数组长度就行，然后刷新一下表达式窗口和结果窗口。

接下来说明 _cal 函数，也是核心部分，中缀表达式转换计算得到结果。

4.3.2 中缀表达式转换

在说明主函数之前，首先介绍几个辅助函数。

栈的结构为一个数组，数组下标 0 存放栈的长度，后面存放栈的元素，转换中缀表达式需要符号栈和数值栈。

栈的插入函数：

```
_push PROC uses esi, p_stack, item
  mov esi, p_stack
  mov ecx, [esi]
  inc ecx
  mov eax, item
  mov [esi+4*ecx], eax
  mov [esi], ecx
  ret
_push ENDP
```

输入栈的指针和元素，因为可能向数值栈或者符号栈插入元素。接下来是删除栈顶元素，实际上就是栈长度减一：

```
_pop PROC uses esi, p_stack
  mov esi, p_stack
  mov eax, [esi]
  .if eax == 0
      mov error, 1
      jmp done
  .endif
  dec eax
  mov [esi], eax
  done:
      ret
_pop ENDP
```

这里需要做一个错误标记，如果栈的长度为 0，则 error 标记为 1，没有返回值，获取栈顶元素函数如下：

```
_top PROC uses esi, p_stack
```

```
    mov esi, p_stack
    mov eax, [esi]
    .if eax == 0
        mov eax, 0
        jmp done
    .endif
    mov eax, [esi+4*eax]
    done:
        ret
_top ENDP
```

输入栈的指针，返回元素值，如果栈为空，为了继续运算，还是返回 0 值，因为最后两个栈一定为空，所以一定会调用 pop ，会触发错误，不会出现错误没被发现的情况。

由于输入的数字是一个一个位输入的，所以需要将输入的数字字符串转换为浮点数存储，采用 sscanf 函数转换：

```
_getnum PROC uses esi
  LOCAL temp:dword
    mov esi, offset str_input_num
    ; temp_out 是 64位数
    invoke sscanf, esi, offset str_in_float, offset temp_out
    ;由于sscanf转换为64位，所以得把得到的数转化为32位才能插入栈中
    fld temp_out
    fstp temp
    invoke _push, offset sta_num, temp
    ret
_getnum ENDP
```

还有就是获取操作符优先级的函数：

```
_getpr PROC uses esi, operation
  mov esi, offset key
  mov eax, operation
  mov eax, [esi+4*eax]
```

```
5     ret
6   _getpr ENDP
```

接下来介绍中缀表达式转换函数，在计算表达式时，在总体表达式左右分别插入左右括号便于计算。

由于前置正负号的存在，导致转换数字有点复杂，下面介绍识别数字过程：

首先前置正负号左边一定为空 (由于开头插入了左括号，所以不会出现空) 或者左括号或者操作符。当识别到正负号时，如果上一个为操作符或者左括号，那么这个为前置正负号。

接下来转换小数，当识别到小数点时，设置一个标记，如果识别的数被插入数值栈，标记清楚；如果识别到小数点时标记还存在，说明一个数存在两个小数点，将 error 标记置 1。

上面就是数字处理部分，下面给出计算表达式的伪代码：

Algorithm 3 计算表达式

```
procedure ASM PROCEDURE
    遍历 id_express 数组.
    if 当前 id 为数字 then
        数字保存过程.
    else if 当前 id 为操作符 then
        将保存的数字转换为浮点数保存在数值栈中.
        if 识别到为 PI then
            数值栈插入 PI.
        else if 当前 id 为右括号 then
            不断弹出符号计算直到遇到左括号.
            计算前置正负号.
        else if 当前 id 为操作符 then
            不断弹出符号并计算，直到栈顶符号优先级小于等于当前符号.
            由于表达式左结合，所以优先级相等时得弹出并计算栈顶符号.
            将当前符号插入符号栈.
        end if
    end if
end procedure
```

4.3.3 计算结果

计算结果过程再 _cal_op 函数中，首先取出操作符和一个数：

```
1   LOCAL n1:dword, n2:dword
```

```
    invoke _top,offset sta_op
    mov edx, eax
    invoke _pop,offset sta_op
    invoke _top,offset sta_num
    mov n1, eax
    invoke _pop,offset sta_num
```

接下来判断操作符是一元还是二元操作数，如果是二元操作数，还得再取一个数，代码如下：

```
    .if edx >= 19
      fld n1
      .if edx == 22    ;sin运算
          fsin
      .elseif edx == 23 ;cos运算
          fcos
      .elseif edx == 24 ;tan运算，fptan得到tan(x)和1，得弹出1
          fptan
          fstp n1
      .elseif edx == 25 ;atan计算方式为atan(st(1)/st(0))，所以得先插入1
          fld1
          fpatan
      .elseif edx == 26 ; st(1)*log2(st(0)),所以修改一下顺序
          fstp n1
          fld1
          fld n1
          fyl2x
      .elseif edx == 27 ;取反，说明有前置负号
          fchs
      .endif
```

接下来是二元运算：

```
    .elseif edx >= 11 && edx < 19
      invoke _top,offset sta_num
```

```
    mov n2, eax
    invoke _pop,offset sta_num
    ; 由于是 n2 出现在前，所以得换一下顺序
    fld n2
    fld n1
    .if edx == 11  ;加法
        fadd
    .elseif edx == 12 ;减法
        fsub
    .elseif edx == 13 ;乘法
        fmul
    .elseif edx == 14 ;除法
        fdiv
    .elseif edx == 16 ;取模
        fstp n1
        fstp n2
        fld n1
        fld n2
        fprem
    .endif
```

最后向数值栈中插入结果

```
fstp n1
invoke _push,offset sta_num, n1
```

4.3.4 结果输出

首先结果出栈:

```
invoke _top,offset sta_num
mov temp, eax
invoke _pop,offset sta_num
```

由于 windows 的 wsprintf 函数不能转换浮点数，所以只能采取折中的方法，将结果乘 1000，再四舍五入得到整数，模 1000 得到结果的小数部分，代码如下:

```
fld temp
fmul pow_num
;判断结果是否为负数
ftst ;测试是否为负数
FSTSW ax  ;取 fpu 状态寄存字，如果为负数，对应标志位为1
and ax, 0100h
.if ax != 0
    mov index, -1 ;index为负数标志
    fchs  ;如果为负数取反
.endif
;判断结果是否为负数
fistp temp
mov eax, temp
mov ebx, 1000
xor edx, edx
div ebx
push edx
invoke wsprintf,offset str_ans_int,offset str_out_ans,eax
pop edx
invoke wsprintf,offset str_ans_float,offset str_out_ans,edx
```

接下来就是将得到的整数部分和小数部分组装为字符串，代码如下：

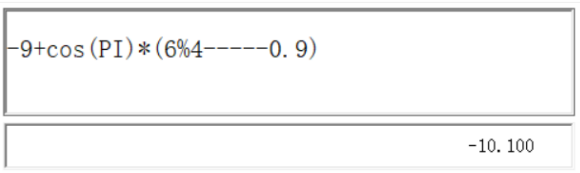
```
mov esi, offset str_ans
.if index == -1
    mov byte ptr [esi], '-'
    inc esi
.endif
invoke        lstrcpy, esi, offset str_ans_int
invoke lstrlen, offset str_ans_int
add esi, eax
mov byte ptr [esi], '.'
inc esi
invoke        lstrcpy, esi, offset str_ans_float
```

接下来进行错误判断，如果计算过程中出现符号栈或者数值栈错误，error 标记会置 1，说明出错；如果符号栈或者数值栈中还存在元素，说明出错；接下来判断计算过程中的错误：fpu 状态寄存器中，无效操作异常位于 0，除零异常位于 2，溢出异常位于 3，栈错误位于 6，所以得到状态字后和 0x004d 进行与运算，如果不为 0，说明出错，代码如下：

```
FSTSW ax
and ax, 004dh
mov esi, offset sta_op
mov ecx, [esi]
mov esi, offset sta_num
mov edx, [esi]
.if error == 1 || ax != 0 || ecx != 0 || edx != 0
    invoke SetWindowText, h_ans, offset str_error
.else
    invoke SetWindowText, h_ans, offset str_ans
.endif
```

4.4 实验结果

下面是计算表达式的结果图：



(a) 前置负号计算

(b) 复杂表达式

图 **4-4** 计算器结果

可以看出，计算器能够得到正确的结果。

下面是错误判断的结果：

(a) 表达式错误　　(b) 数值溢出

(c) 除零异常　　(d) 小数异常

图 **4-5** 错误显示

4.5　心得体会

通过三个实验，我对汇编语言有了一定的了解，能够实现一些功能，有一定的收获。