汇编语言与逆向技术实验报告

Lab3-bubble

学号：2112492 姓名：刘修铭 专业：信息安全

一、 实验目的

1、熟悉汇编语言的整数数组；

2、熟悉基址变址操作数、相对基址变址操作数；

3、掌握排序算法的底层实现细节

二、 实验环境

Windows 记事本的汇编语言编写环境

MASM32 编译环境

Windows 命令行窗口

三、 过程说明

1. 编辑：编写汇编程序bubble_sort.asm。

程序的功能是将Windows命令行输入的十进制无符号整数，转换成对应的十六进制整数，输出在Windows命令行中。

（输入的十进制无符号整数的范围是0到4294967295，输出对应的十六进制整数，对应的范围是00000000h到FFFFFFFFh）。

其中包括：

- StdIn函数：使用StdIn函数获得用户输入的十进制整数。
- StdIn函数的定义在\masm32\include\masm32.inc，库文件是\masm32\lib\masm32.lib
- BubbleSort过程：对输入的数组进行排序。
- StdOut函数：使用StdOut函数在Windows命令函中输出十六进制整数的ASCII字符串。

2. 编译：使用ml将bubble_sort.asm文件汇编到bubble_sort.obj目标文件。

编译命令：“\masm32\bin\ml /c /Zd /coff bubble_sort.asm”

```
C:\Users\98712\OneDrive\桌面>\masm32\bin\ml /c /Zd /coff bubble_sort.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997.  All rights reserved.

 Assembling: bubble_sort.asm

***********
ASCII build
***********
```

3. 链接：使用link将目标文件bubble_sort.obj链接成bubble_sort.exe可执行文件。

链接命令：“\masm32\bin\Link /SUBSYSTEM:CONSOLE bubble_sort.obj”

```
C:\Users\98712\OneDrive\桌面>\masm32\bin\Link /SUBSYSTEM:CONSOLE bubble_sort.obj
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.
```

4. 测试：直接执行bubble_sort.exe可执行文件。

```
C:\Users\98712\OneDrive\桌面>bubble_sort.exe
Please input ten numbers:1 99 64 55 9999 100000 256 512 1024 1017
The result is:1 55 64 99 256 512 1017 1024 9999 10000
```

四、 源代码

```
.386
.model flat, stdcall
option casemap :none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\masm32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\masm32.lib
.stack 4096
.data;定义数据段
     str1 byte "请输入十个无符号整数:",0
     str2 byte "冒泡排序结果为:",0
     istr byte 80 dup(0)
     ostr byte 80 dup(0)
     array dword 12 dup(0)
     const10 dword 10
```

```
.code;定义代码段

main PROC
     invoke StdOut,addr str1
     invoke StdIn,addr istr,80
     call str2array
     call bubble
     call array2str
     invoke StdOut,addr str2
     invoke StdOut,addr ostr
     invoke ExitProcess,0
main ENDP

str2array proc;即上一次的字符串转数字
     mov eax,0
     mov ebx,0
     mov ecx,0
     mov esi,0
L1:
     mov bl,[istr+esi]
     cmp bl,20h
     jne L2
     add ecx,4
     inc esi
     mov bl,[istr+esi]
L2:
     sub bl,30h
     mov eax,[array+ecx]
     mul const10
     add eax,ebx
     mov [array+ecx],eax
     inc esi
     cmp [istr+esi],0
     jne L1
     ret
str2array endp

bubble proc
     mov ecx,10;共十个数
L3:
     dec ecx;十个数，外层循环需 9 次，每次-1
       cmp ecx,0;判断是否循环结束
     je exit;若结束，则退出排序循环
     mov ebx,ecx;将外层循环的值赋给 ebx，用来计数内层循环
```

```
    mov esi,0
L4:
    mov eax,[array+esi];需要用 array 来访问字符串元素！！
    cmp eax,[array+esi+4]
    jle L5
    xchg eax,[array+esi+4];若不相等，则交换值
    mov [array+esi],eax
L5:
    dec ebx
    cmp ebx,0
    je L3
    add esi,4h
    jmp L4
exit:
    ret
bubble endp

array2str proc;
    mov esi,0;访问 ostr 每一位
    mov edi,0;访问 array 每一位
    mov ecx,10;计数外层循环(共 10 个数)
    mov ebx,0;计数出栈次数
    mov eax,[array+edi]
L6:
    mov edx,0;存商
    div const10;商存在 eax，余数存在 edx
    add edx,30h
    push edx;入栈（先算的末位，先进后出）
    inc ebx;计数栈中有几个字符，决定出栈次数
    cmp eax,0;若商为 0，则该数已取完
    jne L6
L7:
    pop eax
    mov [ostr+esi],al
    inc esi
    dec ebx
    cmp ebx,0;ebx 为 0 则说明当前整数的各位已经全部出栈，跳出循环
    jne L7
    mov [ostr+esi],20h;加一个空格
    inc esi
L8:;判断是否终止
    dec ecx
    cmp ecx,0;ecx 为 0 则说明十个整数全被弹出，结束过程
    je L9
```

```
    add edi,4
    mov eax,[array+edi]
    jmp L6
L9:
    ret
array2str endp
end main
```

五、 数组知识点总结

（一）基址变址

基址变址（base-index）操作数把两个寄存器的值相加，得到一个偏移地址。两个寄存器分别称为基址寄存器（base）和变址寄存器（index）。格式为[base + index]，例如 mov eax, [ebx + esi]。在例子中，ebx 是基址寄存器，esi 是变址寄存器。基址寄存器和变址寄存器可以使用任意的 32 位通用寄存器。

相对基址变址（based-indexed with displacement）操作数把偏移、基址、变址以及可选的比例因子组合起来，产生一个偏移地址。常见的两种格式为：[base + index + displacement]和 displacement[base + index]，例子如下：

```
table dword 10h, 20h, 30h, 40h
row_size = ($ - table)
    dword 50h, 60h, 70h, 80h
        dword 90h, 0a0h, 0b0h, 0c0h
mov ebx, row_size
mov esi, 2
mov eax, table[ebx + esi * 4]
```

table 是一个二维数组，共 3 行 4 列。ebx 是基址寄存器，相当于二维数组的行索引，esi 是变址寄存器，相当于二维数组的列索引。

（二）行和列的顺序

从汇编语言程序员的视角来看，二维数组是一维数组的高阶抽象。对于二维数组在内存中行列的存储，高级语言一般采用下面的两种方法：行主序（row-major order）和列主序（column-major order）。使用行主序存储（最常使用）时，第一行放在内存块的开始，第一行的最后一个元素后接第二行的第一个元素。使用列主序存储时，第一列的元素放在内存块的开始，第一列的最后一个元素后接第二列的第一个元素。

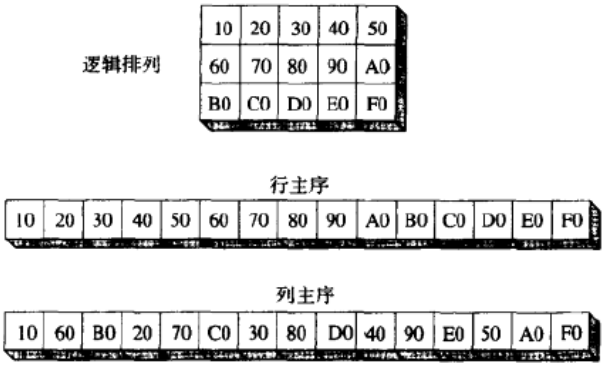

图 9.4　行主序和列主序