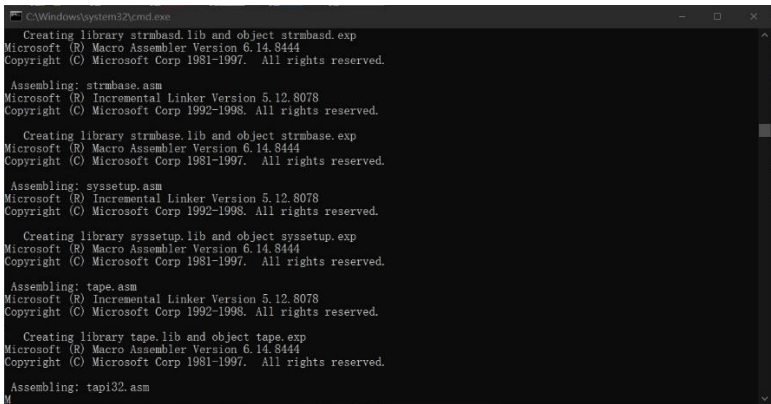
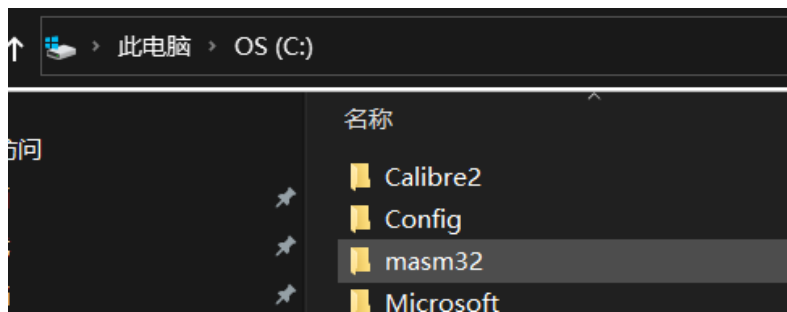


《汇编语言》实验报告 07

班级	2022 秋	实验日期	2022.12.30	实验成绩	
姓名	黄勛	学号	22920212204392		
实验名称	汇编语言第七次实验				
实验目的、要求	1) 汇编指令综合应用 2) 熟悉 32 位 Intel 汇编指令				
实验内容、步骤及结果	<p>1、请使用 32 位的 Intel x86 的指令，打印计算 10000 以内的水仙数的程序（正确的“水仙花数”其实是一个 3 位数，在这里我们不限制它的位数。“水仙花数”是指一个 n 位数，其各个位的数字的 n 次方的和为它本身，例:153 是一个 3 位数，$153=1^3+5^3+3^3$，则 153 即为一个水仙花数。）；并在 32 位的 Intel x86 汇编语言环境下运行通过。需要注意的点是，在 32 位系统下，仍可以使用 16 位寄存器，但是地址变成 32 位了。存地址一般就需要用 32 位寄存器。</p> <p>分析：10000 以内的水仙数通过查找相关资料的值，不存在 1、2、5 位的答案，3 位的被称为水仙花数，4 位的被称为四叶玫瑰数，3 位的数需要每一位的立方和为本身，4 位的需要每一位的 4 次方和为本身，通过这个思路来枚举数字进行计算。</p> <p>编写过程：</p> <p>1) 实验环境设置</p> <p>解压实验发的编译器压缩包，运行 install.exe 文件，安装在 C 盘。</p> 				

安装结束后，在 C 盘目录下会产生一个 masm32 的文件夹



正确配置系统环境变量即可在其他目录进行编译连接操作。

2) 开始编写代码，先声明汇编与链接库

3) 声明数据段

```
.data
    str1 byte "水仙花数共有",0
    str2 byte "四叶玫瑰数共有",0
    str3 byte "个:",0
    endl byte 0Dh, 0Ah, 0
    sxhs dword 10 dup(0) ;存储水仙花数的数组
    symgs dword 10 dup(0) ;存储四叶玫瑰数的数组
    div10 dword 10 ;用于将数字转换为字符串的数字10
    temp byte 6 dup(0) ;用于转换的临时数组
    count1 byte 2 dup(0) ;记录水仙花数个数的数组
    count2 byte 2 dup(0) ;记录四叶玫瑰数个数的数组
```

4) 然后是代码段

```
main proc

    call sxh ;调用水仙花数函数
    call symg ;调用四叶玫瑰数函数
    ret

main endp
```

这是主函数，它调用了求水仙花数函数和四叶玫瑰数的函数

5) 接下来是水仙花数函数

```
sxh proc

    invoke StdOut, addr str1 ;输出"水仙花数共有"
    mov eax,98h ;设置起始数为98h
    xor ecx,ecx ;计数器设为0
    mov esi,eax ;将起始数设为esi
    mov dword ptr[sxhs],0 ;将水仙花数数组初始化为0
```

```

L1:
    mov eax, esi    :将esi赋值给eax
    inc eax        :将eax加1
    mov esi, eax    :将加1后的eax设为esi
    cmp eax, 3E7h   :如果esi大于3E7h, 跳转到L2
    ja L2
    call issxh      :调用判断水仙花数的函数
    test al, al     :如果issxh函数返回的al为0, 跳转到L1
    je L1
    mov eax, esi    :将esi赋值给eax
    mov dword ptr[sxhs+4*ecx], eax :将eax存入水仙花数数组中
    inc ecx         :计数器加1
    jmp L1          :跳转到L1

L2:
    mov byte ptr[count1], cl :将计数器的值存入count1数组中
    add byte ptr[count1], 30h :将count1数组中的数加30h, 以便输出字符
    push ecx        :将计数器的值压入堆栈
    invoke StdOut, addr count1 :输出count1数组中的数
    invoke StdOut, addr str3 :输出"个:"
    pop ecx         :将计数器的值弹出堆栈
    lea ebx, dword ptr[sxhs] :将水仙花数数组的首地址赋值给ebx

```

```

L4:
    push ecx        :将计数器的值压入堆栈
    mov eax, dword ptr[ebx] :将水仙花数数组中的数赋值给eax
    mov dword ptr[temp], eax :将eax存入temp数组中
    call dw2str     :调用将数字转换为字符串的函数
    invoke StdOut, addr temp :输出temp数组中的字符串
    add ebx, 4h     :ebx加4h, 指向下一个水仙花数
    pop ecx         :将计数器的值弹出堆栈
    loop L4         :循环L4
    invoke StdOut, addr endl :输出换行符
    ret
sxh endp

```

6) 判断水仙花数的函数

```

issxh proc
    pushad          :将所有寄存器压入堆栈
    mov eax, dword ptr[esp+32] :将参数赋值给eax
    xor ecx, ecx     :计数器设为0
L1:
    xor edx, edx     :清空edx

```

```

    div div10       :将eax除以10, 余数存入edx
    mov bl, dl       :将余数存入bl
    mul bl           :将bl乘以余数, 结果存入edx
    add ecx, edx     :将ecx加上edx
    cmp eax, 0       :如果eax不为0, 跳转到L1
    jne L1

L2:
    cmp ecx, eax     :如果ecx等于eax, 跳转到L3
    je L3
    xor eax, eax     :清空eax

L3:
    popad           :将所有寄存器弹出堆栈
    ret
issxh endp

```

7) 接下来是将数字转换为字符串的函数

```

dw2str proc
    pushad    ;将所有寄存器压入堆栈
    lea ebx, [temp]    ;将temp数组的首地址赋值给ebx
    mov eax, dword ptr[ebx]    ;将temp数组中的数赋值给eax
    xor ecx, ecx    ;计数器设为0
L1:
    xor edx, edx    ;清空edx
    div div10    ;将eax除以10, 余数存入edx
    add edx, 30h    ;将余数加30h, 以便输出字符
    push edx    ;将余数压入堆栈
    inc ecx    ;计数器加1
    cmp eax, 0    ;如果eax不为0, 跳转到L1
    jne L1

L2:
    pop eax    ;将余数弹出堆栈
    mov byte ptr[ebx], al    ;将余数存入temp数组中
    inc ebx    ;ebx加1, 指向下一个字符
    loop L2    ;循环L2
    popad    ;将所有寄存器弹出堆栈
    ret
dw2str endp

```

8) 编写四叶玫瑰数函数

```

symg proc
    invoke StdOut, addr str2    ;输出“四叶玫瑰数共有”
    mov eax, 3E7h    ;设置起始数为3E7h
    xor ecx, ecx    ;计数器设为0
    mov esi, eax    ;将起始数设为esi
    mov dword ptr[symgs], 0    ;将四叶玫瑰数数组初始化为0
L1:
    mov eax, esi    ;将esi赋值给eax
    inc eax    ;将eax加1
    mov esi, eax    ;将加1后的eax设为esi
    cmp eax, 270Fh    ;如果esi大于270Fh, 跳转到L2
    ja L2
    call issymg    ;调用判断四叶玫瑰数的函数
    test al, al    ;如果issymg函数返回的al为0, 跳转到L1
    je L1
    mov eax, esi    ;将esi赋值给eax
    mov dword ptr[symgs+4*ecx], eax    ;将eax存入四叶玫瑰数数组中
    inc ecx    ;计数器加1
    jmp L1    ;跳转到L1
L2:
    mov byte ptr[count2], cl    ;将计数器的值存入count2数组中
    add byte ptr[count2], 30h    ;将count2数组中的数加30h, 以便输出字符
    push ecx    ;将计数器的值压入堆栈
    invoke StdOut, addr count2    ;输出count2数组中的字符串
    invoke StdOut, addr str3    ;输出“个:”
    pop ecx    ;将计数器的值弹出堆栈

```

```

    lea ebx, dword ptr[syms] ;将四叶玫瑰数数组的首地址赋值给ebx
L4:
    push ecx ;将计数器的值压入堆栈
    mov eax, dword ptr[ebx] ;将四叶玫瑰数数组中的数赋值给eax
    mov dword ptr[temp], eax ;将eax存入temp数组中
    call dw2str ;调用将数字转换为字符串的函数
    invoke StdOut, addr temp ;输出temp数组中的字符串
    add ebx, 4h ;ebx加4h, 指向下一个四叶玫瑰数
    pop ecx ;将计数器的值弹出堆栈
    loop L4 ;循环L4
    ret
syms endp

```

9) 判断四叶玫瑰数的函数

```

issymg proc
    pushad ;将所有寄存器压入堆栈
    mov eax, dword ptr[esp+32] ;将参数赋值给eax
    xor ecx, ecx ;计数器设为0
L1:
    xor edx, edx ;清空edx
    div div10 ;将eax除以10, 余数存入edx
    mov bl, dl ;将余数存入bl
    mul bl ;将bl乘以余数, 结果存入edx
    add ecx, edx ;将ecx加上edx
    cmp eax, 0 ;如果eax不为0, 跳转到L1
    jne L1

L2:
    cmp ecx, 100h ;如果ecx等于100h, 跳转到L3
    je L3

    xor eax, eax ;清空eax
L3:
    popad ;将所有寄存器弹出堆栈
    ret
issymg endp

```

10) 将数字转换为字符串

```
dw2str proc
    pushad ;将所有寄存器压入堆栈
    lea ebx, [temp] ;将temp数组的首地址赋值给ebx
    mov eax, dword ptr[ebx] ;将temp数组中的数赋值给eax
    xor ecx, ecx ;计数器设为0
L1:
    xor edx, edx ;清空edx
    div div10 ;将eax除以10, 余数存入edx
    push edx ;将余数压入堆栈
    inc ecx ;计数器加1
    cmp eax, 0 ;如果eax不为0, 跳转到L1
    jne L1

L2:
    pop eax ;将余数弹出堆栈
    mov byte ptr[ebx], al ;将余数存入temp数组中
    inc ebx ;将ebx加1
    loop L2 ;循环L2
    popad ;将所有寄存器弹出堆栈
    ret
dw2str endp
```

● 运行结果:



经测试，可以正确输出原有的答案。

总结

这一次实验我对汇编语言指令有了更深的理解，并且这一次实验的实践操作颇丰，在练习编码的过程中我加深了 32 位指令的编写的操作的熟练度，我对每一个指令的用途和用法有了更深的认识；通过一步步地解决问题，我的实践能力提高了，这让我受益匪浅；具体遇到问题的解决方案我在下文做了更详细的总结，在此就不多赘述；在未来我还要探索汇编语言的更多应用方面，寻找更多问题，并在发现问题的过程中继续提高我对汇编语言的掌握能力，这是一次颇有意义的实验！

算法及其实现方式总结：

在寻找水仙花数的过程中，汇编语言的基址变址寻址方式和相对基址变址寻址方式起到了重要的作用。

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。

知识点总结:

1、交换指令 `xchg`。`xchg` 指令交换两个操作数的内容, 但不能直接交换两个内存的内容, 可用于数组内的交换。

2、循环指令 `loop`。`loop` 以 `ecx` 为循环计数器进行循环, 可用于遍历数组

3、判断指令 `cmp`, 条件跳转指令, 无条件跳转。

`Cmp` 指令通过修改 `cpu` 的标志位达到比较的目的, 通常和 `je, jne, ja, jb, jg, jl` 等条件跳转配合使用, 以及无条件跳转 `jmp` 指令。

4、间接寻址—变址操作数、基址变址操作数。

形如 `[eax + array1]` 的操作数便称为变址操作数, 最常用于遍历数据。

形如 `[ebx+esi]` 的操作数称为基址变址操作数, 可用于访问二维数组

5、`dup` 操作符用于声明大型数组, 包括需要初始化的数组和不要初始化的数组。

6、寄存器 `esi` 和 `edi` 是常用的变址寄存器。他们类似于指针, 对字符操作非常有用。

7、在处理 `dword` 类型的数组时, 偏移量是以 4 为单位, 而不是 1, 因为一个带符号双字节占用 4 个字节的内存空间

8、操作符 `lengthof` 用于计算数组的元素个数, 操作符 `sizeof` 用于计算数组占用字节空间