

# 第四次作业

## 目录

- 第四次作业
  - 目录
    - 第一题：宏汇编的使用
      - 要求
      - 实现
    - 第二题：多模块实现
      - 要求
      - 实现
      - 问题
      - 解决
    - 多 .obj 文件链接
    - 心得

### 第一题：宏汇编的使用

#### 要求

用macro实现替换过程调用

#### 实现

首先，写一段简单的加法程序，初步感受宏汇编，如下：

```
summ macro x1, x2, result    ;宏定义，有3个哑元
mov ax, x1
add ax, x2
mov result, ax
endm

xor ax, ax    ;清空寄存器
xor bx, bx
xor cx, cx
xor dx, dx
mov bx, 123    ;初始化参数
mov cx, 456

SUMM bx, cx, dx    ;宏调用
;call show
mov ah, 4ch
int 21h
```

输出结果如下：

```
579
```

使用 `DEBUG` 工具查看宏在程序运行过程中的表现，发现一开始并没有执行宏这段程序，后面执行时，也没有地址的跳转，而是直接把宏的代码拿过来执行，如图：

076A:0000	33C0	XOR	AX,AX
076A:0002	33DB	XOR	BX,BX
076A:0004	33C9	XOR	CX,CX
076A:0006	33D2	XOR	DX,DX
076A:0008	BB7B00	MOV	BX,007B
076A:000B	B9C801	MOV	CX,01C8
076A:000E	8BC3	MOV	AX,BX
076A:0010	03C1	ADD	AX,CX
076A:0012	8BD0	MOV	DX,AX
076A:0014	E80400	CALL	001B
076A:0017	B44C	MOV	AH,4C
076A:0019	CD21	INT	21
076A:001B	33C9	XOR	CX,CX
076A:001D	8BDA	MOV	BX,DX
076A:001F	8BC3	MOV	AX,BX

查阅资料，发现这种方式叫[宏展开](#)。

然后，尝试改写上一次作业，王爽教材实验7，部分代码对比如下：

修改前：

```
space:                                ;存储空格
    mov     dl, " "                   ;存入空格
    mov     byte ptr es:[di], dl      ;写入一个字节
    ret                                ;结束
```

调用：

```
call space
```

修改后：

```
space macro x
    mov     x, " "                   ;存入空格
    mov     byte ptr es:[di], x      ;写入一个字节
endm
```

调用：

```
space dl
```

宏与子程序都是编写结构化程序的重要手段,两者各有特色。相同之处，宏和子程序都可以定义为一段功能程序，可以被其他程序调用。不同之处如下：

- 宏指令利用哑元和实元进行参数传递。宏调用时用实元取代哑元，避免了子程序因参数传递带来的麻烦。

- 变元可以是指令的操作码或操作码的一部分，在汇编的过程中指令可以改变。
- 宏调用时没有保护断点和现场的概念，因为在汇编时已经用宏展开把这段程序插入主程序中了。而子程序每执行一次 `CALL` 指令，就要对断点和现场进行保护，把断点处的地址指针和相关寄存器入栈保存，从子程序中返回时要恢复现场和弹出断点地址。
- 宏的缺点是随着宏调用次数的增加，主程序代码会不断加长。

通常当程序较短、传递参数较多或要快速执行时，采用宏比较合适；当程序较长或对内存空间有要求时，选用子程序比较好。

## 第二题：多模块实现

### 要求

把大文件拆分成多个源文件

### 实现

把文件中的部分代码替换到外部文件中，部分代码分别放在单独的文件中并使用伪指令 `INCLUDE` 引入当前文件，即可实现多文件。被引入的文件后缀名为 `.MAC`。

分开后的文件经过 `INCLUDE` 的引入后，会被直接复制到主文件中，所以代码可以被拆的很细，甚至可以只在 `.ASM` 文件中写一句 `INCLUDE`，然后把其他代码放到外面的 `.MAC` 文件中。

下面是一种拆分形式的示例：

```
|
└─project
    extern.asm
    extern1.mac
    extern2.mac
    extern3.mac
    extern4.mac
    extern5.mac
    extern6.mac
    extern7.mac
    extern8.mac
    extern9.mac
    extern10.mac
```

文件之间的引用关系为：

```
|
└─extern.asm
    └─extern1.mac
        └─extern2.mac
            |  栈段和数据段
            |
            └─extern3.mac
                |  代码段
                |
                └─extern10.mac
                    |  space
                    |
                    └─extern4.mac
                        |  start
```

```

|
|—extern5.mac
|   row
|
|—extern6.mac
|   display
|
|—extern7.mac
|   show1
|
|—extern8.mac
|   show
|
|—extern9.mac
|   divdw

```

其中，在 `extern.asm` 中，只有一句代码：

```
include extern1.mac
```

`extern1.mac` 中引入 `extern2.mac` 和 `extern3.mac`：

```
include extern2.mac
include extern3.mac
```

`extern2.mac` 中放数据段和栈段代码，`extern3.mac` 中放代码段代码。代码段通过 `INCLUDE` 伪指令，把所有拆分后的过程调用引入。

## 问题

把下面两句：

```
include extern2.mac
include extern3.mac
```

调换顺序：

```
include extern3.mac
include extern2.mac
```

汇编时报错。

## 解决

由此可知，`INCLUDE` 只是单纯执行了复制的操作，并不会检查顺序，所以，如果要把代码拆分成多个文件，在 `INCLUDE` 的时候要注意顺序。

## 多 .obj 文件链接

在要求更改后，及时对程序进行了调整，修改后的文档如下：

参考文档：[DOSBOX中的简单的masm、link操作以及多个文件的连接输出hello word!](#)

这个人把打印“Hello World!”拆成了两个 .asm 文件，用 link 链接后，生成一个可执行文件，并能成功执行。参考其代码中的如下部分：

```
extrn display:far
```

这是其程序开头第一句，阅读王爽《汇编语言（第4版）》10.4 转移的目的地址在指令中的 call 指令，知加入 far 伪指令实现段间转移。由于外部文件属于另一个代码段，所以要段间转移。

在另一个文件中对应：

```
public display
...
display proc far
...
display endp
...
```

其中，proc 和 endp 伪指令成对使用，标示一个过程调用。

现在，理清了结构，回去改造第三次作业的打表程序。首先需要分两个文件，第一个文件 main.asm 放数据段、栈段和主要的代码段，通过这里调用过程调用。第二个文件 func.asm 放一个代码段，写入函数。需要修改的地方：

在 main.asm 里，开头添加外部过程调用：

```
extrn row:far
extrn display:far
```

在 func.asm 里，开头声明全局过程，保留一个代码段，把两个过程设成远程调用：

```
public row
public display

code segment
    assume cs:code
row proc far                ;每一行循环开始；存储数据过程
...
row endp                    ;存储数据过程结束
display proc far            ;每一行循环开始；显示数据过程
...
display endp                ;显示数据过程结束
code ends
end
```

分别对两个文件执行 masm 指令，得到 main.obj 和 func.obj，再使用 link 指令，输入如下：

```
link main.obj+func.obj
```

得 main.exe，执行得正确结果。

后续，可以把每个函数都拆开，这样的优点是调用方便，直观，面对代码少；缺点是链接困难，函数但凡一点改动要重新汇编和链接。

## 心得

通过学习宏汇编，加深了对参数传递的理解，学到了一种新的调用方式。

通过学习多文件，掌握了模块化编程的一种方法，减少代码在一个文件中的堆积冗余，增强了项目能力。