

- [author](#)

第11节 汇编语言程序

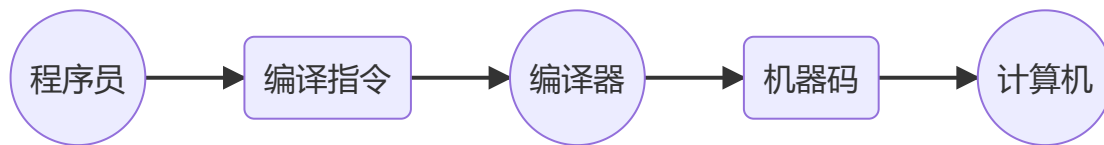
- [回到目录](#)
- [回到项目首页](#)
- [上一节](#)

❤❤❤ 汇编语言目前仍在发挥着不可替代的作用，在效率上无可替代，在底层，学习 linux 内核，计算机外围设备和驱动，都离不开汇编。Myblog: <http://nsddd.top>

- 1 汇编程序编译
- 2 汇编语言的结构
- 3 如何写出一个程序
- 4 程序中可能的错误
- 5 从源程序到程序运行
- 6 验证程序结果
- 7 debug 转载程序
- 8 分析
- 9 [...]和(...)的约定
 - 9.1 再约定 `idata` 表示常量
- 10 END 链接

1 汇编程序编译

- 汇编程序：包含汇编指令和伪指令的文本



汇编指令：对应机器码的指令，可以被编译为机器指令，最终被CPU执行。

伪指令：没有对应的机器码的指令，最终不被CPU所执行。

- 伪指令是由编译器来执行的指令，编译器根据伪指令进行相关的编译工作

```
push ds          ;数据段地址入栈
mov ax,ax        ;ax清零
push ax          ;0入栈
mov ax,data      ;设置段寄存器DS,段地址送ax
mov ds,ax        ;存入ds
mov ax,extra     ;设置段寄存器ES,段地址送ax
mov es,ax        ;存入es
lea si,source_buffer ;源操作数偏移地址送si
lea di,dest_buffer  ;目的操作数偏移地址送di
cld              ;设置方向标志DF=0, 地址增大
mov cx,40        ;设置传送数据数量
rep movsb        ;传送串
ret              ;返回DOS
main endp        ;主程序结束
code ends        ;代码段结束
end start        ;源程序结束
```

https://blog.csdn.net/weixin_45437521

在上面这个程序中，一直到assume都是伪指令。

```

assume cs:codesg

codesg segment
    mov ax,0123H
    mov bx,0456H
    add ax,bx
    add ax,ax

    mov ax,4c00H
    int 21H

codesg ends
end

```

这里我们更清楚地说明一下

```

1 | codesg segment
2 | codesg ends    ;段的结束

```

- (codesg是段的名称) 是一对成对使用的伪指令，也是写汇编程序时必须要用到的一对伪指令。因为一个有意义的汇编程序中至少要有有一个段。
- segment和ends的作用是定义一个段，segment说明一个段开始，ends说明一个段结束。注意区分end和ends
- end也是一个伪指令，是一个汇编程序的结束标记。
- 而ends标志着一个段的结束，可以把ends理解成“end segment”

2 汇编语言的结构

之前在debug中直接写入指令编写汇编程序，就相当于是一个交互式，不需要伪指令

- 适用于功能简单、短小精悍的程序
- 只需要包含汇编指令

对于单独编写的源文件再编译为可执行文件的程序

3 如何写出一个程序

求汇编编写 2^3

1. 定义一个段

```

1 | abc segment
2 | abc ends    ;段的开始和结束

```

2. 实现处理任务

```

1 | abc segment
2 |     mov ax,2    ;2放入ax寄存器
3 |     add ax,ax    ;2 + 2
4 |     add ax,ax    ;4 + 2
5 | abc ends        ;段的开始和结束

```

3. 指出程序在哪结束

```

1 | abc segment
2 |     mov ax,2    ;2放入ax寄存器
3 |     add ax,ax    ;2 + 2
4 |     add ax,ax    ;4 + 2
5 | abc ends        ;段的开始和结束
6 | end

```

4. 段和段寄存器关联

(1) 我们用到 `abc` 段，和代码段关联起来

```

1 | assume cs:abc
2 | abc segment
3 |     mov ax,2    ;2放入ax寄存器
4 |     add ax,ax    ;2 + 2
5 |     add ax,ax    ;4 + 2
6 | abc ends        ;段的开始和结束
7 | end

```

5. 加上程序返回的代码

```

1 | assume cs:abc
2 | abc segment
3 |     mov ax,2    ;2放入ax寄存器
4 |     add ax,ax    ;2 + 2
5 |     add ax,ax    ;4 + 2
6 |
7 |     mov ax,4c00h
8 |     int 21h      ;这个是一个套路，我们要记住
9 | abc ends        ;段的开始和结束
10 | end

```

我们后期在处理复杂任务的时候，无非就是实现处理任务中加入或者是处理多个复杂段。

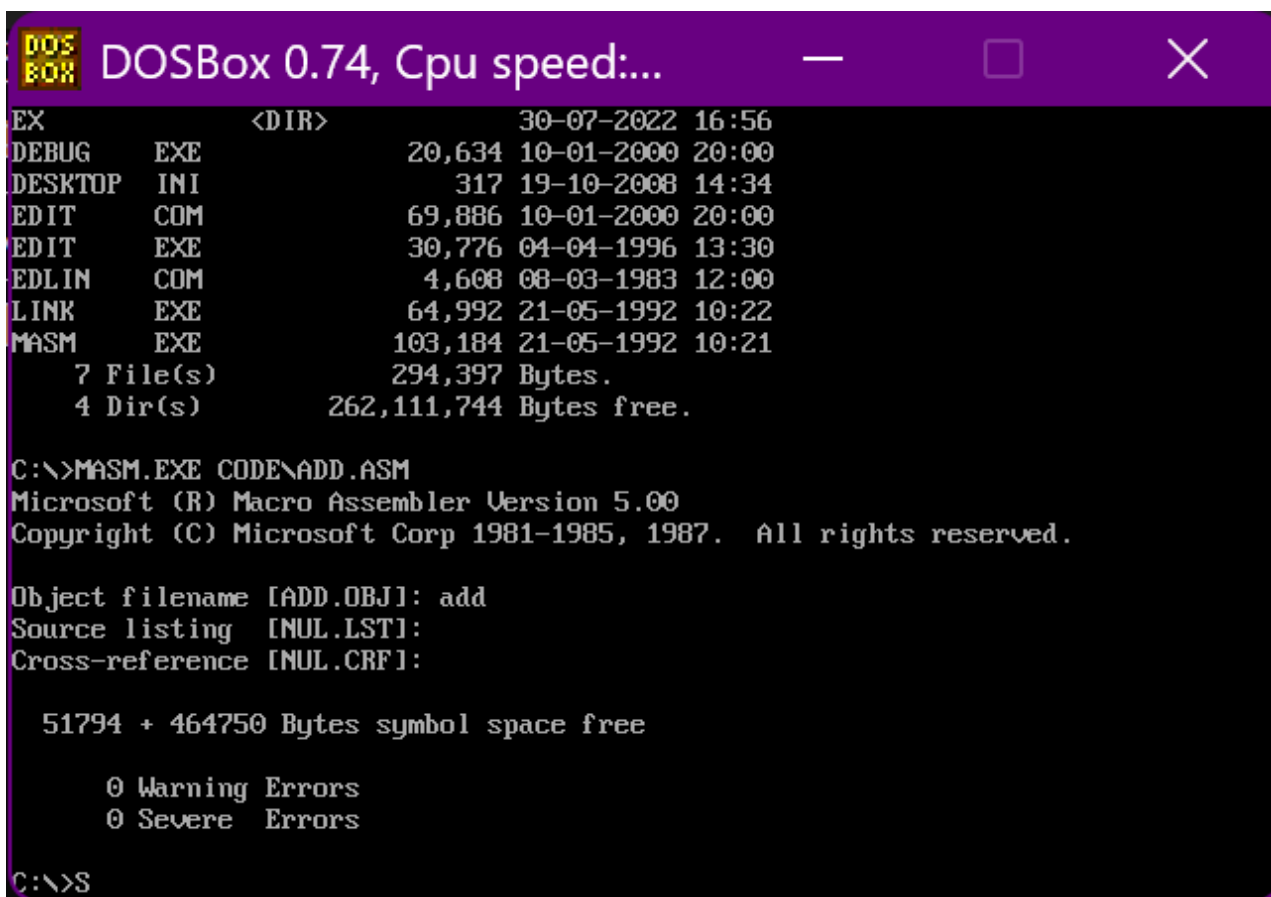
4 程序中可能的错误

我们要避免语法错误，更要注意逻辑错误

- 语法错误：编译中会指明
- 逻辑错误：程序在编译时不表现出来，但是运行时候出现错误

```
1  assume cs:abc
2  abc segment
3      mov ax,2
4      add ax,ax
5      add ax,bx ;不会报错
6
7      mov ax,4c10h ;写错了不会报错但是后面有问题
8      int 21h ;这个是一个套路，我们要记住
9  abc ends ;段的开始和结束
10 end
```

5 从源程序到程序运行



```
DOS
BOX DOSBox 0.74, Cpu speed:...
EX      <DIR>          30-07-2022 16:56
DEBUG   EXE           20,634 10-01-2000 20:00
DESKTOP INI            317 19-10-2008 14:34
EDIT    COM           69,886 10-01-2000 20:00
EDIT    EXE           30,776 04-04-1996 13:30
EDLIN   COM            4,608 08-03-1983 12:00
LINK    EXE           64,992 21-05-1992 10:22
MASM    EXE          103,184 21-05-1992 10:21
  7 File(s)          294,397 Bytes.
  4 Dir(s)           262,111,744 Bytes free.

C:\>MASM.EXE CODE\ADD.ASM
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987. All rights reserved.

Object filename [ADD.OBJ]: add
Source listing [INUL.LST]:
Cross-reference [INUL.CRF]:

51794 + 464750 Bytes symbol space free

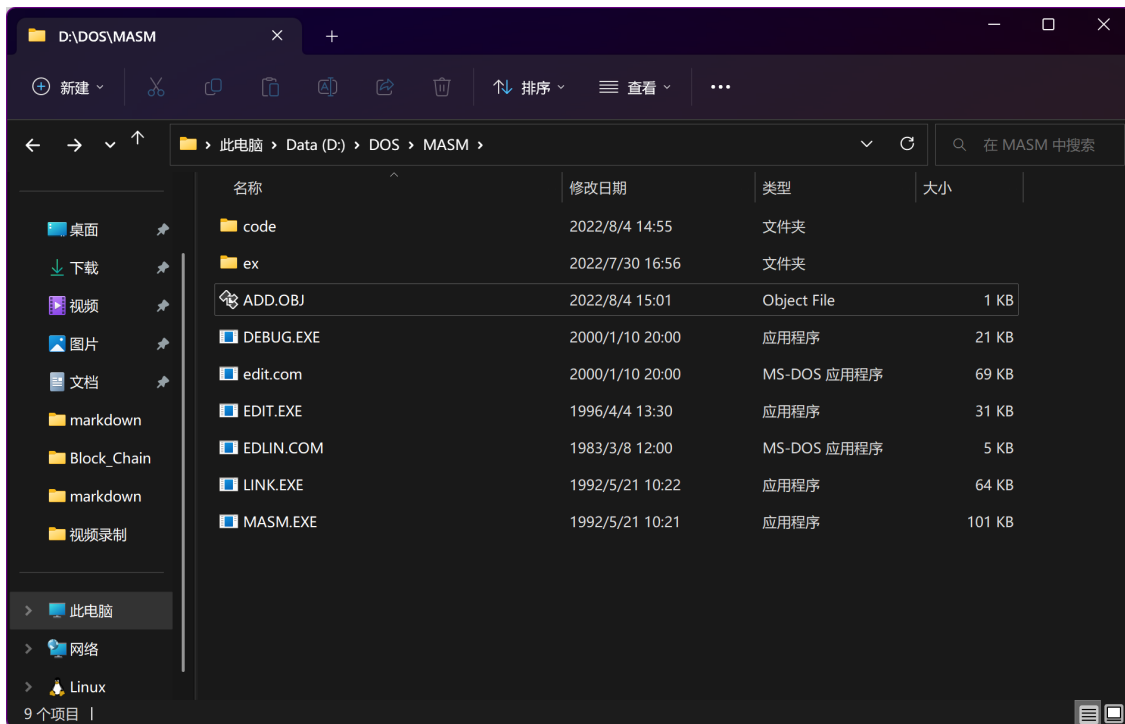
0 Warning Errors
0 Severe Errors

C:\>S
```

我们在MASM文件对应目录下创建 `code` 文件夹，新建文件 `add.asm`，写入上面的程序，编译

1 | `masm.exe code\add.asm`

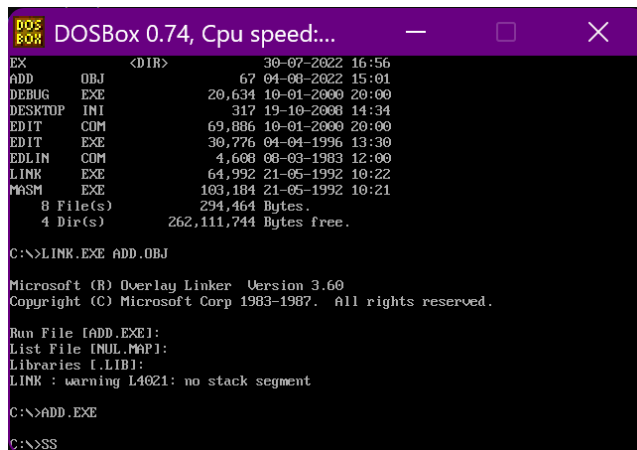
- 目标文件 `*.OBJ` 是我们对一个源程序进行编译要得到的最终结果
- 列表文件 `*.LST` 是编译器将源文件白你为目标文件过程中产生的中间结果
- 最后两行告诉我们这个程序没有警告错误和必须要改正的错误



6 验证程序结果

我们只是编译了程序，但是并没有显示程序的执行结果，这个时候我们可以在 `debug` 中调试。

1. 使用 `masm` 编译，产生目标文件
2. 使用 `link` 命令，把目标文件转为成可执行文件
3. 执行可执行文件



7 debug转载程序

1. 转载

```
1 debug add.exe
```

2. 程序被转在哪里 `-r`

(1) DS=? 075A

(2) CS=? 076A

3. 查看代码段 `-u`

8 分析

程序加载中，DS放程序所在区的段地址，偏移地址为 `0`，则内存区地址为： `DS:0`

这个内存区的前 `256` 个字节存放 `PSP`，DOS用来和程序进行通信

从 `256` 字节后空间存放的是程序， `CS` 的值为 `DS+10H`

程序加载后，CX存放的是代码的长度

`-t` 单步执行

9 [...]和(...)的约定

- [...] – (汇编语言语法要求)表示一个内存地址
- (...) – (为学习方便做出的约定)表示一个内存单元或者一个寄存器中的内容

指令	段地址	偏移地址	操作单位
mov ax, [0]	在DS中	在[0]中	字
mov al, [0]	在DS中	在[0]中	字节
mov ax, [bx]	在DS中	在[bx]中	字
mov al, [bx]	在DS中	在[bx]中	字节

9.1 再约定 `idata` 表示常量

10 END 链接

- [返回目录](#)
- [上一节](#)
- [下一节](#)

-
- [参与贡献](#) ❤️ ❤️ ❤️