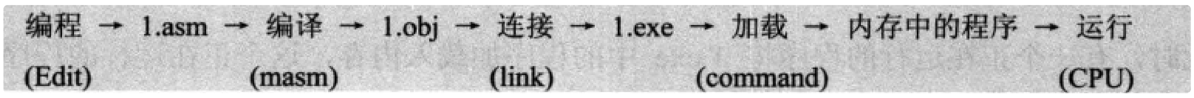
王爽汇编第四章，第一个程序

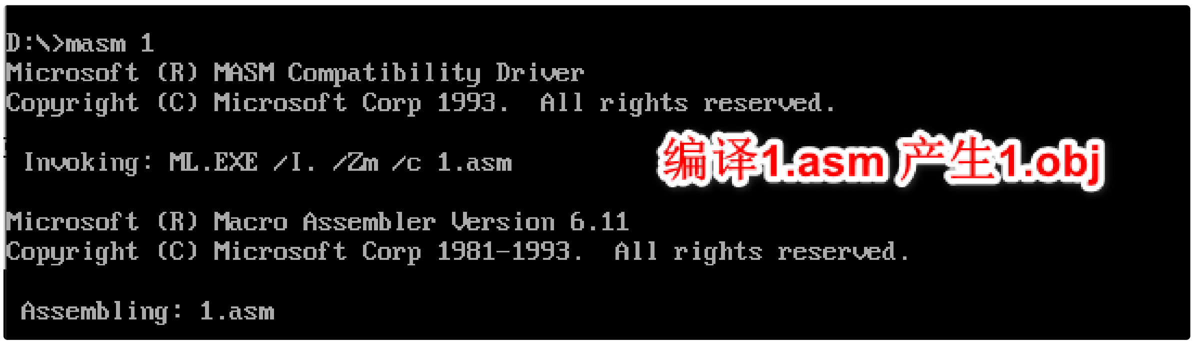
1. 汇编程序从写出到执行的过程



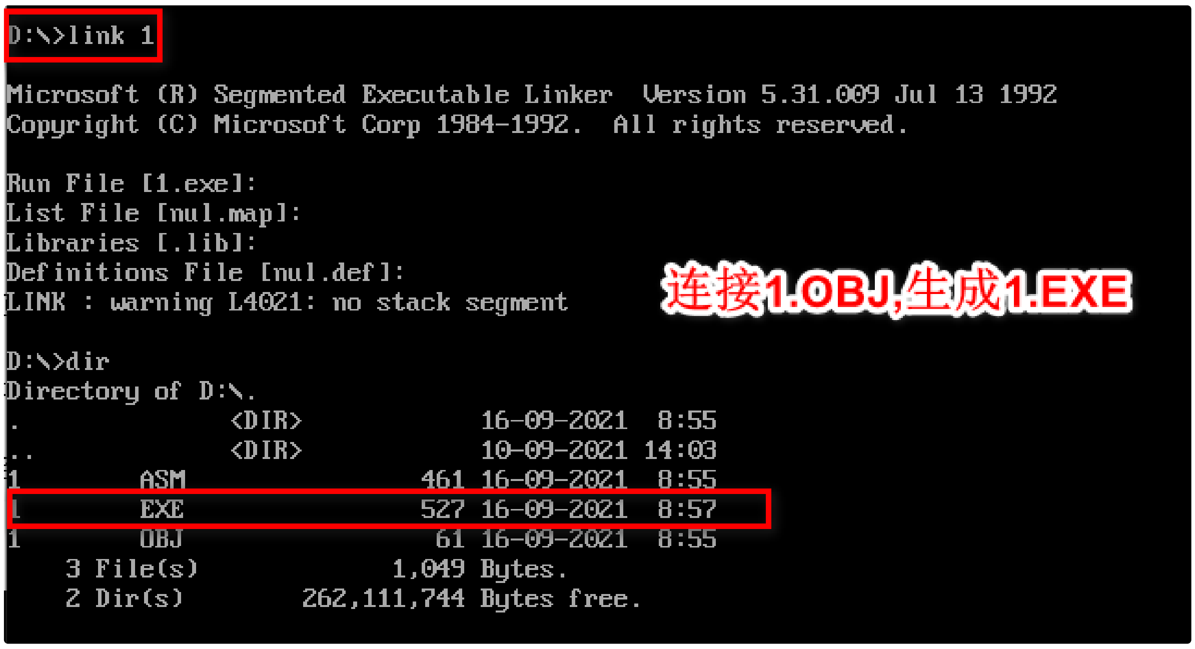
编程 → 1.asm → 编译 → 1.obj → 连接 → 1.exe → 加载 → 内存中的程序 → 运行
(Edit)　(masm)　(link)　(command)　(CPU)

```
;1.asm
assume cs:code ;代码段code和cs关联

;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;代码段
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
code segment ;code是标号,最终会被编译成段地址
;程序入口
start:
        mov ax,0123h
        mov bx,0456h
        add ax,bx
        add ax,ax

        ;返回到DOS
        mov ax,4c00h
        int 21h
code ends
;程序结束
end start
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

编译过程：编译->连接->生成1.exe



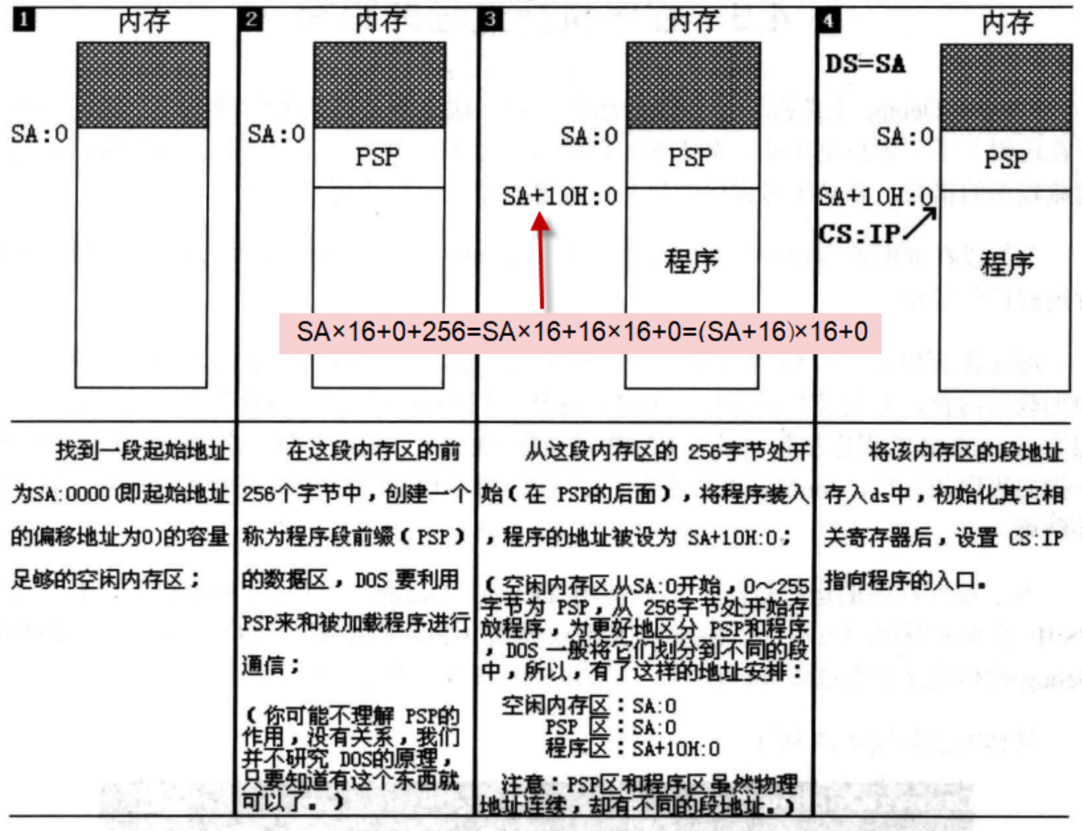
```
D:\>masm 1
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c 1.asm

Microsoft (R) Macro Assembler Version 6.11
Copyright (C) Microsoft Corp 1981-1993.  All rights reserved.

 Assembling: 1.asm
```



```
D:\>link 1

Microsoft (R) Segmented Executable Linker  Version 5.31.009 Jul 13 1992
Copyright (C) Microsoft Corp 1984-1992.  All rights reserved.

Run File [1.exe]:
List File [nul.map]:
Libraries [.lib]:
Definitions File [nul.def]:
LINK : warning L4021: no stack segment

D:\>dir
Directory of D:\.
.              <DIR>            16-09-2021  8:55
..             <DIR>            10-09-2021 14:03
1        ASM               461 16-09-2021  8:55
1        EXE               527 16-09-2021  8:57
1        OBJ                61 16-09-2021  8:55
    3 File(s)            1,049 Bytes.
    2 Dir(s)       262,111,744 Bytes free.
```

2. 程序执行过程的跟踪分析

首先看下图，DOS系统中.EXE文件加载如下，注意和Windows的.EXE有区别。



找到一段起始地址为SA:0000 (即起始地址的偏移地址为0)的容量足够的空闲内存区；

在这段内存区的前256个字节中，创建一个称为程序段前缀(PSP)的数据区，DOS 要利用PSP来和被加载程序进行通信；

(你可能不理解 PSP的作用，没有关系，我们并不研究 DOS的原理，只要知道有这个东西就可以了。)

从这段内存区的 256字节处开始(在 PSP的后面)，将程序装入，程序的地址被设为 SA+10H:0；

(空闲内存区从SA:0开始，0~255字节为 PSP，从 256字节处开始存放程序，为更好地区分 PSP和程序，DOS 一般将它们划分到不同的段中，所以，有了这样的地址安排：

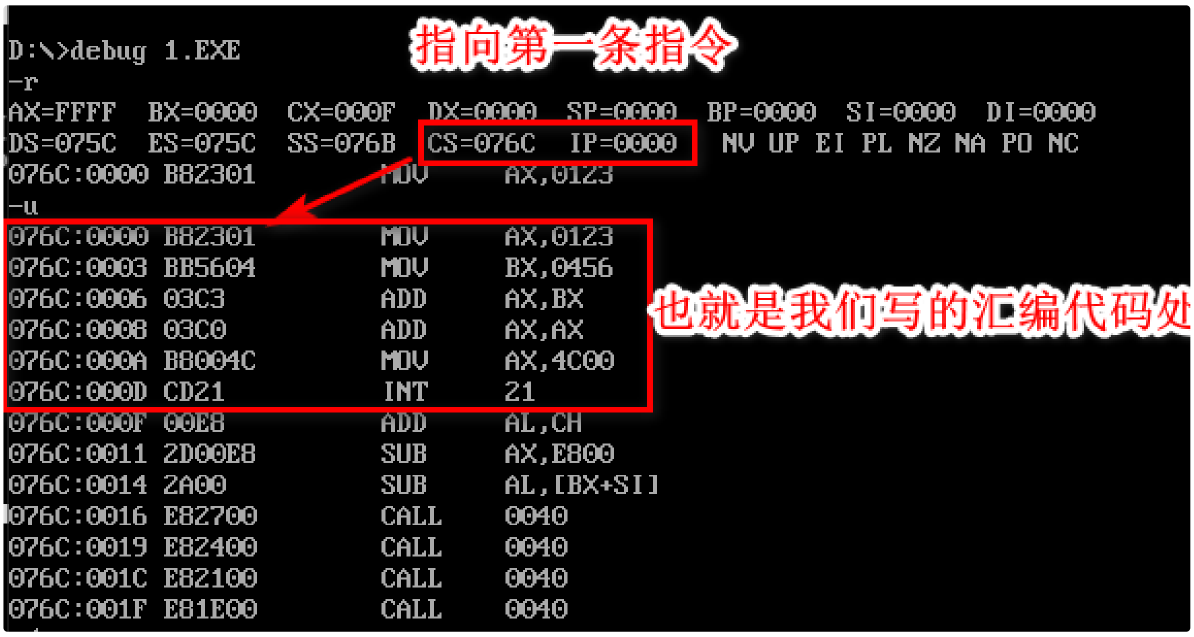
空闲内存区：SA:0
PSP 区：SA:0
程序区：SA+10H:0

注意：PSP区和程序区虽然物理地址连续，却有不同的段地址。)

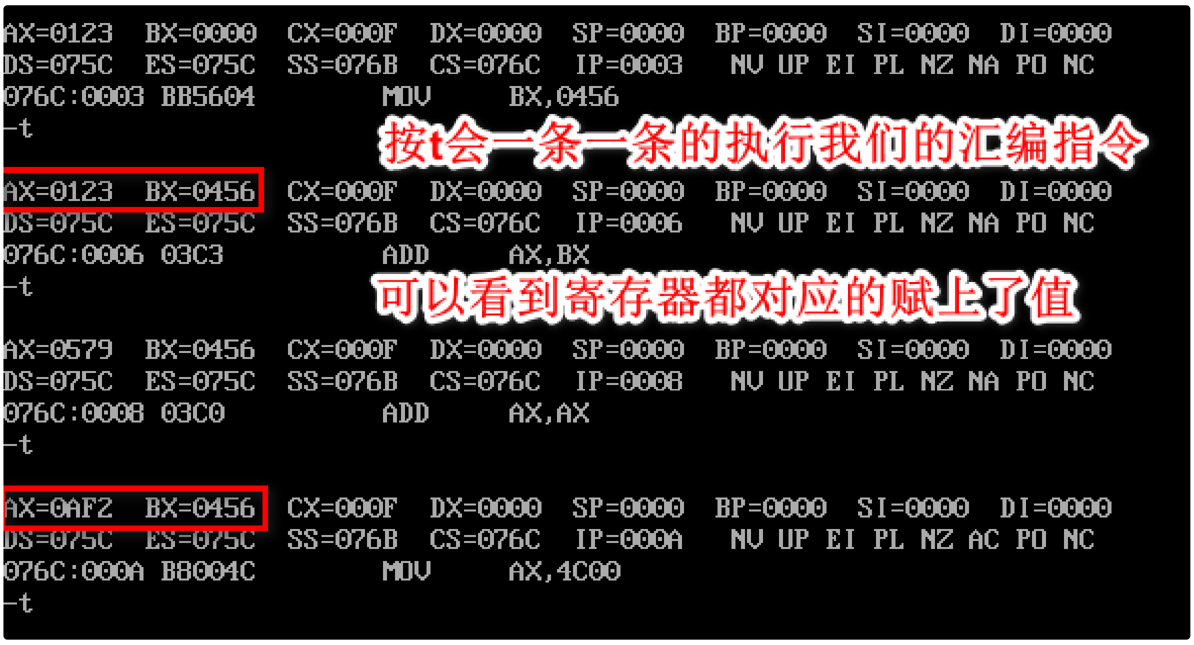
将该内存区的段地址存入ds中，初始化其它相关寄存器后，设置 CS:IP 指向程序的入口。

图 4.20 EXE 文件中程序的加载过程

debug进行跟踪分析

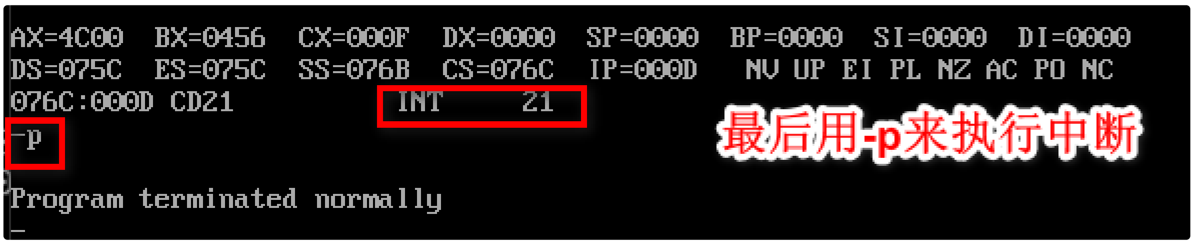


```
D:\>debug 1.EXE
-r
AX=FFFF  BX=0000  CX=000F  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075C  ES=075C  SS=076B  CS=076C  IP=0000   NV UP EI PL NZ NA PO NC
076C:0000 B82301        MOV     AX,0123
-u
076C:0000 B82301        MOV     AX,0123
076C:0003 BB5604        MOV     BX,0456
076C:0006 03C3          ADD     AX,BX
076C:0008 03C0          ADD     AX,AX
076C:000A B8004C        MOV     AX,4C00
076C:000D CD21          INT     21
076C:000F 00E8          ADD     AL,CH
076C:0011 2D00E8        SUB     AX,E800
076C:0014 2A00          SUB     AL,[BX+SI]
076C:0016 E82700        CALL    0040
076C:0019 E82400        CALL    0040
076C:001C E82100        CALL    0040
076C:001F E81E00        CALL    0040
```



```
AX=0123  BX=0000  CX=000F  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075C  ES=075C  SS=076B  CS=076C  IP=0003   NV UP EI PL NZ NA PO NC
076C:0003 BB5604        MOV     BX,0456
-t

AX=0123  BX=0456  CX=000F  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075C  ES=075C  SS=076B  CS=076C  IP=0006   NV UP EI PL NZ NA PO NC
076C:0006 03C3          ADD     AX,BX
-t

AX=0579  BX=0456  CX=000F  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075C  ES=075C  SS=076B  CS=076C  IP=0008   NV UP EI PL NZ NA PO NC
076C:0008 03C0          ADD     AX,AX
-t

AX=0AF2  BX=0456  CX=000F  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075C  ES=075C  SS=076B  CS=076C  IP=000A   NV UP EI PL NZ AC PO NC
076C:000A B8004C        MOV     AX,4C00
-t
```

按t会一条一条的执行我们的汇编指令
可以看到寄存器都对应的赋上了值



```
AX=4C00  BX=0456  CX=000F  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075C  ES=075C  SS=076B  CS=076C  IP=000D   NV UP EI PL NZ AC PO NC
076C:000D CD21          INT     21
-p

Program terminated normally
-
```

文献参考：
https://blog.csdn.net/qq_39654127/article/details/88698911