汇编实验五 编写，调试具有多个段的程序

原创

Skyennnnn 于 2018-11-24 01:52:07 发布 2574 收藏 9

分类专栏： 汇编语言 文章标签： 汇编实验

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/weixin_43491622/article/details/84405509

版权

汇编语言 专栏收录该内容

10 篇文章 1 订阅

订阅专栏

本章的主要内容主要是在讲代码段，数据段，栈段的使用。

实验1

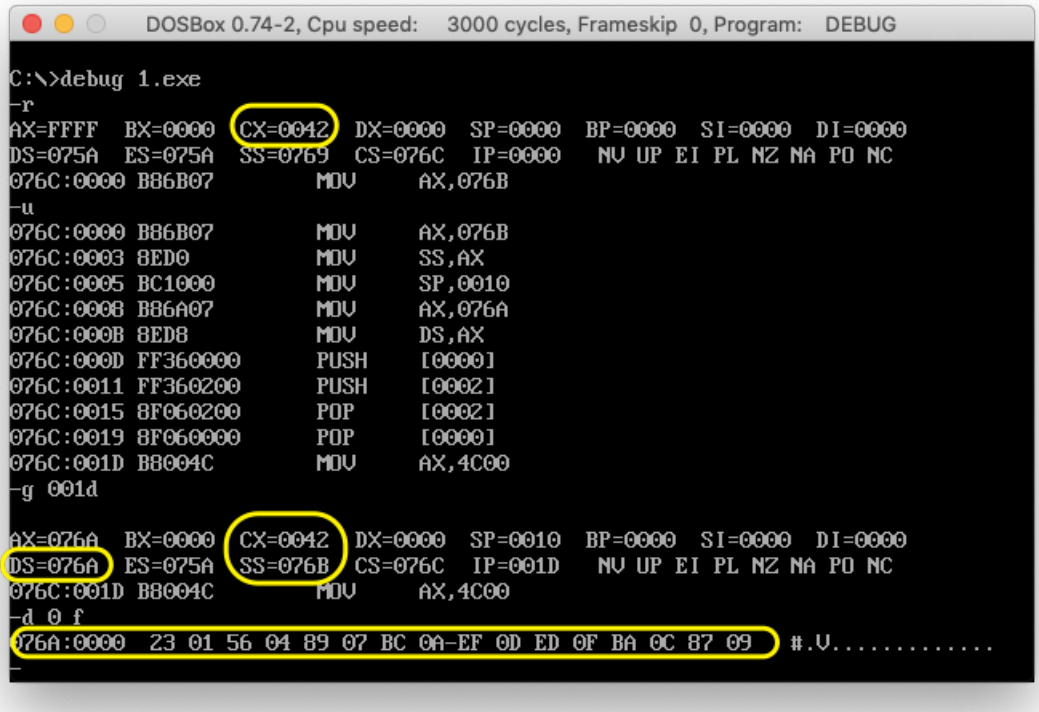
将下面的程序编译，链接，用debug加载，跟踪，然后回答问题。

```
assume cs:code, ds:data, ss:stack

data segment
 dw 0123h, 0456h, 0789h, 0abch, 0defh, 0fedh, 0cbah, 0987h
data ends

stack segment
 dw 0, 0, 0, 0, 0, 0, 0, 0
stack ends

code segment
start: mov ax, stack
 mov ss, ax
 mov sp, 16 ;ss:sp stack

 mov ax, data
 mov ds, ax ;ds data

 push ds:[0]
 push ds:[2]
 pop ds:[2]
 pop ds:[0]

 mov ax, 4c00h
 int 21h

code ends
end start
```

- cpu执行程序，程序返回前，data段中的数据是多少？
- cpu执行程序，程序返回前，cs=**0042h**, ss=**076bh**, ds=**076ah**
- 设程序加载后，code段的地址为X，则data段的地址 **X-2**，stack段为**X-1**。

分析：

先用r命令查看，可以看到cx寄存器中的值为42，然后用u命令进行反汇编，因为，代码段已经用start标注，所以可以直接使用u命令，当然，也可以算出代码段的长度之后再反汇编。（代码段长度为42-20=22h）
然后g命令，g 001D执行mov ax 4c00h之前的内容，最后用d命令查看内存中的内容，如图：



实验2

将下面的程序编译，链接，用debug加载，跟踪，然后回答问题。

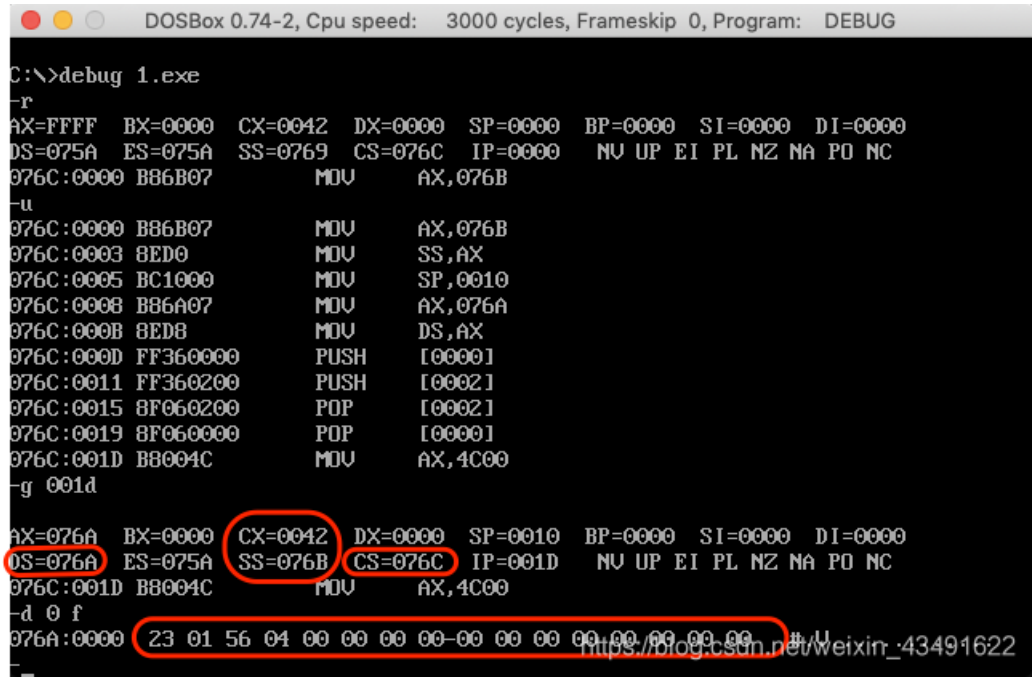
```
assume cs:code, ds:data, ss:stack

data segment
 dw 0123h, 0456h
data ends

stack segment
 dw 0, 0
stack ends

code segment
start:
 mov ax, stack
 mov ss, ax
 mov sp, 16 ;ss:sp stack

 mov ax, data
 mov ds, ax

 push ds:[0]
 push ds:[2]
 pop ds:[2]
 pop ds:[0]

 mov ax, 4c00h
 int 21h
code ends
end start
```

- cpu执行程序，程序返回前，data段中的数据为多少？
- cpu执行程序，程序返回前，cs=**076ch**,ss=**076bh**,ds=**076ah**.
- 设程序加载后，code段的短地址为x，则data段的短地址为**x-2**，stack段的短地址为**x-1**。
- 对于如下定义的段：
  name segment
  …
  name ends
  如果段中的数据占N个字节，则程序加载后该段实际占有的空间为__（[n/16]+1)*16__

分析：

有了实验1的基础，实验2的操作基本和实验1相同，相同的内容就不赘述了。最主要的不同应该就是定义的数据多少，可能就是要探索关于空间的使用问题。
不同的第四个问题，是的，我也不怎么看太明白，看着答案我先简单的推测一下，因为一个段的大小至少为16字节，最大为64k字节，计算地址时有段地址*16 的步骤，那么反过来，若段的数据有n字节，则程序加载后，该段实际占有的空间为（n/16+1）*16（n/16取整数部分）。
还是靠谱的百度一下：
详解：

以下的内容引用于 friendbkf的博客。
对于如下定义的段:
name segment
…
name ends
如果段中的数据占N个字节，则程序加载后，该段实际占有的空间为_____.
答案：
(N/16+1)*16 [说明：N/16只取整数部分] 或 （N＋15）/ 16 ，对16取整
在8086CPU架构上，段是以paragraph(16-byte)对齐的。程序默认以16字节为边界对齐，所以不足16字节的部分数据也要填够16字节。“对齐”是alignment，这种填充叫做padding。16字节成一小段，称为节
一、这首先要从8086处理器寻址原理说起。
8086这种处理器有二十根地址线（20个用于寻址的管脚），可以使用的外部存储器空间可达1MB（00000H～FFFFFH）。 但是，8086内部的寄存器都是16位的，用任何一个寄存器（比如BX或SI），都无法直接寻址8086所支持的1M地址空间，因为16位寄存器只能表示640KB的空间范围（0000～FFFFH）。
所以，Intel想了一个方法，设计出了CS/DS/ES/SS这几个段地址寄存器，用段地址寄存器与普通寄存器组合，来寻址1MB的地址范围，即：对于CPU取指来说，用CS:IP组合来寻址下一个要执行的指令（也在存储器中）；对于堆栈操作PUSH/POP来说，用SS:SP组合来表示当前栈指针（栈也在存储器中）；对于数据操作指令来说，用默认的DS/ES或指定的段地址（段前缀指令）与偏移量寄存器组合寻址。
组合后的实际地址＝段寄存器内容×16＋偏移量寄存器内容
从这个公式可以看到，每一个段的地址都对齐在16的倍数上。比如DS=1234H，则这个段就从 1234H×16＋0000H＝12340H开始，最大到1234H×16＋0FFFFH＝2233FH为止。
二、对同一个内存地址，有不同的段:偏移量组合方法，比如2233FH这个地址，既可以表示为1234H:0FFFFH（在1234H段中），也可以表示为2233H:0000FH（在2233H段中）。
那么，如果汇编程序中有下面两个连续的段定义，汇编编译程序会怎么做呢？
name1 segment
d1 db 0
name1 ends
name2 segment
d2 db 0
name2 ends
编译程序可以将name1和name2编译成一个段，d1和d2在内存中连续存放，这样可以节省内存空间。比如编译程序以name1为基准，将name1作为一个段的起始，程序加载后会被放在xxxx0H的地方，那么d1就放在该段偏移地址为0字节的位置，d2就放在该段偏移为1字节的位置。
这样的处理方式虽然可能会节省一点内存空间，但是对于编译器的智能化要求太高了，它必须将源程序中所有引用到name2和d2的地方，全部调整为以name1段为基准，这实在是太难了，而且也节省不了几个字节的空间，编译器是不会干这种吃力不讨好的事的。
编译器实际的处理方式是将name1中的所有内容放在一个段的起始地址处，name2里的所有内容放在后续一个段的起始地址处（这也是汇编指令segment的本义：将不同数据分段）。这样，即使name1中只包含一个字节，也要占一个段（16个字节），所以，一个段实际占用的空间＝（段中字节数＋15）/ 16。
所以，8086处理器的内部寻址原理和汇编程序编译器共同决定了segment定义的段必须放在按16的倍数对准的段地址边界上，占用的空间也是16的倍数。



实验3

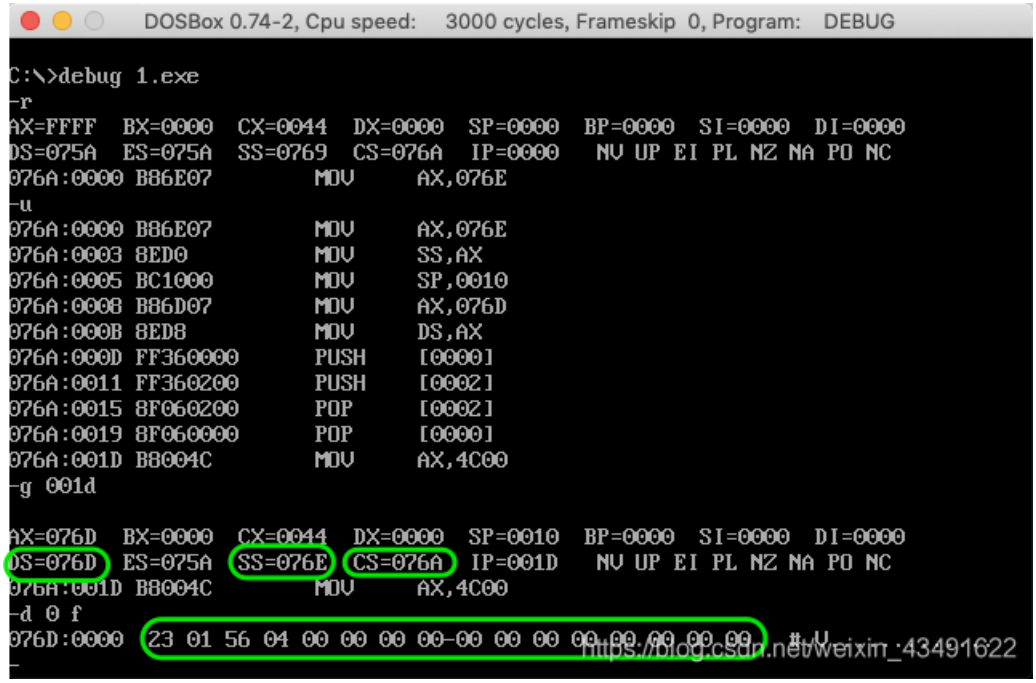
将下面的程序编译，链接，用debug加载，跟踪，然后回答问题。

```
assume cs:code, ds:data, ss:stack

code segment
start:
 mov ax, stack
 mov ss, ax
 mov sp, 16 ;ss:sp stack

 mov ax, data
 mov ds, ax

 push ds:[0]
 push ds:[2]
 pop ds:[2]
 pop ds:[0]

 mov ax, 4c00h
 int 21h

 data segment
 dw 0123h, 0456h
 data ends

 stack segment
  dw 0, 0
 stack ends

code ends
end start
```

- CPU执行程序，程序返回前，data段中的数据为多少？
- CPU执行程序，程序返回前，**cs=076ch**,ss=**076eh**，ds=**076dh**.
- 设程序加载后，code段的短地址为x，则data段的短地址为__x+3__，stack段的短地址为__x+4__。

分析：

操作几乎都和实验1，2一样，不赘述了。



实验4

如果将（1），（2），（3）题中的最后一条伪指令end start改为end（也就是说，不指明程序的入口个），则哪个程序仍然可以正确执行？请说明原因。

第三个程序仍然可以执行，因为不指明程序入口时，cs:code segment默认ip为0，第三个程序正好是程序开始的地方，前两个ip=0开始的地方存的是数据，解析为汇编指令是错误的。也就明白什么时候用end，什么时候用end start。

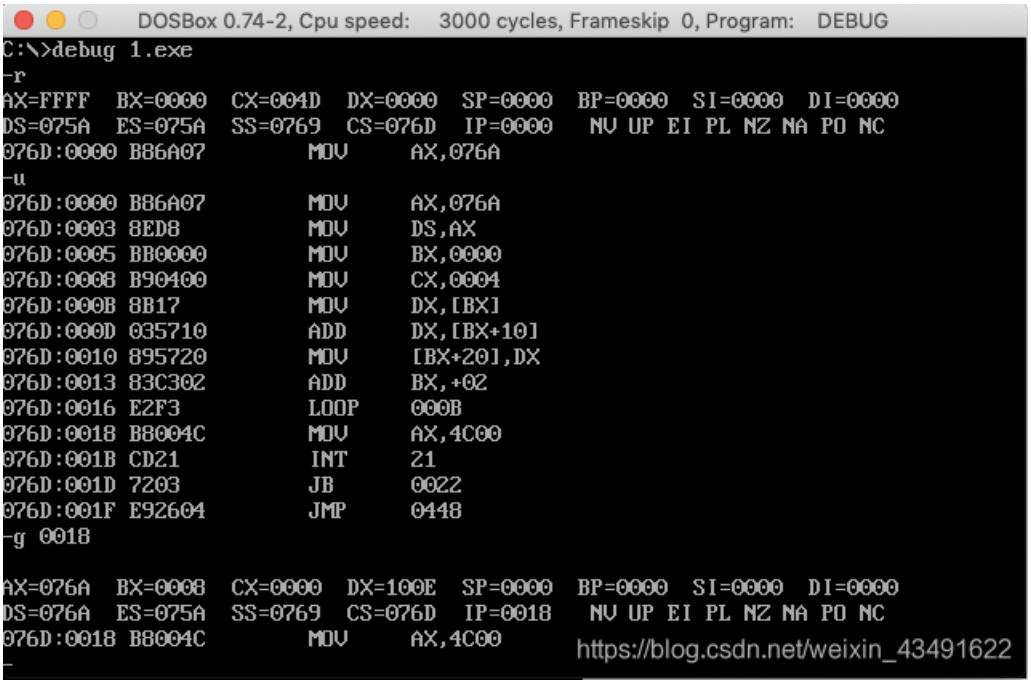
实验5

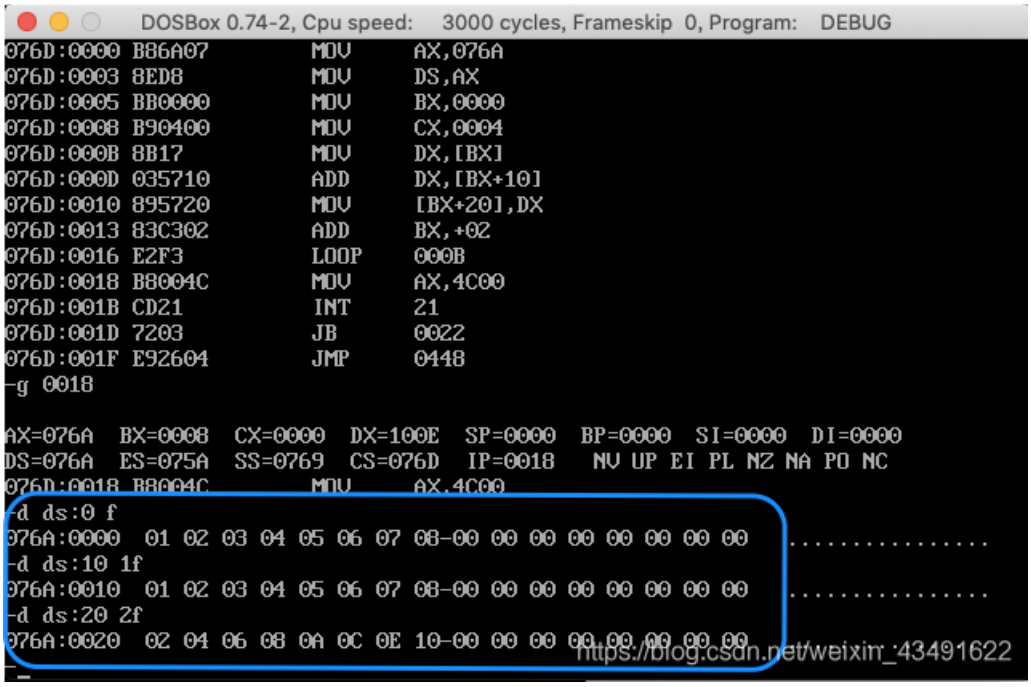
程序如下，编写code段中的代码，用push指令将a段和b段中的数据依次相加，将结果存到c段中。

```
assume cs:code
a segment
    db 1,2,3,4,5,6,7,8
a ends

b segment
    db 1,2,3,4,5,6,7,8
b ends

c segment
    db 0,0,0,0,0,0,0,0
c ends

code segment
start:
     mov ax,a
     mov ds,ax

     mov bx,0
     mov cx,4

s:   mov dx,ds:[bx]
     add dx,ds:[bx+16]
     mov ds:[bx+32],dx
     add bx,2
     loop s

     mov ax,4c00h
     int 21h

code ends
end start
```

分析：

思路：
将a段的数据先存在dx中，然后直接用b段数据相加，然后再将相加后的结果给c。
若a的段地址为x，则b的段地址为x+1，c的段地址为x+2。





很明显，相加成功了。
这里有一个问题，a b段的数据不是很大，相加之后还是小于255，所以不会溢出。如果溢出应该怎么解决呢？

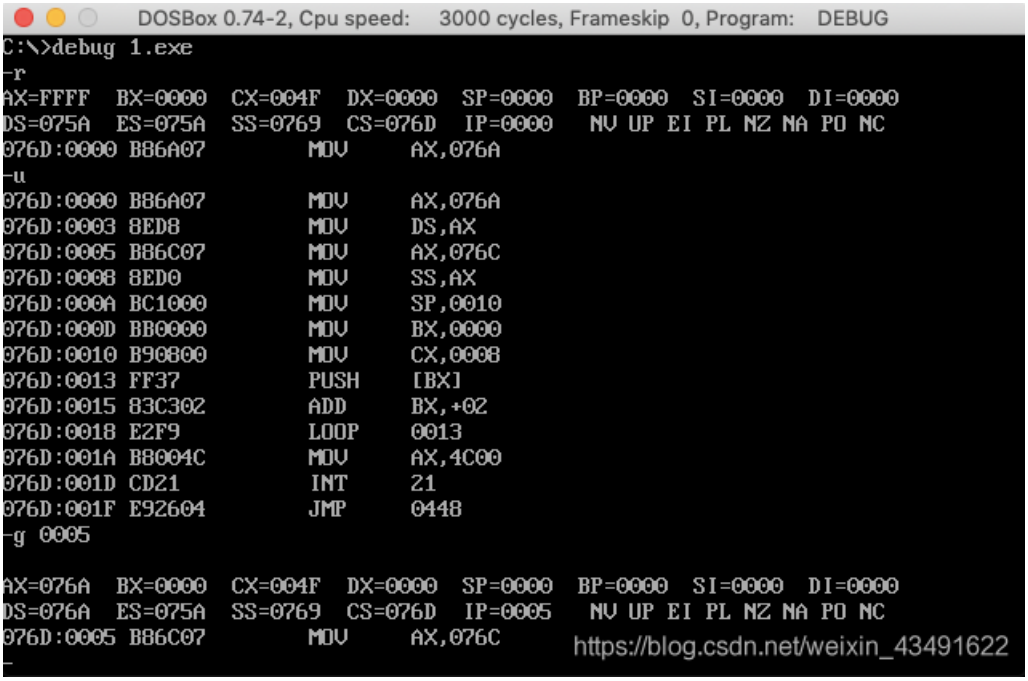
实验6

程序如下，编写code段中的代码，用push指令将a段中的前8个字型数据，逆序存储到b段中。

```
assume cs:code
a segment
    dw 1,2,3,4,5,6,7,8,9,0ah,0bh,0ch,0dh,0eh,0fh,0ffh
a ends

b segment
    dw 0,0,0,0,0,0,0,0
b ends

code segment
    start:
          mov ax,a
          mov ds,ax
          mov ax,b
          mov ss,ax
          mov sp,10h
          mov bx,0
          mov cx,8

s:        push ds:[bx]
          add bx,2
          loop s

          mov ax,4c00h
          int 21h

code ends
end start
```

分析：

记得课上好像讲过类似的。



很明显，b的8个字单元为a段前8个字节的逆序。

注意：

dw 是定义的字型数据，每个数据16个位，两个字节。
db是定义的字节型数据，每个数据8位，一个字节。
段寄存器之间不能直接转移数据，需要通过al寄存器来中转。

不得不说，本次耗费的时间比想像中要长很多。
希望下次不会这么慢了，还是多多练习吧。