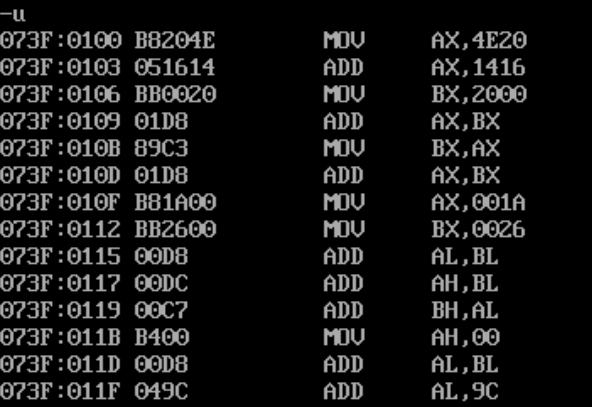
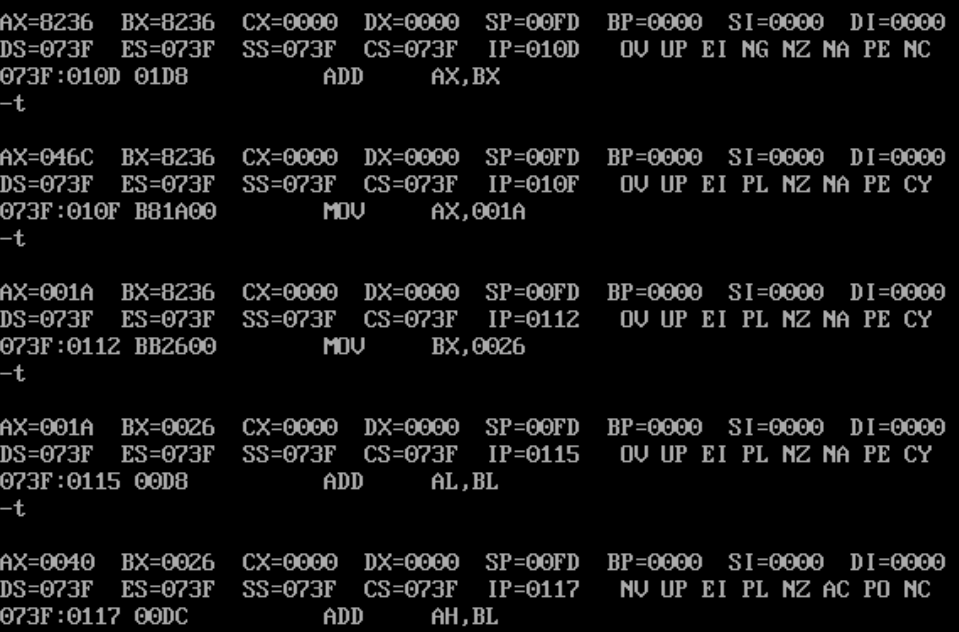
实验1

1、使用debug，将下面程序写入内存，逐条执行并观察每条指令执行后CPU中相关寄存器的变化





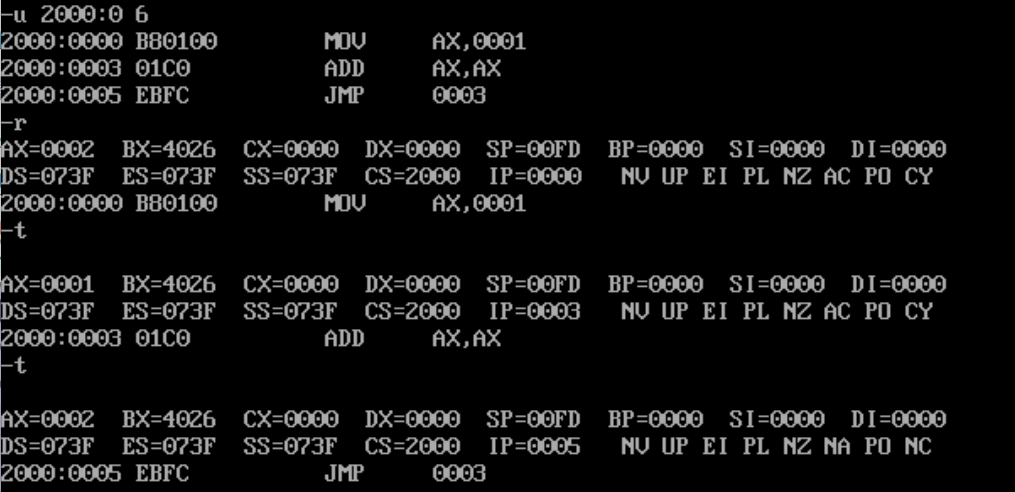


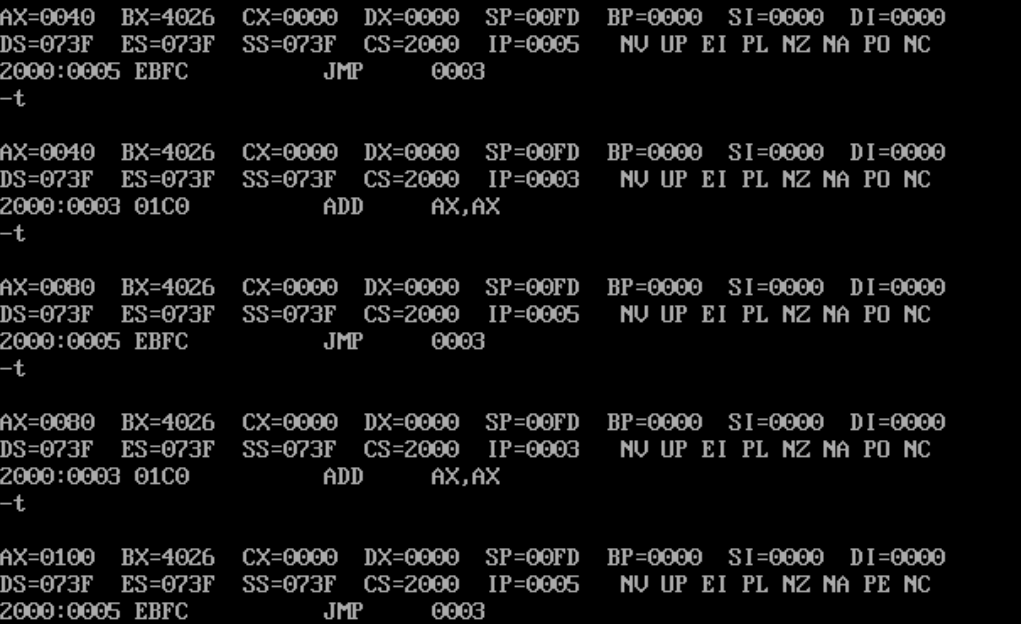
课本P18-19：

使用AH和AL时它们作为独立的寄存器，AL发生溢出时进位不会加给AH

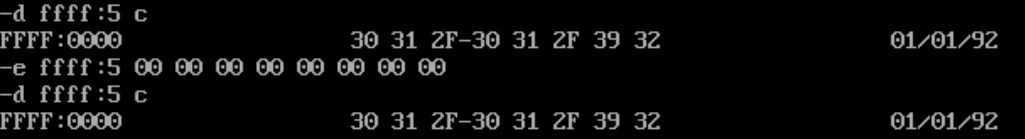
进行数据传送和运算时，指令的两个操作对象的位数必须相同

2、将下面3条指令写入从2000:0开始的内存单元中，计算2的8次方





最终结果为0100H

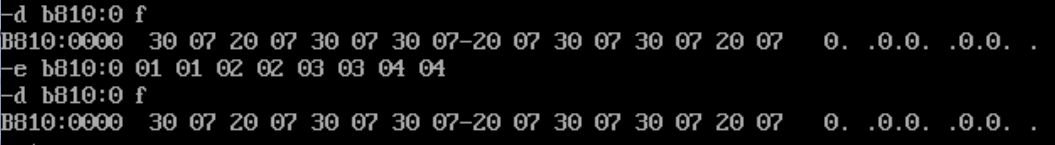
3、PC机主板上的ROM中有一个生产日期，在内存FFF00H-FFFFFH的某几个单元中，请找到这个生产日期并试图改变它

生产日期存储在FFFF5H-FFFFCH这8个单元中

使用e指令尝试将这8个单元全部修改为0后

再次查看，发现没有修改成功

4、尝试向B8100H开始的单元中填写数据

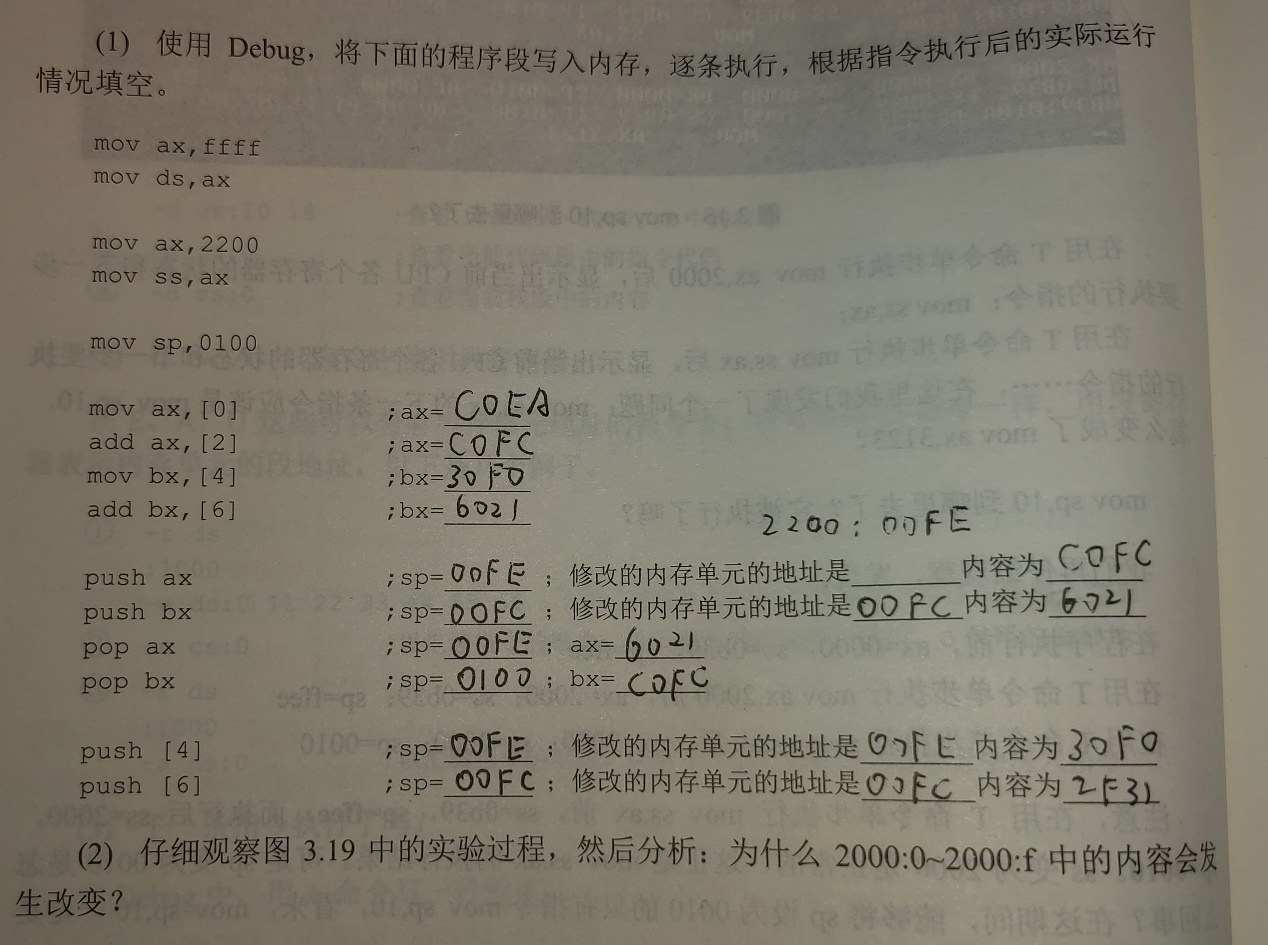


跟第3题的结果相似，没有修改成功

我们向这些内存单元写入数据的操作是无效的

因为这等于改写只读存储器中的内容

实验2



2

给SS和SP赋值后

栈顶的前几个元素会被赋值为CS和IP的值

固定未知含义的值：栈顶前1-2个内存单元

CS：栈顶前3-4个内存单元

IP：栈顶前5-6个内存单元

以上三个值的位置随栈顶的变化而变化（push和pop操作）

存疑：

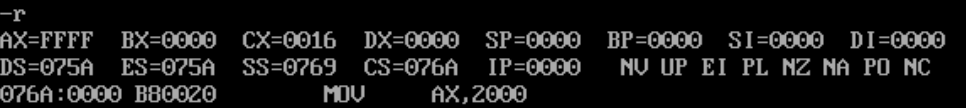
栈顶在初始化时（设置完ss的值后），sp前第8-9个内存单元会被设置为SS

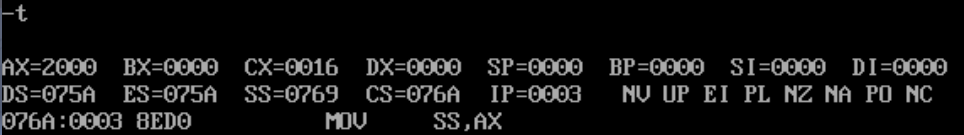
随后此位置设置为SS-1

实验3

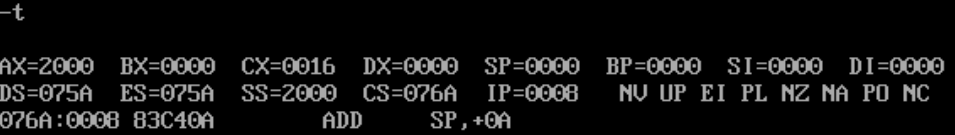
2、跟踪3.exe，写出每一步执行后相关寄存器中的内容

源代码见：3.asm



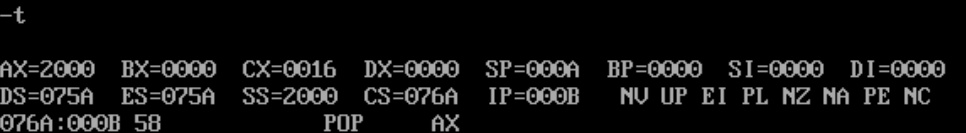


AX=2000H



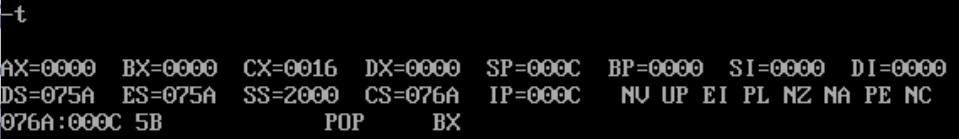
SS=2000H

SP=0



SP=10

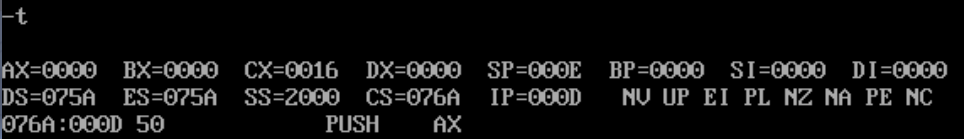
栈顶为0



AX出栈

AX=0

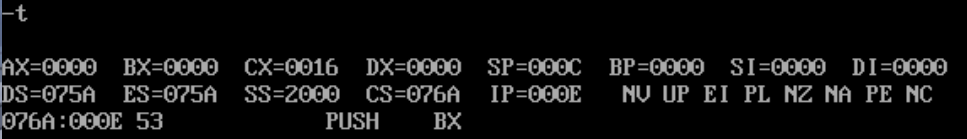
栈顶为0



BX出栈

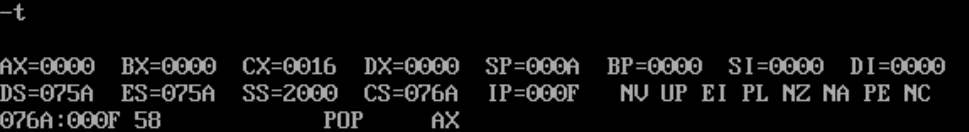
BX=0

栈顶为0



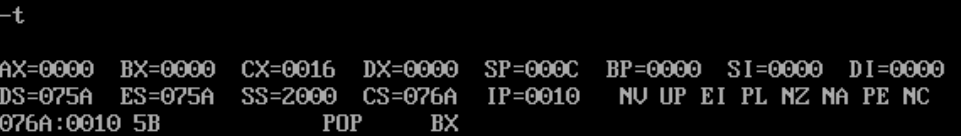
AX入栈

栈顶为0



BX入栈

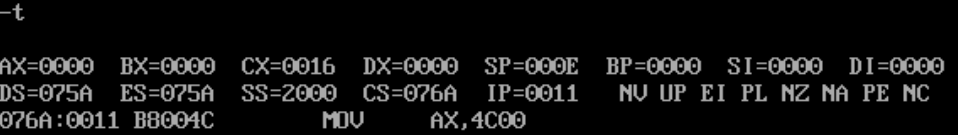
栈顶为0



AX出栈

AX=0

栈顶为0

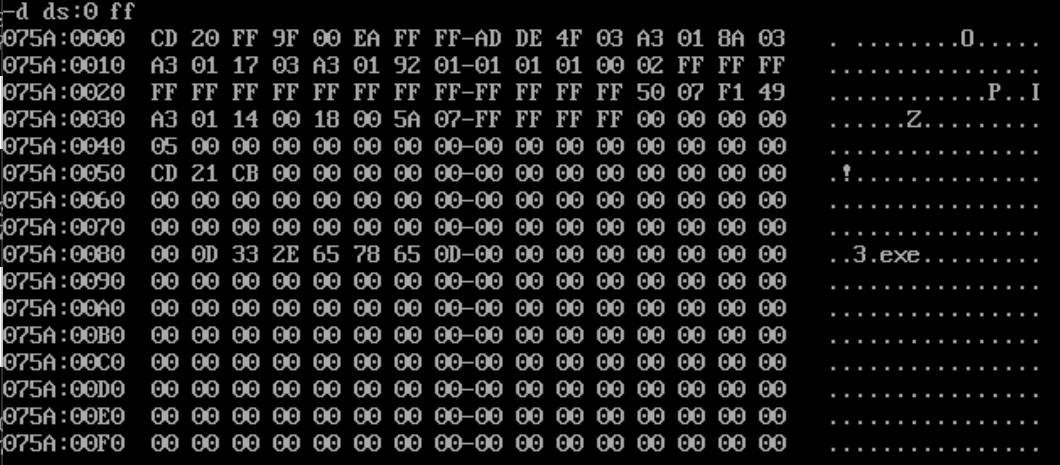


BX出栈

BX=0

栈顶为0

3、PSP头两个字节是CD 20，PSP大小为256个字节，查看PSP中的内容

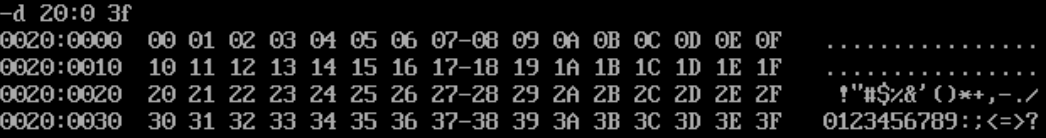


PSP中出现了3.exe的文件名

实验4

2、编程，向内存 0:200-0:23F 依次传输数据 0-63(3FH)，程序中只能使用 9 条指令。

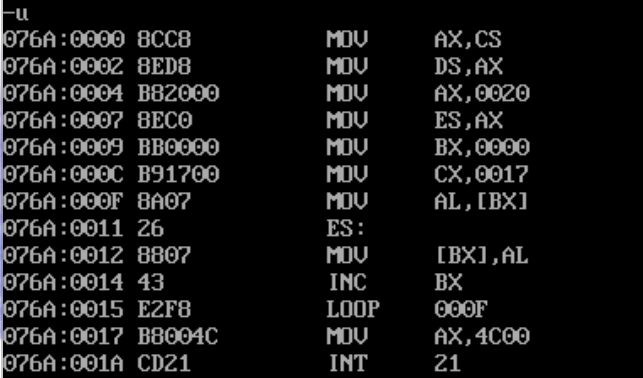
源代码见：4-2.asm

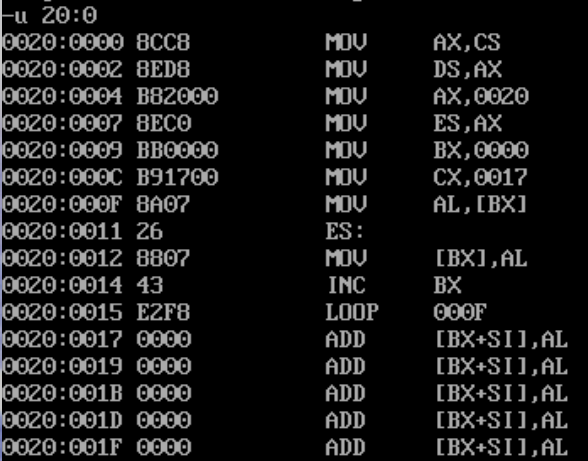


3、下面的程序的功能是将 “mov ax,4c00h” 之前的指令复制到内存 0:200 处，补全程序。上机调试，跟踪运行结果。

源代码见：4-3.asm

复制的是从076A:0000到076A:0016的数据。共复制了23（16h）个字节，通过查看代码在内存中所占的字节数知道需要复制的字节数目





实验5

1、源代码见：5-1.asm

程序返回前，data段保持初始定义时的数据

程序返回前，CS=076C,SS=076B,DS=076A

段地址：若code=X,则data=X-2，stack=X-1

2、源代码见：5-2.asm

程序返回前，data段前四字节不变，后12字节为0

程序返回前，CS=076C,SS=076B,DS=076A

段地址：若code=X,则data=X-2，stack=X-1

数据段中虽然只定义了四个字节的数据，但是数据段占有的空间是 16 字节

栈段没有紧随数据段其后，因为段地址每加1，物理地址增加 16 字节

因此为了将不同的段区分，某个段占用空间不足 16 的倍数字节时会自动补齐

可见段的大小的基本单位是 16 字节

对于占据N个字节数据的段，程序加载后，该段实际占有的空间为字节

3、源代码见：5-3.asm

程序返回前，data段前四字节不变，后12字节为0

程序返回前，CS=076A,SS=076E,DS=076D

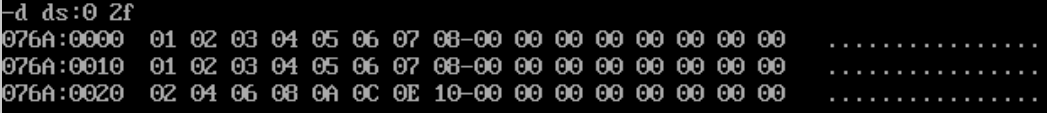
段地址：若code=X,则data=X+3，stack=X+4

4、如果将123题中最后一条伪指令“end start”改为“end”，哪个程序仍然可以正确运行？

程序3，因为程序3的代码段就在程序的入口处

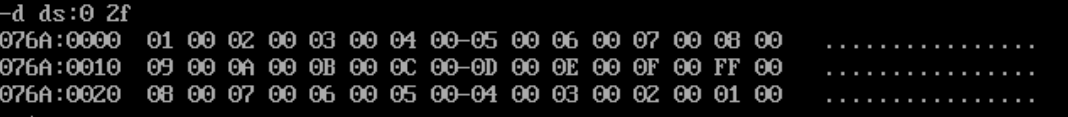
5、编写code段中的代码，将a段和b段中的数据依次相加，将结果存到c段中。

源代码见：5-5.asm



6、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。

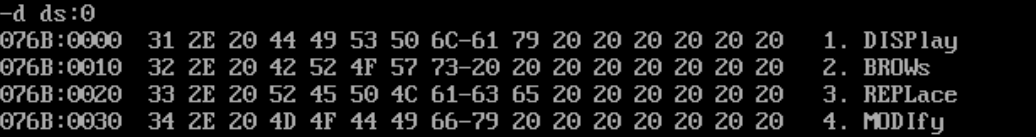
源代码见：5-6.asm



实验6

编程，将 data 段中每个单词的前4个字母改为大写字母

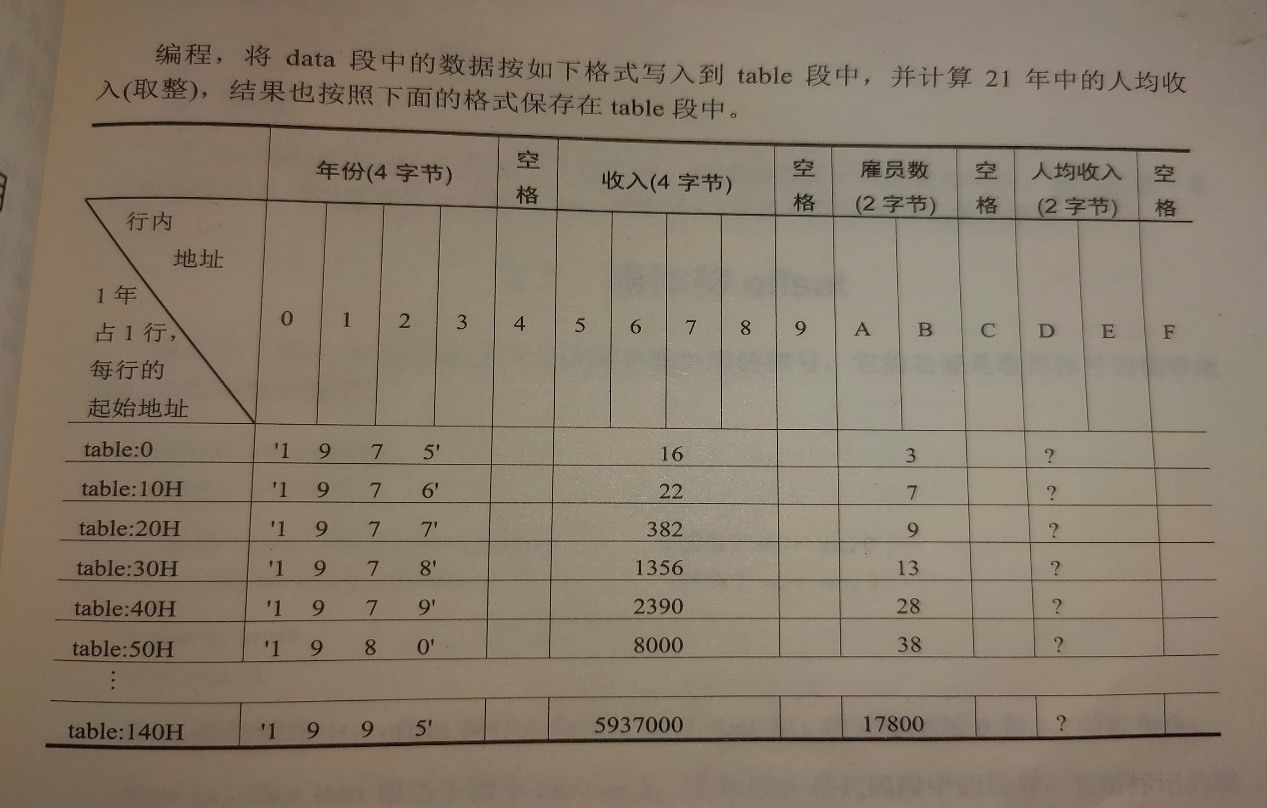
源代码见：6.asm

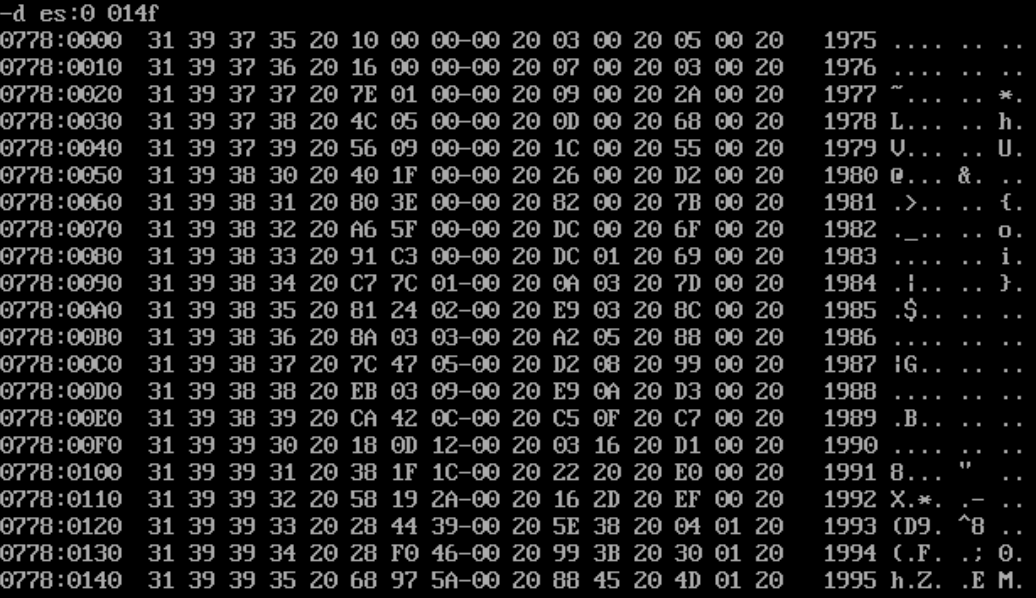


实验7

编程，将data段中的数据按如下格式写入到table段中，并计算21年中的人均收入（取整），结果也按下面格式保存在table段中

源代码见：7.asm





实验8

分析下面程序

运行前思考：可以正确返回吗？

运行后思考：为什么是这种结果？

附件：8.asm

运行前思考：

mov ax,0

mov di,offset s

mov si,offset s2

mov ax,cs:[si]

mov cs:[di],ax ;s处两条nop指令被修改为jmp 0000h

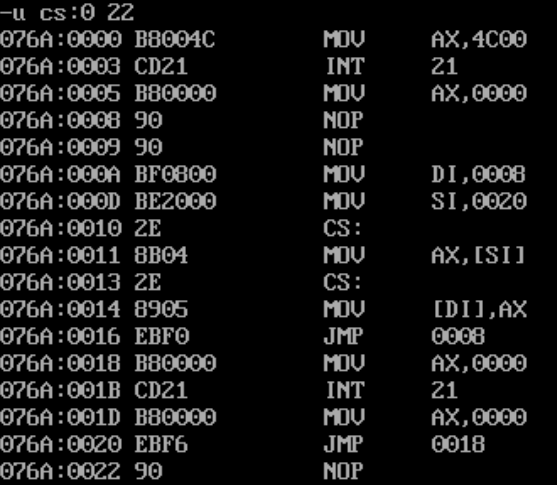
jmp short s ;跳转至s处

jmp short s1 ;跳转至cs:0000h处

mov ax,4c00h

int 21h ;正常返回

运行后发现确实如此



jmp short s1是段内短转移，机器码为EB F6，其中：

EB表示段内短转移指令

F6是8位位移的补码形式，表示执行(ip)=(ip)-10

如果将EB F6复制到cs:0008h处

首先指令进入指令缓冲器，ip=ip+2=0ah

执行EB F6，ip=ip-10=0

此时跳转至cs:0处执行指令mov ax，4c00h和int 21h

程序正常返回

参考课本P177-180

实验9

在屏幕中间分别显示绿色、绿底红色、白底蓝色字符串‘welcome to masm！’

并实现清屏功能

附件：9.asm、9.exe

黑底绿色：0000 0010 = 02h

绿底红色：0010 0100 = 24h

白底蓝色：0111 0001 = 71h

屏幕中间的位置为：12-14行，33-48列

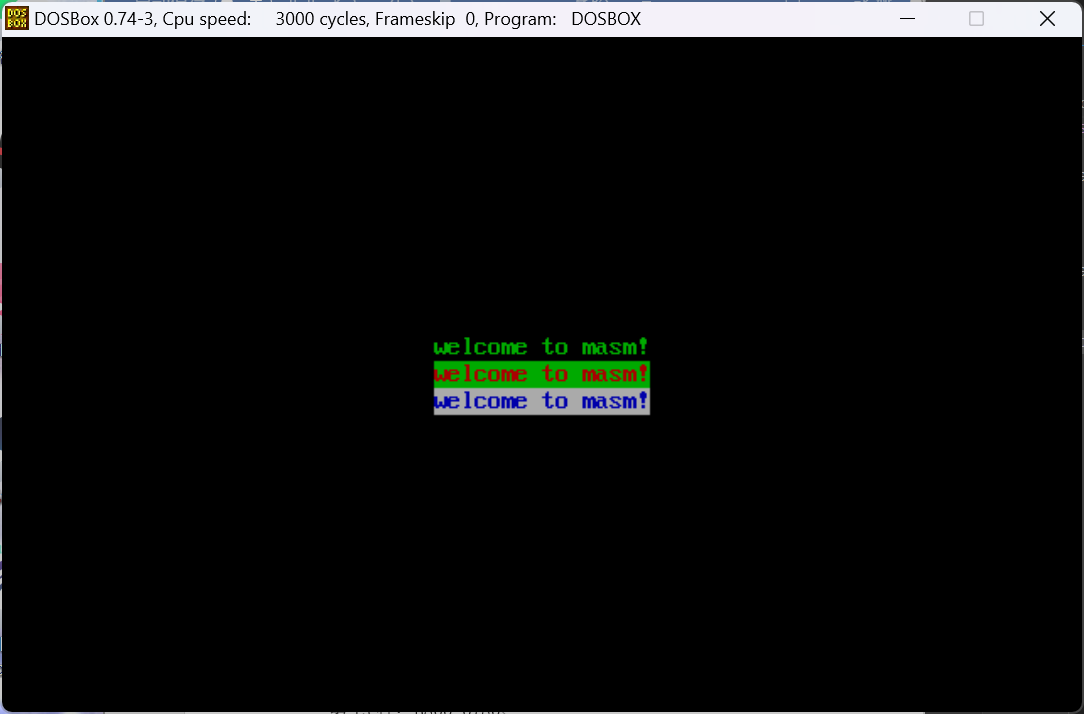
12行：11×00A0H=06E0H

33列：32×2=64

第12行：B800:06E0 +64 ... +94

第13行：B800:0780

第14行：B800:0820



实验10

1、

课程设计1

