山东大学

信息科学与工程学院

2020－2021 学年第二学期

实 验 报 告

课程名称： 微处理器原理与应用

实验名称： 实验 2.2 条件和循环

专 业 班 级 2019 级崇新学堂

学 生 学 号 201900121023

学 生 姓 名 李禹申

实 验 时 间 2021 年 3 月 22 日

实验报告

【实验目的】

1. 掌握分支程序
2. 掌握循环程序

【实验要求】

1. 绘制流程图
2. 对汇编程序进行注释并加以修改观察结果

【实验具体内容】

【第一个实验】分支程序实验：

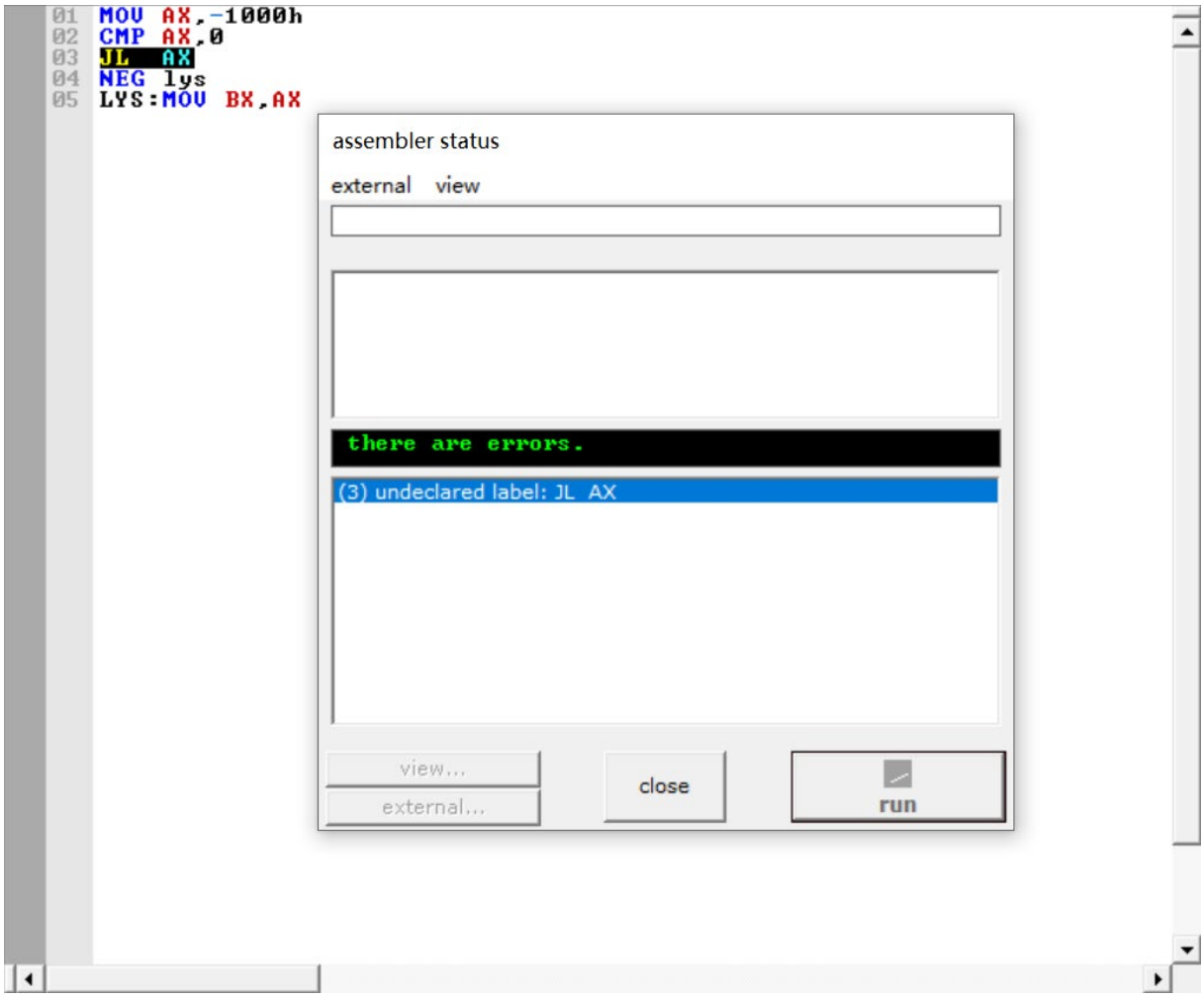
（0）相关知识点：

1. 单分支结构：

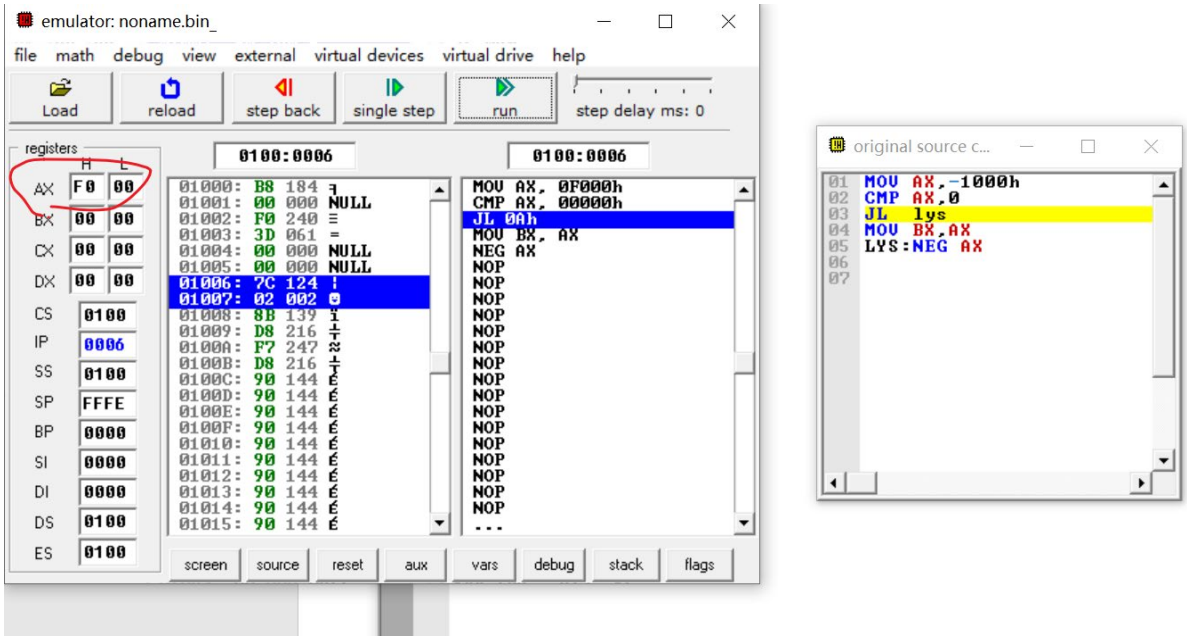
```
1.单分支结构

//比较AX和0
CMP AX,0
//JGE:判断条件:if AX greater than and equal 0,那么执行NONEG中的内容
JGE NONEG
//不满足的条件下就对AX取补,实际上取补码就相当于取的绝对值,老鸟啊!
NEG AX
//条件成立时的执行语句,但是不太理解为什么执行语句可以通过NONEG进行传递
NONEG:MOV RESULT,AX
```

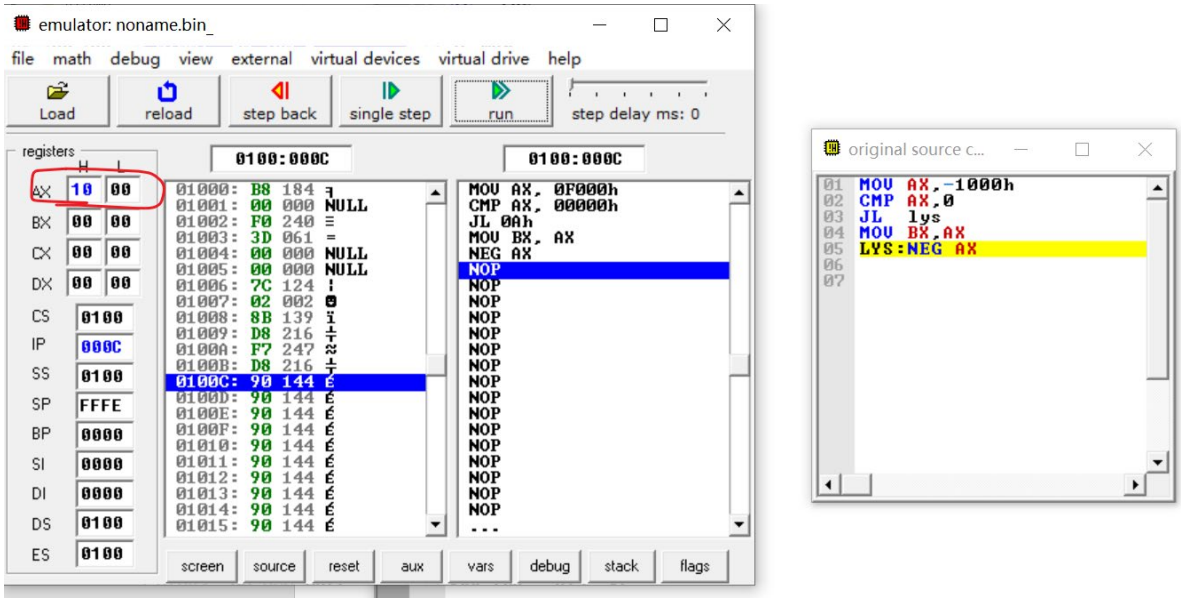
思考：如果条件改成 JL

第一次改了程序：想着既然是小于，那就直接调换一下就好了，然后发现不可



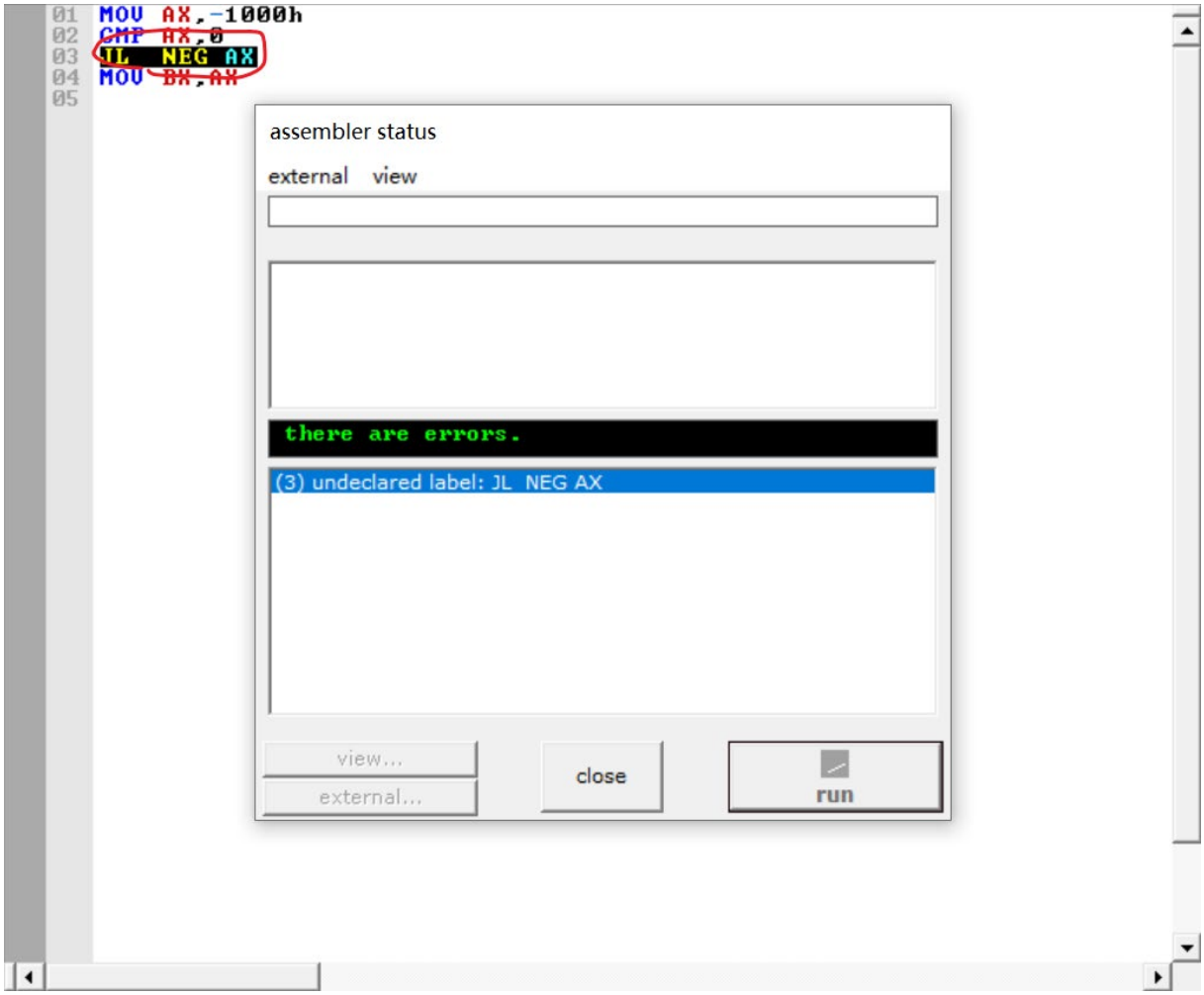
似乎说明 JL，JGE 等条件判断后面就是要通过 CS[IP]来进行转到指令，对应着问题 1，程序更改一下之后变成：





程序运行正确，同时说明问题 1 中的标识符是不区分大小写的

但是此时我发现我之前的判断是错误的，因为我直接在条件后面写了 AX，应该是 NEG AX



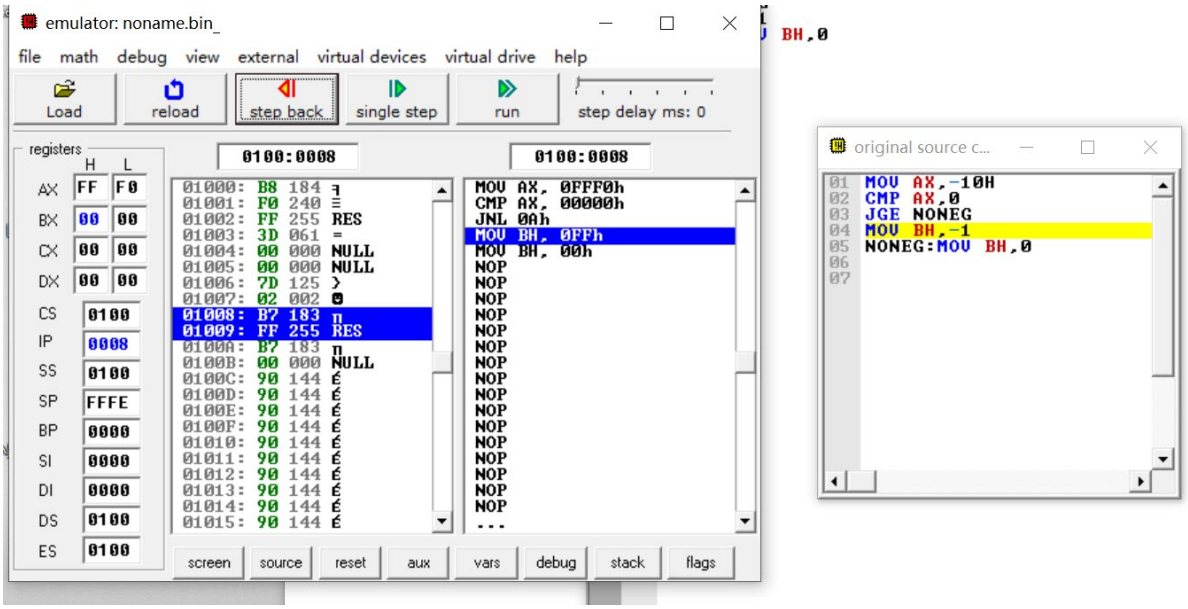
但是更改之后发现仍旧是不对的，说明条件之后就是不能直接加指令，这应该是 Jump

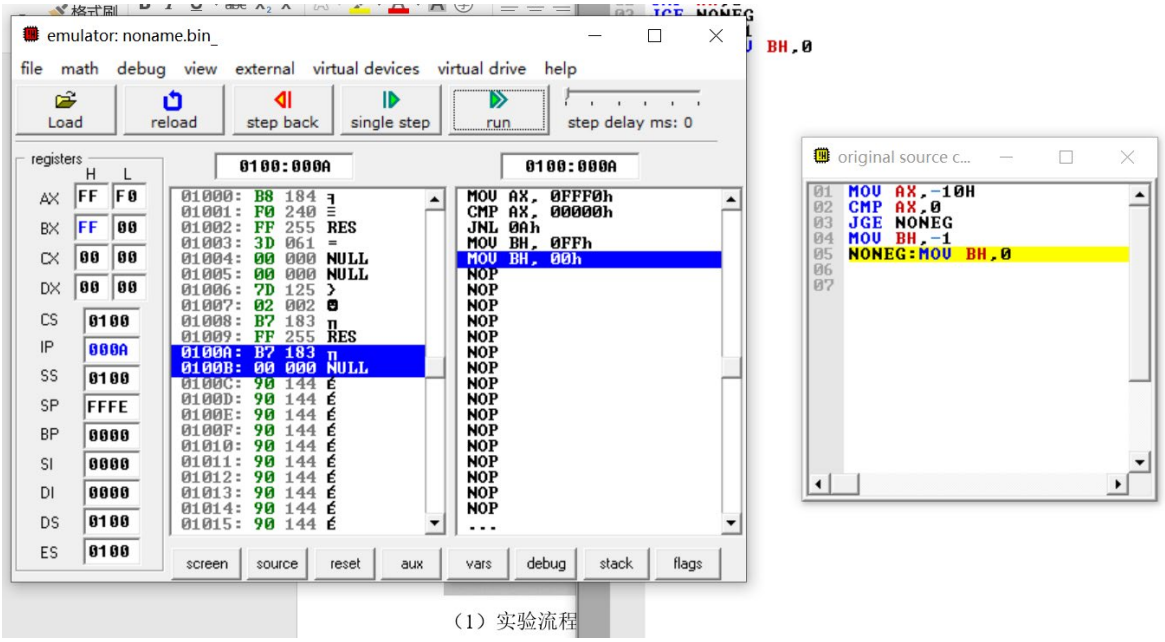
起的作用，结果今天老师上课讲了，原来这个叫子程序哈哈哈！

2.双分支结构

```
2.双分支结构

//比较AX和0
CMP AX,0
//同上
JGE NONEG
//简单的MOV指令
MOV SIGN,-1
//考虑如果没有JMP还是会执行NONEG么
JMP END0
NONEG:MOV SIGN,0
```

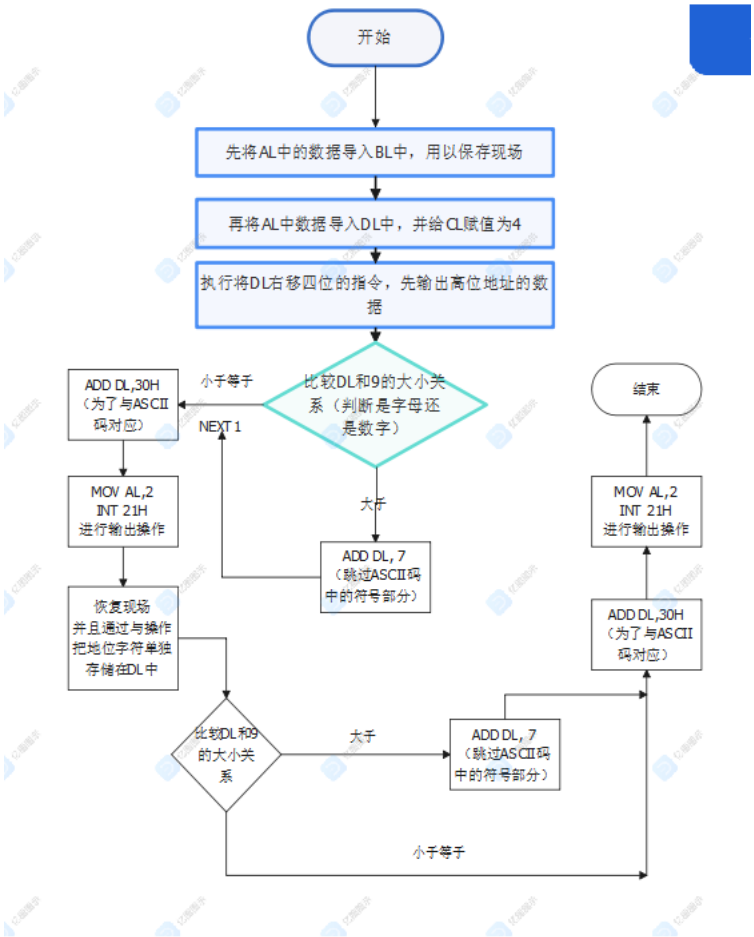
下面对问题进行解答，发现如果没有 JMP END 是会接着执行下面的语句的





（1）实验流程

（1）实验流程图（从实验 2.2 分支程序实验和循环程序实验开始必须画流程图）：



（2）实验源代码（粘贴源代码）：

例如：（第一次实验的源代码）

```
//声明代码段
CODE SEGMENT
//ASSUME是段寄存器关联语句，将CS段寄存器与CODE代码段关联起来，也就相当于CS指向程序罢了
        ASSUME CS:CODE
//入口地址
START:
//一堆MOV都很简单这里就不再写了
        MOV AL, 3EH
//从下面可以看到BL起到保护现场的作用
        MOV BL, AL
        MOV DL, AL
        MOV CL, 4
//SHR是右移指令，由CL中的值可知，移动的位数为4，为的是能够先将高位的输出出来
        SHR DL, CL
//将DL寄存器中的值和9进行比较，是为了判断到底是字母还是数字
        CMP DL, 9
//当DL小于等于9的时候跳转到NEXT1，否则向下执行
        JBE NEXT1
//加7是为了将16进制和ASCII字符关联起来
        ADD DL, 7
//下面对NEXT1子程序进行说明
NEXT1:
        ADD DL, 30H
//下面两行共同作用显示DL中的字符
        MOV AH, 2
        INT 21H

        MOV DL, BL
        AND DL, 0FH
        CMP DL, 9
        JBE NEXT2
        ADD DL, 7
NEXT2:
        ADD DL, 30H
        MOV AH, 2
```

//声明代码段
CODE SEGMENT
//ASSUME 是段寄存器关联语句，将 CS 段寄存器与 CODE 代码段关联起来，也就相当于 CS 指向程序罢了
    ASSUME CS:CODE
//入口地址
START:
//一堆 MOV 都很简单这里就不再写了
    MOV AL, 3EH
//从下面可以看到 BL 起到保护现场的作用

```
    MOV BL, AL
    MOV DL, AL
    MOV CL, 4
//SHR 是右移指令，由 CL 中的值可知，移动的位数为 4，为的是能够先将高位的输出出来
    SHR DL, CL
//将 DL 寄存器中的值和 9 进行比较，是为了判断到底是字母还是数字
    CMP DL, 9
//当 DL 小于等于 9 的时候跳转到 NEXT1，否则向下执行
    JBE NEXT1
//加 7 是为了将 16 进制和 ASCII 字符关联起来
    ADD DL, 7
//下面对 NEXT1 子程序进行说明
NEXT1:
    ADD DL, 30H
//下面两行共同作用显示 DL 中的字符
    MOV AH, 2
    INT 21H

    MOV DL, BL
    AND DL, 0FH
    CMP DL, 9
    JBE NEXT2
    ADD DL, 7
NEXT2:
    ADD DL, 30H
    MOV AH, 2
    INT 21H
CODE ENDS
    ENDS START
```

（3）实验代码、过程、相应结果（截图）并对实验进行说明和分析：

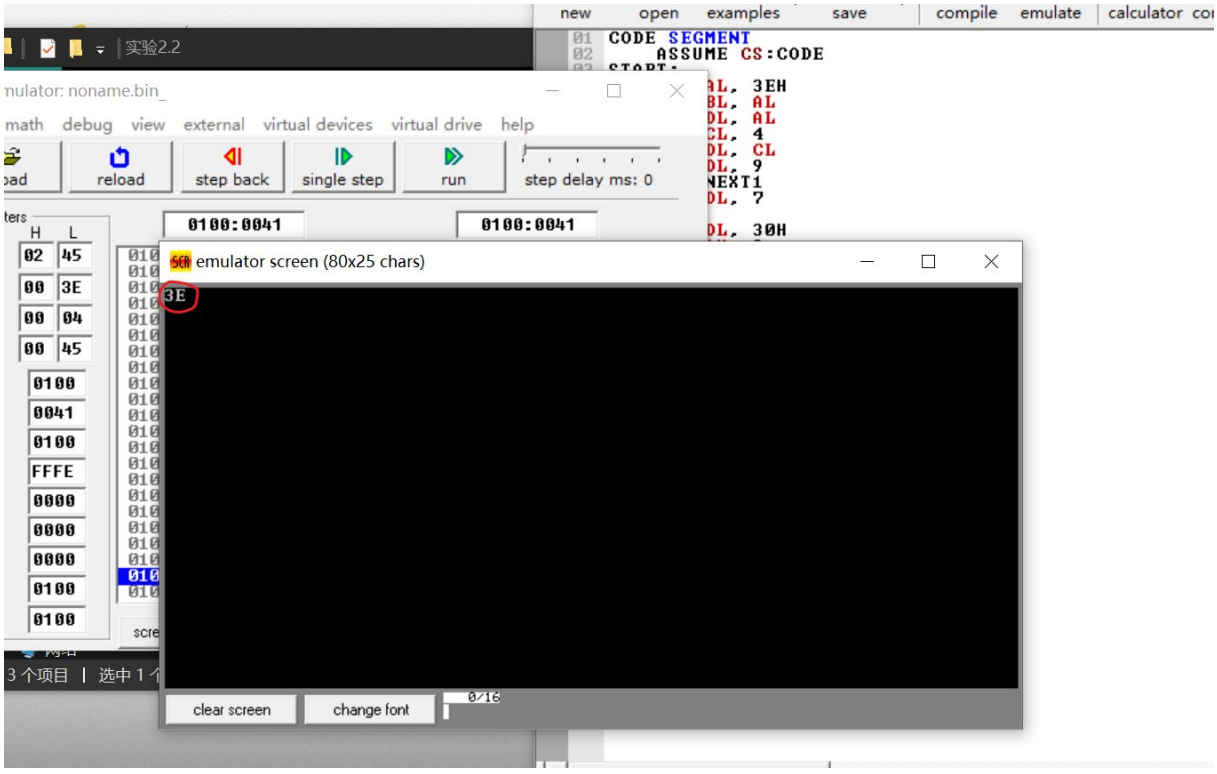
1. 首先俺们先试着运行一下：

敲字敲好代码之后：

```
CODE SEGMENT
        ASSUME CS:CODE
START:
        MOV AL, 3EH
        MOV BL, AL
        MOV DL, AL
        MOV CL, 4
        SHR DL, CL
        CMP DL, 9
        JBE NEXT1
        ADD DL, 7
NEXT1:
        ADD DL, 30H
        MOV AH, 2
        INT 21H
        MOV DL, BL
        AND DL, OFH
        CMP DL, 9
        JBE NEXT2
        ADD DL, 7
NEXT2:
        ADD DL, 30H
        MOV AH, 2
        INT 21H
CODE ENDS
        ENDS START
```

但是直接敲进去是错误的，因此我们在 windows 下进行调试（比较方便）

得到如下运行结果。之后对该程序进行分析和注释，分别画出流程图和写出程序注释放在上面两点。

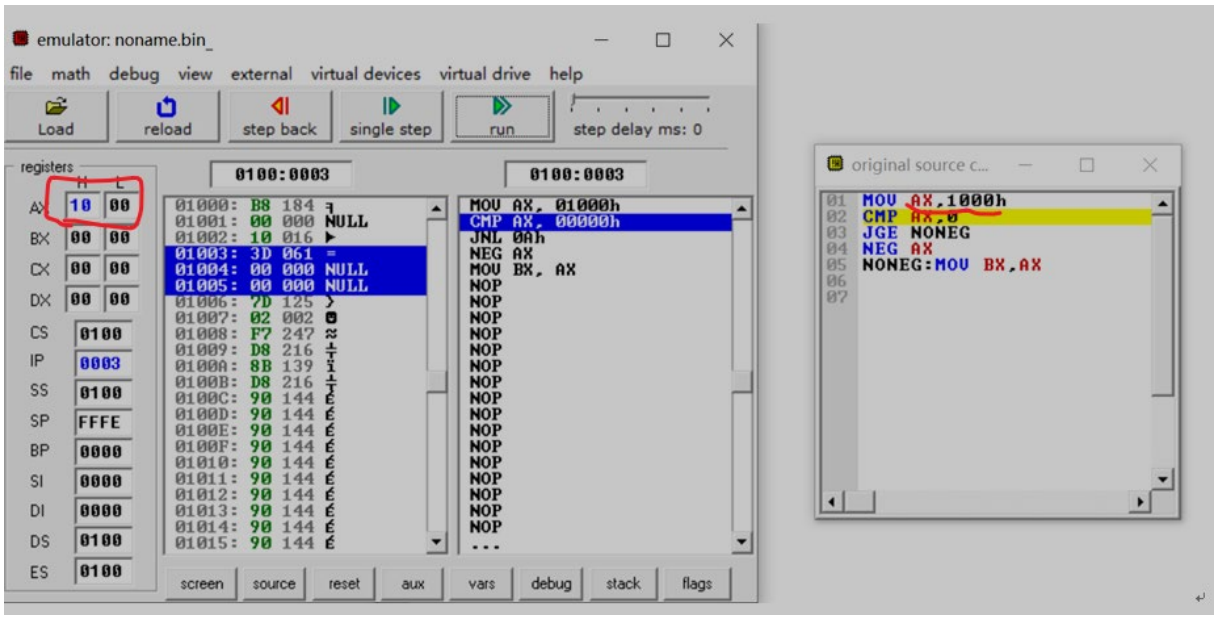


【实验心得】

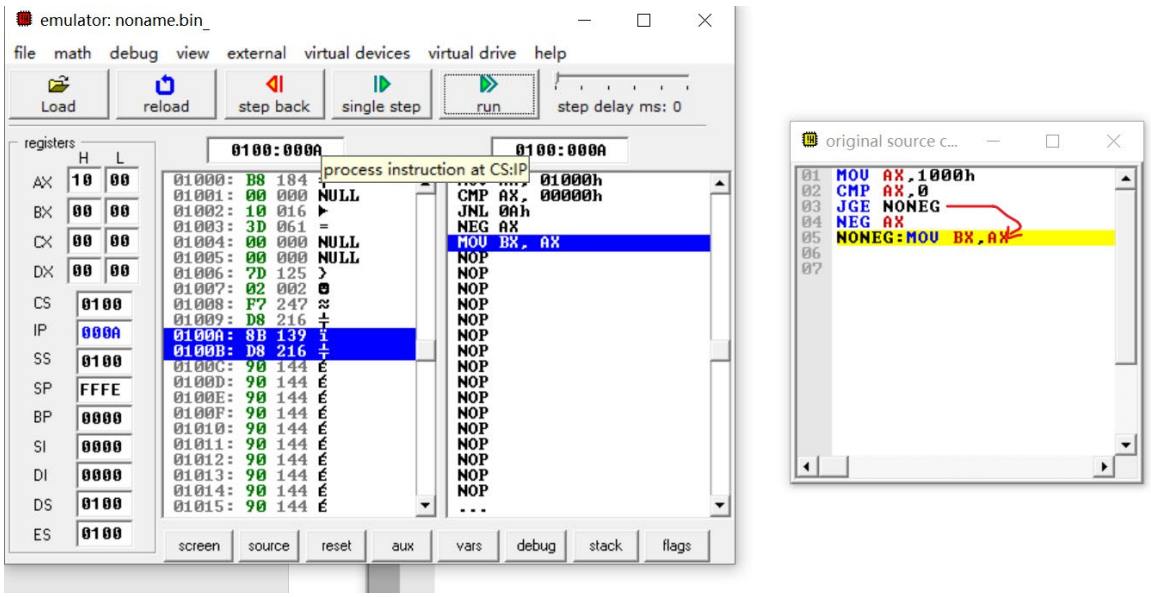
【提出问题】

1. 问题 1：为什么条件成立的执行语句会通过 NONGE 传递：

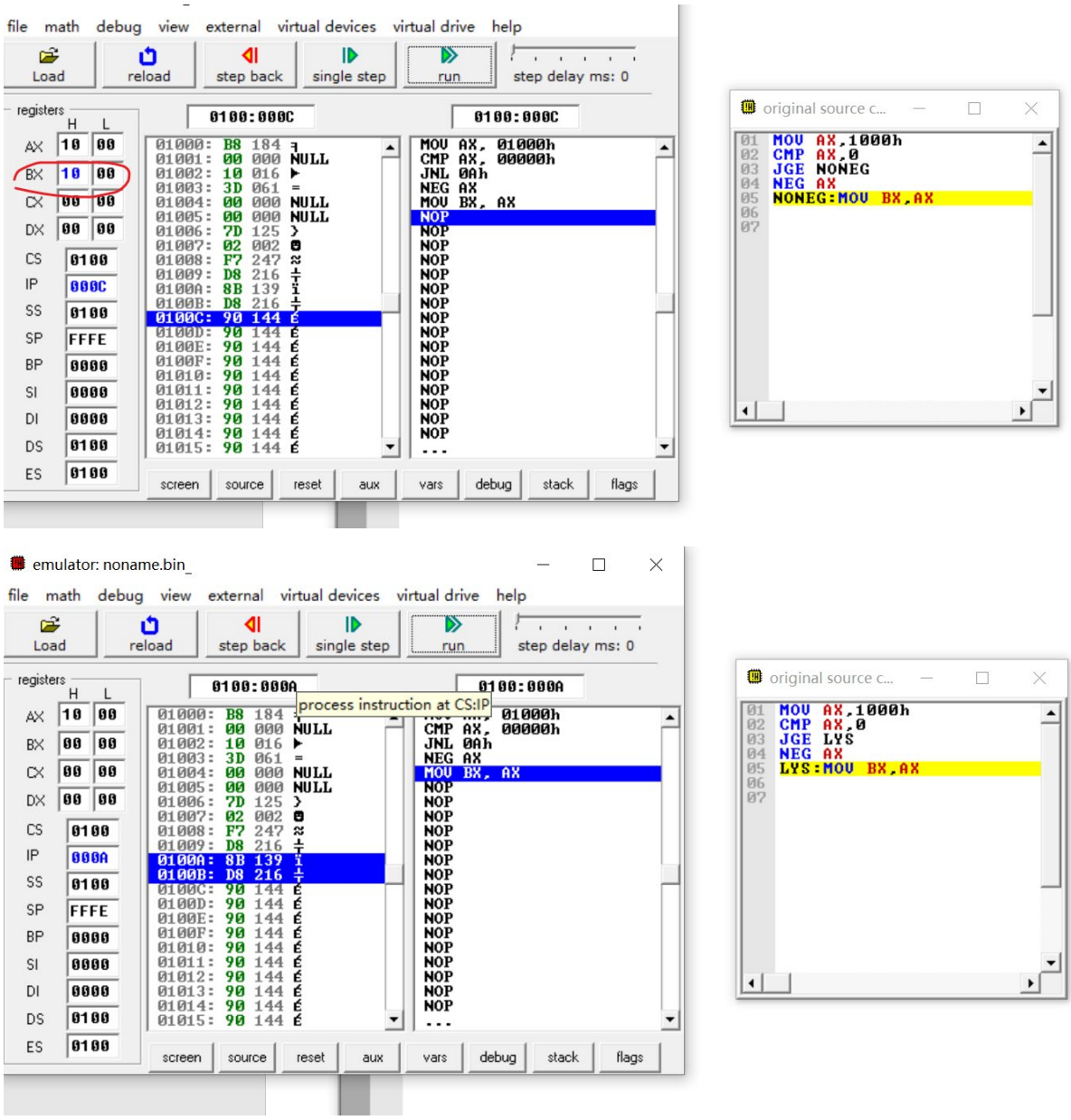
先进行一下测试，看看那个 NONEG 是不是能够随便换



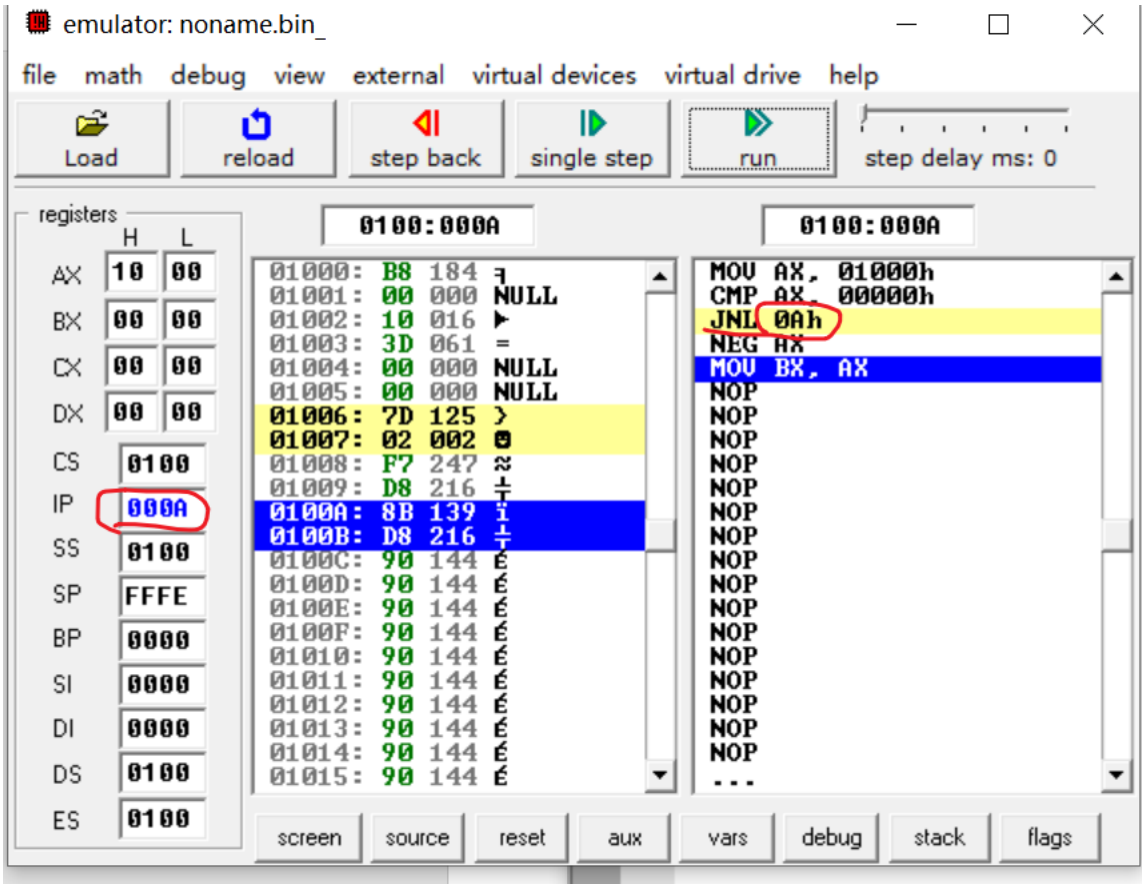
然后发现是直接跳转的



BX 的值发生了改变



换成 LYS 之后发现仍旧是可以运行，这就说明该处的字符同实验 1.1 中的 string 是一样的，仅仅代表是一个标识符。



而且我发现这个是通过 CS[IP]来进行控制的

【实验心得】

这个实验是老师给的程序，在读程序的时候非常快乐就感觉自己好像一位侦探，就看每个线索代表的含义以及背后的意思。最触动我的就是数字和字母是如何通过 AL=2 和 INT 21H 进行输出的，居然可以将数据和字母进行那样的对应关系，实在是让我感觉到很奇妙。流程图也代表着我已经将这个程序怎么运作的搞清楚了，hhh，读懂程序之后很开心！

【第二个实验】

（1）实验流程图：

（2）实验源代码（粘贴源代码）：

```
    MOV BL,DL
//循环体
S: CMP DL,39H
   JGE DECIDE41
```

```
    JMP DECIDE31
loop s
//对 ASCII 码对应的十六进制数进行判断
DECIDE31: CMP DL,31H
          JGE ONENINE
          JMP DECIDEOD
DECIDEOD: CMP DL,0DH
          JGE NONE
          JMP NONE
DECIDE41: CMP DL,41H
          JGE DECIDE5A
          JMP START
DECIDE5A: CMP DL,5AH
          JGE DECIDE61
          JMP OUTPUTC
DECIDE61: CMP DL,61H
          JGE DECIDE7A
          JMP NONE
DECIDE7A: CMP DL,7AH
          JGE NONE
          JMP OUTPUTC
//输出 1~9 段
ONENINE:MOV AH,02H
        INT 21H
        JMP START
//输出 c 段
OUTPUTC:MOV DL,63H
        MOV AH,02H
        INT 21H
//什么都不做段
NONE:    JMP START
//终止程序段
JEND:    MOV AH, 4CH
         INT 21H
```

```
CODE ENDS
    END START
```

（3）实验代码、过程、相应结果（截图）并对实验进行说明和分析：

首先先查找 int 21h 的使用表

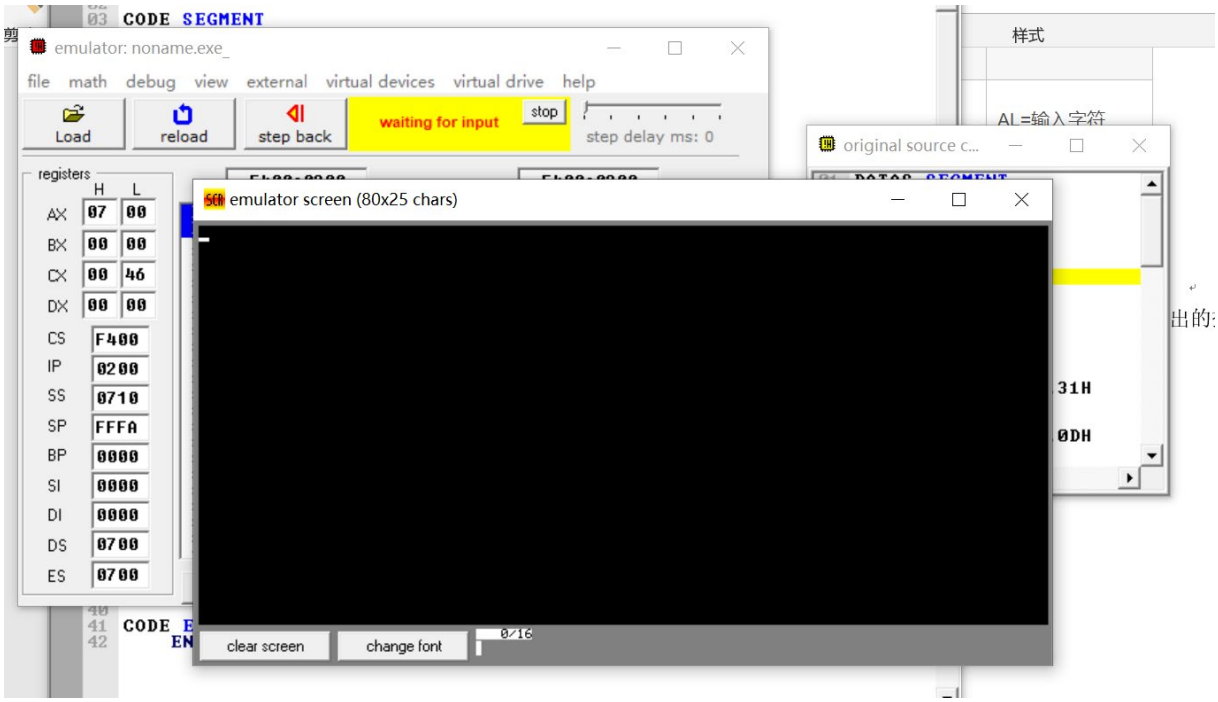
	回显		
02	显示输出	DL=输出字符	
03	异步通讯输入		AL=输入数据
04	异步通讯输出	DL=输出数据	
05	打印机输出	DL=输出字符	
06	直接控制台I/O	DL=FF(输入) DL=字符(输出)	AL=输入字符
07	键盘输入(无回显)		AL=输入字符
08	键盘输入(无回显) 检测Ctrl-Break		AL=输入字符

开始用的是 AH=01 是有回显的，不满足要求，这回改用 07 和 02 进行输入和输出的操作。首先对所需要的语句进行子程序分块书写：每个功能都写成一个代码段

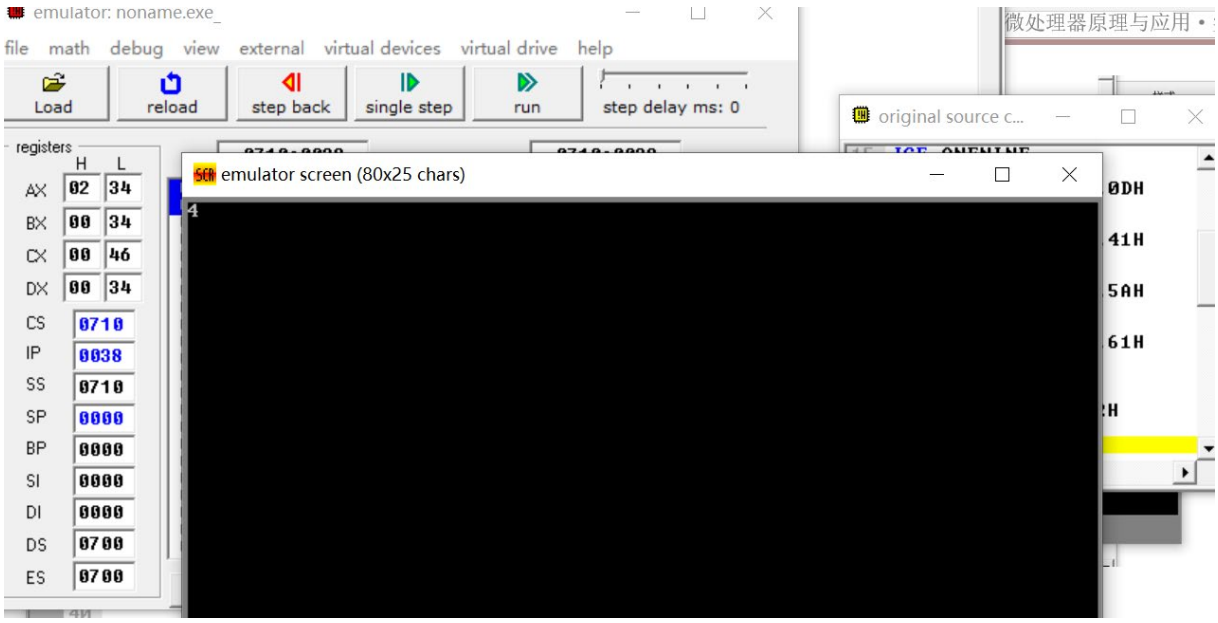
实验过程记录：因为没有回显所以不知道输入的是什么但是肯定没有糊弄老师

①输入 1~9：

等待输入

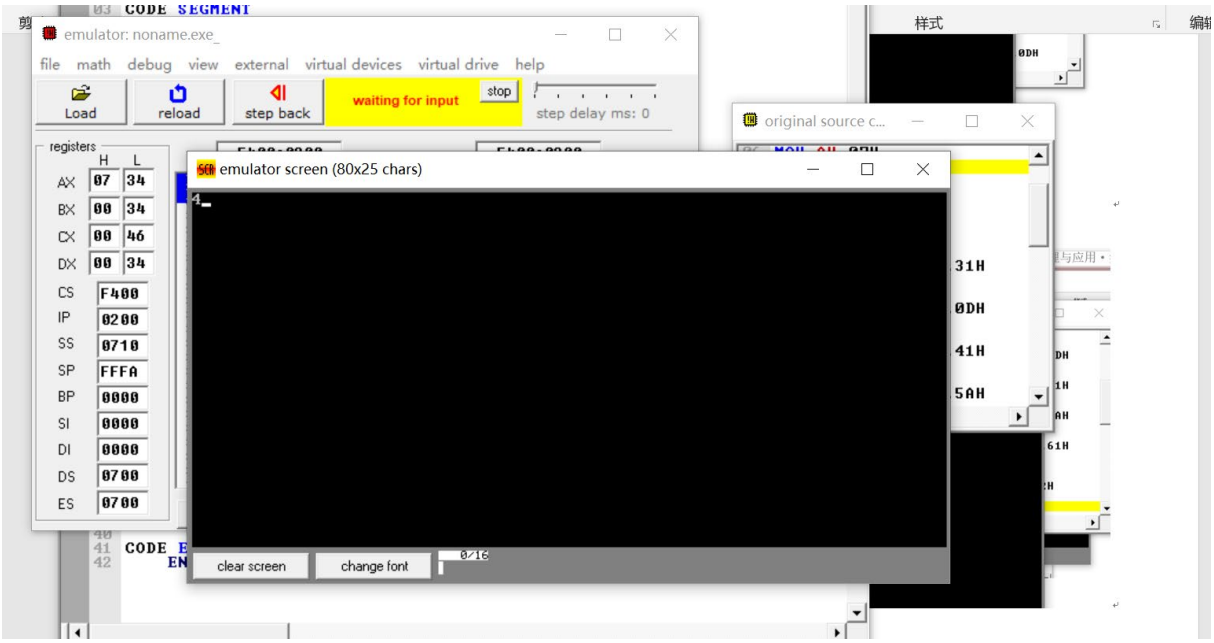


输入了 4，成功显示

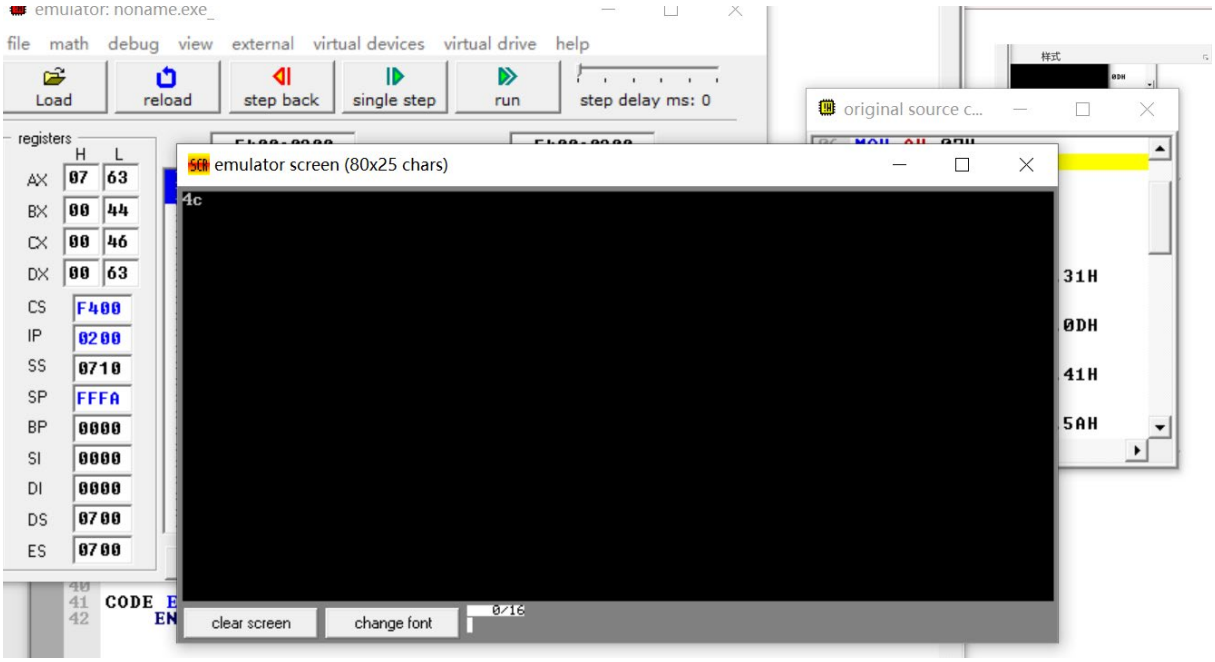


②输入 A~Z

程序继续执行，输入！

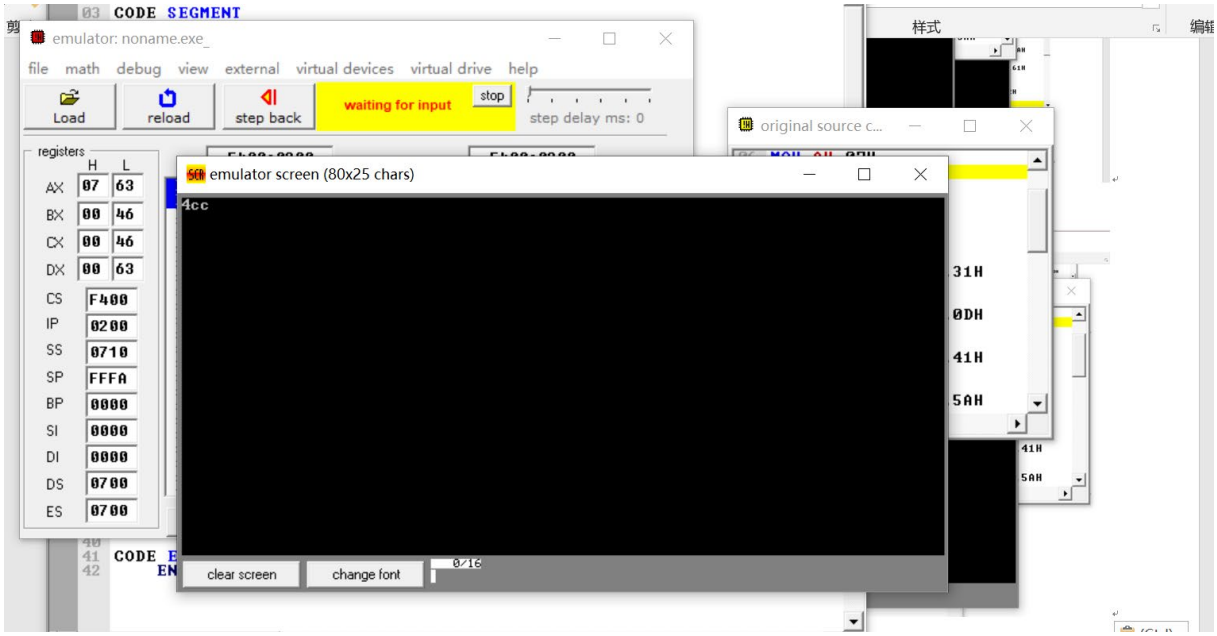


输入 D 和 F 都是显示 c

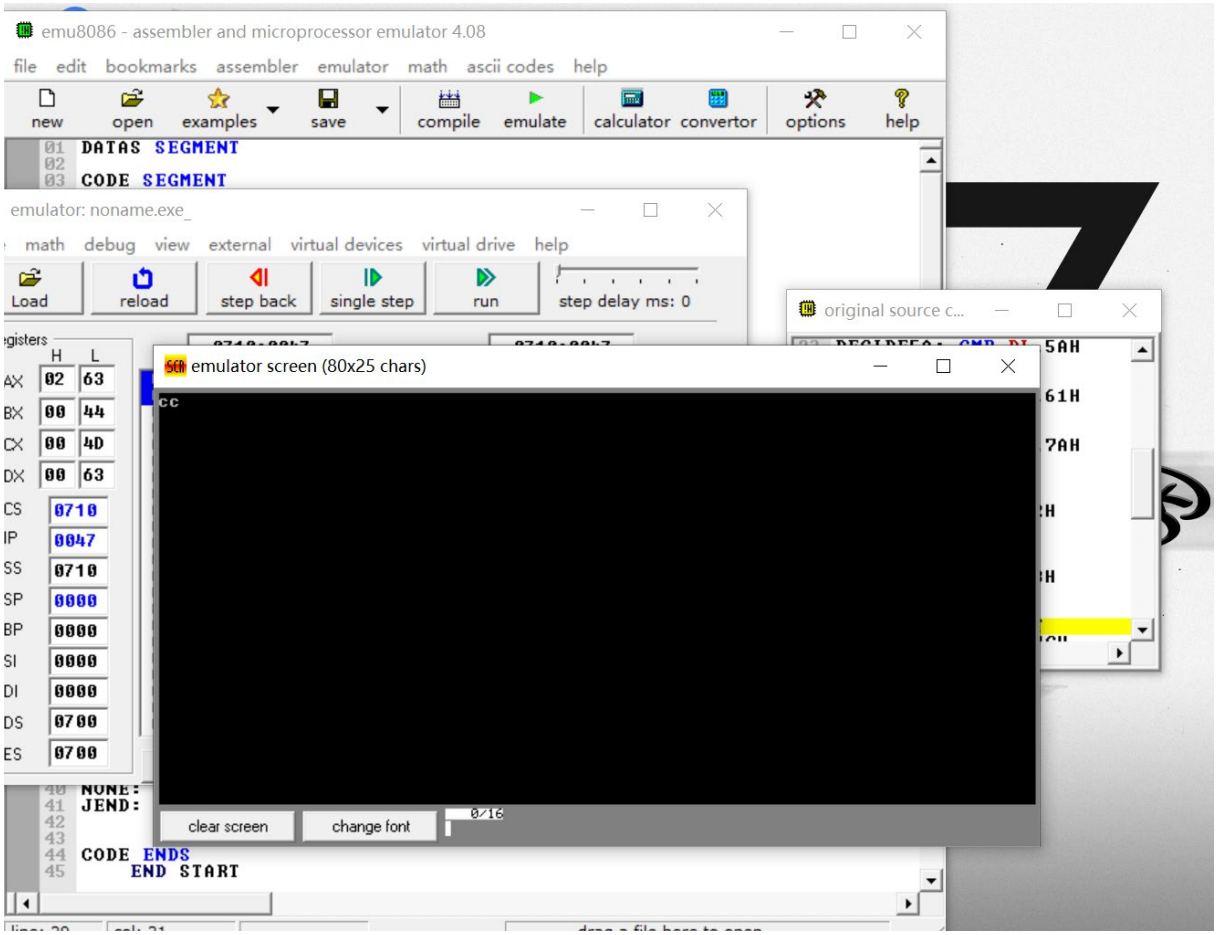


③输入 a~z：

继续输入程序

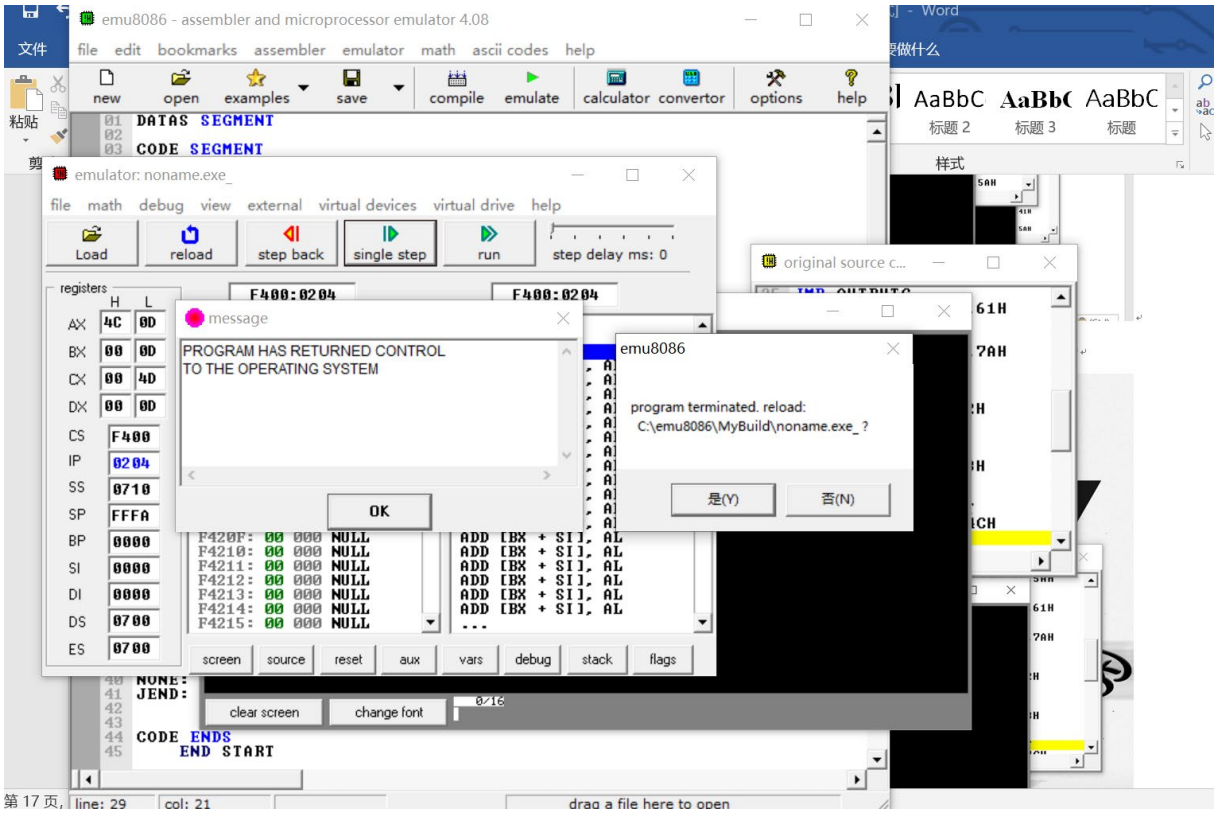


发现输入 d 和 f 也是都输出 c(这里出现了 bug 修改了一下程序，所以只有两个 c)



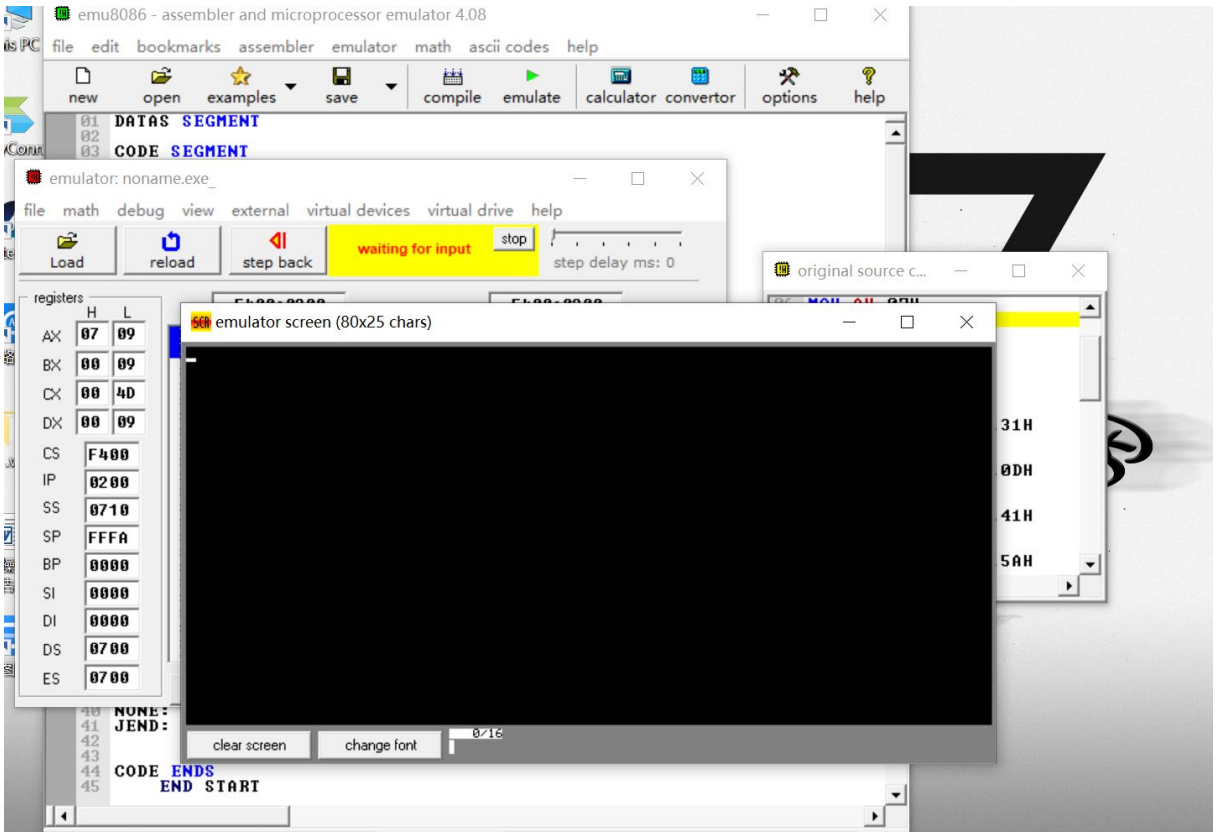
④输入 enter

程序如愿以偿地结束了



⑤输入不相关的如 Tab

在继续等待着输入指令



【实验心得】这个实验的实验心得就一个字，哈哈哈哈哈爽！开始的时候我都想找百度看答案，直接抄一个程序得了，因为见过的程序不多，不知道汇编的规范，但是想了想汇编最重要的就是试一试，我就抱着上课将的代码段的心态试了试，结果还真的出来了！哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈哈，我知道自己还是个菜鸟，许多代码还不够优化，但是这是自己第一个独立写出的汇编指令，太爽了！痛并快乐着，爽死了！

【上课问题】

1. 存储器和 CPU 连接例题

SRAM 与 CPU 的连接

例.试用 2K*8 位 SRAM 芯片，组建 16K*16 位的存储系统，起始地址为 5000H。

（1）计算芯片的数量并分组

（2）写出每组芯片的地址范围

（3）设计芯片的片选逻辑

（4）画出 CPU 与 SRAM 芯片的连接图

解：

（1）16k*16 位/2K*8 位=8*2=16(片)

需要 16 片，分为 8 组，每组 2 片

（2）SRAM1：5000H~57FFH

SRAM2：5800H~5FFFH

SRAM3：6000H~67FFH

SRAM4：6800H~6FFFH

SRAM5：7000H~77FFH

SRAM6：7800H~7FFFH

SRAM7：8000H~87FFH

SRAM8：8800H~8FFFH

（3）SRAM 芯片有 11 位地址，多余的 5 位地址 A15~A11 用于形成片选逻辑。

SRAM1：A15~A11=01010

SRAM2：A15~A11=01011

SRAM3：A15~A11=01100

SRAM4：A15~A11=01101

SRAM5：A15~A11=01110

SRAM6：A15~A11=01111

SRAM7：A15~A11=10000

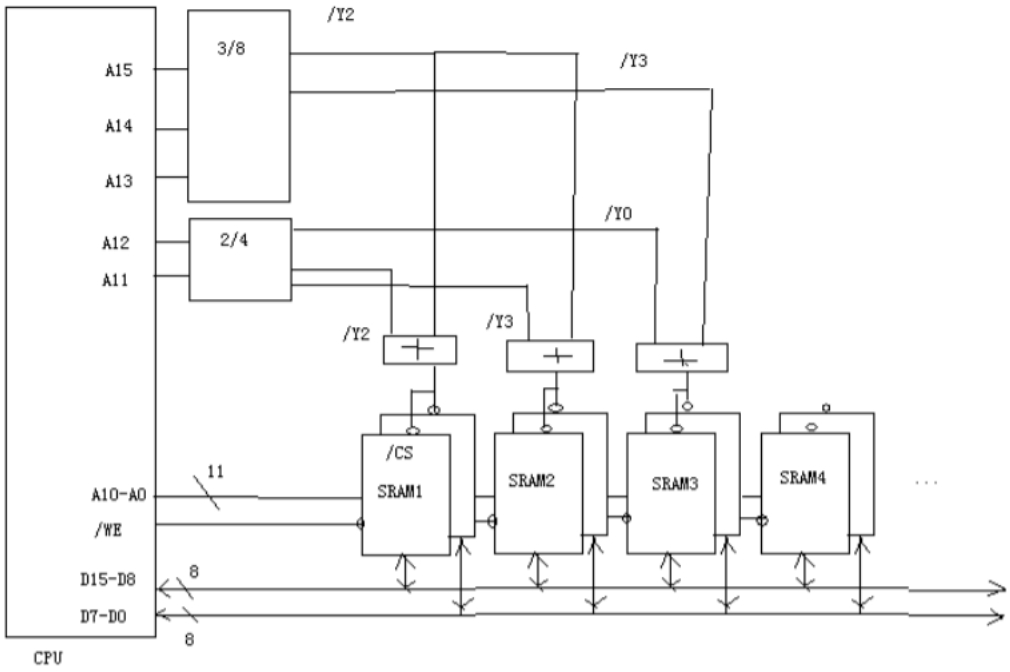
SRAM8：A15~A11=10001

SRAM8：A15~A11=10001

采用一个 3/8 译码器和一个 2/4 译码器组建片选逻辑电路。

	3/8 译码器	2/4 译码器
SRAM1	/Y2 有效	/Y2 有效
SRAM2	/Y2 有效	/Y3 有效
SRAM3	/Y3 有效	/Y0 有效
SRAM4	/Y3 有效	/Y1 有效
SRAM5	/Y3 有效	/Y2 有效
SRAM5	/Y3 有效	/Y3 有效
SRAM6	/Y4 有效	/Y0 有效
SRAM7	/Y4 有效	/Y1 有效
SRAM8	/ Y4 有效	/Y2 有效

（4）CPU 与 SRAM 芯片的连接图如下



2. 8086 中的 AF 和 CF 的区别

AF：辅助进位标志

跟CF一样是进位（借位）的标志寄存器，唯一不一样的是，8位运算或16位运算时如果有进位或借位CF就等于1，而AF也一样是进位或借位的标志，只不过不是8位也不是16位运算时的进位标志，而是4位运算时的进位或借位的标志。

例如：两个8位寄存器相加，AL=1000 0001，BL=1000 0011

结果CF=1，AF=0 因为AL和BL的低四位相加没有进位

AF是为了在BCD码运算时，要用到的，因为BCD码是以4位表示的。。。