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：（基础篇）汇编程序 HelloWorld

专　业　班　级　通信工程 二班

学　生　学　号　201922121209

学　生　姓　名　陈泽宇

实　验　时　间　2020 年 3 月 2 日

实验报告

【实验目的】

1.掌握 windows 的基本 Masm for Windows 集成实验环境 2020 的使用。

2.掌握 win10 使用 DOSBox 实现虚拟 DOS 环境下 masm5 的编译和运行。

【实验要求】

1. 完成 Windows 和 DOS 环境的汇编程序的编译和执行，屏幕上显示 Hello World。

2. 了解如何在 Windows 环境中 Debug 程序，探索性地去了解寄存器的状态和单步执行程序的时候，寄存器的存储过程。

【实验具体内容】

1. DOS 环境下进行源程序的编译链接实现 Hello World 的输出

2. Windows 环境下进行源程序的编译链接实现 Hello World 的输出

【第一个实验】

（1）实验源代码：

```
STACKS  SEGMENT   STACK      ;堆栈段
      DW  128 DUP(?)  ;注意这里只有 128 个字节
STACKS  ENDS

DATAS  SEGMENT
     STRING  DB  13,10,'Hello World!',13,10,'$'
DATAS  ENDS

CODES  SEGMENT
     ASSUME    CS:CODES,DS:DATAS
START:
     MOV  AX,DATAS
     MOV  DS,AX
     LEA  DX,STRING
     MOV  AH,9
     INT  21H

     MOV  AH,4CH
     INT  21H
CODES  ENDS
    END   START
```

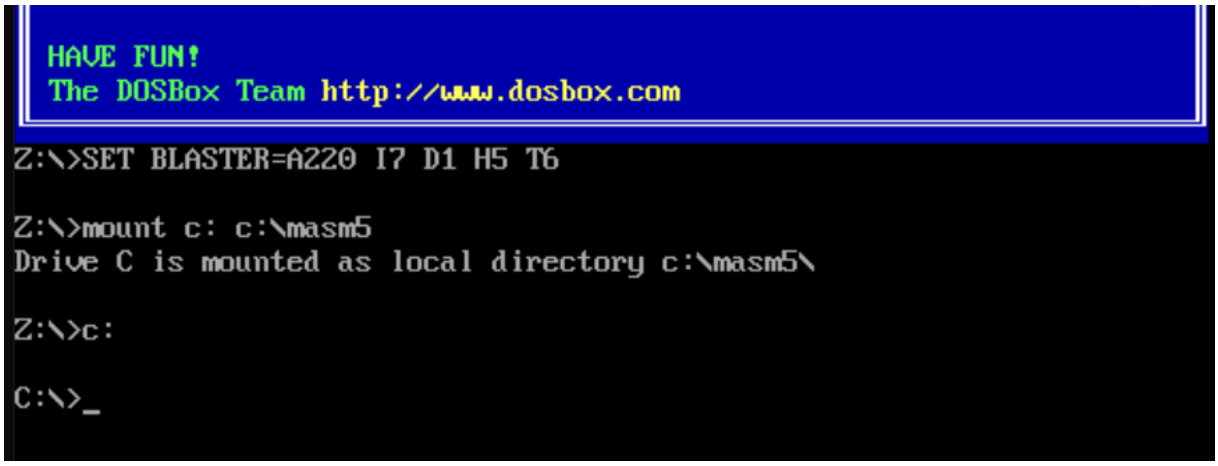
（2）实验代码、过程、相应结果（截图）并对实验进行说明和分析：

- 为直接利用 dos 环境，先将安装好的 dosbox 进行本地的磁盘驱动映射，以便在后续的实验中可以直接编译本地的汇编语言文件。在 dosbox 的安装文件夹中找到 **DOSBox 0.74-3 Options.bat** 进入 **dosbox-0.74-3.conf** 的配置文件编辑，在文档的最后加入如下控制语句

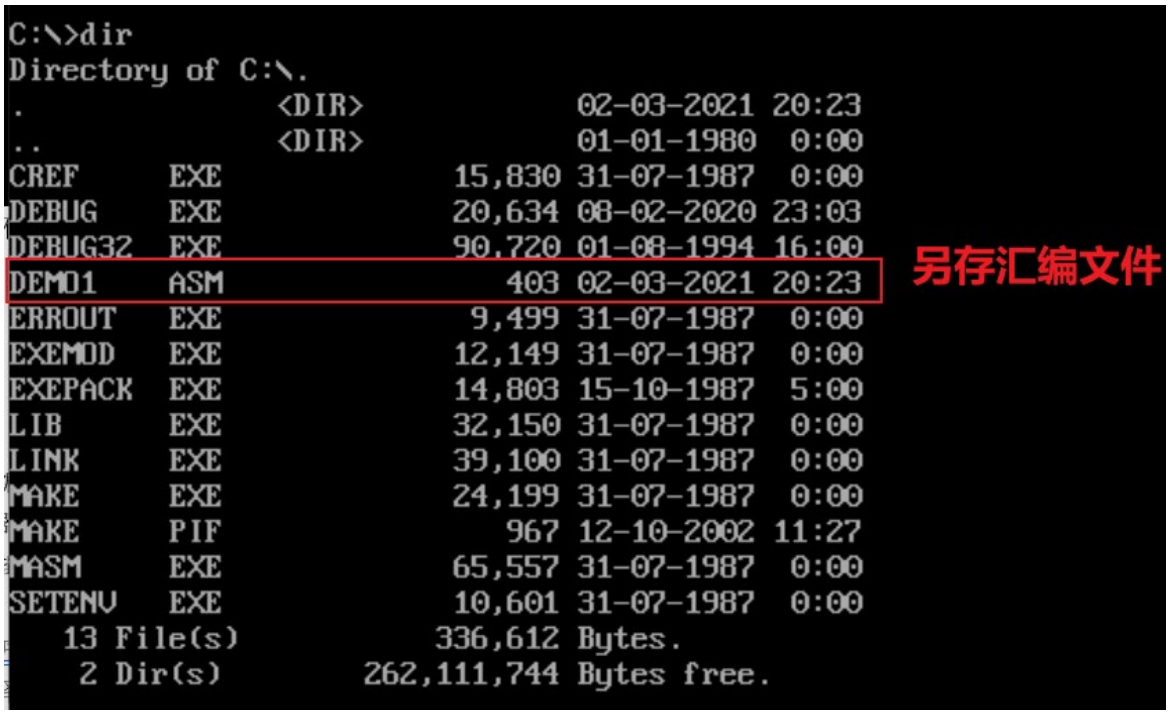
```
mount c: c:\masm5
```

```
c:
```

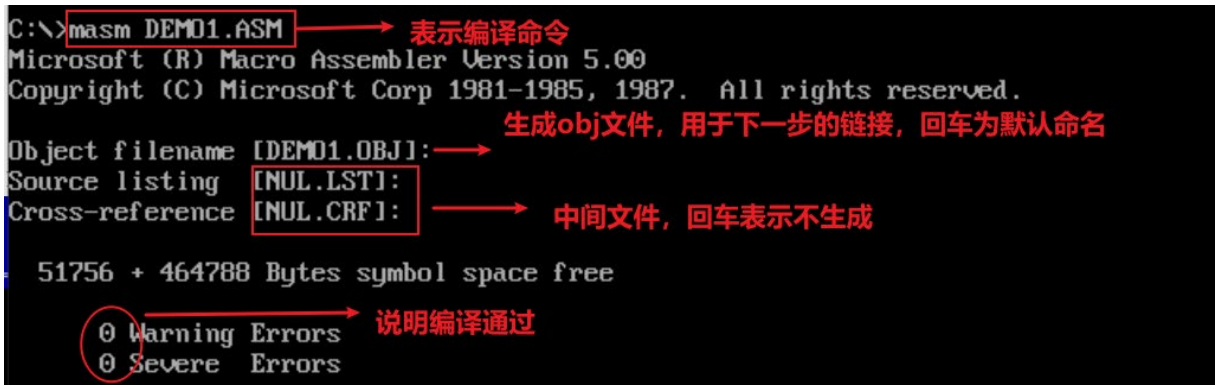
即可将 dosbox 的工作文件夹（工作区）设置为 c:\masm5，如下图所示



- 将上述代码另存为 demo1.asm 放置在已经配置好的 masm 环境的文件夹中，使用 dir 查看文件夹内容，如图所示



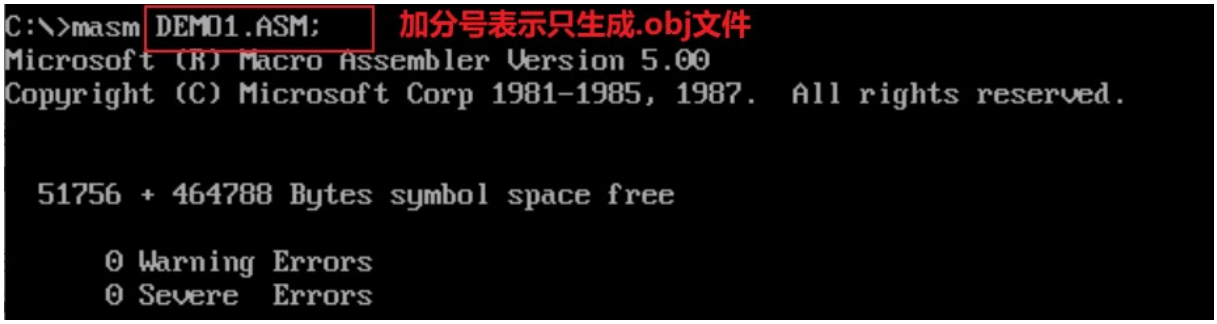
- 利用 masm.exe 进行编译



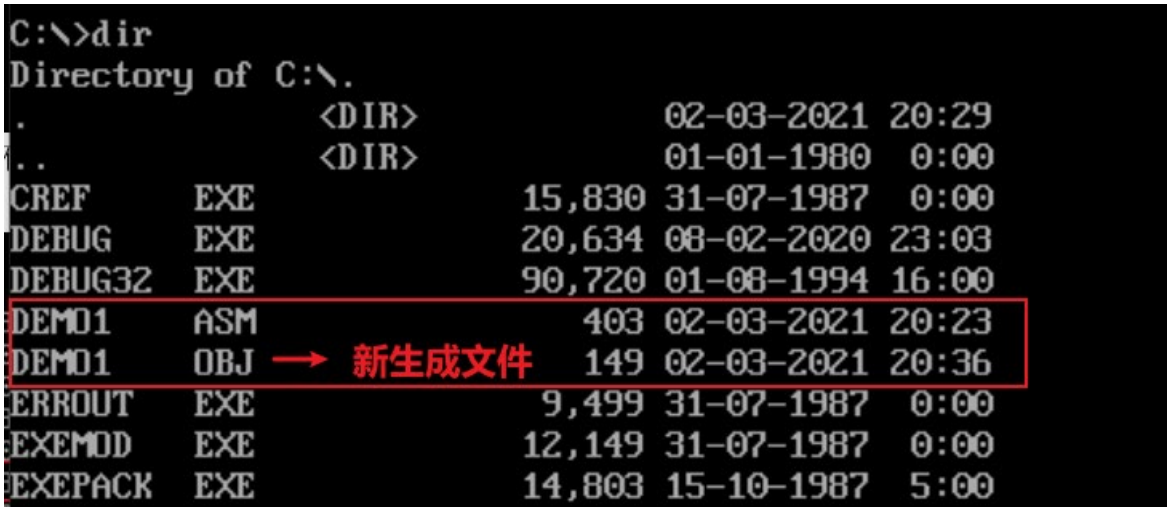
这里涉及到了几个类型的文件：

- source(.asm)： 指定源程序，缺省的扩展名为 ASM
- list(.lst)： 指定输出的列表文件，缺省的扩展名是 LST，缺省情况下不生成列表文件
- cref(.crf)： 指定输出的交叉参考文件，缺省的扩展名是 CRF，缺省情况下不生成交叉参考文件

此外，也可以用如下的方式进行快捷编译

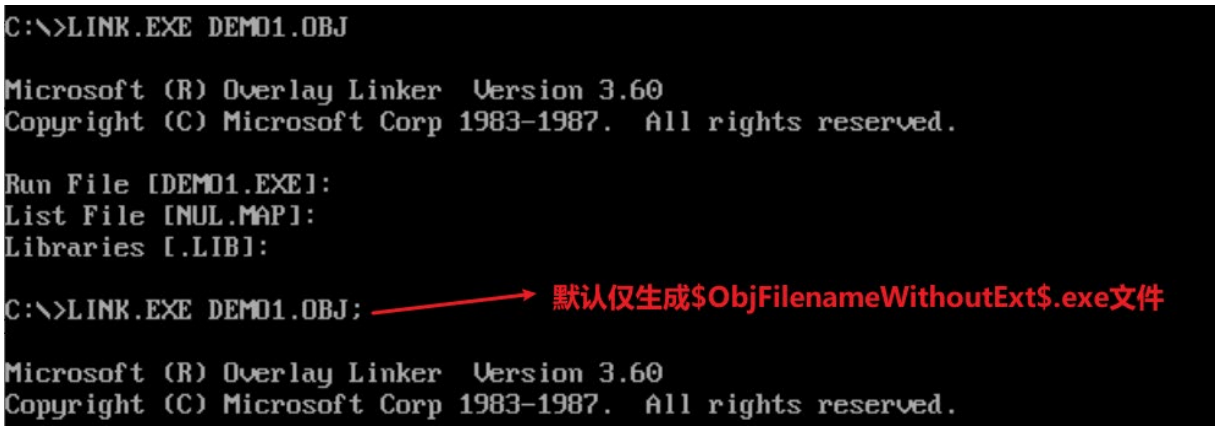


使用 dir 命令进行查看



- 利用 link.exe 进行链接

键入如下命令，如下图所示

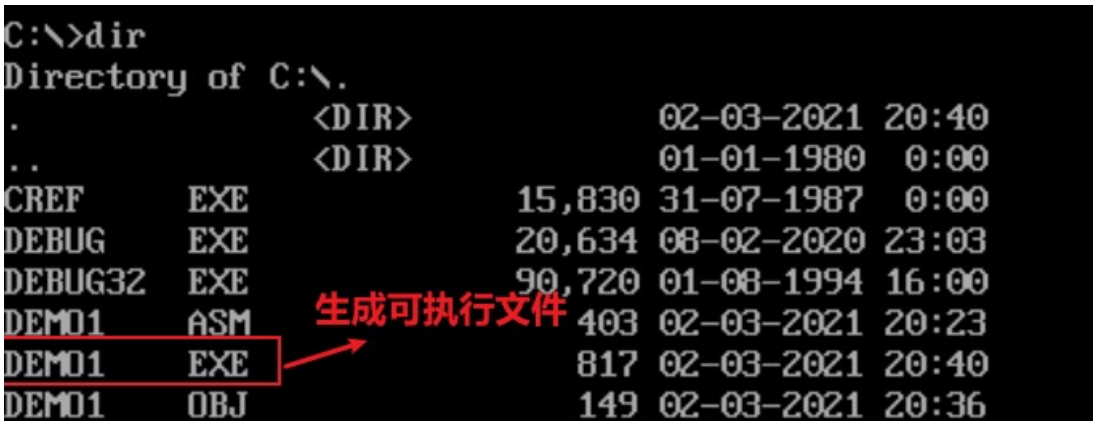


这和之前的编译操作相同，**命令行最后的分号表示其后的缺省项按缺省设置处理**，这里不再赘述。

涉及到的文件类型解释如下

- source(.obj)： 指目标代码文件，缺省的扩展名为 OBJ，可以有多个目标程序代码文件，文件间用加号或空格进行间隔
- out(.exe)： 指定输出的可执行文件。缺省的文件名同第一个目标代码模块的文件名，缺省的扩展名是 EXE
- mapfile(.map)： 指定输出的定位图文件，缺省的扩展名是 MAP，缺省情况下不生成定位图文件
- library(.lib)： 指定连接时使用的库文件，缺省的扩展名是 LIB，可以有多个库，用加号或空格进行间隔，缺省的情况下不使用库

使用 dir 命令进行查看



执行 DEMO1.exe

可见这里成功输出了 `Hello World！`这一字符串

【第二个实验】

说明：本实验在 Windows 平台上采用两种方式进行程序的编写和运行，一种是采用 Masm 集成实验套件 2020，另一种是采用当下流行的文本编辑器 VSCode 结合其应用生态中的 Masm 插件进行运行。但是需要指出的是二者的运行逻辑都是相同的，仅仅在形式上有所不同。

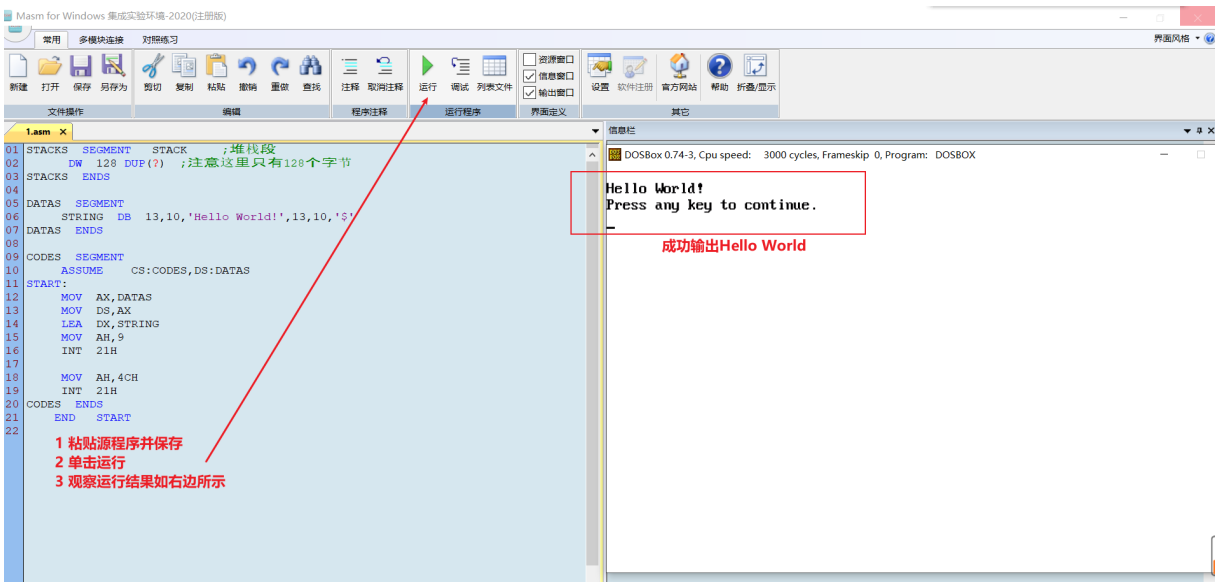
在 Windows 平台上可以利用一些程序和插件实现第三方程序内直接编辑汇编程序，随后该程序直接调用 PC 中的 Dos-Box 平台实现程序的一键编译、链接、运行（或调试），相比第一个实验而言，这样的方式使得编程无论从文本编辑还是程序的编译运行调试工作均得到了一定程度上的简化。

（1）实验源代码（粘贴源代码）：（和第一个实验源代码相同）

```
STACKS  SEGMENT    STACK      ;堆栈段
      DW  128 DUP(?)  ;注意这里只有 128 个字节
STACKS  ENDS

DATAS  SEGMENT
     STRING  DB  13,10,'Hello World!',13,10,'$'
DATAS  ENDS

CODES  SEGMENT
     ASSUME     CS:CODES,DS:DATAS
START:
     MOV  AX,DATAS
     MOV  DS,AX
     LEA  DX,STRING
     MOV  AH,9
     INT  21H

     MOV  AH,4CH
     INT  21H
CODES  ENDS
END   START
```

（2）实验代码、过程、相应结果（截图）并对实验进行说明和分析：

- 首先利用 Masm 2020 实验套件进行操作，这里的操作比较简单，如下图操作即可，可见实验平台会自动调用 dosbox 并且实现一键编译运行。



如果更改实验代码，这里更改了输出的文本和其他部分内容如下所示

```
DATAS   SEGMENT
      STRING   DB   13,10,'Hello Hello Hello Hello”,13,10,'$$$'
DATAS   ENDS
```

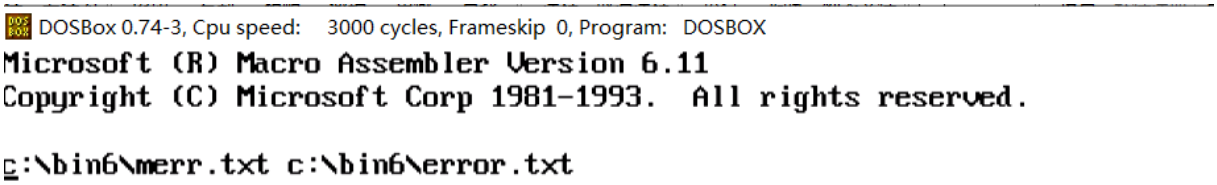
其余代码不变，重复之前的操作如下所示

这里仍然正常输出字符串。查阅资料后得知，**$是字符串结束的标志**，所以后边的多余$被忽略了。

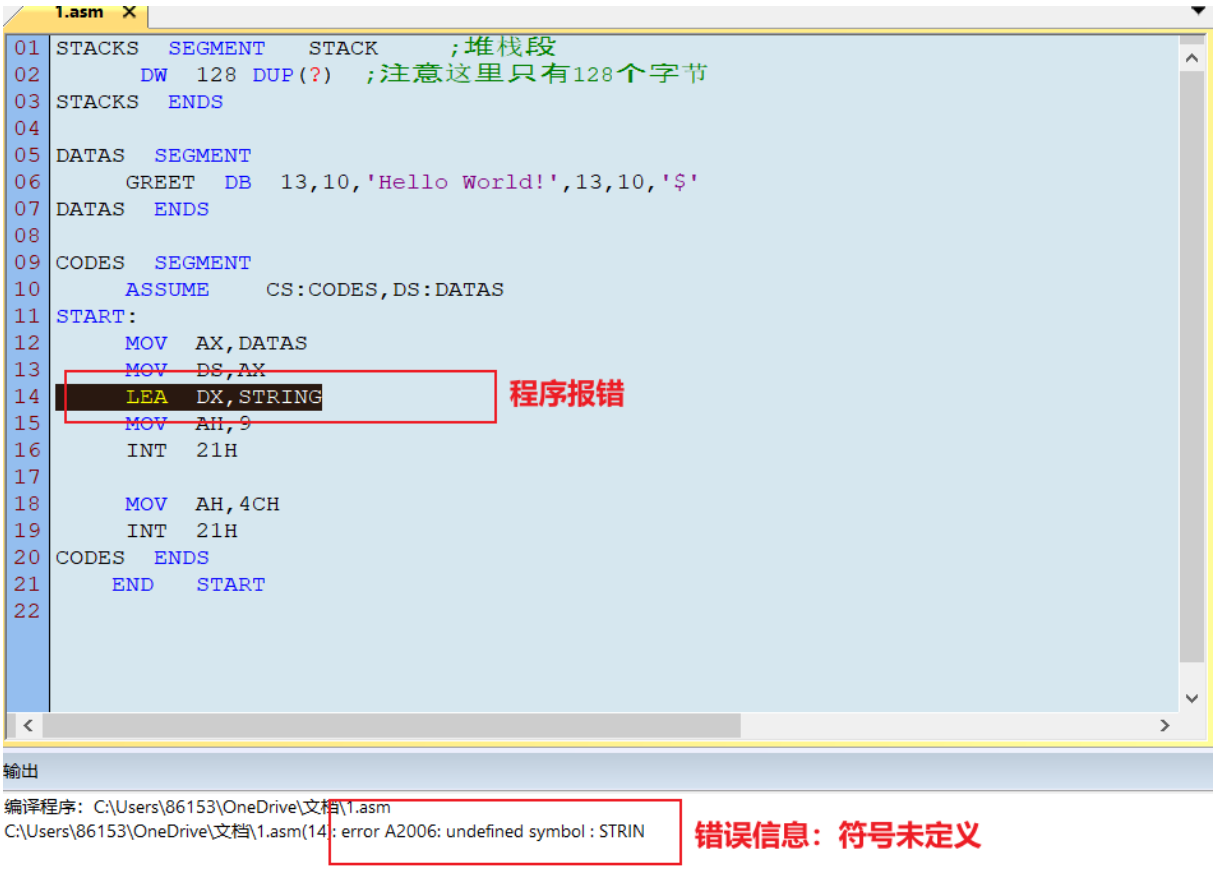
如果继续更改代码，仅改变 STRING 的名称，其他代码不变，如下所示

```
GREET   DB   13,10,'string',13,10,'$'
```

编译运行，观察结果如下图所示



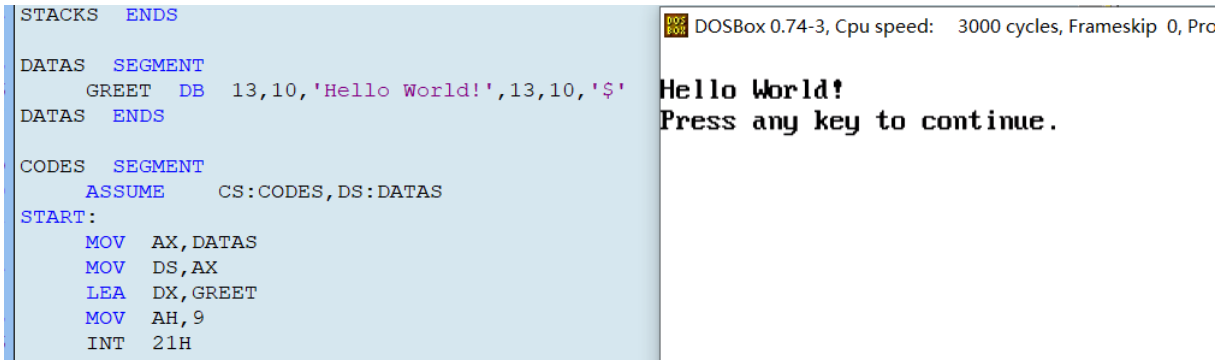
发现弹出错误信息，



经过分析可知，需要对符号进行对应，修改代码如下

```
……
GREET  DB  13,10,'string',13,10,'$'
……
LEA  DX,GREET
```

再次进行运行如下图所示



可知成功输出 Hello World!字符串

查阅相关资料对程序做出分析，对程序做出注释，如下所示

```
ASSUME    CS:CODES,DS:DATAS
;伪指令，指明变量与段寄存器的联系,比如 assume ds:data,它是告诉编译器以后所有在 data 段中定义的变量寻址时，使用 ds 作为段地址，
;不产生机器指令，必须在程序中对 ds 赋值
```

```
STACKS  SEGMENT   STACK      ;堆栈段
      DW  128 DUP(?)  ;注意这里只有 128 个字节（这里不是太明白，Debug 调试显示应该是 256 个字节）
STACKS  ENDS
;dw，define word，即在栈区申请了 128 个字空间，但没有经过初始化，dup 表示重复操作
; 在栈区申请了 1+1+12+1+1+1=17 个字符串空间，以$结尾，13 和 10 分别表示换行和回车所对应的 ASCII 码值
DATAS  SEGMENT
     STRING  DB  13,10,'Hello World!',13,10,'$'
     ; 这里的 string 可理解为变量名，存放着偏移地址
DATAS  ENDS

CODES  SEGMENT
START:
     MOV  AX,DATAS
     MOV  DS,AX
     ;datas 送入 ax 寄存器中，把 ax 存入 ds 段寄存器中，间接实现 ds=datas


     LEA  DX,STRING
     ;Load effect address——取有效地址，也就是取偏移地址，这里把偏移地址存到 DX

     MOV  AH,9
     INT  21H
     ; 可以理解为组合指令，该功能为显示 DS：DX 地址处的字符，从而实现 Hello World 的输出


     MOV  AH,4CH
     INT  21H
     ; 组合指令，功能为结束程序

CODES  ENDS
END   START
;伪指令，表示程序结束
```

利用 Debug 进行内存的查看，如下图所示：

```
-r
AX=FFFF  BX=0000  CX=0131  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=075C  ES=075C  SS=076C  CS=077E  IP=0000   NV UP EI PL NZ NA PO NC
077E:0000 B87C07        MOV     AX,077C
```

根据汇编知识，考虑在栈空间中找文本内容，而考虑到 SS:SP 始终指向栈顶，再结合程序中先申请了 128 个字空间，故考虑进行如下内存查找

```
-r
AX=FFFF  BX=0000  CX=0131  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=075C  ES=075C  SS=076C  CS=077E  IP=0000   NV UP EI PL NZ NA PO NC
077E:0000 B87C07        MOV     AX,077C
-d 076C:0100
076C:0100  0D 0A 48 65 6C 6C 6F 20-57 6F 72 6C 64 21 0D 0A   ..Hello World!..
076C:0110  24 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   $...............
076C:0120  B8 7C 07 8E D8 8D 16 00-00 B4 09 CD 21 B4 4C CD   .|..........!.L.
076C:0130  21 89 97 C0 22 5E 8B E5-5D C3 55 8B EC 81 EC 86   !..."^..].U.....
076C:0140  00 57 56 B8 BE 05 50 E8-C3 71 83 C4 02 8B F0 0B   .WV...P..q......
076C:0150  F6 74 61 8D 86 7E FF 8B-F8 80 3C 3B 74 05 80 3C   .ta..~....<;t..<
076C:0160  00 75 45 8D 86 7E FF 3B-C7 73 42 8B C7 2B C5 04   .uE..~.;.sB..+..
076C:0170  82 88 86 7E FF 80 3D 3A-74 1B B8 5C 00 50 8D 86   ...~..=:t..\.P..
```

根据上图的 实验结果，推知字符存在了第 256 个内存单元中（16^2=256）

此外，实验代码中涉及到了几个新的操作符，查阅资料后总结如下

db 定义一个字节 define byte (1 Byte)
dw 定义一个字 define word (2 Bytes)
dd 定义一个双字 define double word (4 Bytes)
dup 是一个操作符，在汇编语言中同 db,dw,dd 一样，也是由编译器识别处理的符号。
它是和 db,dw,dd 等数据定义伪指令配合使用的，用来进行数据的重复。
比如
db 3 dup (0) 相当于 db 0,0,0
db 3 dup (0,1,2) 相当于 db 0,1,2, 0,1,2, 0,1,2
db 3 dup ('abc','ABC')
使用格式如下：
db 重复的次数 dup (重复的字节型数据)
dw 重复的次数 dup (重复的字型数据)
dd 重复的次数 dup (重复的双字型数据)

另外需要注意一些细节，字符中的 10 和 13 有特殊的含义：回车、换行。

但是这里说的回车指回车符，不是键盘上的回车。

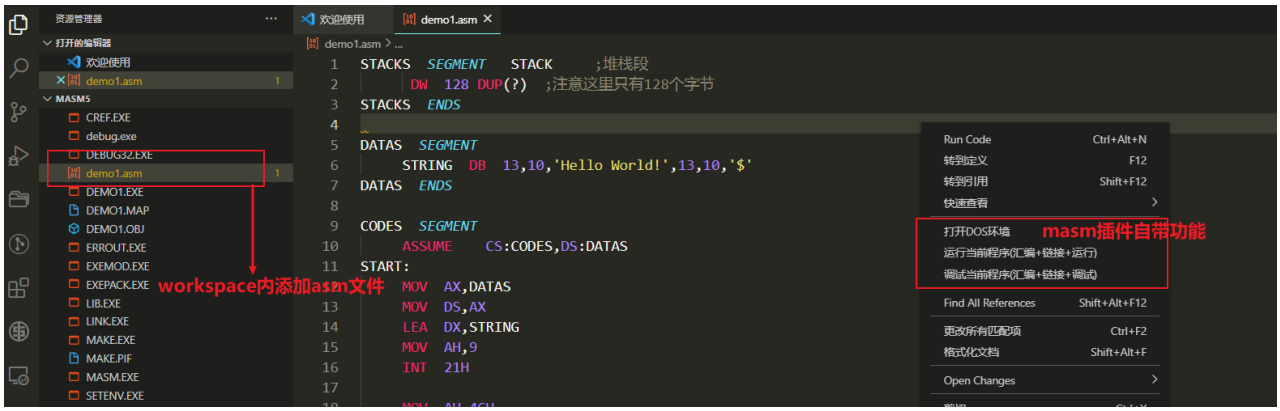
回车（Carriage return），常缩写为 CR，是指将定位设备重置到文本行首的控制字符或过程，用以把一设备的位置重设到一行字的开头。

换行（newline），或称为 Line break 或 end-of-line（EOL），是一种加在文字最后位置的特殊字元，在换行字元的下一个字元将会出现在下一行。

- 最后简单介绍一下 Visual Studio Code 结合 masm 插件进行汇编的“一站式”编写，亦即使用 VSCode 编辑器，通过 dosbox 和 MASM/TASM 运行、调试汇编代码。

Visual Studio Code 是当下流行的文本编辑器，适配很多语言。我平时在编程时也习

惯了 VSCode 的某些操作逻辑和插件应用。因此这里考虑使用插件进行汇编的编写和运行调试。为节省篇幅，这里仅介绍和汇编相关的配置内容，VSCode 自身的下载、主界面配置简单带过，如下图所示：



需要指出的是，asm 文件的位置可以是任意文件夹（工作区），不必和 masm.exe 在一个文件夹，这和 Windows 中的集成开发环境是类似的。

综上，利用 VSCode 结合相应插件也可以实现实验平台类似的功能，并有更好的文本编辑体验（例如代码补全，高亮，快捷键等）。

【补充问题】

1. 堆和栈的区别

接触堆、栈的概念是在 C++的高级语言学习中开始的，我了解到程序在执行时，将内存大方向划分为 4 个区域，也称为内存模型

-代码区：存放函数体的二进制代码，由操作系统进行管理的

-全局区：存放全局变量和静态变量以及常量

-栈区：由编译器自动分配释放，存放函数的参数值,局部变量等

-堆区：由程序员分配和释放,若程序员不释放,程序结束时由操作系统回收

二者区别较大。后续也在数据结构中了解过堆和栈，二者分别作为两种数据结构而存在。

但在学习汇编时，目前并没有发现二者太大的区别，甚至堆的单独概念都没有接触过，《汇编语言》（第三版）中目前也并未对二者作过于明显的区分。

2. 主板中南桥和北桥的作用

北桥芯片（North Bridge）是主板芯片组中起主导作用的最重要的组成部分，也称为主桥（Host Bridge）。一般来说，芯片组的名称就是以北桥芯片的名称来命名的，例如英特尔 845E 芯片组的北桥芯片是 82845E，875P 芯片组的北桥芯片是 82875P 等等。

北桥芯片负责与 CPU 的联系并控制内存、AGP、PCI 数据在北桥内部传输，提供对 CPU 的类型和主频、系统的前端总线频率、内存的类型（SDRAM，DDR SDRAM 以及 RDRAM 等等）和最大容量、ISA/PCI/AGP 插槽、ECC 纠错等支持，整合型芯片组的北桥芯片还集成了显示核心。**北桥芯片就是主板上离 CPU 最近的芯片，这主要是考虑到北桥芯片与处理器之间的通信最密切，为了提高通信性能而缩短传输距离**。

南桥芯片（South Bridge）是主板芯片组的重要组成部分，一般位于主板上离 CPU 插槽较远的下方，PCI 插槽的附近，这种布局是考虑到它所连接的 I/O 总线较多，离处理器远一点有利于布线。相对于北桥芯片来说，其数据处理量并不算大，所以南桥芯片一般都没有覆盖散热片。南桥芯片不与处理器直接相连，而是通过一定的方式（不同厂商各种芯片组有所不同，例如英特尔的英特尔 Hub Architecture 以及 SIS 的 Multi-Threaded “妙渠”）与北桥芯片相连。

南桥芯片负责 I/O 总线之间的通信，如 PCI 总线、USB、LAN、ATA、SATA、音频控制器、键盘控制器、实时时钟控制器、高级电源管理等，这些技术一般相对来说比较稳定，所以不同芯片组中可能南桥芯片是一样的，不同的只是北桥芯片。所以现在主板芯片组中北桥芯片的数量要远远多于南桥芯片。

3. Celeron 300A 的相关介绍，为什么成本低？在功能上有何特点？

赛扬 300A 是超频历史上的经典之作，于 1998 年诞生，以超高的性价比实现超频“平民化”，甚至几乎造就了一条专门为它而生的产业链：主板、转接卡……标志着 DIY 的超频时代正式到来。

赛扬 300A 采用 Socket 370 架构，PPGA 封装，0.25um 工艺和集成全速 L2 256KB cache（PentiumII 为半速 512KB），同频率下与高端的 PentiumII 性能基本相当。采用 Mendocino 核心的它本来是非常低端的，但它的经典之处在于几乎每一颗 cpu 都能够将前端总线从 66MHz 直接提升到 100MHz，相当于主频 450MHz，媲美 500 美元的 Pentium II 450。这样大大降低了成本。

特点如下

1、128KB 全速二级缓存

相比之下，Celeron 266 和 300（原版）没有二级缓存，性能照 Pentium II 差出不少，而 Pentium II 虽然有 512K 二级缓存，却因为搭载在 PCB 板上，频率只有 cpu 主频的一半，而 Celeron 300A 的二级缓存在芯片内部，运行频率和 CPU 主频一致，这导致 Celeron 300A 在某些情况下性能甚至比 Pentium II 还要好！

2、包超 Pentium II 450

Celeron 300A 默认运行在 66Mhz 外频，但当时的 Celeron 体质实在太好，不少批次都可以超频到 100Mhz 外频和 450Mhz 主频，这已经是 Pentium II 的最高频率了。而且由于 100Mhz 外频可以三分频为 33Mhz，这样不会影响 PCI 总线的速度和稳定性。

3、飞线打孔上双 U（DIY 特性）

标准版的 Celeron 300A 是不支持双 CPU 的，但经过魔改打孔飞线之后的 Celeron 300A 可以上双插槽服务器主板，而且速度不输昂贵的双 Pentium II 服务器。

4. **程序改写与调试**（已在实验过程中提及）
5. **CISC 和 RISC 的区别联系是什么**

CISC 和 RISC 的区别为：存储器操作不同、汇编语言程序不同、响应中断不同。RISC 和 CISC 都是设计制造微处理器的典型技术，它们都试图在体系结构、操作运行、软件硬件、编译时间和运行时间等诸多因素中做出某种平衡，以求达到高效的目的。

一、存储器操作不同

1、CISC：CISC 机器的存储器操作指令多，操作直接。

2、RISC：RISC 对存储器操作有限制，使控制简单化。

二、汇编语言程序不同

1、CISC：CISC 汇编语言程序编程相对简单，科学计算及复杂操作的程序设计相对容易，效率较高。

2、RISC：RISC 汇编语言程序一般需要较大的内存空间，实现特殊功能时程序复杂，不易设计。

【实验心得】

这次汇编实验虽然整体内容不难，但是在没有足够了解汇编知识的情况下理解其中的代码对我而言还是有一定的难度的。第一次在 DOS 环境下运行程序，为了搞清楚程序代码的含义，我查阅了很多的资料，同时也初步使用了 Debug 的一些功能辅助理解程序。总之收获了很多知识。

同时，这次实验也让我对以前学习过的一些知识有了巩固和加强，同时有了新的理

解和体会，比如先前曾经在大一下学期接触过的 C/C++的学习使得我接触这门课程不至于一头雾水，能够以类比的方式进行学习。例如在本次样例程序中换行符和回车符的细微区别，以及在整个利用 DosBox 进行黑窗口编译运行的过程，大都可以从 C/C++编程中找到借鉴的地方，例如借鉴之前在 cmd 手动输入 `gcc demo.c -o demo.exe/g++ demo.c -o demo.exe` 实现的 C/C++编译过程，可以很快上手 asm 文件的编译连接运行操作 `masm demo.asm;`。

本次实验还大大加强了我的信息搜集能力和自学能力，使得我对编程更加有兴趣。先前在网上也了解过，对编程技术开始有质的上升的时候就是对底层有充分了解的时候。这也大大增强了我学好这门课的决心。

总之，学习的过程就是一个不断深化所学知识、并结合新学知识进行应用的过程。这次实验是一个很好的学习开端，我对此充满信心。