



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 1.2 掌握 DEBUG 基本  
命令及其功能

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 14 日

# 实验报告

## 【实验目的】

1. 掌握 DEBUG 的基本命令及其功能
2. 掌握 DOSBox 使用 Debug 的方法

## 【实验要求】

1. 通过对某部分代码的修改从而加深对于 debug 所涉及命令的理解
2. 不懂的知识及时百度

## 【实验具体内容】

### 【上半部分】

#### 【预备知识】

##### (1) R 指令

- ① 使用 R 指令显示所有寄存器内容和标志位状态如下图所示：

```
C:\>debug
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0100  NU UP EI PL NZ NA PO NC
073F:0100 0000      ADD     [BX+SI],AL      DS:0000=CD
-:~
```

- ② 使用 R 命令修改指定寄存器的内容如下图所示：

```
C:\>debug
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0100  NU UP EI PL NZ NA PO NC
073F:0100 0000      ADD     [BX+SI],AL      DS:0000=CD
-R AX
AX 0000
:1000
-R
AX=1000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0100  NU UP EI PL NZ NA PO NC
073F:0100 0000      ADD     [BX+SI],AL      DS:0000=CD
-:~
```

##### (2) D 指令

- ① 使用 D 命令查看内存中的内容（上面是寄存器 register，这个是 Memory），给出段地址和偏移地址将会列出 128 个内存单元的内容

```
C:\>debug
-D 1000:0
1000:0000  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
1000:0010  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
1000:0020  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
1000:0030  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
1000:0040  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
1000:0050  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
1000:0060  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
1000:0070  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
```

给出一些说明：00H—70H 是 128 个内存单元，每个内存单元里面有 2 个 16 字符也就是 8 位数据，一个字节

- ② 指定内存的偏移地址：

```
-D 1000:0 1
1000:0000 00 00
-;:
```

### (3) E 指令

#### ① 向内存中写入数据

```
:\>debug
-e 1000:0 01 02 03 04 05 06 07 08 09
-d 1000:0 10
1000:0000 01 02 03 04 05 06 07 08-09 00 00 00 00 00 00 00
1000:0010 00
-;
```

这里发现已经写入了，这里需要注意前面的地址是十六进制的，数量并不能那么普通的一一对应

#### ② 向内存中写入字母

```
-E 1000:0 0x61
^ Error
-E 1000:0 61 62 63
-E 1000:0 61H 62H 63H
^ Error
-d 1000:0 4
1000:0000 61 62 63 04 05
-e 1000:0 'a' 'b' 'c'
-d 1000:0 3
1000:0000 61 62 63 04
-;
```

本来想考虑是否能够直接用 E 指令去写如 ASCII 码使之在内存中能够对应相应的字符，但是内存中只能存放的是二进制所以根本不可能显示出来字符

```
-e 1000:0 61 62 64
-d 1000:0 4
1000:0000 61 62 64 B9 02
abd.
```

但是在后面是能够显示出来 abd 的。问题 1

#### ③ 向内存中写入字符串

```
-e 1000:0 'a+b','d'
-d 1000:0 9
1000:0000 61 2B 62 64 05 06 07 08-09 00
```

字符串就是将每个字符分别存在一个内存单元中罢了

#### ④ 写入汇编指令

```
-E 1000:0 B8 01 00 B9 02 00 01 C8
-U 1000:0 8
1000:0000 B80100 MOV AX,0001
1000:0003 B90200 MOV CX,0002
1000:0006 01C8 ADD AX,CX
1000:0008 0000 ADD [BX+SI],AL
```

### (4) U 指令:

#### ① 反汇编的范围是 32 个字符

```

-U 1000:0
1000:0000 B80100      MOV     AX,0001
1000:0003 B90200      MOV     CX,0002
1000:0006 01C8        ADD     AX,CX
1000:0008 0000        ADD     [BX+SI],AL
1000:000A 0000        ADD     [BX+SI],AL
1000:000C 0000        ADD     [BX+SI],AL
1000:000E 0000        ADD     [BX+SI],AL
1000:0010 0000        ADD     [BX+SI],AL
1000:0012 0000        ADD     [BX+SI],AL
1000:0014 0000        ADD     [BX+SI],AL
1000:0016 0000        ADD     [BX+SI],AL
1000:0018 0000        ADD     [BX+SI],AL
1000:001A 0000        ADD     [BX+SI],AL
1000:001C 0000        ADD     [BX+SI],AL
1000:001E 0000        ADD     [BX+SI],AL
    
```

② 对一定地址范围进行反汇编

```

-E 1000:0 B8 01 00 B9 02 00 01 C8
-U 1000:0 8
1000:0000 B80100      MOV     AX,0001
1000:0003 B90200      MOV     CX,0002
1000:0006 01C8        ADD     AX,CX
1000:0008 0000        ADD     [BX+SI],AL
    
```

(5) A 指令:

一条一条地输入汇编指令，下面在实验中进行展示

【第一个实验】

(1) 实验流程图

(2) 实验源代码（粘贴源代码）:

该实验 1.2 暂无源代码，所有实验均在 dos 中完成

(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 使用 E 命令完成对内存的写入:

```

DS=073F  ES=073F  SS=073F  CS=1000  IP=000D  NU UP EI PL NZ NA PE NC
1000:000D 01C0      ADD     AX,AX
-t
AX=000C  BX=0002  CX=0003  DX=0000  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=1000  IP=000F  NU UP EI PL NZ NA PE NC
1000:000F 0002      ADD     [BP+SI],AL      SS:0000=D0
-e 1000:0 b8 20 4e 05 16 14 bb 00 20 01 d8 89 c3 01 d8 b8 1a 00 bb 26 00 00 d8 0
-u 1000:0
1000:0000 B8204E      MOV     AX,4E20
1000:0003 051614      ADD     AX,1416
1000:0006 B80020      MOV     BX,2000
1000:0009 01D8        ADD     AX,BX
1000:000B 89C3        MOV     BX,AX
1000:000D 01D8        ADD     AX,BX
1000:000F B81A00      MOV     AX,001A
1000:0012 B82600      MOV     BX,0026
1000:0015 0000        ADD     [BX+SI],AL
1000:0017 0000        ADD     [BX+SI],AL
1000:0019 0000        ADD     [BX+SI],AL
1000:001B 0000        ADD     [BX+SI],AL
1000:001D 0000        ADD     [BX+SI],AL
1000:001F 0000        ADD     [BX+SI],AL
    
```

这里是错误  
根源

使用 E 命令进行写入时，发现本来输入指令到 00 d8 但是在使用 u 指令查看时发现并未将该机器码写入内存中，因此查找原因。经分析可以首先排除地址无法达到的问题，因为 IP 地址的寻址能力是 64KB 也就是说写到 IP = 0012 时远远没有超过 IP 所能寻址的最大长度，因此排除无法寻址而产生的问题，后续我将问题聚焦到因为不了解 dos 不可换行输入的原因从而多输入了一个 0，下图为原因的寻找:

```

1000:000F BB1A00      MOV     AX,001A
1000:0012 BB2600      MOV     BX,0026
1000:0015 0000      ADD     [BX+SI],AL
1000:0017 0000      ADD     [BX+SI],AL
1000:0019 0000      ADD     [BX+SI],AL
1000:001B 0000      ADD     [BX+SI],AL
1000:001D 0000      ADD     [BX+SI],AL
1000:001F 0000      ADD     [BX+SI],AL
-e 1000:0  b8 20 4e 05 16 14 bb 00 20 01 d8 89 c3 01 d8 b8 1a 00 bb 26 00 00 d8
-u 1000:0
1000:0000 BB204E      MOV     AX,4E20
1000:0003 051614      ADD     AX,1416
1000:0006 BB0020      MOV     BX,2000
1000:0009 01D8      ADD     AX,BX
1000:000B 89C3      MOV     BX,AX
1000:000D 01D8      ADD     AX,BX
1000:000F BB1A00      MOV     AX,001A
1000:0012 BB2600      MOV     BX,0026
1000:0015 00D8      ADD     AL,BL
1000:0017 0000      ADD     [BX+SI],AL
1000:0019 0000      ADD     [BX+SI],AL
1000:001B 0000      ADD     [BX+SI],AL
1000:001D 0000      ADD     [BX+SI],AL
1000:001F 0000      ADD     [BX+SI],AL
- ▲ ;
    
```

如图再次将指令输入一遍，在即将换行的情况下就将输入停止，可以发现上述未输入的 00 d8 这回出现在了内存之中，可知上述所提到的问题是有效的，**同时可以知道 dos 系统是无法换行输入的**

紧接着上面所写到的内存继续向下，将代码写完

```

1000:000F BB1A00      MOV     AX,001A
1000:0012 BB2600      MOV     BX,0026
1000:0015 00D8      ADD     AL,BL
1000:0017 0000      ADD     [BX+SI],AL
1000:0019 0000      ADD     [BX+SI],AL
1000:001B 0000      ADD     [BX+SI],AL
1000:001D 0000      ADD     [BX+SI],AL
1000:001F 0000      ADD     [BX+SI],AL
-e 1000:0017 00 dc 00 c7 b4 00 00 d8 04 9c
-u 1000:0
1000:0000 BB204E      MOV     AX,4E20
1000:0003 051614      ADD     AX,1416
1000:0006 BB0020      MOV     BX,2000
1000:0009 01D8      ADD     AX,BX
1000:000B 89C3      MOV     BX,AX
1000:000D 01D8      ADD     AX,BX
1000:000F BB1A00      MOV     AX,001A
1000:0012 BB2600      MOV     BX,0026
1000:0015 00D8      ADD     AL,BL
1000:0017 00DC      ADD     AH,BL
1000:0019 00C7      ADD     BH,AL
1000:001B B400      MOV     AH,00
1000:001D 00D8      ADD     AL,BL
1000:001F 049C      ADD     AL,9C
- ;
    
```

U 命令查看发现无误之后，更改 CS 和 IP 的值，使得其指向 1000:0，如下图：

```

-r cs
CS 1000
:
-r ip
IP 000F
:0
    
```

然后使用 t 命令执行即可，按步骤执行如下

```

-t
AX=4E20 BX=0002 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0003 NU UP EI PL NZ NA PE NC
1000:0003 051614 ADD AX,1416
-t
AX=6236 BX=0002 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0006 NU UP EI PL NZ NA PE NC
1000:0006 BB0020 MOV BX,2000
-t
AX=6236 BX=2000 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0009 NU UP EI PL NZ NA PE NC
1000:0009 01D8 ADD AX,BX
-t
AX=8236 BX=2000 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=000B NU UP EI NG NZ NA PE NC
1000:000B 89C3 MOV BX,AX
-;_

AX=8236 BX=8236 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=000D NU UP EI NG NZ NA PE NC
1000:000D 01D8 ADD AX,BX
-t
AX=046C BX=8236 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=000F NU UP EI PL NZ NA PE CY
1000:000F B81A00 MOV AX,001A
-t
AX=001A BX=8236 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0012 NU UP EI PL NZ NA PE CY
1000:0012 BB2600 MOV BX,0026
-t
AX=001A BX=0026 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0015 NU UP EI PL NZ NA PE CY
1000:0015 00D8 ADD AL,BL
-t
AX=0040 BX=0026 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0017 NU UP EI PL NZ AC PO NC
1000:0017 00DC ADD AH,BL
-;_

AX=0040 BX=0026 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0017 NU UP EI PL NZ AC PO NC
1000:0017 00DC ADD AH,BL
-t
AX=2640 BX=0026 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0019 NU UP EI PL NZ NA PO NC
1000:0019 00C7 ADD BH,AL
-t
AX=2640 BX=4026 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=001B NU UP EI PL NZ NA PO NC
1000:001B B400 MOV AH,00
-t
AX=0040 BX=4026 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=001D NU UP EI PL NZ NA PO NC
1000:001D 00D8 ADD AL,BL
-t
AX=0066 BX=4026 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=001F NU UP EI PL NZ NA PE NC
1000:001F 049C ADD AL,9C
-;_
    
```

实验要求观察 IP 的值，我们可以看见 IP 的值就是随着指令向下进行而发生改变的，也就是说，执行完一条指令之后 IP 的值会和上述使用了 U 指令所得到底的列表中 IP 的值是一致的。

使用 R 命令查看运行结果

```

-r
AX=0002 BX=4026 CX=0003 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0021  NU UP EI PL NZ AC PO CY
1000:0021 0000      ADD     [BX+SI],AL      DS:4026=00
-;
    
```

② 使用 A 命令完成相应的内容:

```

-a 1000:0
1000:0000 MOV AX,4E20
1000:0003 ADD AX,1416
1000:0006 MOV BX,2000
1000:0009 ADD AX,BX
1000:000B MOV BX,AX
1000:000D ADD AX,BX
1000:000F MOV AX,001A
1000:0012 MOV BX,0026
1000:0015 ADD AL,B1
1000:0015 ^ Error
1000:0015 ADD AL,BL
1000:0017 ADD AH,BL
1000:0019 ADD BH,AL
1000:001B AH,0
1000:001B ^ Error
1000:001B MOV AH,0
1000:001D ADD AL,BL
1000:001F ADD AL,9C
1000:0021
    
```

A 指令就是将其一条一条的输入，所执行的内容都是和上面相同的，下面对所输入的汇编指令进行注释

```

//把4E20H写入AX寄存器
MOV AX,4E20H
//将AX寄存器中的数据加上1416H然后放入AX中
ADD AX,1416H
//将2000H写入BX中
MOV BX,2000H
//将AX加上BX所得数据加入AX中
ADD AX,BX
//把AX移入BX中
MOV BX,AX
//把AX和BX相加所得数据存入AX中
ADD AX,BX
//下面相同很简单就不写了
MOV AX,001AH
MOV BX,0026H
ADD AL,BL
ADD AH,BL
ADD BH,BL
MOV AH,0
ADD AL,BL
ADD AL,9CH
    
```

## 【第二个实验】熟悉 JMP 指令

(1) 实验流程图:

(2) 实验源代码(粘贴源代码):

(3) 实验代码、过程、相应结果(截图)并对实验进行说明和分析:

使用 A 指令写入所给代码

```

-A 2000:0
2000:0000 MOV AX,1
2000:0003 ADD AX,AX
2000:0005 JMP 2000:0003
2000:0007
-RCS
CS 2000
:2000
-RIP
IP 0005
0
    
```

进行执行指令, JMP 2000:0003 指令跳转到 2000:0003 地址,即所对应的汇编指令是 ADD AX,AX 实现累加,变相的实现次方功能

```

-T
AX=0001 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0003  NU UP EI PL NZ NA PO NC
2000:0003 01C0          ADD     AX,AX
-T
AX=0002 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0005  NU UP EI PL NZ NA PO NC
2000:0005 EBFC          JMP     0003
-T
AX=0002 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0003  NU UP EI PL NZ NA PO NC
2000:0003 01C0          ADD     AX,AX
-T
AX=0004 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0005  NU UP EI PL NZ NA PO NC
2000:0005 EBFC          JMP     0003
-T
    
```

执行 T 指令 17 次后得到 2 的 8 次方

```

AX=0040 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0005  NU UP EI PL NZ NA PO NC
2000:0005 EBFC          JMP     0003
-T
AX=0040 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0003  NU UP EI PL NZ NA PO NC
2000:0003 01C0          ADD     AX,AX
-T
AX=0080 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0005  NU UP EI PL NZ NA PO NC
2000:0005 EBFC          JMP     0003
-T
AX=0080 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0003  NU UP EI PL NZ NA PO NC
2000:0003 01C0          ADD     AX,AX
-T
AX=0100 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0005  NU UP EI PL NZ NA PO NC
2000:0005 EBFC          JMP     0003
-T
    
```

### 【第三个实验】

找到 DOSBOX 虚拟环境中所示的 CPU 生产日期



```

-D FFFF:0
FFFF:0000 EA C0 12 00 F0 30 31 2F-30 31 2F 39 32 00 FC 55 ....01/01/92.U
FFFF:0010 60 10 00 F0 BB 13 A3 01-08 00 70 00 B1 13 A3 01 .....p.....
FFFF:0020 08 00 70 00 60 10 00 F0-60 10 00 F0 60 10 00 F0 ..p.....
FFFF:0030 A5 FE 00 F0 87 E9 00 F0-55 FF 00 F0 60 10 00 F0 .....U.....
FFFF:0040 60 10 00 F0 60 10 00 F0-80 10 00 F0 60 10 00 F0 .....e...
FFFF:0050 00 13 00 F0 00 11 00 F0-20 11 00 F0 40 11 00 F0 .....e...
FFFF:0060 A0 11 00 F0 C0 11 00 F0-E0 11 00 F0 20 12 00 F0 .....e...
FFFF:0070 C0 12 00 F0 C0 12 00 F0-40 12 00 F0 60 10 00 F0 .....e...
-D FFFF:0 F
FFFF:0000 EA C0 12 00 F0 30 31 2F-30 31 2F 39 32 00 FC 55 ....01/01/92.U
- ;
    
```

之所以是错误的是因为该 8086CPU 是由现在的 CPU 映射出来的，某种程度上也是虚拟的所以生产日期是不对的，而且 ROM 存储器中的生产日期是无法修改的

## 【下半部分】

### 【预备知识】

#### (1) 关于 D 指令的说明：

```

C:\>debug
-r ds
DS 073F
:1000
-d ds:0
1000:0000 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0030 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-r ds
DS 1000
:1000
-d ds:10 18
1000:0010 00 00 00 00 00 00 00 00-00 .....
- ;
    
```

#### ① 直接使用 ds 段寄存器：IP 的方式得到

#### ② 查看指令代码和栈段

```

1000:0070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-r ds
DS 1000
:1000
-d ds:10 18
1000:0010 00 00 00 00 00 00 00 00-00 .....
-d cs:0
073F:0000 CD 20 3E A7 00 EA FD FF-AD DE 4F 03 A3 01 BA 03 .>.....0.....
073F:0010 A3 01 17 03 A3 01 92 01-01 01 01 00 02 FF FF FF .....
073F:0020 FF FF FF FF FF FF FF FF-FF FF FF FF 00 00 00 00 .....
073F:0030 00 00 14 00 18 00 3F 07-FF FF FF FF 00 00 00 00 .....?.....
073F:0040 05 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
073F:0050 CD 21 CB 00 00 00 00 00-00 00 00 00 00 20 20 20 .!.....
073F:0060 20 20 20 20 20 20 20 20-00 00 00 00 00 20 20 20 .....
073F:0070 20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 .....
-d ss:0
073F:0000 CD 20 3E A7 00 EA FD FF-AD DE 4F 03 A3 01 BA 03 .>.....0.....
073F:0010 A3 01 17 03 A3 01 92 01-01 01 01 00 02 FF FF FF .....
073F:0020 FF FF FF FF FF FF FF FF-FF FF FF FF 00 00 00 00 .....
073F:0030 00 00 14 00 18 00 3F 07-FF FF FF FF 00 00 00 00 .....?.....
073F:0040 05 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
073F:0050 CD 21 CB 00 00 00 00 00-00 00 00 00 00 20 20 20 .!.....
073F:0060 20 20 20 20 20 20 20 20-00 00 00 00 00 20 20 20 .....
073F:0070 20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 .....
- ;
    
```

#### (2) 同上其他指令亦可以使用段寄存器

### 【第一个实验】

将指令写入内存中，就像老师说的一样，这是在练打字而已

```
-A DS:0
1000:0000 mov ax,ffff
1000:0003 ds,ax
^ Error
1000:0003 mov ds,ax
1000:0005 mov ss,ax
1000:0007 mov sp,0100
1000:000A mov ax,[0]
1000:000D add ax,[2]
1000:0011 mov bx,[4]
1000:0015 add bx,[6]
1000:0019 push ax
1000:001A push bx
1000:001B pop ax
1000:001C pop bx
1000:001D push [4]
1000:0021 push [6] ;
```

将 CS IP 指向相应的指令位置

```
-rCS
CS 073F
:1000
-rIP
IP 0100
:0
- ;
```

开始执行指令

```
-t
AX=FFFF BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=1000 IP=0003  NU UP EI PL NZ NA PO NC
1000:0003 8ED8          MOV     DS,AX
-t
AX=FFFF BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=073F CS=1000 IP=0005  NU UP EI PL NZ NA PO NC
1000:0005 8ED0          MOV     SS,AX
-t
AX=FFFF BX=0000 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=FFFF CS=1000 IP=000A  NU UP EI PL NZ NA PO NC
1000:000A A10000        MOV     AX,[0000]          DS:0000=C0EA
-t
AX=C0EA BX=0000 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=FFFF CS=1000 IP=000D  NU UP EI PL NZ NA PO NC
1000:000D 03060200     ADD     AX,[0002]          DS:0002=0012
- ;
```

```
-t
AX=C0FC BX=0000 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=FFFF CS=1000 IP=0011  NU UP EI NG NZ NA PE NC
1000:0011 8B1E0400     MOV     BX,[0004]          DS:0004=30F0
-t
AX=C0FC BX=30F0 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=FFFF CS=1000 IP=0015  NU UP EI NG NZ NA PE NC
1000:0015 031E0600     ADD     BX,[0006]          DS:0006=2F31
-t
AX=C0FC BX=6021 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=FFFF CS=1000 IP=0019  NU UP EI PL NZ NA PE NC
1000:0019 50          PUSH    AX
-t
AX=C0FC BX=6021 CX=0000 DX=0000 SP=00FE BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=FFFF CS=1000 IP=001A  NU UP EI PL NZ NA PE NC
1000:001A 53          PUSH    BX
- ;
```

```

-t
AX=C0FC BX=6021 CX=0000 DX=0000 SP=00FC BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=FFFF CS=1000 IP=001B  NU UP EI PL NZ NA PE NC
1000:001B 58      POP      AX
-t
AX=6021 BX=6021 CX=0000 DX=0000 SP=00FE BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=FFFF CS=1000 IP=001C  NU UP EI PL NZ NA PE NC
1000:001C 5B      POP      BX
-t
AX=6021 BX=C0FC CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=FFFF CS=1000 IP=001D  NU UP EI PL NZ NA PE NC
1000:001D FF360400  PUSH    [0004]      DS:0004=30F0
-t
AX=6021 BX=C0FC CX=0000 DX=0000 SP=00FE BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=FFFF CS=1000 IP=0021  NU UP EI PL NZ NA PE NC
1000:0021 FF360600  PUSH    [0006]      DS:0006=2F31
- ;
    
```

```

AX=6021 BX=C0FC CX=0000 DX=0000 SP=00FC BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=FFFF CS=1000 IP=0025  NU UP EI PL NZ NA PE NC
1000:0025 0000      ADD     [BX+SI],AL      DS:C0FC=00
- ;
    
```

相应的填空如下：

0

DS=0B39 ES=0  
0B39:0108 B8?  
-d 2000:0 f  
2000:0000 0

; ax= C0EA  
 ; ax= C0FC  
 ; bx= 30F0  
 ; bx= 6021  
  
 ; sp= 0102 ; 修改的内存单元的地址是 100f1100f2 内容为 C0FC  
 ; sp= 0104 ; 修改的内存单元的地址是 100f3100f4 内容为 6021  
 ; sp= 0102 ; ax= 6021  
 ; sp= 0100 ; bx= C0FC  
  
 ; sp= 0102 ; 修改的内存单元的地址是 100f1100f2 内容为 30F0  
 ; sp= 0104 ; 修改的内存单元的地址是 100f3100f4 内容为 6021  
 2F31

3.19 中的实验过程，然后分析：为什么 2000:0~2000:f 中的内容

验才能发现其中的规律 如果你在这里就不该问这个问题了

### 【第二个实验】

输入指令之后发现 add ds, ax 出现错误，因为 sub 和 add 不能对段寄存器进行操作

```

-a ds:0
1000:0000 mov ax,1000
1000:0003 mov ds,ax
1000:0005 mov ds,[0]
1000:0009 add ds,ax
1000:0009 ^ Error
    
```

下面继续执行可以发现，`mov ds, [0]`实际上是把该 1000:0 地址处存的机器指令给了 DS 寄存器

```

-t
AX=1000 BX=C0FC CX=0000 DX=0000 SP=00FC BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=FFFF CS=1000 IP=0003 NU UP EI PL NZ NA PE NC
1000:0003 8ED8          MOV     DS,AX
-t
AX=1000 BX=C0FC CX=0000 DX=0000 SP=00FC BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=FFFF CS=1000 IP=0005 NU UP EI PL NZ NA PE NC
1000:0005 8E1E0000     MOV     DS,[0000]          DS:0000=00B8
-t
AX=1000 BX=C0FC CX=0000 DX=0000 SP=00FC BP=0000 SI=0000 DI=0000
DS:00B8 ES=073F SS=FFFF CS=1000 IP=0009 NU UP EI PL NZ NA PE NC
1000:0009 01A10000     ADD     [BX+DI+0000],SP      DS:C0FC=0000
-;
    
```

### 【第三个实验】

值发生改变的原因，就像问题 2 中提到的那样栈是程序执行中必备的，T 指令为中断指令，在执行的过程中会把当前状态的 CS 和 IP 的值入栈，因为栈的数据不会消失只会被覆盖所以就仍保留在内存中

### 【实验心得】

### 【实验问题】

#### 问题 1:

##### 1.1 汇编指令中如何识别输入的是十进制还是十六进制

Debug 就是从 CPU 的视角去看问题的，也就是说 Debug 中是不存在 10 进制，所有的数都会被理解为 16 进制，masm 中是要区别 H 和 D 之间的区别的

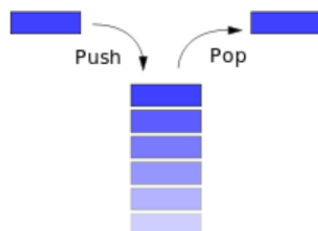
##### 1.2 汇编指令中如何区别 ASCII 码和数之间的不同之处

反正在内存中存储的都是 0 和 1 的二进制组合，所以是 ascii 码还是就只是 16 进制的数，是看变量的定义到底是什么类型的

**问题 2:**既然说栈是一种以特殊方式访问的内存空间，那么为什么还要设置栈，这种 LIFO 的方式有什么特殊的作用么？

## 栈的作用

计算机里面的栈其实有着举足轻重的作用。大学刚学c语言的时候，教的是堆栈，传达的是一种后入先出的算法思想。但其实我们知道，堆和栈是两个截然不同的东西。而这里面说到的栈，则是更融入到计算机系统里面，CPU结构的一部分。



一个函数设计里面，有2个问题：

1.是参数传递的问题。传递参数的目的，是为了代码可以重用，让一种方法可以应用到更多的场合，而不需要为N种情况写N套类似的代码。那用什么方法来做参数的传递，可以选择：

a.为了速度快，使用cpu的寄存器传递参数。这会碰到一个问题，cpu寄存器的数量是有限的，当函数内再想调用子函数的时候，再使用原有的cpu寄存器就会冲突了。想利用寄存器传参，就必须在调用子函数前把寄存器存储起来，然后当函数退出的时候再恢复。

b.利用某些ram的区域来传递参数。这和上面a的情况几乎一样，当函数嵌套调用的时候，还是会出现冲突，依然面临要把原本数据保存到其他地方，再调用嵌套函数。并且保存到哪里，也面临困难，无论临时存储到哪里，都会有上面传递参数一样的困境。

2.函数里面必然要使用到局部变量，而不能总是用全局变量。则局部变量存储到哪里合适，即不能让函数嵌套的时候有冲突，又要注重效率。

以上问题的解决办法，都可以利用栈的结构体来解决，寄存器传参的冲突，可以把寄存器的值临时压入栈里面，非寄存器传参也可以压入到栈里面，局部变量的使用也可以利用栈里面的内存空间，只需要移动下栈指针，腾出局部变量占用的空间。最后利用栈指针的偏移来完成存取。于是函数的这些参数和变量的存储演变成记住一个栈指针的地址，每次函数被调用的时候，都配套一个栈指针地址，即使循环嵌套调用函数，只要对应函数栈指针是不同的，也不会出现冲突。利用栈，当函数不断调用的时候，不断的有参数局部变量入栈，栈里面会形成一个函数栈帧的结构，一个栈帧结构归属于一次函数的调用。栈的空间也是有限的，如果不限制的使用，就会出现典型的栈溢出的问题。有了栈帧的框架在，我们在分析问题的时候，如果能获取到当时的栈的内容，则有机会调查当时可能出现的问题。



由此可以看出，一个任务状态可以利用如下信息来表征：1.main函数体代码。2.main的栈指针位置（即存储了局部变量等信息）。3.当前cpu寄存器的信息。假如我们可以保存在这些信息，则完全可以强制让cpu去做别的事情，只是将来想继续执行main任务的时候，把上面的信息恢复就可以。有了这样的先决条件，多任务就有了存在的基础。也可以看出栈存在的意义所在。在多任务模式下，当CPU认为有必要切换到别的任务上运行时，只需要保存好当前任务的状态，即上面说的三个内容。恢复另一个任务的状态，然后跳转到上次运行的位置，就可以恢复继续运行。可见每个任务都有自己的独立的栈空间。正是有了独立的栈空间，为了代码重用，不同的任务甚至可以混用任务的函数体本身，例如可以一个main函数有2个任务实例。有不同的栈空间，这完全可以实现。



### 【心得体会】

每次进行汇编的学习，我都感觉对我手中的电脑更加了解了，这种感觉让我感觉很爽，同时我也对前人所设计的这个精妙的机器表示惊叹，尤其是在查找栈的作用的时候，我就在想那些人是怎么想出来用如此巧妙的一个对内存的特殊存储方式来进行对多任务处理的，真是太神奇了。崇新是可以选择微处理器还是高频的，说实话微处理器的实验报告压力确实很大，但是我却很喜欢这种了解电脑工作原理的感觉，我们现在是信息时代，作为信息学院的学生，电脑就是我们的伙伴，只有更加了解它才能走得更远，所以我选择痛并快乐着。加油！奥利给！