

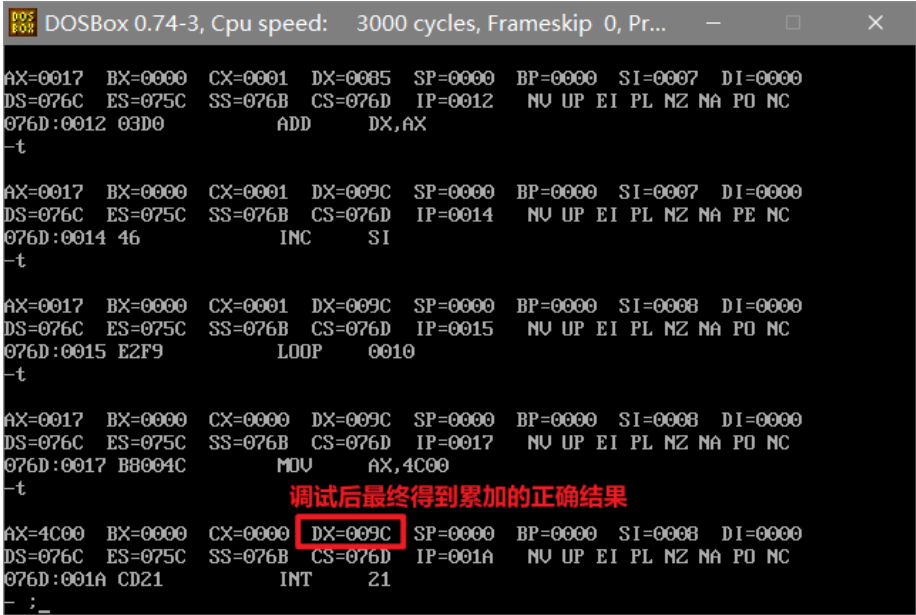
《汇编语言》实验报告 03

班级	2022 秋	实验日期	2022.10.14	实验成绩	
姓名	黄勛	学号	22920212204392		
实验名称	汇编语言第三次实验				
实验目的、要求	<div>1. 熟练使用 Debug（更加推荐采用程序形式），理解数据在内存中的存放，并理解并练习各种寻址方式。</div> <div>2. 学习汇编语言的基本指令，学会阅读汇编代码并动手尝试编写一些算法</div> <div>3. 学习虚拟机的使用方法，了解 BIOS 的基础运行方式，尝试破解 BIOS 密码</div>				
实验内容、步骤及结果	<div>(一)在数据段中依次存入 10H,11H,12H,13H,14H,15H,16H,17H，将其相加，并将结果存入 DX 寄存器。</div> <div>1) 在 code（lab3.asm）中编写代码</div> <div>① 首先在数据段中依次存入 10H,11H,12H,13H,14H,15H,16H,17H</div> <div><pre>1 data SEGMENT 2 DB 10H,11H,12H,13H,14H,15H,16H,17H 3 data ENDS</pre></div> <div>② 用 SI 源变址寄存器存放相对于 DS 段之源变址指针，初始化指向第一个数据（10H），CX 存放循环次数（8），DX,AH 初始化为 0（AH 是 ax 的高 8 位，而 AL 是 ax 的低 8 位）</div> <div><pre>start: MOV AX, DATA MOV DS, AX MOV CX, 8 MOV SI, 0 MOV DX, 0 MOV AH, 0</pre></div> <div>③ 用 AL 接收字节型数据（AH 保持为 0），然后将 DX 和 AX 相加，SI 指向下一个数据，循环执行，最后 DX 即为累加的结果</div> <div><pre>S: MOV AL,[SI] ADD DX, AX INC SI LOOP S</pre></div>				

④ 最后编写退出程序指令

```
MOV AX,4C00H
INT 21H
code ENDS
END start
```

2) （编译连接过程省略）在 debug 中实际调试程序



得到答案 DX = 009C

(二)练习使用 debug 命令破解 BIOS 密码，写出自己对破解密码的理解。
1) 首先为了保证实验的安全性，我在网络上查找到 win7 的 32 系统可以直接在命令行运行 debug，于是搜索下载了相关的镜像并在 vmware 中安装了虚拟机系统

[注]笔者使用 32 位旗舰版镜像: <http://www.msdnwogaosuni.com/win7/13765.html>

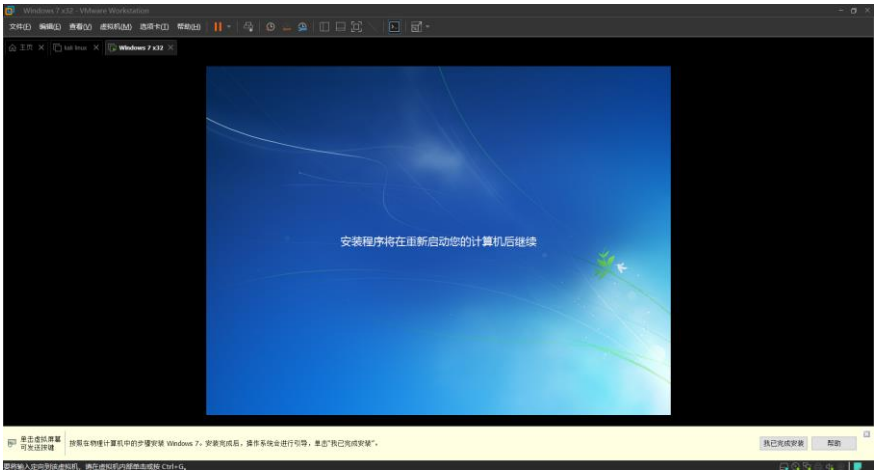


图 2-1 安装系统镜像的截图



图2-2 安装 win7x32 完毕 (好古早)

2) 启动 win7 系统后，再选择顶部菜单启动按钮，向下的三角形--打开电源时进入固件，即可自动启动虚拟机并进入 VMware 虚拟机 BIOS 设置界面。

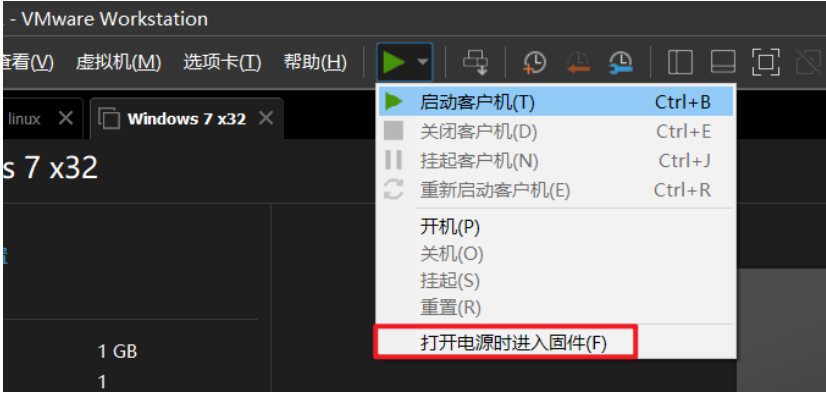


图2-3 vmware 进入 BIOS 的操作图示

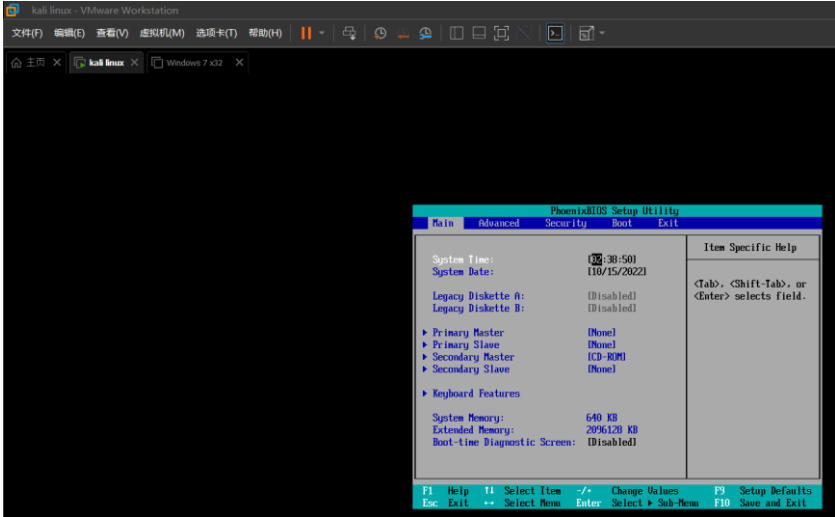


图2-4 BIOS 界面

3) 用键盘操作在 BIOS 中的 Security→Set Password 设置管理员密码与普通用户密码

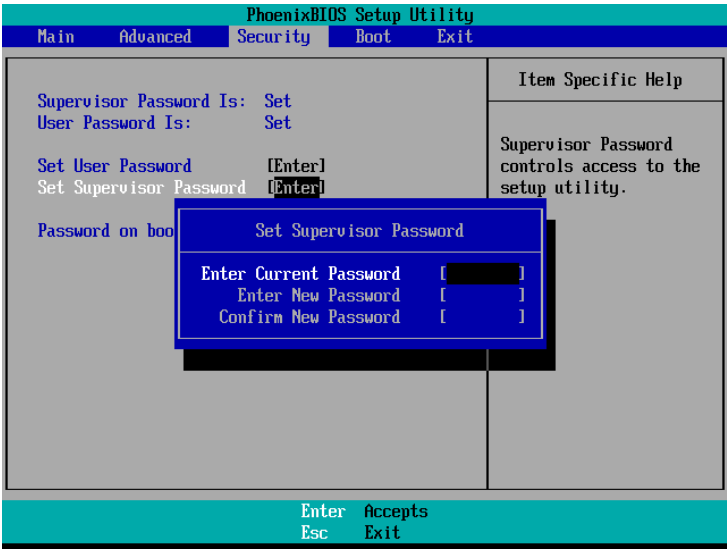


图 2-5 BIOS 设置密码的界面

4) 重启系统并再次进入 BIOS 界面，提示要输入密码（此时假设不知道密码，则无法进入 BIOS 系统）

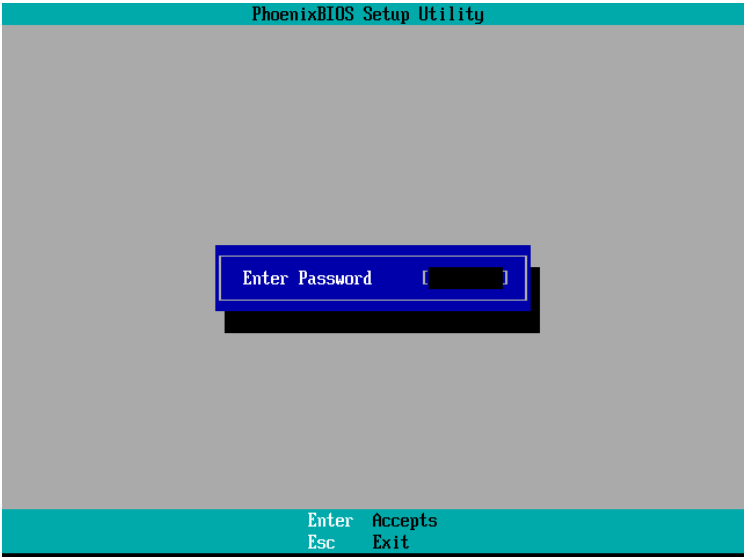


图 2-6 进入 BIOS 提示输入密码

5) 经过查阅资料，此时需要进入纯 DOS 状态利用 debug 代码破解 BIOS 代码，在此处我利用 maxdos 工具箱软件在系统引导中添加进入纯 dos 系统环境的选项

[注]笔者使用工具地址 <https://www.xitongzhijia.net/soft/94105.html?1643103248>



图 2-7 MaxDOS 控制台界面



图 2-8 利用 vmtools 将文件传入虚拟机

6) 此时需要进入纯 DOS 系统



图 2-9 系统引导界面-进入 MaxDOS



图 2-10 选择进入 MaxDOS 模式



图 2-11 现在就可以选择进入纯 DOS 环境了



图2-12 纯正DOS 界面

7) 此时启动 debug 并运行破解 BIOS 密码指令
输入以下破解程序：

```
debug
-o 70 10
-o 71 ff
-q
```

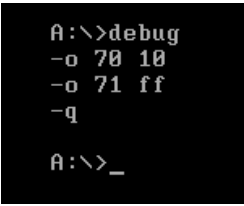


图2-13 输入指令

8) 重启，进入 BIOS 界面，发现密码已被破解，可以直接进入并修改 BIOS 配置！

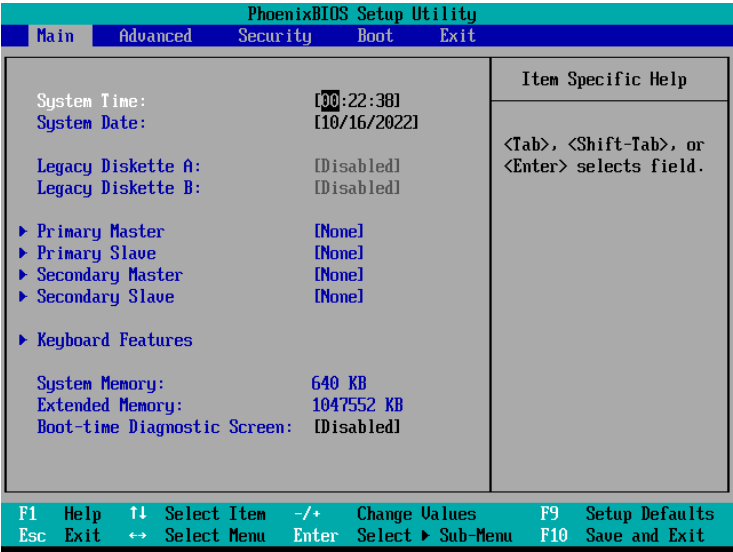


图2-14 破解成功画面

(三) (选做题) 在长度为 8 的字节数组 (无符号数) 中, 查找大于 42H 的无符号数的个数, 存放在字节单元 up 中; 等于 42H 的无符号数的个数, 存放在字节单元 equa 中; 小于 42H 的无符号数的个数, 存放在字节单元 down 中。程序显示 up equa down 的值。

八个数: 31H,21H,42H,52H,87H,23H,98H,01H

① 在数据段中存入八个数, 为 up,equa,down 分配内存并初始化为 0

```

3  data1 DB 31H,21H,42H,52H,87H,23H,98H,01H
4  count DB 8H
5  search DB 42H
6  up DB 0
7  down DB 0
8  equa DB 0
9  upString db 0ah,0dh,'UP Count:$'
10 downString db 0ah,0dh,'Down Count:$'
11 equaString db 0ah,0dh,'Equa Count:$'

```

② 初始化, 把数组数据和 42H 进行比较, 如果相等就转移到 EQUAL_NUMBER, 如果小于 42H 就转移到 DOWN_NUMBER, 如果大于 42H 就转移到 UP_NUMBER, 并利用 count 变量判断是否已经计数完毕 (如果 count 为 0, 说明比较完转移到 RESULT, 否则转移到 COMPARE)

```

COMPARE:
    MOV AH,search
    MOV BL,count
    MOV AL,[data1][BX]
    CMP AL,AH
    JZ EQUAL_NUMBER
    JS DOWN_NUMBER
    JNS UP_NUMBER

```

对每次的比较结果将每次的计数结果变量对应加一

```

EQUAL_NUMBER:
    INC equa
    DEC count
    MOV AL,count
    MOV AH,01H
    CMP AL,AH
    JS RESULT
    JZ COMPARE
    JNS COMPARE

```



```
UP_NUMBER:
```

```
    INC up
    DEC count
    MOV AL,count
    MOV AH,01H
    CMP AL,AH
    JS RESULT
    JZ COMPARE
    JNS COMPARE
```

```
DOWN_NUMBER:
```

```
    INC down
    DEC count
    MOV AL,count
    MOV AH,01H
    CMP AL,AH
    JS RESULT
    JZ COMPARE
    JNS COMPARE
```

③ 输出结果，显示字符串 upString，调用功能号 02H，显示字符

```
RESULT:
```

```
    MOV DX,OFFSET upString
    MOV AH,09H
    INT 21H
    MOV DL,up
    ADD DL,30h
    MOV AH,02h
    INT 21h
    MOV DX,OFFSET downString
    MOV AH,09H
    INT 21H
    MOV DL,down
    ADD DL,30h
    MOV AH,02h
    INT 21h
    MOV DX,OFFSET equaString
    MOV AH,09H
    INT 21H
    MOV DL,equa
    ADD DL,30h
    MOV AH,02h
    INT 21h
```

④ 退出程序

```
    MOV AX,4C00H
    INT 21H
END START
```

⑤ 连接编译运行查看结果，得到 up 为 3，equa 为 1，down 为 4，结果正确！

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pr...
076D:0017 B8004C      MOV     AX,4C00
-t

AX=4C00 BX=0000 CX=0000 DX=009C SP=0000 BP=0000 SI=0008 DI=0000
DS=076C ES=075C SS=076B CS=076D IP=001A NU UP EI PL NZ NA PO NC
076D:001A CD21      INT     21
-q

C:\>link lab3_2.obj

Microsoft (R) Segmented Executable Linker Version 5.31.009 Jul 13 1992
Copyright (C) Microsoft Corp 1984-1992. All rights reserved.

Run File [lab3_2.exe]:
List File [nul.map]:
Libraries [.lib]:
Definitions File [nul.def]:
LINK : warning L4021: no stack segment

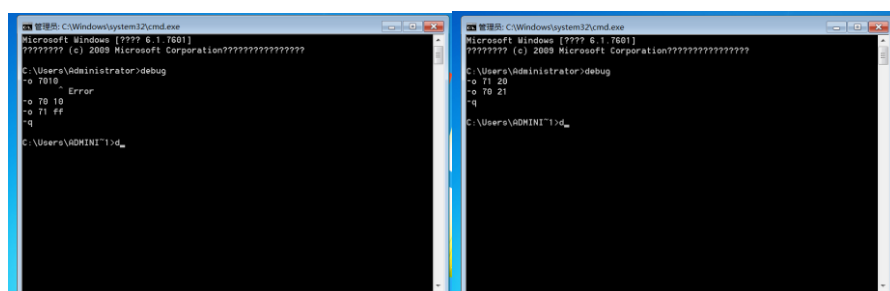
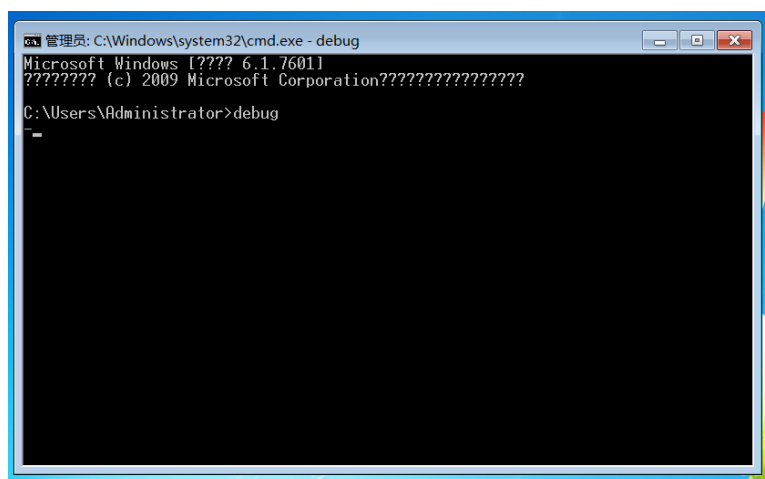
C:\>lab3_2.exe

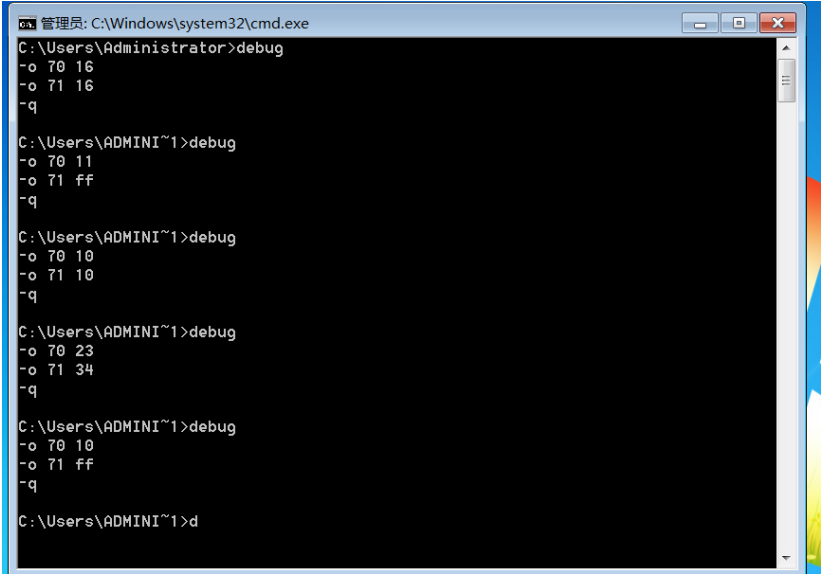
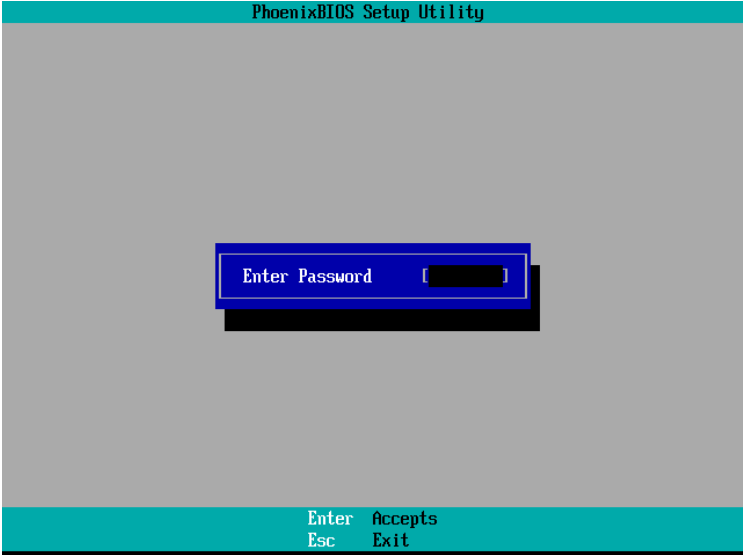
UP Count:3
Down Count:4
Equa Count:1
C:\>
```

*(四)遇到的问题

1) 在尝试破解密码的时候, 由于对教程的错误理解在 win7 的 cmd 窗口中执行 dos 指令, 最后没有效果。(花了很长时间)

错误操作如下图:



	<div data-bbox="371 197 1195 768"></div> <div data-bbox="371 786 1117 1339"></div> <p>解决办法：在实际实验过程中，进入纯 dos 系统操作则成功。</p> <p>2) 汇编语言中,为什么 SI 和 DI 不能同时使用汇编 解决：见总结。</p> <p>3) 破解 BIOS 密码的原理？ 解决：见总结。</p>
总结	<p>这一次实验我对汇编语言指令和 BIOS 有了更深的理解，并且这一次实验的实践操作颇丰，在练习编码和输入指令的过程中我对每一个指令的用途和用法有了更深的认识，通过一步步地解决问题，我的实践能力提高了，这让我受益匪浅；具体遇到问题的解决方案我在后文做了更详细的总结，在此就不多赘述；在未来我还要探索汇编语言的更多应用方面，寻找更多问题，并在发现问题的过程中继续提高我对汇编语言的掌握能力，这是一次颇有意义的实验！</p>

问题总结:

一、破解 BIOS 密码的原理与理解?

出现了-o 70 -o71, 因为 CMOS 中数据访问是通过 70 和 71 这两个 I/O 端口来实现的。端口 70H 是一个字节的地址端口, 用来设置 CMOS 中数据的地址, 而端口 71H 则是用来读写端口 70H 设置 CMOS 地址中的数据单元内容。70 写的是地址, 71 写的是数据, o 是 out, 这两条指令会把 20 写到 RTC 地址为 21 的寄存器里面。RTC (Run Time Cloc)位于芯片, 就是主板上的时钟, 用一小块钮扣电池驱动, 这块电池同时还一直在刷新一个 256 字节的小存储器, 里面存放的就是 CMOS 里的数据。这里往一个地址随意写一个值, 会导致校验错误, BIOS 在 boot 的时候遇到校验错误会 load default, 就是会恢复默认值, 这样密码就没了。同样的, 拿掉 CMOS 电池并短接也可以导致 RTC 存储器里的数据丢失, 校验基本上就错了, 也会恢复默认值。

二、汇编语言中,为什么 SI 和 DI 不能同时使用汇编

这两个寄存器的意思, SI 源变址寄存器, DI 目地变址寄存器, 既然是变址寄存器, 那么他们肯定是在某个地址的基础上进行偏移变化, 由此我们就得出了需要基址寄存器。

要是把这两个寄存器同时使用, 那地址变化的基址都没有, 该怎么变化呢? 在谁的基础上变化 (也就是地址偏移)?

对于汇编中的规定, 其实有时并不需要书上详细的介绍, 我们都应该可以从中推导出这些规则, 书上的那些介绍只是用来验证我们的推测的。或是对我们所掌握的知识的进行检测, 用来说明我们所掌握的是对的!

三、寄存器的总结

1:数据寄存器,一般称之为通用寄存器组

8086 有 8 个 8 位数据寄存器,

这些 8 位寄存器可分别组成 16 位寄存器:

AH&AL=AX: 累加寄存器, 常用于运算;

BH&BL=BX: 基址寄存器, 常用于地址索引;

CH&CL=CX: 计数寄存器, 常用于计数;

DH&DL=DX: 数据寄存器, 常用于数据传递。

2:地址寄存器/段地址寄存器

为了运用所有的内存空间, 8086 设定了四个段寄存器, 专门用来保存段地址:

CS (Code Segment): 代码段寄存器;

DS (Data Segment): 数据段寄存器;

SS (Stack Segment): 堆栈段寄存器;

ES (Extra Segment): 附加段寄存器。

3: 特殊功能的寄存器

IP (Instruction Pointer): 指令指针寄存器, 与 CS 配合使用, 可跟踪程序的执行过程;

SP (Stack Pointer): 堆栈指针, 与 SS 配合使用, 可指向目前的堆栈位置。

BP (Base Pointer): 基址指针寄存器, 可用作 SS 的一个相对基址位置;

SI (Source Index): 源变址寄存器可用来存放相对于 DS 段之源变址指针;

DI (Destination Index): 目的变址寄存器, 可用来存放相对于 ES 段之目的变址指针。