Ares ME

个人实验过程中遇到的各种小问题的小笔记，可以小参考一下。
1、先说说 MOVSB（MOVe String Byte）：即字符串传送指令，这条指令按字节传送数据。通过 SI 和 DI 这两个寄存器控制字符串的源地址和目标地址，比如 DS:SI 这段地址的 N 个字节复制到 ES:DI 指向的地址，复制后 DS:SI 的内容保持不变。

而 REP（REPeat）指令就是“重复”的意思，术语叫做“重复前缀指令”，因为既然是传递字符串，则不可能一个字（节）一个字（节）地传送，所以需要有一个寄存器来控制串长度。这个寄存器就是 CX，指令每次执行前都会判断 CX 的值是否为 0（为 0 结束重复，不为 0，CX 的值减 1），以此来设定重复执行的次数。因此设置好 CX 的值之后就可以用 REP MOVSB 了。

CLD（CLear Direction flag）则是清方向标志位，也就是使 DF 的值为 0，在执行串操作时，使地址按递增的方式变化，这样便于调整相关段的的当前指针。这条指令与 STD（SeT Direction flag）的执行结果相反，即置 DF 的值为 1。

；example：把当前数据段中偏移 1000H 开始
；的 100 个字节数据传送到偏移 2000H 开始的
；单元中

```
cld ；指针增值
push DS ；当前数据段，因此压入栈中保存
pop ES ；使 ES=DS
mov SI，1000H ；源串指针初值
mov DI，2000H ；目的串指针初值
mov CX，100 ；循环次数
Next：
lodsb ；取一个字节
stosb ；存一个字节
loop Next ；循环 CX 次
```

；以下程序片段与上面的等价：

```
cld ；地址指针增值
……
mov CX，100 ；循环 CX 次
Next：
movsb ；每次传送一字节数据
loop Next
```

；或者用更简单的写法：
cld
……
mov CX,100
rep movsb

2、repe cmpsb 指令：
repe 是一个串操作前缀，它重复串操作指令，每重复一次 ECX 的值就减一
一直到 CX 为 0 或 ZF 为 0 时停止。

cmpsb 是字符串比较指令，把 ESI 指向的数据与 EDI 指向的数一个一个的进行比较。

当 repe cmpsb 配合使用时就是字符串比较啦，当相同时继续比较，不同时不比较

mov edi,[ebp+08] 将你输入的密码的地址付给 EDI
mov esi,[ebp+0c] 真正的地址付给 ESI
mov ecx,[ebp+10] 将他们的长度附给 ECX
repe cmpsb 进行比较
jecxz 00401260 如果 CX 等于 0，即密码正确跳
mov eax,00 密码不正确时，会 EAX 为 0

再补充一下吧：
cmpsb 是将 ESI 指向的字节与 EDI 指向的字节进行减操作
如果两个字符相等，即 ZF 为 1，当不相等时 ZF 为 0

而 REPE 停止重复的条件是 ZF 为 0 或 CX 为 0，说明啦，当 ZF 为 0 时
肯定就是字串不同啦，当 CX 为 0 时，表明字符串比对成功没有出现
不相等的情况。

不过我觉得这样比不太好，如果你的密码打了 8 位
而真正的密码 4 位，如果 ECX 是 4 的话，他真会比对 4 位，正好你前 4 位
打对啦，那输入的密码也是正确的啦。如果他的 ECX 取得是你输入的密码
的个数，就没问题啦。不知道具体程序是什么样的，乱评一下，别介意。

3、CBW ：字节转换为字执行的操作：AL 的内容符号扩展到 AH,形成 AX 中的字。即如果(AL)
的最高有效位为 0，则(AH)=0;如(AL)的最高有效位为 1，则(AH)=0FFH
CBW 字节转换为字指令
执行的操作：AL 的内容扩展到 AH,形成 AX 中的字。

例如:CBW
指令执行前 AL=98H 即 10011000B
则执行后 AX=0FF98H

4、org 指定绝对地址的好处是，维护代码的时候不必再行计算相应代码被安排的地址。换句话说，如果你接手一份代码的时候，尚未完全了解硬件平台，未必清楚它的中断向量表大小，如果没有 org 指定主程序入口地址，你又如何能快速的判断呢

举例说，你的“一般的 org 2000H；ajmp main；org 2013H；ajmp Int_ADC.....”
实际上是指 ajmp main 这条指令被放在 rom 的 0x2000，ajmp Int_ADC 这条指令被放在 0x2013，而并不是说“main 函数从 0x2000 开始、Int_ADC 函数从 0x2013 开始”，所以你不必担心。只需要确保一条 ajmp 指令的长度不要大于 0x13 就没有问题（当然不会大于，64 位系统的 jmp 也只有 5 字节）
如果程序强行指定那两个函数的入口地址，xxx 是这样写的：
“

```
org 2000H
main:

xxx
...

org 2013H
Int_ADC:
xxx
xxx
```

”
如果是这样写的，那么就表示 main 被强制指定到 0x2000，Int_ADC 被强制指定到 0x2013，那么如果 main 后面的代码多于 0x13 字节，就会出问题了

5、
在 8086 系列的汇编语言中
IN 指令的意思是从端口中读取数据，比如 IN AL，80H，将 80H 端口数据读入到 AL 中
OUT 指令的意思是往端口输出数据，比如 OUT 80H，AL，将 AL 输出到 80H 端口

in 和 Out 都是 IO 操作指令 ， 例如 out 指令：
OUT PortNo/DX, AL/AX
功能为把 AL/AX 的数据送到 IO 地址，IO 地址如果大于 FFH，则地址需要放入 DX，要输出字节，默认使用 AL 寄存器，要输出字，默认使用 AX 寄存器。
例如:

```
MOV   AL,10H              //(将字节 10H 从 I/O 端口 70H  输出)
OUT   70H,AL

MOV   AX,0010H            //（将 字 0010H   从 I/O 端口 37FH  输出。）
MOV   DX,37FH
```

OUT　DX,AX

6、

STI(Set Interrupt) 中断标志置 1 指令 使 IF = 1；

CLI(Clear Interrupt) 中断标志置 0 指令 使 IF = 0；

它们只影响本指令指定的标志，而不影响其他标志位（即 STI 和 CLI 只影响 IF）。

7、汇编中 test 操作：

为举例方便说一下 jnz 和 jz

测试条件

JZ　ZF=1

JNZ　ZF=0

即 Jz＝jump if zero (结果为 0 则设置 ZF 零标志为 1,跳转)

Jnz＝jump if not zero

test 属于逻辑运算指令

功能: 执行 BIT 与 BIT 之间的逻辑运算

测试(两操作数作与运算,仅修改标志位,不回送结果).

Test 对两个参数(目标，源)执行 AND 逻辑操作,并根据结果设置标志寄存器,结果本身不会保存。TEST AX,BX 与 AND AX,BX 命令有相同效果

语法: TEST r/m,r/m/data

影响标志: C,O,P,Z,S(其中 C 与 O 两个标志会被设为 0)

运用举例:

1.Test 用来测试一个位,例如寄存器:

test eax, 100b;　　b 后缀意为二进制

jnz　******;　　如果 eax 右数第三个位为 1,jnz 将会跳转

我是这样想的,jnz 跳转的条件是 ZF=0,ZF=0 意味着 ZF(零标志)没被置位,即逻辑与结果为 1.

2.Test 的一个非常普遍的用法是用来测试一方寄存器是否为空:

test ecx, ecx

jz somewhere

如果 ecx 为零,设置 ZF 零标志为 1,Jz 跳转。

8、汇编语言查表指令 XLAT （Translate）。

查表指令 XLAT (XLAT： 字节查表转换)。　　指令格式：XLAT TABLE 其中 TABLE 为一待查表格的首地址。　　指令功能：把待查表格的一个字节内容送到 AL 累加器中。在执行该指令前，应将 TABLE 先送至 BX 寄存器中，然后将待查字节与其在表格中距表首地址位移量送 AL,即 AL<--((BX)+(AL)). 执行 XLAT 将使待查内容送到累加器。　　本指令不影响状态标位，表格长度不超过 256 字节

9、

HLT halt，停止执行指令，执行后 cpu 进入停止状态。不在执行指令。直到被其他设备的信号或中断信号来激活。用来等待设备输入和节能。

IRET(interrupt return)中断返回，中断服务程序的最后一条指令。IRET 指令将推入堆栈的段地址和偏移地址弹出，使程序返回到原来发生中断的地方。其作用是从中断中恢复中断前的状态。

10、宏与子程序的区别

宏和子程序都是为了简化源程序的编写，提高程序的可维护性，但是它们二者之间存在着以下本质的区别：

1、在源程序中，通过书写宏名来引用宏，而子程序是通过 CALL 指令来调用；

2、汇编程序对宏通过宏扩展来加入其定义体，宏引用多少次，就相应扩展多少次，所以，引用宏不会缩短目标程序；而子程序代码在目标程序中只出现一次，调用子程序是执行同一程序段，因此，目标程序也得到相应的简化；

3、宏引用时，参数是通过“实参”替换“形参”的方式来实现传递的，参数形式灵活多样，而子程序调用时，参数是通过寄存器、堆栈或约定存储单元进行传递的；

4、宏引用语句扩展后，目标程序中就不再有宏引用语句，运行时，不会有额外的时间开销，而子程序的调用在目标程序中仍存在，子程序的调用和返回均需要时间。

总之，当程序片段不长，速度是关键因素时，可采用宏来简化源程序，但当程序片段较长，存储空间是关键因素时，可采用子程序的方法来简化源程序和目标程序

11、宏与子程序都可以用一个名字定义一段程序，以简化源程序的结构和设计。二者的区别：

1、在源程序中，通过书写宏名来引用宏，而子程序是通过 CALL 指令来调用；

2、宏调用是通过宏扩展来实现的，宏引用多少次，就相应扩展多少次，所以，引用宏不会缩短目标程序；而子程序代码在目标程序中只出现一次，调用子程序是执行同一程序段，因此，目标程序也得到相应的简化

子程序

优点：模块化，节省内存，可被多次调用，编程效率高。

缺点：额外开销（保存返回地址，计算转向地址，传递参数等）大，增加了执行时间。适用于子功能代码较长、调用比较频繁的情况。

宏调用：

优点：参数传送简单，执行效率高。

缺点：不节省空间，适用于子功能代码较短、传参较多的情况。