开启dosbox 使用几个汇编程序 masm link debug

需要注意实模式 保护模式的区别 找专门的书看看

Debug命令 单步t 堆栈 内存g 反汇编u 寄存器r e修改内存

一个程序 是由高级语言编译成机器码（可以认为是汇编）后连接上系统头（用于执行），执行时 所有机器码被载入分配的虚拟内存 ，然后从入口点开始一行行执行代码，执行过程中会对 虚拟内存的数据进行读写 ，比如说读取文件到自己的内存，缓存修改数据，复制，因为可以修改自己分配的地址中任何数据，所以甚至可以动态修改自己的代码来改变执行，这也是壳加密的原理吧，以及反破解

机器语言 01

指令：汇编指令对应机器码 伪指令 符号+-\*/ 由编译器识别

存储器 指令就是数据 存储在内存或磁盘上 存储单元,一个单元8bit 一个字节 byte

Cpu读写内存 地址 读还是写 数据 分成三条地址总线、控制总线、数据总线

一根线只能传一个数字0/1 10个可以传10位 地址线根数就是总线宽度，即寻址上限

数据总线根数则是传输数据的位数

Cpu控制的主板 接口卡 存储器 随机存储ram 只读rom bios 显卡上也有ram rom装显卡bios 这些存储器都和cpu总线相连，被看成同一个空间

地址分段

寄存器 运算器计算 值输出给寄存器 寄存器从内存接收值 通用寄存器abcd 16位可以当成两个八位 ax=ah+al 高位低位

字节byte 字word

Move ax,18 add ax,8

二、物理地址 两个16位合成一个20位地址 段\*16+偏移 \*16实际上是左移四位

也就是基础地址+偏移地址 段寄存器cs ss ds es

CS段寄存器 IP当前指令 地址加法器合并寄存器地址 改变ip的方法是jmp

代码段 把程序代码放在一个段中 实模式 虚拟8086模式dos

内存中存值是连续的

Debuug r查看改变寄存器 d查看内存 e改写内存 u内存机器指令翻译成汇编 t执行 a内存写入指令

R 会显示当前的汇编指令 ds 修改值 r 寄存器 ax 也可以改cs ip

D 段:偏移 d 1000:0

E 起始 e 100:0 数据 数据 数据 数据 都是一个字节 两位16进制 空格 改写下一个 回车 改写完毕

可以写入字符串 打上引号就行

E 写入机器码 写入汇编对应的机器码

u可以翻译机器码 u 1000:0

t 执行 ip指向的指令 要执行指定指令先要cs ip跳转 r修改 ip指令每次跳转 指令长度的字节 跟据命令 寄存器位数而定 指令分三段 别忘了

e 用机器码写指令到内存很不方便 可以用汇编写 a

a 1000：0 mov ax 1000 想要测试汇编指令直接用a 就行了

jmp回退导致循环 可能是无限循环 自我相加 实际就是\*2 生产日期在ff段

三、寄存器 内存访问

放一个字 两个字节

2、 读取内存地址 ds段地址 mov ds,1000 mov al,[0] 把ds:0 的数据读取到al中

[]中是偏移地址 段地址自动读取ds 段寄存器不允许直接传入数字，必须用寄存器中转

3、一次性传入16位 两个字节 用 bx []内存直接传

4、mov add sub mov可以在寄存器 数据 内存 段寄存之间

5、数据段 专门用来存数据 方便顺序读取 累加等

6、栈 先进后出 push pop pop ax 把返回值赋给ax 栈可以实现倒序输出 倒序赋值

栈指针 ss:sp指向栈顶 栈段 栈顶 先开辟sp 每次push-2

超出空间 会覆盖外部数据 push pop可以是寄存器 段寄存器 内存

Ss sp可改 栈满时sp=0

开辟栈 mov ss,ax mov sp,10 修改ss的时候会直接跳到下一行语句，中断机制

7、当处理 debug d 命令 访问不同段时 段地址入ds e u r a ds:1000

四、程序 源程序 编译 连接

1、伪指令 段名 segment 段名ends

End 完了

Assume 假设段关联 和段寄存器关联起来

2、定义 一个段 abc segments: ends 指出结束 将abc和cs关联（不一定）

3、运行过程 系统程序将 程序载入内存 cpu控制权给程序 系统暂停运行？ 结束后交还控制权（windows不是这样的，是并发的）

关于程序返回 mov ax,4c00 in21 不写的话会导致不能返回，但是可以运行

语法错误 逻辑错误

Severe error

Masm link 连接可以把多个源程序连接，连接调用库

简洁编译masm 文件地址/文件名 link文件名

4、是谁运行的程序 操作系统 shell command 运行完成后返回到command

5、debug 程序.exe t单步 看寄存 程序装到内存哪了？ Cx是程序长度

程序加载后 ds是程序段地址 ds:0 前256个字节是psp程序头 用来通信 psp占100h

所以程序开头 ds:0+10:0 就是十位+1

用U直接看源代码

Debug中默认是十六进制 编程时不要忘记H 后缀

到了Int21 这句，不是用t 而是用P执行 Q退出debug

Pop 可以用来 清零ax和清空栈

五、[bx] loop 内存单元的表示 地址 单元长度或者类型 偏移地址存到寄存器

假设（）代表寄存器或者内存的内容 表达式法表示mov add push pop

Idata表示常量

Inc 加1

2、loop 用cx作为计数器 每次loop cx-1 如果cx为0跳出loop

Loop 地址号 之前先mov cx,次数

给循环位置做标记 s: 语句 loop s 计算2的n次方 计算两数相乘，转换为加法循环

源程序中数字不能以字母开头要加0

循环一次跑完 用P命令 源程序不能读[数字] 可以间接[bx]或者加上段ds:[0]

Mov ax,[bx] ds:[bx] cs:[bx] ss:[bx] es:[bx] 随意写内存可能导致死机，应该使用操作系统分配的空间 安全空间 0：200 -0：2ff

内存复制需要用寄存器中转 利用循环来读内存很好 loop inc bx 每次bx加1 读[bx]

注意debug中的地址也是用的16进制

六、多个段的程序 数据代码栈放一起 分开

大量数据计算，应该有连续空间存放

数据定义 放在最开头的话 dw 1,2,3,4,5 cs是代码段地址 偏移地址是0

数据在 0 2 4 6 8 A C 用循环取 [bx] add bx,2

此时 DS+10h指向的是数据开头，过了数据才是指令区，此时程序无法直接运行，必须修改ip

方法：指定入口 start: end start

程序如何识别入口

创建栈 可以dw 一串数据 开空间 然后mov ss,cs mov sp,30h 或者add也行

2、写多个段 data segemnt dw data ends stacksegment dw stack ends

Mov ax,stack mov ss,ax mov ax,data mov ds,ax 指针指向数据和栈段 段寄存器不能直接传入数字，必须用寄存器中转

Data code stack

3、 用assume cs:code ss:stack ds:data 将段相连 这语句没太大意义

七、内存寻址

1、and or 与计算 或

2、ASCII 3、源程序中用’ ‘字符或串给出数据 data段 db‘hello world ’ 4、大小写转换 加减20H 即32 判断数字码范围 实现 （还没学）

方法2：二进制第5位 为0则大写 为1则小写 强制转换

5、间接寻址写法 [bx+100h] 200[bx] [bx].200

6、处理数组 data段的 字符串可看成数组 7、si di和bx差不多 用来自增移位[si][di] [si+100] 8、[bx+di] 9、[bx+si+100] 100[bx][si]像二维数组

单变量 双变量 变量+常量 双变+常量

二维数组 双重循环 由于只有一个cx loop只能用cx计数，每次外循环必须把内循环的cx恢复，需要一个寄存器保存， 如果寄存器都不够，应该用内存来保存

暂存数据一般用栈

八、数据处理 位置和长度

假疔 寄存器reg 段寄存器 sreg 寻址用bx si di bp [bp]默认段地址在ss中，堆栈访问

数据位置：cpu 内存（比如堆栈或者数据段） 端口

数据类型 立即数 寄存器 内存地址（段+偏移）

寻址 直接 寄存器间接 寄存器相对（结构体，数组，二维） 变址寻址（双变量）二维

数据操作长度 字，字节 1、 靠寄存器类型ax al 2、没有寄存器时靠 word ptr 格式 3、push默认字操作

寻址[bx]

7、除法 div 8、db字节 dw字 dd双字 9、重复定义数据 db 3 dup (0) 重复3 次

10、访问员工信息

九、转移指令 可以修改cs:ip 段内转移 段间转移

短转移 范围比较小

Loop jmp 条件转移 过程 中断

Offset 标号 取得标号的地址值（偏移）

Jmp short 标号

Jmp的时候不需要转移目的地址 而是用的相对偏移 eb 03 代表移动3位

远转移 jmp far ptr 标号 jmp ax jmp word ptr [内存地址] 段内

Jmp dword ptr 段：偏移 段间

显示颜色

十、call ret 转移指令 retf 用栈数据修改cs:ip 近转移 远转移

Call ip或者cs:ip压入栈 再转移 和jmp相同

用来实现子程序设计 （函数）

ret 命令 cs：ip 获取堆栈栈顶中的地址 相当于pop 然后堆栈恢复

ret

call不能短转移 1、call 标号 ip入栈 之后ip会自动加上位移 相当于push ip jmp prt

2、段间转移 call far ptr 标号 cs:ip 先push 后修改cs ip 3、寄存器 call ax ip=ax

4、内存call word prt ds:[] call dword prt ds:[] 此时内存中存的值是地址

7、call ret配合 stack segment db 8 dup(0)

给子程序打一个标号 call func1

8、乘法 mul 默认是和ax相乘 结果高位默认在dx,低位在ax

Mul 寄存器 mul 内存

10、子程序传参 3的n次方 参数和返回值应该在哪 （1）用寄存器 ax bx dx

(2) 利用设置的数据段 传入实参 si di bx 都可以用 （3）如果参数很多，应该用内存保存参数 比如转换一个字符串为大写，数据段 and byte ptr [di] loop

(4)内存不一定要用动态寻址 完全可以用栈 push pop 那么如何在程序中取出栈中的参数呢？ 如果栈中保存数组和字符串呢？

11、显示字符串 字符串数组的遍历都是靠循环实现 显示需要调用显存地址

（2） 除法溢出 div 默认 是ax/被除 商太大就会溢出 可以自己编写程序 分成多位（3） 数据以十 进制显示 转换为字符串 acsii码

十一、标志寄存器 存储指令执行结果 作为cpu判断标准

Flag寄存器只有一个，但是每位都有意义

Zf 每行指令 结果是否为0 是0则1非0则0 包括加减乘除位运算 不包括mov push pop pf 奇偶 数字是奇数还是偶数 （代表二进制中1的个数） 奇数0 偶1

Sf 结果是否为负 补码代表有符号数 一个数可能是正数也可能是负数 当成无符号数时 sf无意义 SF就代表一个数是正还是负 但是如果发生溢出的话sf无意义，没溢出的话sf 取正 溢出sf取反

CF 无符号计算时借位或者进位

溢出 OF 有符号数 是否溢出 溢出1

Adc 带进位加法 adc ax,bx =ax+bx+cf 用来和add组合进行大数据计算 ，其他位存在另外的寄存器

Sbb 带借位减法

Cmp ax,bx 实际上就是sub 但是不存储结果

9、条件转移 cmp je相等则转移 jne不等 jb below 小于 jnb ja大于 above

可以进行条件判断了