DEBUG和MASM的区别：

DEBUG使用的是机器语言，但是借助指令助记符。

打了几行程序：

MOV AX, 1234

MOV [1000], BX



A: 地址

D: 看内存, 使用方式是：d【地址】，地址可以使用绝对地址和相对的地址

如：d 100 或者 d 12df：100 甚至直接输入 d

另外，还可以输入范围，如： d 100 110 或者 d 12df：100 110

C：看源程序

……

之后就是用的黑板：

1. 指令系统

C706A8592B1A

但是这个人肯定记不住

其实这句话等同于：MOV [59A8], 1A2BH

其中，C706/MOV是指令码，后面的[59A8], 1A2BH是操作数

其中，[59A8]是目的操作数，1A2B是源操作数

目的操作数有时候是变量，因为很少直接操作地址。大部分情况下带两个操作数，但是有的时候只带一个操作数，还有时候不带操作数。

操作数类型：

操作数包括：1. ；立即数， 2. 寄存器， 3. 内存

立即数就是直接的地址；寄存器比如AX之类的，也可能从内存里面取，往内存中放。

而**操作数的寻址方式**，也算是立即数、寄存器和内存三种方式。

MOV AX, 12H 前为寄存器，后为立即数

MOV AX, BX 前为寄存器，后为寄存器

MOV [1000H], DX 前为内存， 后为寄存器

源操作数有三种，但是目的操作数只有两种。

内存操作数寻址方式

段：偏移（一般情况下省略了段，段就是ds，等价于：DS : 偏移）

根据偏移的合成方式，一般分成五种寻址方式：

1. 直接寻址：比如MOV [1000H], AX；MOV BX, [2000H]

这种用[ ]直接表示的就是直接寻址方式。

2. 寄存器间接寻址：MOV [BX], AX，然而BX事先放入1000h；或者MOV BX. [DI]

在16位下，寄存器间接取址只有BX, BP, SI, DI

在32位下，寄存器EAX, EBX,ECX,EDX,ESP,EBP,EDI,ESI都可以间接取址

只有内存取址[BP]是ss段，其他的都是ds段

3. 寄存器相对寻址：MOV [BX+10H], AX, MOV BX, [DI+20H]或者MOV BX, [DI+var]

4. 基址变址：基址+变址

MOV [BX+SI], AX(不能MOV [DI+SI], AX这么写)不能两个变址或者两个基址相加

MOV BX, [BP+DI](默认ss段+di，而上面的默认ds段

32位下，基址还可以用eax，ebx，ecx，edx……除了esp

变址则8个全能用

5．相对基址变址：

MOV [BX+SI+10H], AX; MOV BX, [BP+SI+var]

MOV 目的操作数(reg, mem)，源操作数(imm, reg, mem)

但是mem->mem是不支持的，所以只有五种匹配方式（是没有这个电路）

1. 指令类别:
   1. 传输类：a=b，如mov，push,pop,xchg
   2. 运算类：a+b，如add
   3. 转移类：ifelse，如IF
2. MOV 目的，源（将源内容传输到目的内容）

Reg，mem/reg/imm

Mem，reg/imm

Seg，reg/mem

Reg/mem，seg 一共有9种

在写mov的时候，如果源操作数是imm，那么前面要加 byte/word/dword/qword/fword ptr，不然的话会报错，就是因为电脑不知道到底是多大长度的

CS不能作为目的操作数，code segment是不可以被指令改变的，只能够电脑自己去改。Mov cs，ax是不被允许的。

C语言的 short int a； MOV a, WORD PTR 0AH/MOV a, WORD PTR 10（10进制）

1. ADD 目的，源（用途：源+目的->目的）

Reg，mem/reg/imm

Mem，reg/imm

SUB 目的，源（用途：目的-源->目的）

Reg，mem/reg/imm

Mem，reg/imm

对于char a, b, c;

a = 10; b = 20; c = a+b;

b = c; c = a – b;

(对于第一行)

MOV a, BYTE PTR 10

MOV b, BYTE PTR 20

MOV AL, BYTE PTR a

ADD AL, BYTE PTR b

MOV c, BYTE PTR AL

（这才是对的）

Ex:MOV [1000H], BYTE PTR a

ADD [1000H], BYTE PTR b

MOV c, BYTE PTR [1000H]

(原来我是这么写的，是错的，因为a和b都在内存区域，内存和内存不行)

(对于第二行)

MOV BL, BYTE PTR c

MOV c，BYTE PTR BL （为什么会冲突？？？？）

MOV AL, BYTE PTR a

SUB AL, BYTE PTR b

MOV c, BYTE PTR AL

1. XCHG OP1, OP2 （两个内容互换）

Reg，reg/mem

Mem，reg

1. PUSH 源 （将源压入栈，之后调整sp）
2. POP 目的（将栈上最上方弹出，之后调整sp）
3. LEA 目的，源

Reg，mem

Mem，mem（？？？？）

1. XLAT
2. CBW 没有操作数，将AL符号扩展刀AX

如AL为01110000，那么扩展为00000000 01110000

如AL为10001111，那么扩展为11111111 10001111

1. CWD 没有操作数，将AX扩展为DX AX(32 bits)

如果AL最高位为0，那么DX为0，反之。

1. INC 操作数（reg/mem）， 操作数自身加一，然后又存入操作数
2. DEC ………………………………………………………减……………………………………
3. ADC 和ADD相仿，不同的是要加上CF（也就是进位）
4. SBB 和SUB相仿，不同的是要减去CF（借位）
5. NGA 操作数，求相反数，但是真正的做法却是0-操作数。
6. CMP 目的操作数，源操作数

功能是两个操作数比较大小，根据目的操作数减去源操作数的运算结果，从而设置标志位

操作数的寻址方式：

CMP reg，reg/mem/imm

CMP mem，reg/imm

1. MUL/IMUL 源操作数

8b： AL \* 源操作数-》AX

16b：AX \* 源操作数-》DX:AX

32b：EAX \* 源操作数-》EDX:EAX

操作数寻址方式：

MUL/IMUL reg/mem

两者的区别在于：mul将操作数视为无符号数，imul操作数内容看作补码

1. DIV/IDIV 源操作数

8b：AX/源操作数，商在AL，余数在AH

16b：DX,AX/源操作数，商在AX，余数在DX

32b：EDX,EAX/源操作数，商在EAX，余数在EDX

操作数寻址方式：

Div/idiv reg/mem

两者区别在于，div的操作数是无符号数，商和余数都是无符号数

Al是不是不影响ah？？？？

直接在当前目录输入MASM EXP2.ASM

如果汇编通过，那么就进行连接：LINK EXP2.OBJ

之后运行：EXP2.EXE