目录

[DEBUG 基本命令 1](#_Toc199340485)

[基本命令使用方式 1](#_Toc199340486)

[命令全称 2](#_Toc199340487)

[8086处理器寄存器 3](#_Toc199340488)

[32位和64位寄存器 3](#_Toc199340489)

[通用寄存器 3](#_Toc199340490)

[指针和索引寄存器 4](#_Toc199340491)

[段寄存器 4](#_Toc199340492)

[指令指针寄存器 5](#_Toc199340493)

[标志寄存器 5](#_Toc199340494)

[MASM基本指令 5](#_Toc199340495)

[数据定义指令 5](#_Toc199340496)

[数据传输指令 6](#_Toc199340497)

[算术运算指令 7](#_Toc199340498)

[逻辑运算指令 8](#_Toc199340499)

[控制转移指令 9](#_Toc199340500)

[移位指令 9](#_Toc199340501)

[循环指令 9](#_Toc199340502)

[函数指令 10](#_Toc199340503)

[其他指令 11](#_Toc199340504)

## DEBUG 基本命令

### 基本命令使用方式

1. **A [address]**: 在指定地址处开始汇编指令。如果不提供地址，默认从 CS:IP 当前指向的位置开始。
2. **D [address] or D [start address] [end address]**: 显示内存内容。不带参数时显示当前段内的部分数据。
3. **E [address] "data"**: 向指定内存地址写入数据。可以是单个字节的数据序列，也可以是一串字符。
4. **G [=address] [address ...]**: 运行程序。如果不提供起始地址，则从 CS:IP 当前值开始执行。你还可以指定断点地址。

g [= 起始地址] [断点地址1 断点地址2 ...]

起始地址（可选）：从哪里开始执行（默认从当前 CS:IP 指向的位置）。

断点地址（可选）：运行到这些地址时会自动停止。

1. **R [register name]**: 查看所有寄存器的内容，或者查看/修改特定寄存器的值。
2. **T [=address] [count]**: 单步执行指令。可以从指定地址开始执行给定数量的指令。
3. **U [address] or U [start address] [end address]**: 反汇编代码。将机器码转换为汇编代码以供阅读。
4. **Q**: 退出 DEBUG 程序
5. **P**: 自动执行到循环结束

### 命令全称

1. **A (Assemble)**：汇编命令。允许用户输入汇编语言指令并将其转换为机器码存储在指定的内存位置。
2. **C (Compare)**：比较命令。比较两个内存区域的内容，并显示不同之处。
3. **D (Dump/Display)**：转储或显示命令。显示指定内存区域的内容，以十六进制和ASCII码两种格式展示。
4. **E (Enter)**：输入命令。允许用户手动向指定的内存地址写入数据。
5. **F (Fill)**：填充命令。用指定的数据填充一段内存区域。
6. **G (Go)**：执行命令。从当前的CS:IP开始运行程序，或者从指定的地址开始执行直到遇到断点或程序结束。
7. **H (Hex Computation)**：十六进制计算命令。对给定的两个十六进制数执行加法和减法运算，并显示结果。
8. **I (Input)**：输入端口命令。从指定的I/O端口读取数据。
9. **L (Load)**：加载命令。从磁盘文件或指定的扇区加载数据到内存中。
10. **M (Move)**：移动命令。将数据从一个内存位置复制到另一个位置。
11. **N (Name)**：命名命令。指定要加载或写入的文件名。
12. **O (Output)**：输出端口命令。向指定的I/O端口发送数据。
13. **P (Proceed)**：继续执行命令。执行过程调用或循环，然后暂停。
14. **Q (Quit)**：退出命令。退出 DEBUG 工具返回操作系统。
15. **R (Register)**：寄存器命令。查看或修改CPU寄存器的内容。
16. **S (Search)**：搜索命令。在指定的内存区域内查找特定的数据模式。
17. **T (Trace)**：跟踪命令。逐条指令执行程序，并显示每一步的结果。
18. **U (Unassemble)**：反汇编命令。将机器码转换回汇编语言指令。
19. **W (Write)**：写入命令。将内存中的数据保存到磁盘文件或指定的扇区。

## 8086处理器寄存器

### 32位和64位寄存器

带有 ”e” 前缀表示32位寄存器

带有 ”r” 前缀表示64位寄存器

### 通用寄存器

**1. AX (Accumulator Register)**

* **功能**：累加器寄存器，通常用于算术运算、输入输出操作。
* **特点**：
  + 在执行乘法(MUL)和除法(DIV)指令时，默认使用AX作为其中一个操作数或存放结果的位置
  + 输入输出指令通常也使用AX来传输数据。

**2. BX (Base Register)**

* **功能**：基址寄存器，常用于存放**数据段**内的偏移地址。
* **特点**：
  + 当需要访问内存中的数据时，BX可以直接用作地址偏移量，如[bx]
  + 其他通用寄存器不能用作偏移量
  + DI和SI也可以用作偏移量

**3. CX (Count Register)**

* **功能**：计数寄存器，通常用于循环计数。
* **特点**：
  + 循环控制指令如LOOP会自动递减CX并检查是否达到零以决定是否继续循环。
  + 也可用于重复前缀指令，如字符串操作指令（REP MOVSB等）。

**4. DX (Data Register)**

* **功能**：数据寄存器，通常与AX一起用于双字长的数据操作
* **特点**：
  + 在进行高精度的乘法和除法运算时，DX用来保存高16位的结果或被除数的高位部分
  + 常用于I/O端口地址指定（特别是在间接寻址模式下）。

### 指针和索引寄存器

* + **SP (Stack Pointer)**：栈指针寄存器，指向栈顶的位置。
  + **BP (Base Pointer)**：基址指针寄存器，通常用于访问堆栈中的数据。

可以作为SS的偏移量

* + **SI (Source Index)**：源变址寄存器，常用于字符串操作。

可以作为DS的偏移量

* + **DI (Destination Index)**：目标变址寄存器，也常用于字符串操作。

可以作为DS的偏移量

### 段寄存器

* + **CS (Code Segment)**：代码段寄存器，指向当前执行指令的代码段地址。
  + **DS (Data Segment)**：数据段寄存器，指向程序使用的数据段地址。
  + **SS (Stack Segment)**：栈段寄存器，指向程序使用的栈段地址。
  + **ES (Extra Segment)**：附加段寄存器，通常用于扩展数据段或其他特殊目的。

**注意：**

* **直接将立即数加载到段寄存器是不允许的**。例如，mov ds, 1234h 是非法的语法。需要先将立即数移到一个通用寄存器，然后从那里移到段寄存器，如：

mov ax, 1234h ; 将16位值1234h移动到AX寄存器

mov ds, ax ; 将AX中的值加载到DS段寄存器

* CS寄存器可以通过设置start: 标签自动加载，其他段寄存器只能手动写代码加载

### 指令指针寄存器

1. **IP (Instruction Pointer)**：指令指针寄存器，存储下一条将要执行的指令的偏移地址。与CS寄存器一起确定实际的内存地址。

### 标志寄存器

**FLAGS**：这是一个16位寄存器，包含了多个单个位的状态标志，如进位标志(CF)、零标志(ZF)、符号标志(SF)等，用于控制CPU的操作流程。

|  |  |  |  |
| --- | --- | --- | --- |
| 标志位 | 名称 | 位位置 | 含义说明 |
| CF | 进位标志位 | 位0 | 算术运算中有无进位或借位（如无符号溢出） |
| PF | 奇偶标志位 | 位2 | 运算结果低8位中1的个数为偶数则为1，奇数为0 |
| AF | 辅助进位标志位 | 位4 | 用于BCD运算，表示第3位向第4位进位或借位 |
| ZF | 零标志位 | 位6 | 运算结果为0则置1，否则为0 |
| SF | 符号标志位 | 位7 | 运算结果最高位（符号位），1为负，0为正 |
| TF | 单步标志位 | 位8 | 控制调试功能，置1后每执行一条指令会触发中断 |
| IF | 中断允许标志位 | 位9 | 为1表示允许响应外部中断请求 |
| DF | 方向标志位 | 位10 | 控制字符串处理指令的方向（递增/递减） |
| OF | 溢出标志位 | 位11 | 表示有符号运算溢出 |

## MASM基本指令

### 数据定义指令

* **DB (Define Byte)**：定义字节大小(8 bit)的数据。

var1 DB 5 ; 定义一个值为5的字节

* **DW (Define Word)**：定义字大小（16 bit）的数据。

var2 DW 0FFFFh ; 定义一个值为0xFFFF的字

* **DD (Define Doubleword)**：定义双字大小（32 bit）的数据。

var3 DD 12345678h ; 定义一个值为0x12345678的双字

* **DQ** :定义四字大小(64 bit)的数据

**注：定义的数值默认为10进制**

* 重复数据定义指令

[count] DUP (value)

count：表示要重复的次数，这个数字默认为10进制

value：表示每次重复的内容（可以是一个常量、表达式、或者一组值）。

### 数据传输指令

**MOV**：将数据从源地址复制到目标地址。

MOV AX, BX ; 将BX寄存器的值移动到AX

MOV [**var**], AL ; 将AL寄存器的值存储到变量**var**中

**注：只能有一个操作数是内存地址**

**OFFSET**

返回某个符号（如变量、标号）在其所在段中的偏移地址。它是一个数值常量，表示该符号距离段起始位置的字节偏移量

MOV register, OFFSET symbol

register：通常是 AX、BX、SI、DI 等通用寄存器。

symbol：可以是变量名、标号名等。

**LEA** - **Load Effective Address（加载有效地址）**

将一个内存操作数的偏移地址（即有效地址）加载到某个寄存器中。

不传递内容，而是**传递地址**

**语法格式**

lea destination, source

destination：必须是一个寄存器（如 AX, BX, SI, DI, BP 等）

source：必须是一个内存操作数（如 [bx], [si+2], [bx+di\*2] 等）

**LODSB**

* **功能**：从由 **DS:SI指向的内存位置**加载一个字节的数据**到 AL 寄存器**，并根据方向标志（Direction Flag, DF）自动调整 SI 的值。如果方向标志为0，则增加 SI；如果方向标志为1，则减少 SI。
* **用途**：常用于**遍历或检查一段内存区域**中的每个字节。

**STOSB**

* + **功能**：将 **AL 寄存器**的内容存储**到由 ES:DI指向的内存位置**，并根据方向标志自动调整 DI 的值。
  + **用途**：用于将数据**写入连续的内存地址**，通常用于填充或设置内存块。

**MOVSB**

* + **功能**：从由 **DS:SI 指向的内存位置**移动一个字节的数据**到由 ES:DI 指向的内存位置**，并根据方向标志调整 SI 和 DI 的值。
  + **用途**：实现**两个内存区域之间的数据复制**。

**SCASB**

* + **功能**：比较 AL 寄存器与 ES:DI 指向的内存位置处的字节，并根据结果和方向标志调整 DI 的值。
  + **用途**：用于**搜索特定字节值**在内存块中的出现位置。

**CMPSB**

* + **功能**：比较 DS:SI 和 ES:DI 指向的内存位置处的字节，并根据结果和方向标志调整 SI 和 DI 的值。
  + **用途**：用于**比较两个内存区域的内容是否相同**。

这些指令可以与重复前缀如 REP、REPE/REPZ、REPNE/REPNZ 结合使用，以实现对字符串或内存块的多次操作。例如，REP MOVSB 将重复执行 MOVSB 直到计数寄存器 CX 归零，从而实现高效的大规模数据复制。

### 算术运算指令

* **ADD**：加法运算。

ADD AX, BX ; AX = AX + BX

* **SUB**：减法运算。

SUB CX, DX ; CX = CX - DX

* **INC**：递增操作数。

INC AX ; AX = AX + 1

* **DEC**：递减操作数。

DEC BX ; BX = BX – 1

* **ADC**：带进位加法指令

指令格式：

adc destination, source

功能说明：

低位add，高位adc

destination = destination + source + CF(进位标志)

* **SBB**：带借位减法指令

指令格式：

sbb destination, source

功能说明：

低位sub，高位sbb

destination = destination - source – CF(借位标志)

#### 乘法指令

1. **无符号乘法**
   * **MUL**：用于执行无符号整数乘法。
     + 对于字节操作数（8位），AL = AL \* operand，结果存放在 AX 中。
     + 对于字操作数（16位），AX = AX \* operand，结果的低16位存放在 AX，高16位存放在 DX。
     + 对于双字操作数（32位，在32位模式下），EAX = EAX \* operand，结果的低32位存放在 EAX，高32位存放在 EDX。
2. **有符号乘法**
   * **IMUL**：用于执行有符号整数乘法。
     + IMUL 可以接受一个或两个操作数。当提供一个操作数时，行为类似于 MUL，但是处理的是有符号数。
     + 当使用两个操作数时，格式为 IMUL dest, src 或者三个操作数 IMUL dest, src1, src2，其中 dest = src1 \* src2。结果可能会存储在一个较大的寄存器中以容纳可能的溢出。

#### 除法指令

1. **无符号除法**
   * **DIV**：用于执行无符号整数除法。
     + 对于字节操作数（8位），AL = AX / operand（商），AH = AX % operand（余数）。
     + 对于字操作数（16位），AX = DX:AX / operand（商），DX = DX:AX % operand（余数）。注意这里需要先将被除数放入 DX:AX 中，并确保 DX 的值是正确的（通常需要先清零）。
     + 对于双字操作数（32位，在32位模式下），EAX = EDX:EAX / operand（商），EDX = EDX:EAX % operand（余数）。
2. **有符号除法**
   * **IDIV**：用于执行有符号整数除法。
     + 使用方式与 DIV 类似，但适用于有符号数。同样需要注意正确设置高位寄存器（如 DX 或 EDX）的值。

### 逻辑运算指令

* **AND**、**OR**、**XOR**：执行按位逻辑运算。

**AND** AX, BX ; 对AX和BX进行按位与运算，并将结果存入AX

**OR** AX, BX ; 对AX和BX进行按位或运算，并将结果存入AX

**XOR** AX, BX ; 对AX和BX进行按位异或运算，并将结果存入AX

### 控制转移指令

* **JMP**：无条件跳转。

JMP label ; 跳转到标签label处

* **JE/JZ**, **JNE/JNZ**, **JG**, **JL**等：基于比较结果有条件跳转。

CMP AX, BX ; 比较AX和BX

JG 0000:0000 ; 如果AX大于BX则跳转到0000:0000

**注：**默认情况下，MASM会根据上下文自动决定使用近跳转或远跳转

### 移位指令

|  |  |  |
| --- | --- | --- |
| SHL dest, count | 逻辑左移（Logical Shift Left） | shl ax, 1 |
| SHR dest, count | 逻辑右移（Logical Shift Right） | shr al, cl |
| SAL dest, count | 算术左移（Arithmetic Shift Left） | sal dx, 1 |
| SAR dest, count | 算术右移（Arithmetic Shift Right） | sar bx, cl |

### 循环指令

**LOOP**

**语法：**LOOP label ;label 是一个标号（跳转目标）

**执行过程:**

CX = CX - 1

如果 CX ≠ 0，则跳转到 label

如果 CX == 0，则继续执行后续指令

**注意**：

CX 必须在使用 LOOP 前预先赋值。

label 目标必须在 LOOP 指令附近的 ±128 字节内（短跳转）。

### 函数指令

函数名 PROC [NEAR|FAR]

（具体代码）

RET

函数名 ENDP

**参数说明：**

NEAR：近程调用（默认），仅在当前段中跳转；

FAR：远程调用（跨段调用），需要同时保存 IP 和 CS，使用RETF返回

**调用方式：**

CALL MyProc CALL FAR PTR MyProc

**注：**

CALL命令执行后会自动将CALL下面一条指令的地址压入栈中

RET命令执行后会自动出栈，把出栈的值赋给IP

使用远程调用时一定要有CALL FAR PTR 函数名 和RETF

#### 例：有参函数

MOV AX, 3

PUSH AX ; 参数1

MOV AX, 5

PUSH AX ; 参数2

CALL AddTwoNums

ADD SP, 4 ; 清理参数（2×2字节）

MOV result, AX ; AX中返回值

AddTwoNums PROC

PUSH BP

MOV BP, SP

MOV AX, [BP+4] ; 第一个参数（第二次压栈）

ADD AX, [BP+6] ; 第二个参数（第一次压栈）

MOV SP,BP

POP BP

RET

AddTwoNums ENDP

**引入PUSH BP和MOV BP,SP的原因：**

在函数执行过程中，栈指针SP可能由于PUSH/POP不断变化，因此不能通过 SP 定位参数或变量。这时可以将函数刚开始时的栈偏移**保存到BP作为一个不变的基准**，即MOV BP,SP。不过这样会导致BP原来的值被覆盖，因此需要在MOV BP,SP之前有PUSH BP保存原始BP值。

### 其他指令

#### XCHG：交换两个操作数的内容

* 支持寄存器之间、寄存器与内存之间的交换。

**🔹 语法：**XCHG operand1, operand2

#### NEG：对操作数取负（求补）

**🔹 功能：**

* 对操作数取负数（相当于 0 - operand）。
* 常用于实现有符号数的取反或减法运算。

**🔹 语法：**NEG register\_or\_memory

#### INT：调用中断服务程序

**🔹 功能：**

* 触发一个软件中断，跳转到指定的中断向量表中的中断处理程序。
* 在实模式下常用于调用 BIOS 或 DOS 功能。

**🔹 语法：INT** type\_number

其中 type\_number 是中断类型号（0 ~ 255）。

**🔹 常见中断示例（DOS 中断）：**

**INT 21h：DOS 功能调用**

#### <size> ptr: 指示内存操作数大小

用于指定操作数的大小，如：

mov ax,word ptr bl指定bl中的数据为16bit