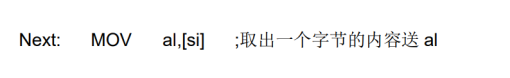
第六章 汇编语言源程序组织

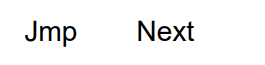
1. 语句种类和格式
2. 指令语句

一般格式：

标号: 指令助记符 目标操作数表达式, 源操作数表达式 ;注释



其中，标号是指令语句的符号地址。其实源程序经编译（汇编）后，每条指令语句的代码都会按先后顺序自然包含程序运行时在代码段的存放位置，即每条指令语句都有各自的地址，而标号是给要使用这条语句地址的代码作了一个标记，程序编译（汇编）时会将其他使用到这个符号地址的语句中的这个标号替换成相应的地址。比如在同一代码段内其他位置有一条跳转指令：

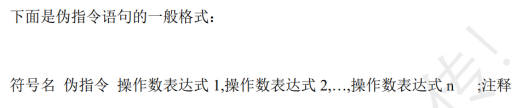


无论一条指令语句是否有标号，编译后的代码是没有区别的。

1. 伪指令语句

伪指令是用于指示编译（汇编）程序如何编译（汇编）源程序的，例如描述源程序如何分段和各段分别由哪个段寄存器指向，以及常量和变量及子程序的定义等

伪指令是用于控制源程序编译（汇编）而不是控制 CPU 运行的；伪指令是在源程序编译时被执行的，而指令是在程序运行时被执行的



和指令语句在形式上的区别除语句核心字段是伪指令而不是指令外，符号名后没有冒号也是明显和重要的区别特征。

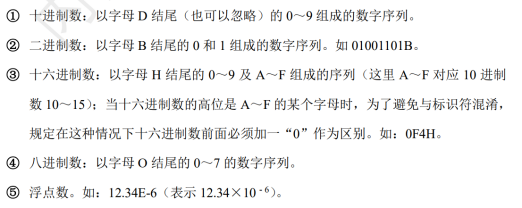
语句中的“符号名”是常量、变量、子程序或段结构等的命名。

1. 标识符
2. 标识符是长度不超过31 的字符串
3. 标识符的字符集是英文字母、数字和下述特殊字符：



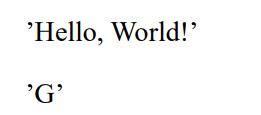
的ASCII 码，但第一个字符不能是数字

1. 标识符不能和汇编语言的保留字冲突。即不能和寄存器、指令、伪指令等同名。
2. 常量与变量
3. 常量
4. 数字常量

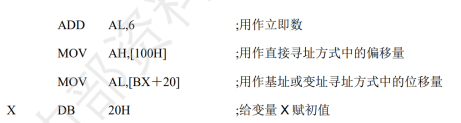


1. 字符串常量

字符串常量由单引号或双引号括起来的 1 个或多个字符组成，如：

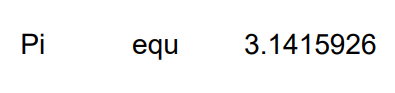


编译（汇编）后每个字符被翻译成对应的 ASCII 码，每个字符占一个字节。如’ABC’ 编译（汇编）后所占 3 个字节的内容是 41H、42H、43H。



1. 符号常量定义伪指令

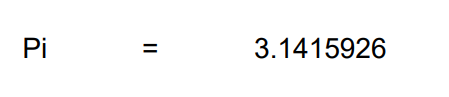
① 等值伪指令 equ



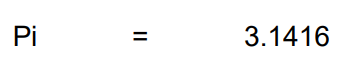
这条语句给编译（汇编）程序的指示是： 将源程序中所有的 Pi 都用 3.1415926 替代。

② 等号伪指令“=”

简单地将“equ”改写成“=”：



“=”伪指令和“equ”的差别是：后者可以对同一常量作多次定义，比如在程序的另一处可以：



即此后遇到 Pi 的地方都被编译成3.1416而不是 3.1415926。需要说明的是，尽管这 条伪指令有点灵活性，但笔者认为这样做会增加编程时为避免混乱的额外负担，所以建议定义常量就使用 equ 伪指令即可。

1. 简单变量定义

计算机存储单元的内容是可以被读取或被修改的，所以编程语言用符号命名的若干存储单元来表示一个变量。汇编语言中，常见的数据类型有字节、字和双字等。

1. 变量定义的一般形式



① 变量名必须是一个合法的标识符；

② 数据定义符用于确定变量的数据类型，常用的定义符有：DB、DW 和 DD 等；

③ 表达式用于定义内存单元的初值，一个定义语句可以有多个表达式，各表达式 之间必须用西文逗号‘,’分开；如果某个存储单元没有初值表达式，则必须用一 个西文问号‘?’来替代； ④ 变量定义语句的注释根据具体需要，可写可不写。

其实变量名就是某个或某若干个字节存储单元的符号地址，这和指令语句的标号很 相似：变量名是所描述的变量在数据段的偏移量，这是汇编程序在编译（汇编）时，将 操作数第一个字节的偏移地址赋给了这个符号。

1. 定义字节变量

字节变量的定义符为 DB，如



该语句定义 X 为字节型变量，分配给 X 一个字节，且初值为零。

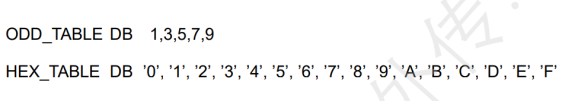
这条伪指令语句和下面 C 语言的变量定义语句一样的





这里的西文问号表示没有给 X 赋初值，但分配一个字节的空间。这和C语言的变量定义语句是一样的：



Eg：

前一语句为字节型变量 ODD\_TABLE分配了 5 个字节并依序被初始化成了 10 以内 的奇数；后一语句定义了一个 16 进制数的ASCII 码表。

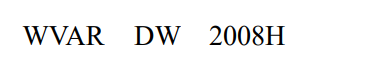
DB 伪指令还支持用“字符串”表达式定义变量初值，如



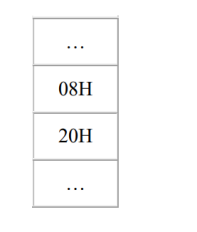


1. 字变量定义

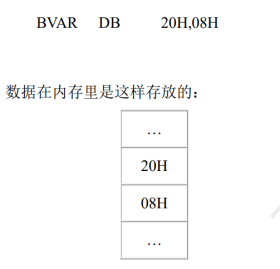
字变量的定义符为 DW；每个字占用两个连续的字节单元。如：

****

上述定义初始化后的内存分配如下所示：

****

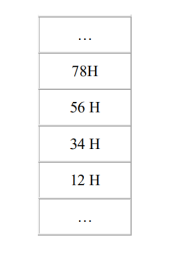
注意字变量数据是按“高位存放到高地址字节、低位存放在低地址字节”的方式存 于存储单元之中的，而字节数据是按照顺序排列在存储单元中的

****

1. 双字变量

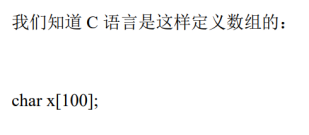
双字变量的定义符为 DD，每个双字变量占用二个连续的字单元共四个字节。



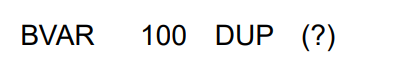


字和双字类型变量多用作存储较大值域的数据，不适合用作存储 8 位字符的字符串。

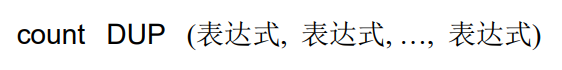
1. 重复说明符DUP



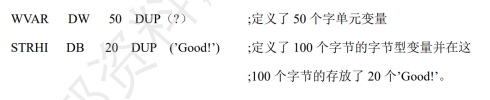
而汇编语言用“重复说明符”DUP 来定义同类型变量：



带重复说明符 DUP 的表大式的一般形式是：

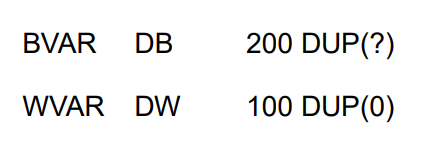


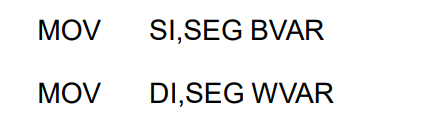
其中：count是重复次数，括号内的若干表达式是被重复的部分；“表达式”可以直 接是存储单元的初值，也可以是另一个被嵌套的DUP 表达式；表达式括号中的多个表达式用西文逗号“,”分开。下面是一些使用 DUP 的例子：



1. 标号和内存变量的属性及属性操作符
2. 段属性运算符SEG

SEG 指示编译程序返回变量或标号操作数所在段的段基址。



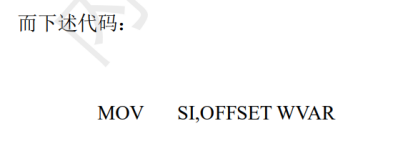


因为 BVAR 和 WVAR 在同一数据段中，所以编译后，SEG BVAR 和 SEG WVAR 是相同的，且和变量本身的类型无关。

1. 偏移量属性运算符 OFFSET

偏移量属性运算符指示编译程序返回变量或标号操作数的偏移量。





的源操作数表达式在程序编译时将得出一个立即数的结果；程序执行这条语句时，SI 将获得变量 WVAR 的偏移量

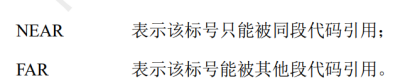
和 LEA 指令相比，指令执行后的效果是一样的，但实现的手段是有区别的：LEA 是纯指令实现的功能，而偏移量运算符主要是靠编译程序的功能实现的； 另外 LEA 的目标操作数只能是 16 位通用寄存器，而带 OFFSET 运算符及变量的表达式经编译后是一个立即数，所以目标操作数还可以是 16 位存储器单元。

1. 类型属性运算符TYPE

类型属性运算符 TYPE 返回变量的字节数或语句标号的 FAR 或 NEAR 类型



变量的类型数字就是变量的字节数，直观实用；而标号的属性用于说明该标号能否被其他段的代码引用

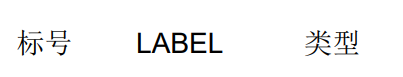


语句标号的属性在语句定义时，被默认成 NEAR 属性；标号的 FAR 属性的须通过 LABEL 伪指令设置。

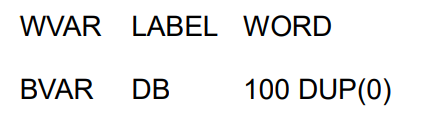
1. LABEL THIS PTR

LABEL 伪指令用于设置或更改变量或标号的属性：





其中变量的数据类型可以是 BYTE，WORD，DWORD；标号的类型可以是 NEAR 或 FAR。



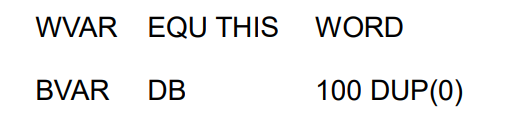
这样定义后，就可以用字节或字属性访问这 100 个字节了。比如在做数据块清零或 复制等操作时，用字属性操作是 16 位的，代码效率会更高。

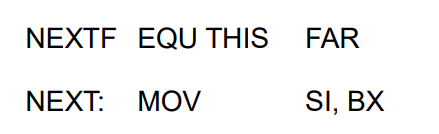


为上述数值语句定义了不同属性的标号，段内转移指令使用标号 NEXT，段间使用 NEXTF

目标代码和没有标号说明时占用的空间是一样的，只是编译（汇编）程序会将同段其他位置的标号替换成该语句的段内偏移量

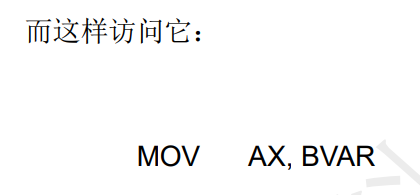
在上述语句中使用 EQU THIS 替换 LABEL 可以实现同样的目的

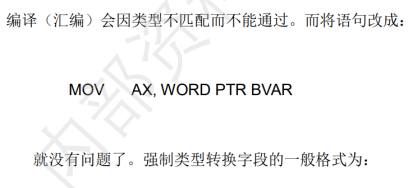




如果只是个别地方需要用和定义时不同的属性对该变量进行访问， 可使用强制类型转换运算符 PTR 来实现。









强制类型转换给变量赋予一种临时性的属性，**只在本条语句有效**

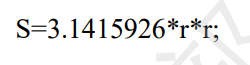
1. LENGTH 和 SIZE

长度属性操作符 LENGTH 是内存变量操作符，它指示编译（汇编）程序返回重复操作符 DUP 中的重复数。如果有嵌套的 DUP，则只返回最外层的重复数；如果DUP，则返回1。显然，这个操作符的功能是很有限的，完全不能和 C 语言的 Strlen 函数相提并论。



1. 汇编语言表达式

高级语言涉及到变量的运算表达式将用于在目标代码运行时控制计算机的运作，而汇编语言的表达式是用于控制编译（汇编）的

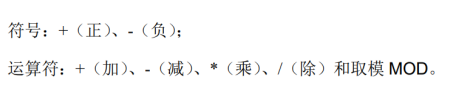


如果r是变量，那么表达式 3.1415926\*r\*r 就是用于控制计算机运作的代码，其结果是在该语句执行后确定的；而汇编语言的表达式在编译（汇编）后已经是被确定的常量了。

1. 数值表达式

数值表达式是在编译（汇编）过程中能够确定计算值的表达式，即编译（汇编）完成后表达式的值就完全确定

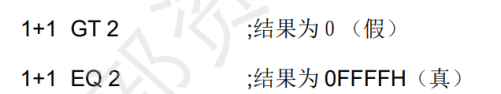
1. 算术运算符



1. 关系运算符

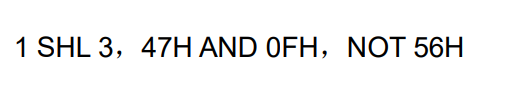
关系运算符包括符号：EQ（相等）、NE（不等）、LT（小于）、GT（大于）、LE（小 于等于）和 GE（大于等于）。

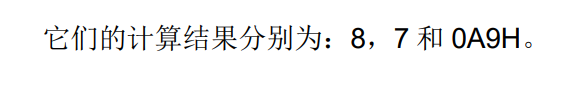
若关系不成立，数值表达式的结果为0，否则为 0FFFFH。



1. 逻辑运算符

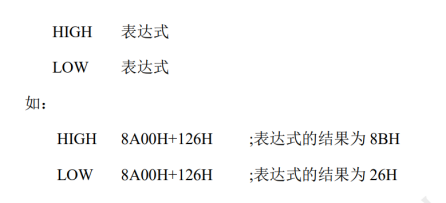
AND（逻辑与）、OR（逻辑或）、 NOT（逻辑非）、XOR（异或）、SHL（左移位）和 SHR（右移位）

****

****

1. HIGH和LOW操作符

HIGH 和 LOW 操作符分别获取表达式计算结果的高 8 位和低 8 位

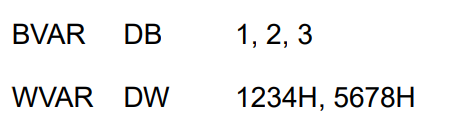


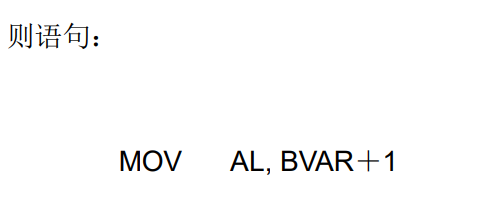
1. 运算符和操作符的优先级



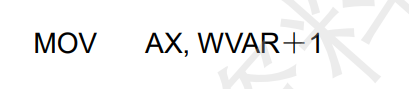
1. 地址表达式

地址表达式是计算存储单元地址的表达式，它可由标号、变量名和由括号括起来的基址或变址寄存器组成。其计算结果表示一个存储单元的地址，而不是该存储单元的值。

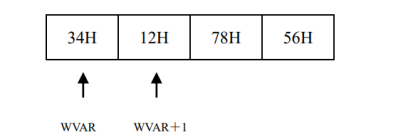




执行后，AL的内容为 2，即编译（汇编）后，地址表达式 BVAR＋1 指向变量 BVAR之后内容为2的那个字节



执行后，AX 的内容不是 5678H，而是7812H。这是因为地址表达式 WVAR＋1 没有指向变量 WVAR 之后的那个字而是之后的那个字节

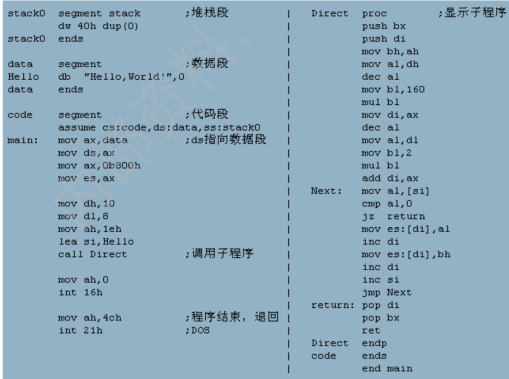






似乎定义 VOFF 和 VPTR 的表达式包含了变量，事实上 BVAR 是一个符号地址，经编译（汇编）后已经是一个确定的常量，故表达式 BVAR+1 也是一个常量：变量 BVAR 第二个字节的偏移量，作为变量 VOFF 的初值存放到偏移量为 VOFF 的字单元中了；而当双字变量 VPTR 的初值为包含变量的表达式时，编译（汇编）程序将地址表达式指向的变量的 32 位地址指针赋给 VPTR。

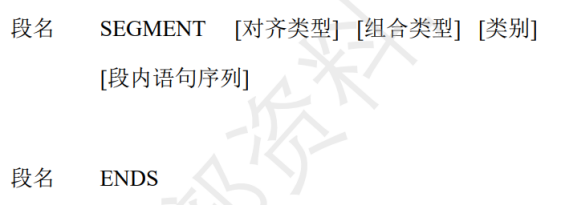
1. 段定义伪指令与源程序框架



1. 段定义伪指令

SEGMENT 和 ENDS，这是汇编语言 段定义伪指令。

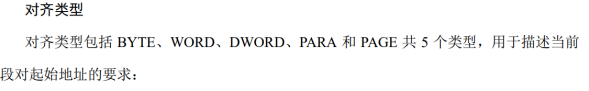
一般格式：



段名是由编程者根据段的用途按标识符的规则定义的，且前后二个段名要相同。和其 他写在一行的伪指令语句不同，段定义伪指令和结构定义语句相似，或者也可以看作是一种语句括号：以关键字 SEGMENT 开头，以段结束伪指令ENDS结束。

类似的这种伪指令语句还有过程定义语句和宏定义语句等。和其他伪指令语句定义的符号名的另一个不同是，汇编语言程序的段名可以是唯一的，也可以与其它段同名。如果有二个段同名，则后者被认为是前段的后续，它们属同一段。

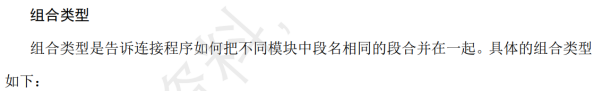
关键字 SEGMENT 之后的“对齐类型”、“组合类型”和“类别”为段属性。段属性必须按如上所述的顺序说明，而是否描述可根据需要选择



¬ BYTE：逻辑段从未占用的下一个字节开始。 ¬ WORD : 逻辑段从未占用的下一个字开始。有可能空闲 1 个字节；段的起始地址必须是偶数。

¬ DWORD : 逻辑段从未占用的下一个双字开始。有可能空闲 3 个字节；段起始物理 地址的低 2 位为 0。 ¬ PARA：逻辑段从下一个未占用的节（16 个字节）开始，有可能空闲 15 个字节；段起始物理地址的低 4 位为 0。当未指定对齐类型时，节对齐为默认对齐类型。

¬ PAGE : 逻辑段从下一个未占用的页（256 字节）开始。有可能空闲 255 个字节， 段的起始物理地址的低 8 位为 0



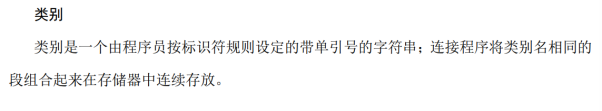
¬ NONE：表示当前段在逻辑上独立于其它模块，并有其自己的基地址。NONE 是缺省的组合类型。

¬ PUBLIC：表示当前段与其它模块中同段名的 PUBLIC 类型段组合成一个段。组合 的先后次序取决于 LINK 程序中目标模块排列的次序。在组合时，后续段的起始地址要按其对齐类型进行定位，所以，同名段之间可能有间隔。

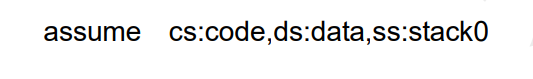
¬ COMMON：表示当前段与其它模块中同名段重叠，也就是说，它们的起始地址相同。最终段的长度是同名段的最大长度。由于段覆盖，所以，前一同名段中的初始化数据被后续段的初始数据覆盖掉。

¬ STACK：组合类型 STACK 表示当前段是堆栈段，其组合情况与 PUBLIC 相同。

¬ AT 数值表达式：以数值表达式的值作为前段所的起始地址。 例子中除了堆栈段的组合类型是 STACK 外，其他的段都没有明确描述组合类型，即为默认的 NONE 独立组合类型。



1. 段声明伪指令的段初值

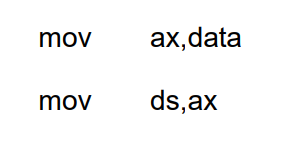


这条语句通过建立了段名和段寄存器间的对应关系，让编译（汇编）程序知道了程序的段是如何安排的。ASSUME 语句的一般形式是：



虽然 assume 伪指令指明了某段名和某段寄存器的关系，但并没有确定段寄存器的初值。

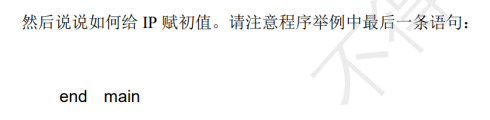
当典型的 DOS 程序运行时，操作系统会根据内存的使用情况把执行程序引导、安放到内存的适当位置，这里所说的“安放”就是通过初始化段寄存器来实现的。这样做不仅让我们在编写单个程序时不用去考虑段间冲突，也不用担心两个执行程序在内存出现冲突。DOS 在引导程序进入内存时对各个段寄存器的初始化是有区别的：CS 和 SS（当 对应段的组合类型定义成 STACK 时）被直接初始化了，而在装入程序过程中 DS 和 ES 寄存器有其它用途，所以只能在用户程序中用指令专门对其作初始化，以装入用户数据段和附加段的基址。



通过 ax 搭桥的原因是由于所有的段寄存器都不能接收立即数；而 data 和其他段名一样是一种特殊的立即数：它们的性质一直是立即数，但它们的值是在程序装入时由操作系统确定的。

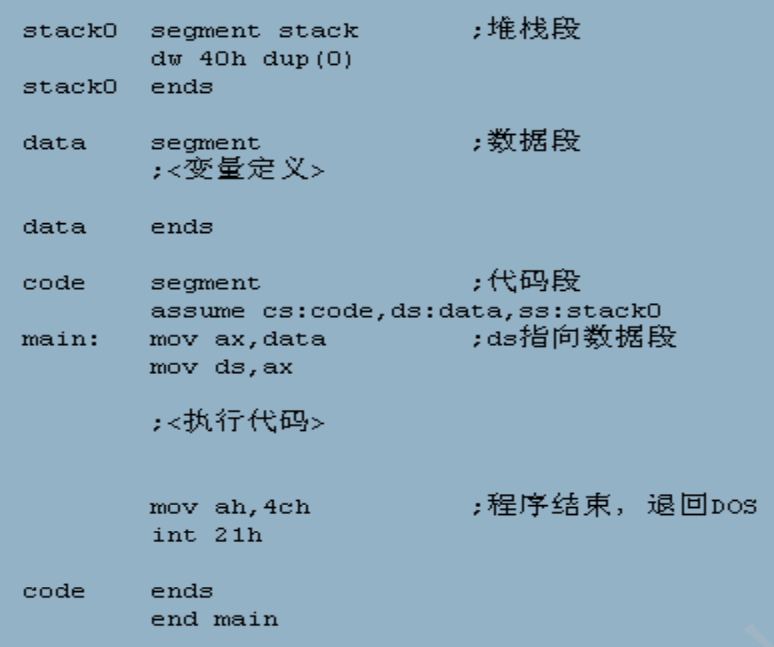
1. IP和SP的初值

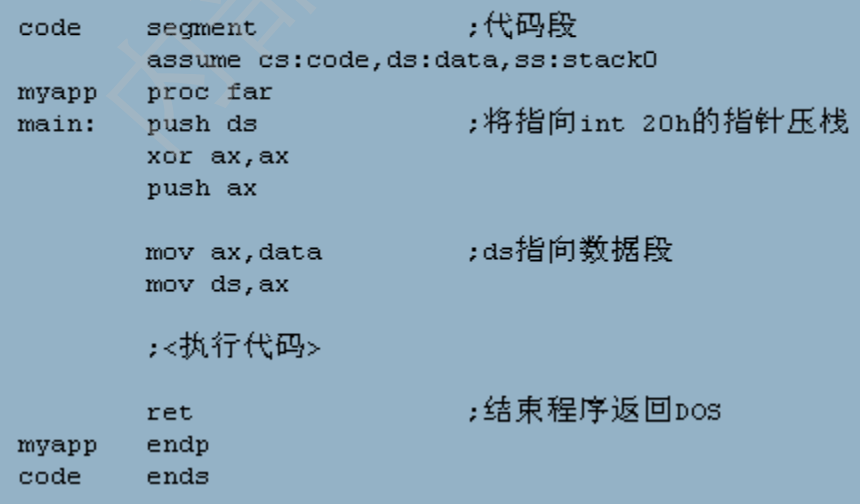
我们先说说给堆栈指针 SP 赋初值。如果对应段的组合类型选择 stack，栈指针 SP 将被设置成该段定义的字节数的总和；如果没有一个段的段组合属性设置成 stack，就必须用类似初始化 DS 和 ES 那样让 SS 指向你设置的堆栈段并让 SP 指向栈顶。



end 是一条伪指令，表示源程序到此为止，编译（汇编）程序不处理该语句之后的任何内容。另外，end 后面可以附带一个在代码段中已定义的标号，用以说明程序启动的偏移量。 如果源程序是一个独立程序或主模块，end 后面一定要带一个如上所述的标号，否则 end 之后就不能跟除注释之外的任何东西。 所谓程序的运行，就是程序文件被操作系统引导到内存中，然后让指令指针 CS:IP 指向程序的起点，比如 main，具体地讲就是给 CS 和 IP 赋值；而 IP 所赋的值，就是由伪指令 end 指定的。

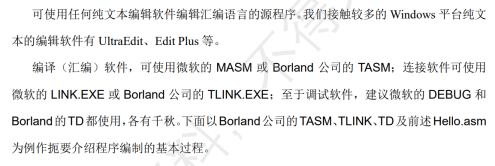
1. 源程序基本框架





即把代码段做成一个被 DOS 调用的子程序的形式。

1. 编制汇编语言程序的完整过程
2. 编程工具及经典过程

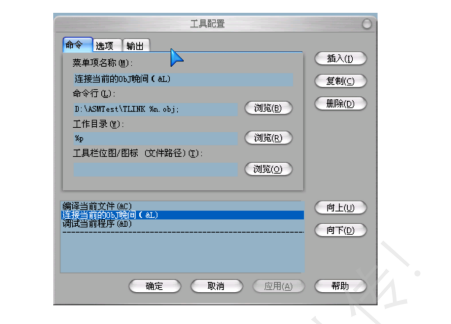






1. 用 UltraEdit 设置简易的汇编语言编程环境

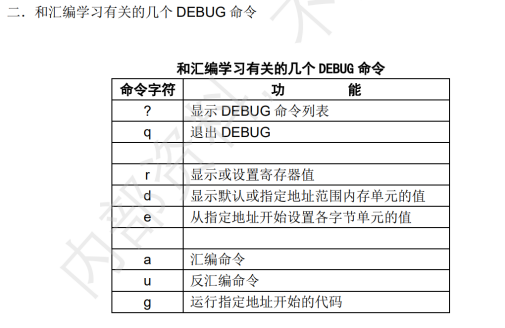


****

（未完 看不懂）

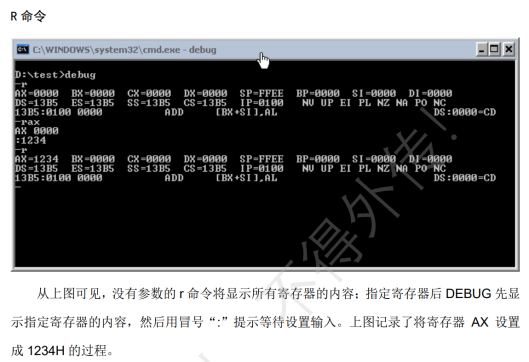
1. DEBUG常用命令简介





DEBUG只显示和接受 16 进制数

R命令：

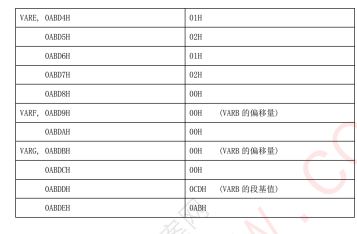


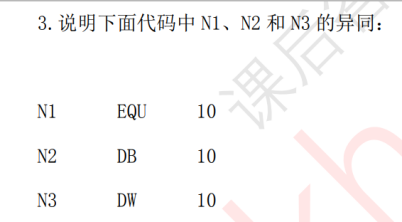


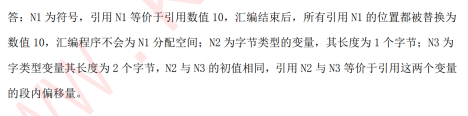
习题：

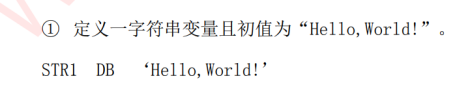




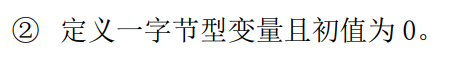










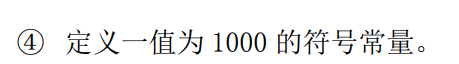




DW：字变量定义



VAR1 DW 100 DUP(?)

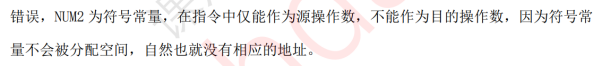


SYM1 EQU 1000

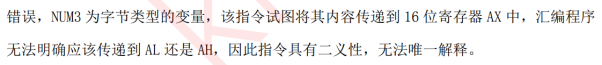
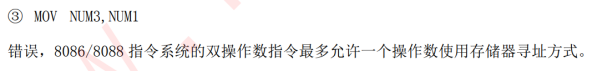


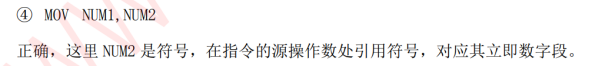


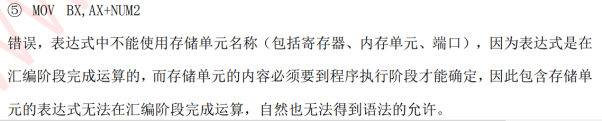


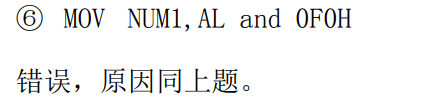


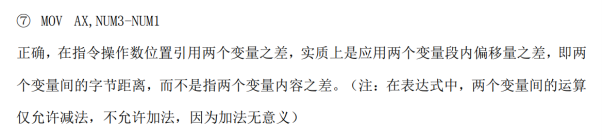


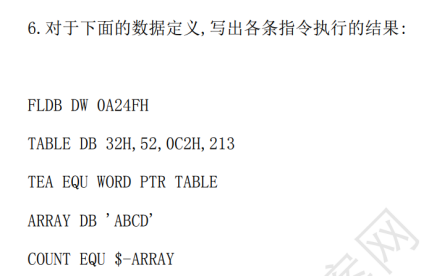














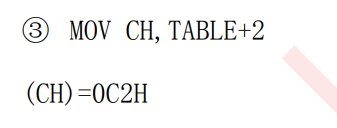
AX=0A24FH

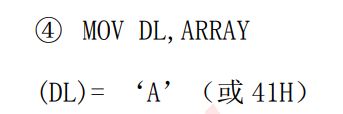




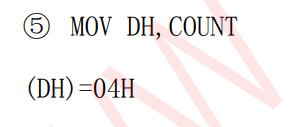
Equ是符号常量 TEA表示偏移量

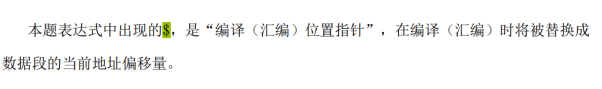
初始偏移量为0000->下一个偏移量为0002



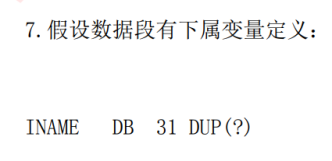


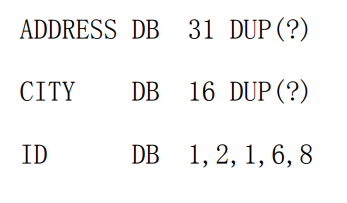
A的ascii码为65，16进制为41





0002+2=0004





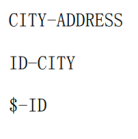




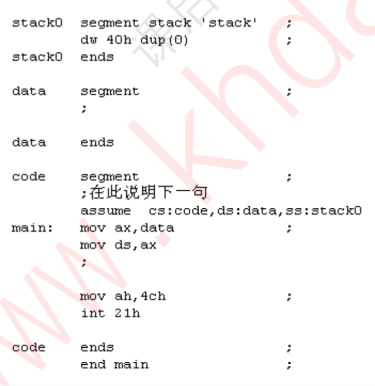


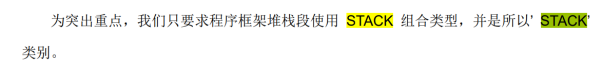




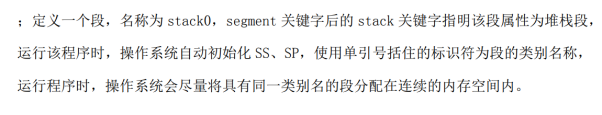




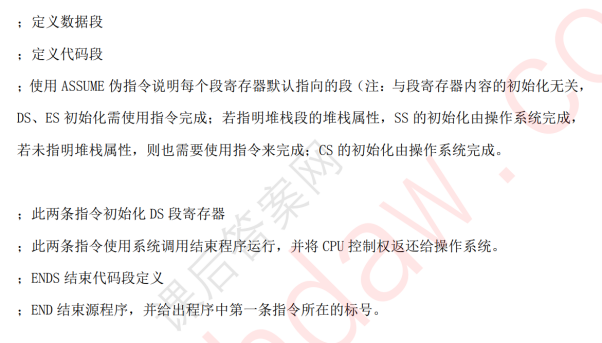


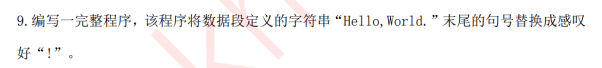












DATA SEGMENT

VAR1 DB ‘Hello，World.’

LEN EQU $-VAR1

DATA ENDS

CODE SEGMENT

ASSUME DS:DATA

BEGIN:MOV AX,DATA

MOV DS,AX

LEA SI,VAR1

ADD SI,LEN-1

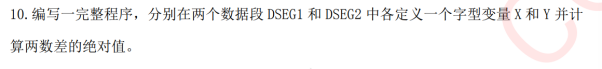
MOV BYTE PTR[SI], ‘!’

MOV AH,4CH

INT 21H

CODE ENDS

END BEGIN



YY1 SEGMENT

X DW 4723H

Z DW ?

YY1 ENDS

YY2 SEGMENT

Y DW 528AH

YY2 ENDS

CODE SEGMENT

ASSUME DS:YY1,ES:YY2

BEGIN:

MOV AX,YY1

MOV DS,AX

MOV AX,YY2

MOV ES,AX

SUB AX,Y

MOV Z,AX

MOV AH,4CH

INT 21H

CODE ENDS

END BEGIN