哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

汇编语言程序设计
第3讲：汇编语言程序格式

裴文杰

第3讲：汇编语言程序格式

汇编程序功能

汇编语言程序格式

伪指令

表达式操作符

wenjiecoder@outlook.com

汇编程序功能

汇编语言程序格式

伪指令

表达式操作符

wenjiecoder@outlook.com

汇编程序功能

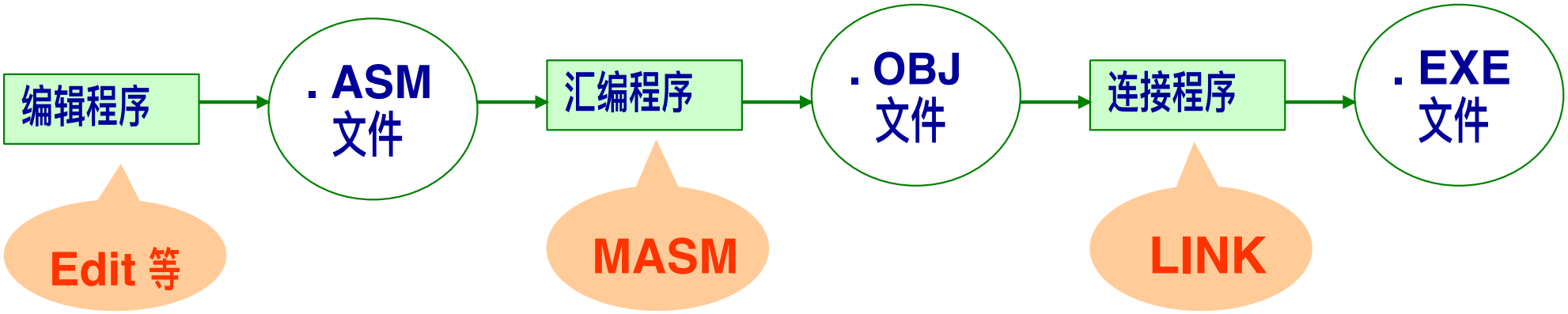
汇编语言是一种面向CPU指令系统的程序设计语言，它采用指令系统的助记符来表示操作码和操作数，用符号地址表示操作数地址。相对于机器语言，因而易记、易读、易修改，给编程带来很大方便。

用汇编语言编写的程序

能够直接利用硬件系统的特性，直接对位、字节、字寄存器、存储单元、I/O端口等进行处理，同时也能直接使用CPU指令系统和指令系统提供的各种寻址方式编制出高质量的程序，这种程序不但占用内存空间少，而且执行速度快。

wenjiecoder@outlook.com

汇编程序功能



汇编语言程序的建立和汇编过程

用汇编语言编写的源程序在输入计算机后，需要将其翻译成目标程序，计算机才能执行相应指令，这个翻译过程称为汇编，完成汇编任务的程序称为汇编程序。

汇编程序以汇编语言源程序文件作为输入，并由它产生两种输出文件：目标程序文件和源程序列表文件。

OBJ文件虽然已经是二进制文件，但它还不能直接运行，必须经过连接程序（link）把目标文件与库文件或其它目标文件连接在一起形成可执行文件（EXE文件）。

wenjiecoder@outlook.com

汇编程序功能

常用的汇编程序有Microsoft公司的宏汇编程序MASM（Macro Assembler）和Borland公司推出的TASM（Turbo Assembler）两种。

汇编程序的主要功能：

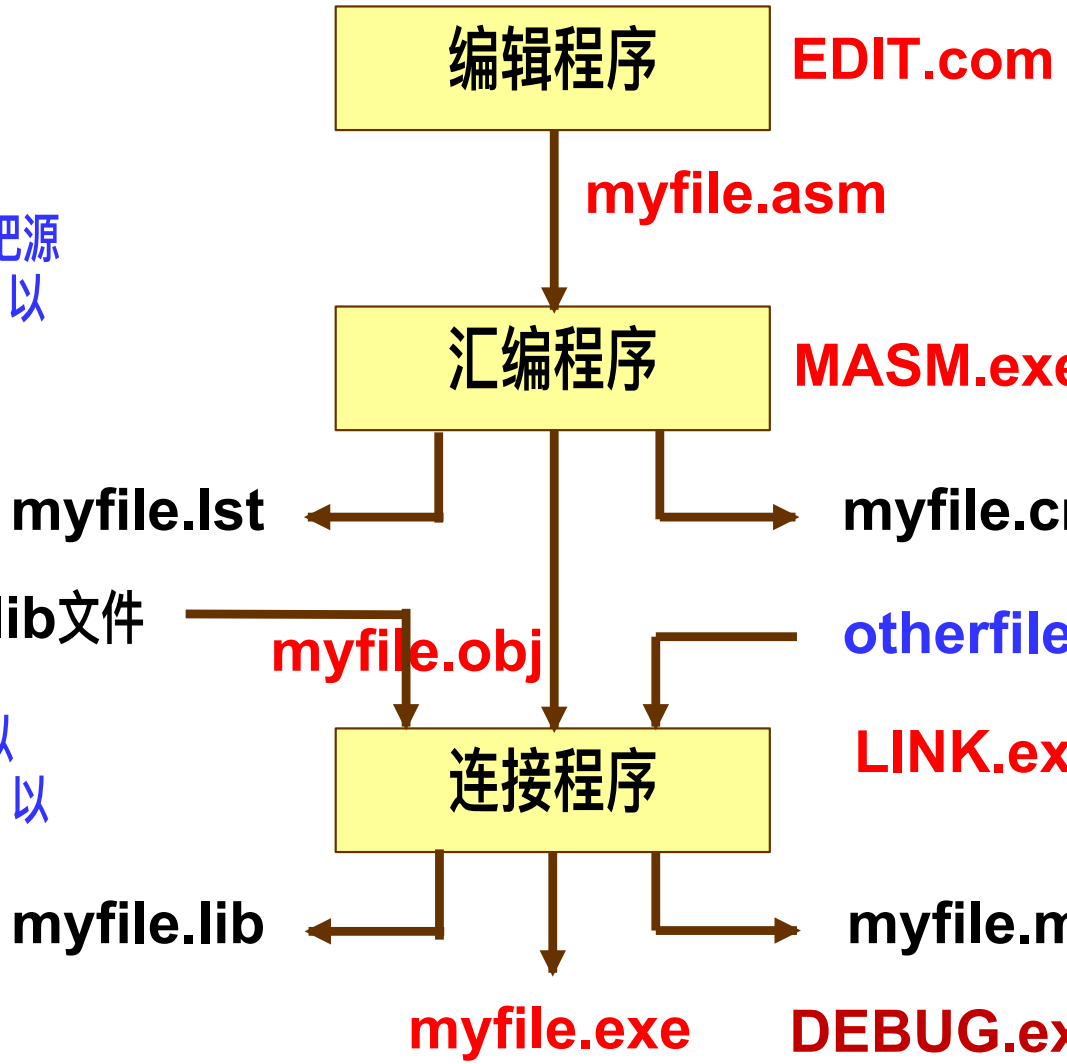
（1）检查源程序，给出出错信息。

（2）产生目标文件(.obj)和列表文件(.lst)。

（3）展开宏指令。

wenjiecoder@outlook.com

程序运行步骤及生成的文件



列表文件(*.LST)：把源程序和目标程序列表，以供检查程序用

交叉索引文件(*.CRF)：它是一个对源程序所用的各种符号进行前后对照的文件

比如多模块编程

库文件(*.LIB)：可以把函数编译成函数库，以便其他程序调用

映像文件(*.MAP)：是一种文本文件，列出各段在存储器中的分配情况

汇编程序功能

建立，
运行汇编语言程序

```
C>EDIT  MYFILE . ASM ↙
C>MASM  MYFILE . ASM ↙
    Microsoft (R) Macro Assembler Version 5.10
   Copyright (C) Microsoft Corp 1981,1988.All rights reserved.
    Object filename [MYFILE.OBJ]:  ↙
    Source listing  [NUL.LST]: MYFILE.LST ↙
    Cross-reference [NUL.CRF]:  ↙
        47962 + 413345Bytes symbol space free
            0 Warning Errors
            0 Severe  Errors
C>LINK  MYFILE . OBJ  ↙
    Microsoft (R) Overlay Linker  Version 3.64
   Copyright (C) Microsoft Corp 1983-1988. All rights reserved.
    Run File [MYFILE.EXE]:  ↙
    List File [NUL.MAP]:  ↙
   Libraries [.LIB]:  ↙
  LINK : warning L4021: no stack segment
   C>MYFILE  ↙
```

汇编程序功能

汇编语言程序格式

伪指令

表达式操作符

wenjiecoder@outlook.com

汇编语言语句的基本格式

语句是汇编语言源程序的基本组成单位。一个汇编语言
指令语句、伪指令语句和宏指令语句（宏指令语句就是
。前两种是最常见、最基本的语句。指令语句和伪指令
而且实现其功能的方法和时间也不同。

1、指令语句

指令语句就是计算机中指令系统的各条指令，每条指令
U执行的机器目标代码。

计算机中每条指令语句表示一种基本功能，这些基本功
件来实现的。一条指令语句由四个字段组成，其一般格式如下

[标号名：] 操作码 [操作数] [；注释]

例：result=a+b

```
data segment
  a       db  1
  b       db  2
  result  db  ?
  string  db  'result=$'
data ends
code segment
  assume cs:code, ds:data
start:  mov   ax,data
        mov   ds,ax
              mov   al,a
        add   al,b
        mov   result,al
        lea   dx,string
        mov   ah,09
        int   21h
        add   result,30h
        mov   dl,result
        mov   ah,2
        int   21h
              mov   ah,4ch
              int   21h
code    ends
        end   start
```

汇编语言语句的基本格式

2、伪指令语句

伪指令语句指示汇编程序在汇编源程序时完成某些工
储区、指示程序结束等。

伪指令属于汇编控制命令，它所指示的操作是由汇编
汇编时，它不产生目标代码，在将源程序汇编成目标程
一条伪指令语句也由四个字段组成，其一般格式如下：

[符号名] 伪操作码 操作数 [；注释]

由上可知，伪指令语句与指令语句的主要区别是：

- 伪指令语句经汇编后不产生机器目标代码，而指
器目标代码；
- 伪指令语句所指示的操作是在程序汇编时完成的
运行时才能完成。

例：result=a+b

```
data segment
  a       db  1
  b       db  2
  result  db  ?
  string  db  'result=$'
data ends
code segment
  assume cs:code, ds:data
start:  mov   ax,data
        mov   ds,ax
            mov   al,a
        add   al,b
        mov   result,al
        lea   dx,string
        mov   ah,09
        int   21h
        add   result,30h
        mov   dl,result
        mov   ah,2
        int   21h
            mov   ah,4ch
            int   21h
code    ends
        end   start
```

汇编语言语句的基本格式

在格式上，指令语句和伪指令语句略有差别。

指令性语句的格式如下：

[标号名：] 操作码 [操作数[，操作数]] [；注释]

伪指令语句的格式如下：

[符号名] 伪操作码 [操作数[，操作数，...]] [；注释]

wenjiecoder@outlook.com

汇编语言语句的基本格式

名字项（标号、符号名、变量等）：

- 名字也就是由用户按一定规则定义的标识符。
- 名字可用下列字符组成：
  - 英文字母（A～Z，a～z）
  - 数字（0~9）
  - 特殊字符（?、·、@、—、$）
- 名字的定义要满足如下规则：
  - 数字不能作名字项的第一个字符。
  - 圆点（.）仅能用作第一个字符。
  - 单独的“?”不能作为名字。
  - 汇编语言中有特定含义的保留字，如操作码、寄存器名等，不能作为名字使用。
  - 可以用很多字符来说明名字，但只有前面的31个字符能被汇编程序所识别。

例如：一些有效的名字

var123 Count myFile

_main $first .mylife

(合法但不可取)

wenjiecoder@outlook.com

汇编语言语句的基本格式

标号:

标号用来代表一条指令所在单元的地址，在代码段中定义及使用。

标号放在语句的前面，并用冒号“:”与操作项分开。

标号不是每条指令所必需的，它也可以用LABEL或EQU伪指令来定义。

标号经常在转移指令或循环指令或CALL指令的操作数字段出现，用以表示转向的目标地址。

wenjiecoder@outlook.com

汇编语言语句的基本格式

标号:

```
        MOV   CX，COUNT                ；设置循环次数
        MOV   SI，OFFSET STRING
        XOR   BX，BX                   ；BX清0，用于记录空格数
        MOV   AL，20H                  ；空格的ASC码为20H
AGAIN:  CMP   AL，ES:[SI]
        JNZ   NEXT                     ；ZF=0，非空格，转移
        INC   BX                       ；ZF=1，是空格，个数加1
NEXT:   INC   SI
        LOOP  AGAIN                    ；字符个数减1，不为0继续循环
        MOV   RESULT，BX               ；保存结果
```


wenjiecoder@outlook.com

汇编语言语句的基本格式

变量：

变量在数据段、附加段和堆栈段中定义，

它也可以用LABEL或EQU

伪指令来定义。变量是一个可以存放

即存放数据的存储单元的地址符号名

变量可以是用DB、DW、DD定义的字

可以被定义为一个数据区（有具体数值

区域，而不指定具体的数值）。此时

变量名仅表示该数据区或存储区的第

。变量经常在操作数字段出现。

例：result=a+b

```
data segment
  a       db  1
  b       db  2
  result  db  ?
  string  db  'result=$'
data ends
code segment
  assume cs:code, ds:data
start:  mov   ax,data
        mov   ds,ax
            mov   al,a
        add   al,b
        mov   result,al
        lea   dx,string
        mov   ah,09
        int   21h
        add   result,30h
        mov   dl,result
        mov   ah,2
        int   21h
            mov   ah,4ch
            int   21h
code    ends
        end   start
```


wenjiecoder@outlook.com

汇编语言语句的基本格式

标号和变量都具有三种属性：段属性、偏移属性及类型属性

段属性定义标号或变量的段起始地址，此值必须在一个段寄存器中。

偏移属性

是标号或变量所在的地址距段基址的偏移量。它们通常在指令中以显式方式出现，并最终能确定其有效地址EA。

类型属性

标号的类型属性用来指出该标号是在本段内引用还是在其它段中引用。如在段内引用的，则称为NEAR，指针长度为2个字节；如在段外引用，则称为FAR，指针长度为4个字节。

变量的类型属性定义该变量所保留的字节数。如BYTE（1字节），WORD（2字节），DWORD（4字节），FWORD（6字节），QWORD（8字节），TBYTE（10字节）。

在同一个程序中，同样的标号或变量的定义只允许出现一次，否则汇编程序会提示出错。

wenjiecoder@outlook.com

汇编语言语句的基本格式

操作项：

操作项可以是指令、伪指令或宏指令的操作码，或称助记符。

对于指令：汇编程序将其翻译成机器语言指令。

对于伪指令：汇编程序将根据其所要求的功能进行处理。

对于宏指令：则将根据其定义展开。（后续章节介绍）

助记符表示指令语句的功能，如INC，MOV等。

操作项是由系统定义的，编程时必须照写不误，既不能多写，也不能少写。

wenjiecoder@outlook.com

汇编语言语句的基本格式

操作数

操作数项是操作项的操作对象，可以是数据本身、标号、寄存器名字或算术表达式。

该项由一个或多个表达式组成，多个操作数项之间一般用逗号分开。

对于指令，操作数项一般给出操作数地址，它们可能有一个、两个、三个、或一个也没有。

对于伪指令或宏指令则给出它们所要求的参数。

操作数表达式是由运算量和运算符组成的表示操作数或操作数地址的运算式。

wenjiecoder@outlook.com

汇编语言语句的基本格式

常量

常量是没有任何属性的纯数值，它的值在汇编期间已能完全确定，且在程序运行中也不会发生变化。常量分为数值常量、字符串常量和符号常量，它主要用于指令语句中的立即数或伪指令语句中给变量赋初值等。

1）数值常量：

- 基数后缀：h、q/o、d、b
- 如果整数常量后面没有后缀，就被认为是十进制的
- 以字母开头的十六进制常量前面必须加一个0

例如：

26 十进制　　26d 十进制　　11011110b 二进制
42q 八进制　　42o 八进制　　1Ah 十六进制　　0A3h 十六进制

wenjiecoder@outlook.com

汇编语言语句的基本格式

2）字符串常量

字符串常量是用单引号或双引号括起来的一个字符或多个字符。

字符串常量以单引号或双引号中各字符的ASCII码形式存储在内存中，如‘H’，在内存中就是48H，‘12’就是31H，32H。使用时可在单引号内直接写字符序列，如‘12AB’，也可写字符的ASCII码，ASCII码之间用逗号分隔（此时不需要用单引号），如31H，32H，41H，42H，该序列等同字符串‘12AB’。

引号可以嵌套：

“This isn’t a test”

‘Say “Goodnight,” Gracie’

3）符号常量

符号常量是指用EQU伪指令或赋值语句“=”定义过的符号名，可作操作数项或在表达式中使用。

wenjiecoder@outlook.com

汇编程序功能

汇编语言程序格式

伪指令

表达式操作符

wenjiecoder@outlook.com

伪指令

伪指令无相应的目标代码，因此也称为伪操作。

伪指令语句又称为说明性语句或管理语句。它不同于指令性语句，不是直接命令CPU去执行某一操作，而是命令汇编程序应当如何生成目标代码。

伪指令是汇编程序对源程序进行汇编时处理的操作，完成逻辑段的定义、存储模式定义、数据定义、存储器分配、指示程序开始结束等功能。

wenjiecoder@outlook.com

伪指令

常用伪指令：

- 段定义伪操作
- 程序开始和结束伪操作
- 数据定义及存储器分配伪操作
- 表达式赋值伪操作
- 地址计数器与对准伪操作
- 基数控制伪操作

有关过程定义、宏汇编的伪指令将在后续章节介绍。

wenjiecoder@outlook.com

伪指令

♦ 完整的段定义伪指令：

汇编语言源程序是用分段的方法来组织程序、数据
千个逻辑段组成。

段定义伪指令的格式如下：

```
段名  SEGMENT  [定位类型]  [组合类型]  [使
        ……
        ……                          ;语句序列
段名  ENDS
```

SEGMENT和ENDS前边的段名表示定义的逻
将无法辨认。起什么名字可由程序员自行决定，
伪指令重名。

当定义除代码段以外其他段时，段内不能包括指令语

例：result=a+b

```
data segment
  a       db  1
  b       db  2
  result  db  ?
  string  db  'result=$'
data ends
code segment
  assume cs:code, ds:data
start:  mov   ax,data
        mov   ds,ax
            mov   al,a
        add   al,b
        mov   result,al
        lea   dx,string
        mov   ah,09
        int   21h
        add   result,30h
        mov   dl,result
        mov   ah,2
        int   21h
            mov   ah,4ch
            int   21h
code    ends
        end   start
```


wenjiecoder@outlook.com

伪指令

```
;*********************************
data_seg1 segment     ;定义数据段
  .
  .
 data_seg1ends
;*********************************
 data_seg2 segment              ;定义附加段
  .
  .
data_seg2 ends
;*********************************
code_seg segment               ;定义代码段
start:                    ;程序执行的起始地址
        .
code_seg ends                ;  代码段结束
   ;*********************************
   end    start                    ;源程序结束，且start表示程序执行的入口.
```


wenjiecoder@outlook.com

伪指令

```
code_seg segment
        mov ax, datas
        mov ds, ax
    ......
code_seg ends
    end
```

区别?

```
 code_seg segment
start:
        mov ax, datas
        mov ds, ax
    ......
 code_seg ends
      end   start
```

code_seg ends 表示代码段结束,

End表示源程序结束。

start：一个标号，定义了程序的入口，既程序从start :处开始执行，

若程序的第一条指令就是程序的入口，则start可以缺省。

其中 start 可以用其他字符代替，但是对应的end start 中的start 也必须用同字符代替。

若第一个start缺省，则end start中的 start 也必须去掉。

```
 code_seg segment
        mov ax, datas
start:
        mov ds, ax
    ......
 code_seg ends
      end   start
```


wenjiecoder@outlook.com

伪指令

♦ ASSUME伪指令：

段定义后，还要规定段的性质，也就是要明确段和段寄存器的关系，这可用ASSUME伪指令来实现。

ASSUME伪指令的格式为：

ASSUME　段寄存器：段名[，段寄存器：段名……]

或者：

ASSUME　段寄存器：NOTHING　；取消原段寄存器的指定

段寄存器可以是CS、DS、ES、SS。

ASSUME伪指令只是指出各逻辑段应该装填的地址，但并未真正将段基址装入相应的段寄存器中，所以在程序的代码段开始处就应该先进行DS、ES、SS段基址的装填，否则无法正确对数据进行寻址操作。CS由系统自动装填。

堆栈段SS也可以不用用户装填，而由系统自动装填。但是在定义堆栈段时，必须把参数写全。其格式为：

STACK SEGMENT PARA PUBLIC 'STACK'

wenjiecoder@outlook.com

伪指令

♦ 完整的段定义伪指令:

```
data   segment        ; 定义数据段
       …
data   ends
;--------------------------------------
extra  segment        ; 定义附加段
       …
extra  ends
;--------------------------------------
code   segment        ; 定义代码段
       assume cs:code, ds:data, es:extra
start:
       mov   ax, data
       mov   ds, ax     ; 数据段段地址 ? 段寄存器
       mov   ax, extra
       mov   es, ax     ; 附加段段地址 ? 段寄存器

       …
code   ends
       end   start
```

段名将被编译为一个表示段地址的数值，而立即数不能直接送段寄存器

wenjiecoder@outlook.com

伪指令

- **简化的段定义伪指令：**

```
.code [name]          ；代码段
.data                 ；初始化数据段
.data?                ；未初始化数据段
.fardata [name]       ；远（调用）初始化数据段
.fardata? [name]      ；远（调用）未初始化数据段
.const                ；常数段
.stack [size]         ；堆栈段
```

这种分段方法把数据段分的更细：

- 初始化数据段和未初始化数据段分开。
- 近和远的数据段分开。
- 常数段和数据段分开。
- 便于与高级语言兼容。若汇编程序独立，则不必细分。

wenjiecoder@outlook.com

伪指令

```
DATA SEGMENT
            ;此处输入数据段代码
DATA ENDS

STACK SEGMENT 'STACK'
            ;此处输入堆栈段代码
STACK ENDS

CODE SEGMENT
ASSUME CS:CODE,DS:DATA,SS:STACK
START:
   MOV AX, DATA
   MOV DS, AX
            ;此处输入代码段代码
   MOV AH, 4CH
   INT 21H
CODE ENDS
   END START
```

在简化段定义中，如何在存储器中组织安放各个段，即存储模式

```
.MODEL SMALL
.DATA
            ;此处输入数据段代码

.STACK
            ;此处输入堆栈段代码

.CODE
START:
   MOV AX, @DATA
   MOV DS, AX
            ;此处输入代码段代码
   MOV AH, 4CH
   INT 21H
   END START
```

用预定义符号@DATA给出了数据段的段名

使用简化段定义时，必须有存储模式.MODEL语句，且位于所有简化段定义语句之前。

伪指令

注意：

完整段定义利用SEGMENT和ENDS
一对伪指令定义逻辑段。同时需要配合ASSUME
伪指令指明逻辑段是代码段、堆栈段、数据段还是附加段。

完整段定义的优势是可以指明逻辑段的定位、组合、类别等属性。

而简化段定义只能采用系统默认的属性。

简化的段定义ASSUME伪指令可以省掉（如果只有.Data, .STACK, .CODE）。

完整段定义和简化段定义的实质是一致的。

wenjiecoder@outlook.com

伪指令

♦ 存储模式伪指令

MODEL 伪指令

.MODEL 存储模式 [,语言类型] [,操作系统类型] [,堆栈选项]

存储模式：tiny 、small、medium、compact、large、huge、flat

语言类型：C、BASIC、PASCAL等 （被高级语言调用时）

操作系统：OS_DOS（默认）或OS_OS2

堆栈选项：NEARSTACK或FARSTACK（其中NEARSTACK是指把堆栈段和数据段组合到一个DGROUP中：即DS，SS均指向DGROUP段；FARSTACK指堆栈段和数据段不合并。）

wenjiecoder@outlook.com

伪指令

存储模式	特 点
TINY（微型模式）	COM类型程序，只有一个小于64KB的逻辑段
SMALL （小型模式）	小应用程序，只有一个代码段和一个数据段（含堆栈段），每段不大于64KB
COMPACT （紧凑模式）	代码少、数据多的程序，只有一个代码段，但有多个数据段
MEDIUM （中型模式）	代码多、数据少的程序，可有多个代码段，只有一个数据段
LARGE （大型模式）	大应用程序，可有多个代码段和多个数据段（静态数据小于64KB）
HUGE （巨型模式）	更大应用程序，可有多个代码段和多个数据段（对静态数据没有限制，可超过64KB）
FLAT （平展模式）	32位应用程序，运行在32位80x86CPU和Windows 9x或NT环境

wenjiecoder@outlook.com

伪指令

例如：

.Model small, C

. Model large, pascal, os_dos, farstack

存储模式为Tiny, Small, Medium,
Flat时，默认项为NEARSTACK（DS和SS合并）。

存储模式为Compact, Large,
Huge时，默认项为FARSTACK（DS和SS不合并）。

wenjiecoder@outlook.com

伪指令

♦ 段组定义伪操作(GROUP):

Groupname　　Group　segname1, segname2…

```
dseg1  segment
       ……
dseg1  ends
dseg2  segment
       ……
dseg2  ends
datagroup  group  dseg1, dseg2
cseg   segment
       assume   cs:cseg, ds: datagroup
start:
       mov  ax, datagroup
       mov  ds, ax
       ……
       mov  ax, 4c00h
       int  21h
cseg   ends
       end  start
```

共用一个段寄存器

伪指令

◆ 数据定义及存储器分配伪操作：

[变量] 助记符 操作数 [, 操作数 , …] [; 注释]

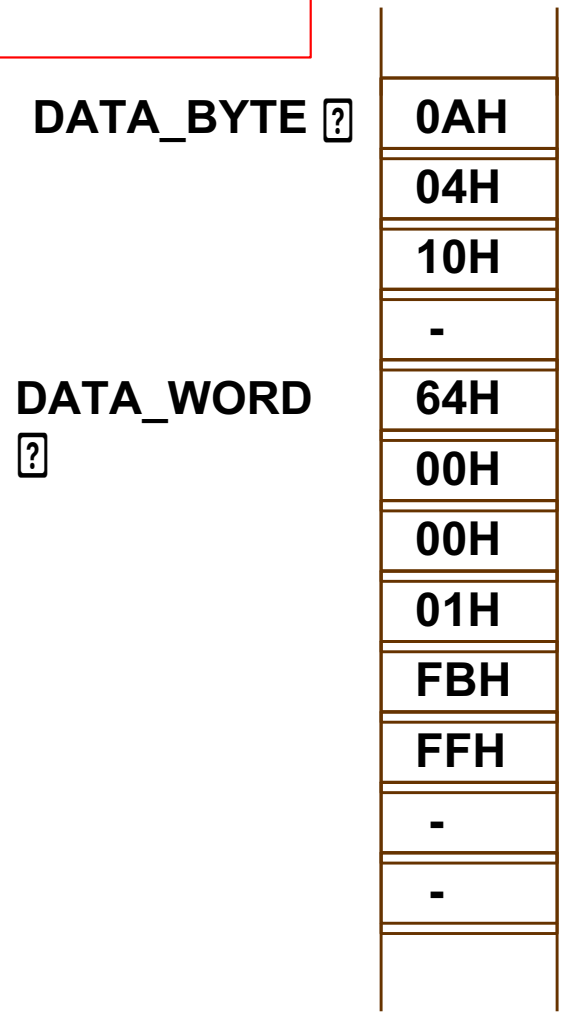
助记符： DB DW DD DF DQ DT

变量指向第一个字节的偏移地址

? :
只分配存储空间并不存入确定的值，形成未初始化数据

例：

```
DATA_BYTE  DB  10,4,10H,?
DATA_WORD  DW  100,100H,-5,?
```



伪指令

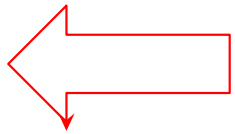
伪操作	含义	说明
DB	一个操作数占用一个字节单元，定义的变量为字节变量。	可用来定义字符串。
DW	一个操作数占用一个字单元（两个字节单元），定义的变量为字变量。	在内存中存放时，低字节在低地址单元，高字节在高地址单元。
DD	一个操作数占用一个双字单元（4 个字节单元），定义的变量为双字变量。	在内存中存放时，低字节在低地址单元，高字节在高地址单元。
DF	一个操作数占用一个三字单元（6 个字节单元），定义的变量为三字变量。	该伪操作符仅用于 80386 以上 CPU，定义的变量作为指针使用，其低 4 字节存放偏移地址，高 2 字节存放段地址。
DQ	一个操作数占用一个四字单元（8 个字节单元），定义的变量为四字变量。	在内存中存放时，低字节在低地址单元，高字节在高地址单元。
DT	一个操作数占用 10 个字节单元，定义的变量为十字节变量。	使用该伪操作符时，对于十进制操作数，必须给出后缀 D，没有后缀则默认为压缩 BCD 码。

伪指令

ARRAY DB ‘HELLO’

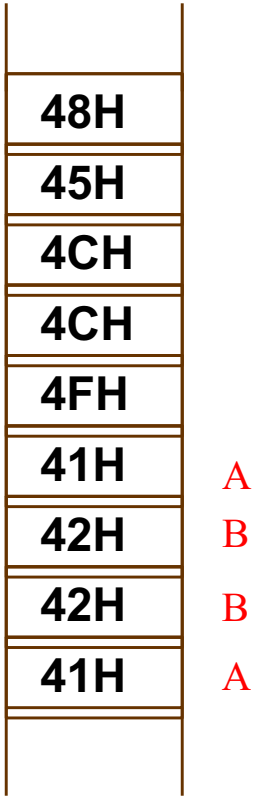
DB ‘AB’

DW ‘AB’



区别?

ARRAY ?



Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂

伪指令

◆ 复制操作符（与前面的数据定义操作符联合应用）：

Repeat_count DUP (操作数1, 操作数2, …) [; 注释]

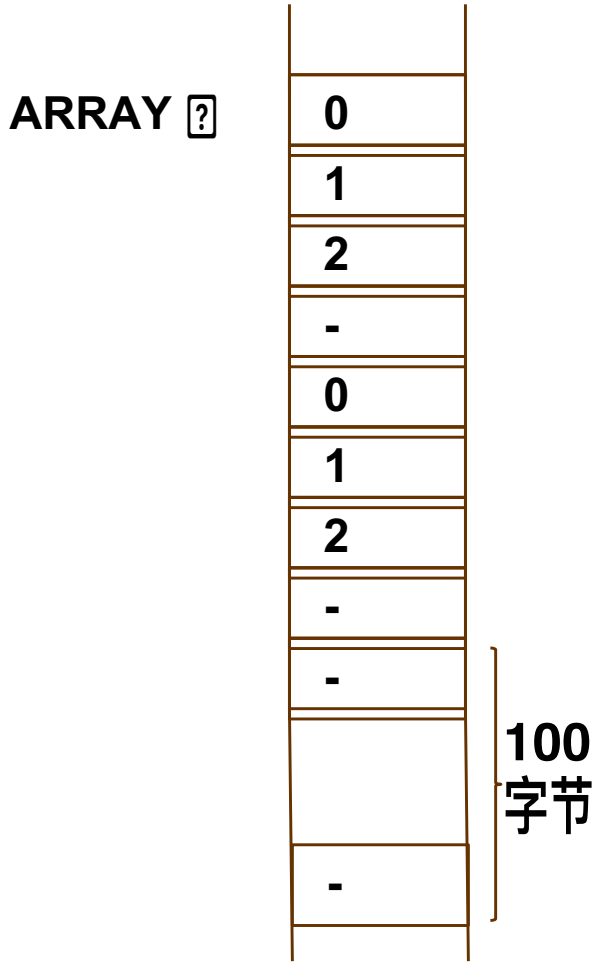
例:

ARRAY1 DB 2 DUP(0,1,2,?)

ARRAY2 DB 100 DUP(?)

DUP指令可以嵌套使用:

ARRAY3 DB 2 DUP(0,2 DUP(1,2),0,3)



伪指令

PTR伪指令： type PTR 变量或常量

指定操作数的类型属性

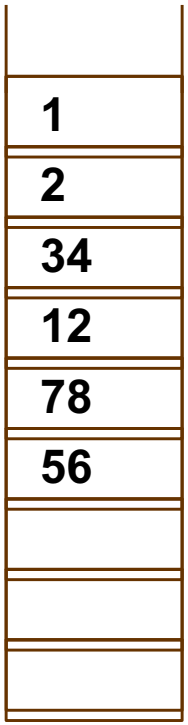
，其中type可以是BYTE，WORD，DWORD，FWORD，QWORD，TBYTE等等

```
OPER1  DB  ?, ?
OPER2  DW  ?, ?
       ……
       MOV  OPER1, 0   ;字节指令
       MOV  OPER2, 0   ;字指令

OPER1  DB  1, 2
OPER2  DW  1234H, 5678H
       ……
       MOV  AX, OPER1+1
       MOV  AL, OPER2          ;X
                               ;X   类型不匹配
```

OPER1 ?

OPER2 ?




```
MOV  AX, WORD PTR  OPER1+1
```

(AX)=3402H

```
MOV  AL, BYTE PTR  OPER2
```

(AL)=34H

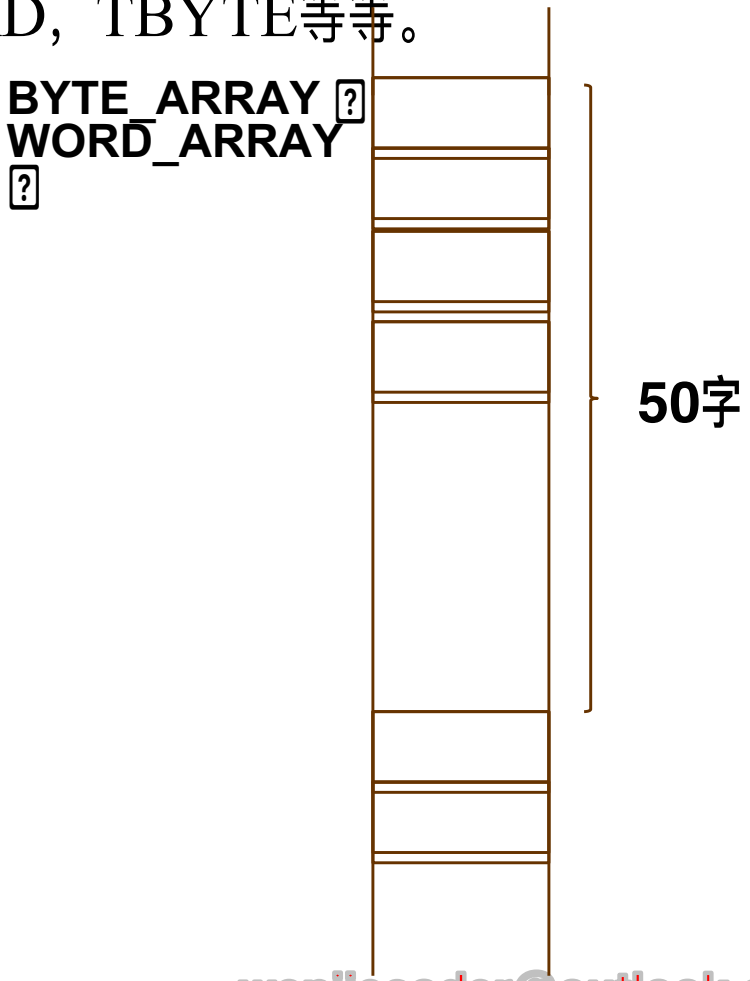
伪指令

LABEL 伪操作： name LABEL type

除了PTR伪操作，还可以用LABEL来定义操作数类型，其中type可以是BYTE，WORD，DWORD，FWORD，QWORD，TBYTE等等。

例:

```
BYTE_ARRAY  LABEL  BYTE
WORD_ARRAY  DW  50  DUP (?)
```



思考:

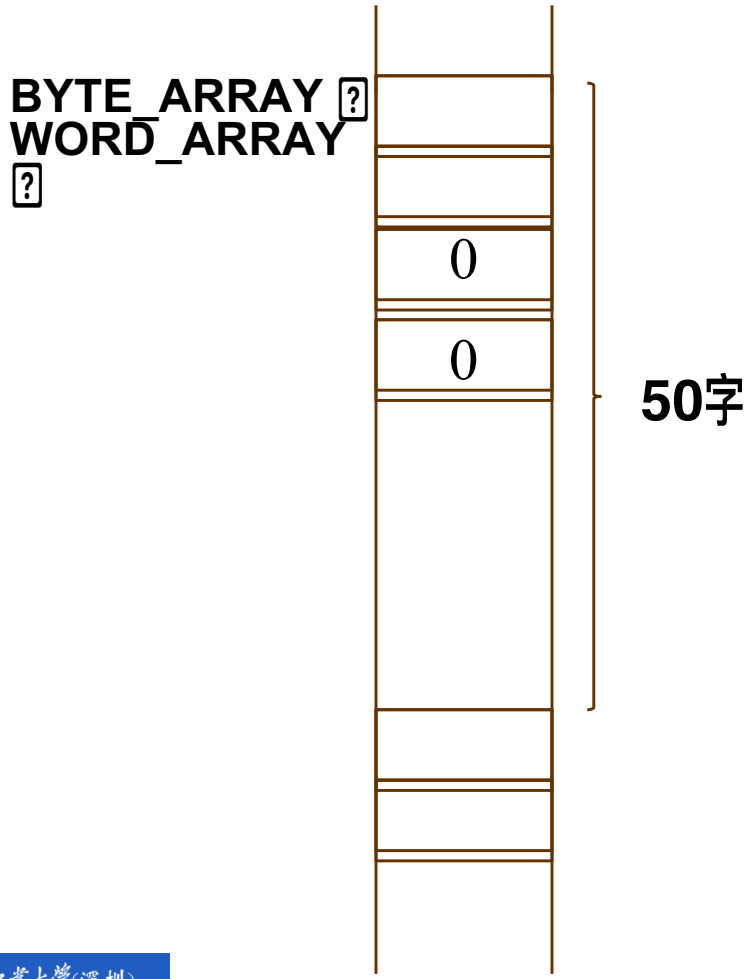
```
MOV WORD_ARRAY+2，0
MOV BYTE_ARRAY+2，0
```

分别执行的结果是?

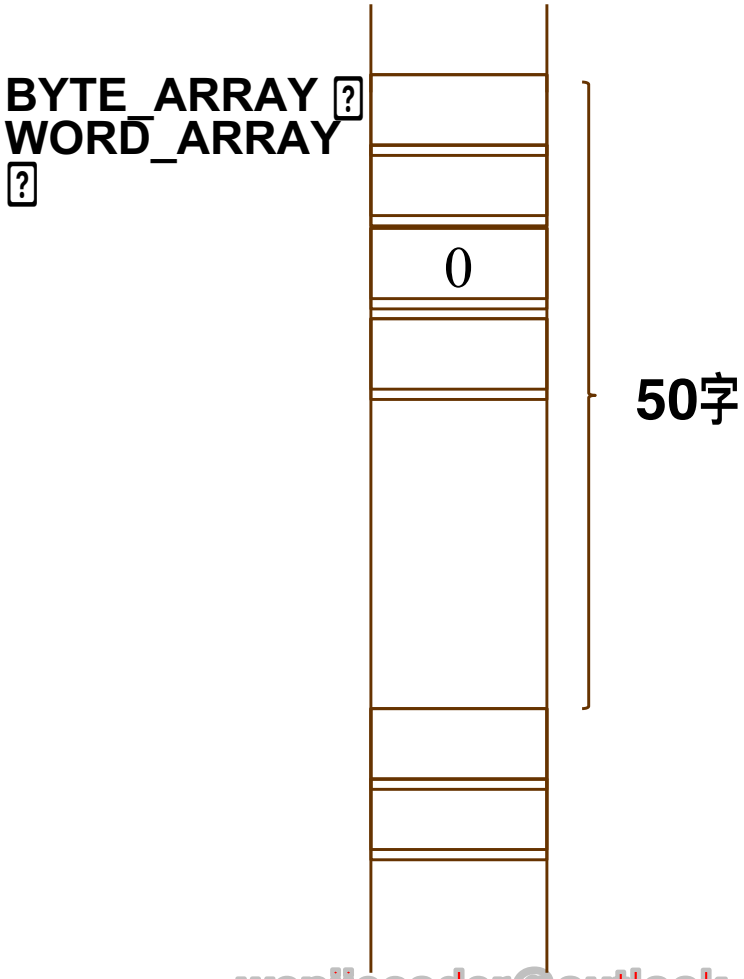
伪指令

思考:

MOV WORD_ARRAY+2, 0



MOV BYTE_ARRAY+2, 0



伪指令

当同一个表达式在同一个程序中出现很多次时

表达式赋值伪操作：

- EQU伪指令的功能是给各种形式的表达式赋予一个名字。

表达式一旦赋予了一个名字，在以后的程序语句中，凡是出现该表达式的地方，均可用它的名字来代替。格式：

表达式名　EQU　表达式

上式中的表达式可以是任何有效的操作数格式，可以是任何可以求出常数值的表达式，也可以是任何有效的助记符。

```
ALPHA   EQU  9
BETA    EQU  ALPHA+18
BB      EQU  [BP+8]
```

功能与“=”相似，区别在于EQU不允许重复定义，而“ = ”伪操作（允许重复定义）：

```
……
EMP = 7
……
EMP = EMP+1
……
```


wenjiecoder@outlook.com

伪指令

◆ 地址计数器与对准伪操作：

地址计数器 $

：保存当前正在汇编的指令的偏移地址。每处理一条指令，$就增加一个值（此值为该指令所需要的字节数）。

- 当$

  用在指令中时，它表示本条指令的第一个字节的地址

  。例：

```
JNE  $+6        ;转向地址是 JNE 的首址 +6
```

- $

  用在伪操作的参数字段时，表示地址计数器的当前值

  。例：

```
ARRAY   DW   1, 2 , $+4 , 3 , 4 , $+4
```

如汇编时，ARRAY的首地址是0074。

单元	地址
01	0074
00	0075
02	0076
00	0077
7C	0078
00	0079
03	007A
00	007B
04	007C
00	007D
82	007E
00	007F

伪指令

ORG 伪指令：ORG 数值表达式

- ORG伪指令用来设置当前地址计数器的值。
- 其
  功能是告诉汇编程序，其后的指令或数据从数值表达式所指定的偏移地址开始存放。直到遇到下一个ORG命令。
- 表达式的值在0000H~FFFFH（0?65535）之间。
- 例：

```
SEG1    SEGMENT
        ORG   10
        VAR1  DW  1234H
        ORG   20
        VAR2  DW  5678H
        ORG   $+8
        VAR3  DW  1357H
SEG1    ENDS
```

VAR1的偏移地址是10

建立一个8字节的未初始化的数据缓冲区

```
Buffer  Label  BYTE
        ORG    $+8
```

↓

```
Buffer  DB  8  DUP(?)
```


wenjiecoder@outlook.com

伪指令

◆ 基数控制伪操作：

- 汇编语言默认的数是十进制，.RADIX伪指令可以把默认的基数改变为2~16范围内的任何基数。
- 指令格式：. RADIX 表达式

```
MOV  BX, 0FFH
MOV  BX, 178
```

⟺

```
.RADIX  16
MOV  BX, 0FF
MOV  BX, 178D
```

注意：在用“.RADIX 16”
把基数改为16进制后，十进制后面的数都应该加D。在这种情况下，如果某个16进制的末尾字符为D，则应在其后跟字母H，以免产生混淆。

第3讲：汇编语言程序格式

汇编程序功能

汇编语言程序格式

伪指令

表达式操作符

wenjiecoder@outlook.com

表达式操作符

汇编指令：[标号：] 操作码 [操作数] [；注释]

-

  其中的操作数项可以是常数、寄存器、标号、变量、或表达式。这里将专门对表达式加以说明。

- 表达式是常数、寄存器、标号、变量与一些操作符组合的序列，可以有数字表达式和地址表达式两种。

算术操作符： 、 、 、 、Mod

算术操作符可以用于数字表达式或地址表达式，但用于地址表达式时，只有其结果有明确的物理意义时才是有效的结果。例如：地址+或-是有意义的，但两个地址*或/是无意义的。

计算数组长度

```
ARRAY   DW   1,2,3,4,5,6,7
ARYEND  DW   ?
MOV  CX, (ARYEND-ARRAY)/2
```

```
ADD  AX, BLOCK+2   ；符号地址常数 有意义
                   ;  时意义不明确
MOV  AX, BX+1      ;
MOV  AX, [BX+1]    ; 寄存器间接寻址
```

表达式操作符

关系操作符：EQ、NE、LT、LE、GT、GE

指令格式：变量1　EQ　　变量2

计算结果为逻辑值：真　0FFFFH

假　0000H

```
MOV AX, (OFFSETY - OFFSETX) LE 128

若 ?128 (真)  汇编结果：  MOV  AX, 0FFFFH
若 ?128 (假)  汇编结果：  MOV  AX, 0
```

```
MOV BX, ((val1 LT 5) AND 20) OR ((val1 GE 5) AND 30)

若val1<5, 汇编结果：MOV  BX, 20
否则，汇编结果：     MOV  BX, 30
```

表达式操作符

逻辑和移位操作符：AND、OR、XOR、NOT

指令格式：变量1　AND(OR, XOR)　变量2

NOT　变量

```
OPR1  EQU  25      ; 00011001B
OPR2  EQU  7       ; 00000111B

ADD  AX, OPR1 AND OPR2              ; ADD AX, 1
```

表达式操作符

数值回送操作符：TYPE、LENGTH、SIZE、OFFSET、SEG

指令格式：操作符　变量

- TYPE：返回变量以字节数表示的类型

DB	DW	DD	DF	DQ	DT	NEAR	FAR	常数
1	2	4	6	8	10	-1	-2	0

- LENGTH：对于变量中使用DUP的情况，返回分配给该变量的单元数，其他情况返回1.
- SIZE：返回分配给该变量的字节数，等于TYPE值*LENGTH值。
- OFFSET:回送变量或标号的偏移地址
- SEG：回送变量或标号的段地址

wenjiecoder@outlook.com

表达式操作符

例:

```
Data segment
        ARRAY  DW   100 DUP (?)
        TABLE  DB   'ABCD'
Data ends
…
ADD  SI,  TYPE    ARRAY          ; ADD  SI, 2
ADD  SI,  TYPE    TABLE          ; ADD  SI, 1
MOV  CX,  LENGTH  ARRAY          ; MOV  CX, 100
MOV  CX,  LENGTH  TABLE          ; MOV  CX, 1
MOV  CX,  SIZE    ARRAY          ; MOV  CX, 200
MOV  CX,  SIZE    TABLE          ; MOV  CX, 1

MOV BX, OFFSET TABLE             ; 返回变量TABLE的偏移地址

MOV BX, SEG TABLE                ; 返回变量TABLE所在段的段地址
```


wenjiecoder@outlook.com

表达式操作符

属性操作符：SHORT、HIGH、LOW、HIGHWORD、LOWWORD

- SHORT：用来修饰跳转指令中转向地址的属性，指出转向地址在下一条指令的-127~+127个字节范围之内。如：

```
JMP  SHORT  NEXT
```

- HIGH和LOW

：字节分离操作符。对一个数或表达式，HIGH取其高字节，LOW取其低字节。如：

```
CONS  EQU  1234H
MOV   AH, HIGH  CONS
MOV   AL, LOW   CONS
```

- HIGHWORD、LOWWORD：字分离操作符。对一个数或表达式，HIGHWORD取其高位字，LOWWORD取其低位字。

wenjiecoder@outlook.com

表达式操作符

运算符的优先级别

运算符	优先级
圆括号中的项、方括号中的项、LENGTH、SIZE、WIDTH、MASK	从高到低
段超越前缀符（:）	
PTR、OFFSET、SEG、TYPE、THIS	
HIGH、LOW	
乘法和除法：*、/、MOD	
加法和减法：+、-	
关系操作：EQ、NE、LT、GT、LE、GE	
逻辑：NOT	
逻辑：AND	
逻辑：OR、XOR	
SHORT	

wenjiecoder@outlook.com

第3讲作业：

Page 155 -158：4.1、4.14

wenjiecoder@outlook.com