第4章
汇编语言程序格式

- 4.1 汇编程序功能
- 4.2 伪操作
- 4.3 汇编语言程序格式
- 4.4 汇编语言程序的上机过程

4.1 汇编程序功能

- 运行汇编语言程序的步骤

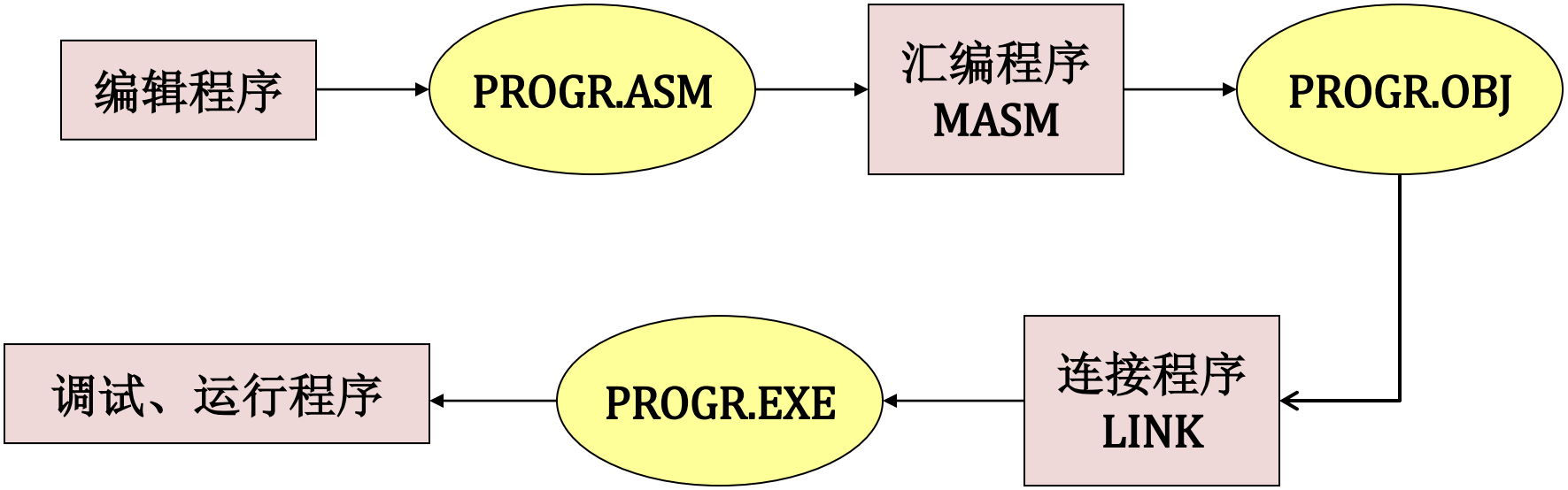


图4.1 汇编语言程序的建立及汇编过程

4.2 伪操作

- 4.2.1 处理器选择伪操作
- 4.2.2 段定义伪操作
- 4.2.3 程序开始和结束伪操作
- 4.2.4 数据定义及存储器分配伪操作
- 4.2.5 表达式赋值伪操作
- 4.2.6 地址计数器与对准伪操作
- 4.2.7 基数控制伪操作

伪操作

- 汇编语言程序的语句的组成
  - 指令: 在程序运行期间由计算机执行
  - 伪操作: 又称伪指令，在汇编程序对源程序汇编期间由汇编程序处理的操作
  - 宏指令: 在汇编程序对源程序汇编期间由汇编程序展开宏操作

4.2.1 处理器伪操作

.8086	选择8086指令系统
.286	选择80286指令系统
.286 P	选择保护方式下的80286指令系统
.386	选择80386指令系统
.386 P	选择保护方式下的80386指令系统
.486	选择80486指令系统
.486 P	选择保护方式下的80486指令系统
.586	选择Pentium指令系统
.586 P	选择保护方式下的Pentium指令系统

4.2.2 段定义伪操作

- 1. 完整的段定义伪操作
- 2. 存储模型与简化段定义伪操作
- 3. 段组定义伪操作

1. 完整的段定义伪操作

(1) 段定义伪操作

(2) SEGMENT伪操作类型及属性

(1) 段定义伪操作

- 格式：

```
segment_name    SEGMENT
        . . .
segment_name    ENDS
```

- 说明：

  删节号部分，对于数据段、附加段和堆栈段来说，一般是存储单元的定义、分配伪操作；对于代码段则是指令及伪操作

明确段和段寄存器的关系

- ASSUME伪操作格式：

  ASSUME segreg: seg_name, …

  – 段寄存器名必须是CS、DS、ES或SS

  – 段名必须是由SEGMENT定义的段中的段名

段地址装入段寄存器

- 指令：

  MOV　　ax , segment_name

  MOV　　ds , ax

- 代码段不需要这样做

例4.1

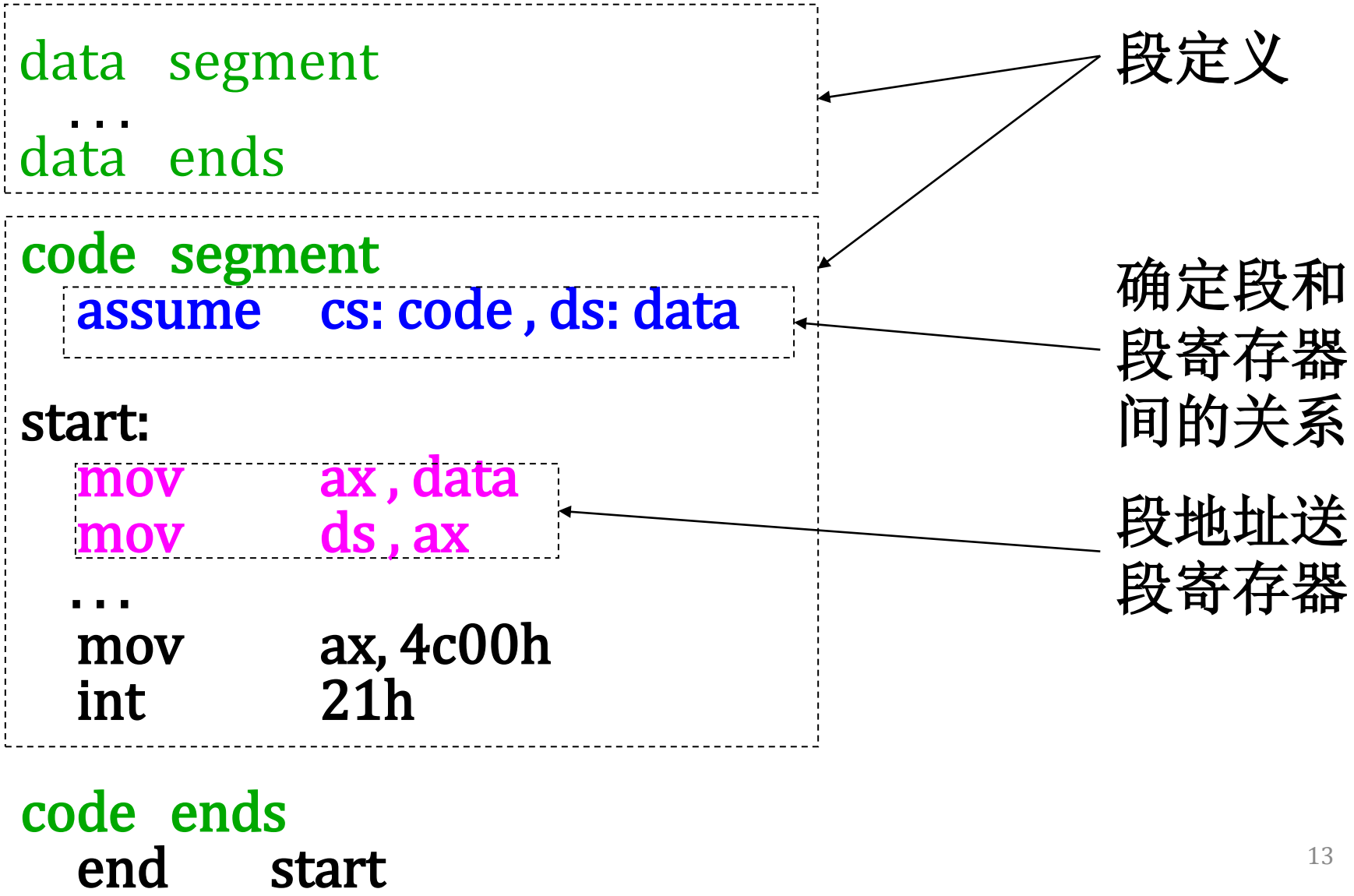
```
data   segment
   . . .
data   ends

code   segment
    assume    cs: code , ds: data

start:
    mov         ax , data
    mov         ds , ax
   . . .
    mov         ax, 4c00h
    int         21h

code   ends
    end       start
```

例4.1


data segment
...
data ends
code segment
assume cs: code , ds: data
start:
mov ax , data
mov ds , ax
...
mov ax, 4c00h
int 21h
code ends
end start
段定义
确定段和
段寄存器
间的关系
段地址送
段寄存器

(2) SEGMENT伪操作的类型和属性

- 格式：

segname SEGMENT [align][combine][use]['class']

- 说明：

一般情况下，这些类型和属性可以不用。但是，如果需要用连接程序把本程序与其它程序模块相连接时，就需要使用这些说明。

- 例4.5：

```
DSEG1     SEGMENT  WORD     PUBLIC     'DATA'
          …        …
DSEG1     ENDS
```

2. 存储模型与简化段定义伪操作

- (1) MODEL伪操作
- (2) 简化的段定义伪操作
- (3) 与简化段定义有关的预定义符号
- (4) 用MODEL定义存储模型时的段默认属性
- (5) 简化段定义举例

例4.2(p124)

```
. MODEL     SMALL
. STACK     100H
. DATA
      . . .
. CODE
START:
    MOV     AX , @DATA
    MOV     DS , AX
      . . .
    MOV     AX, 4C00H
    INT     21H
    END     START
```

(1) MODEL伪操作

- 格式：

  . MODEL memory_model [, model options]

- 说明：

  用来表示存储模型，即说明在存储器中如何安放各个段。有7种存储模型。

.MODEL memory_model [,model options]

- Tiny: 所有数据和代码都放在一个段内
- Small: 数据和代码各自放在一个64KB段内，最常用的一种模型
- Medium: 代码使用多个段，数据合并成一个64KB的段组
- Compact: 代码放在一个64KB的代码段内，数据可放在多个段
- Large: 代码和数据都可用多个段
- Huge: 与Large模型相同，差别是允许数据段的大小超过64KB
- Flat: OS/2或其它保护模式的操作系统下允许使用32位偏移量

例如

```
.MODEL      SMALL
.MODEL      SMALL , C
```

(2) 简化的段定义伪操作

- 汇编程序给出的标准段

标准段	简化段定义伪操作
code	. CODE [name]
initialized data	. DATA
uninitialized data	. DATA?
far initialized data	. FARDATA
far uninitialized data	. FARDATA?
constants	. CONST
stack	. STACK [size]

(3) 预定义符号

例4.2

```
.model          small
.stack          100h
.data
    array       db      0, 1, 2, 3, 4, 5, 6, 7, 8
.code
start:
    mov         ax, @data
    mov         ds, ax
          …
    mov         ax, 4c00h
    int         21h
end     start
```

4.2.3 程序开始和结束伪操作

```
data1  segment
  ...
data1  ends

code1  segment
   assume      ----
start:
   mov         ax , data1
   mov         ds , ax
  ...
   mov         ax , 4c00h
   int         21h
code1  ends

end    start
```

指示程序开始执行的起始地址

结束汇编

```
        .model small
        .data
        ......
        .code
start:
        mov     ax, @data
        mov     ds, ax
                ......
        mov     ax, 4c00h
        int     21h
        end start
```

4.2.4 数据定义及存储器分配伪操作

- 格式：

  [Variable] Mnemonic Operand1, … , Operandn [;Comments]

- 说明：

1)Variable: 作为符号地址使用，其值为操作数第1个字节的偏移地址

2) Comments: 可有可无，说明该伪操作的功能

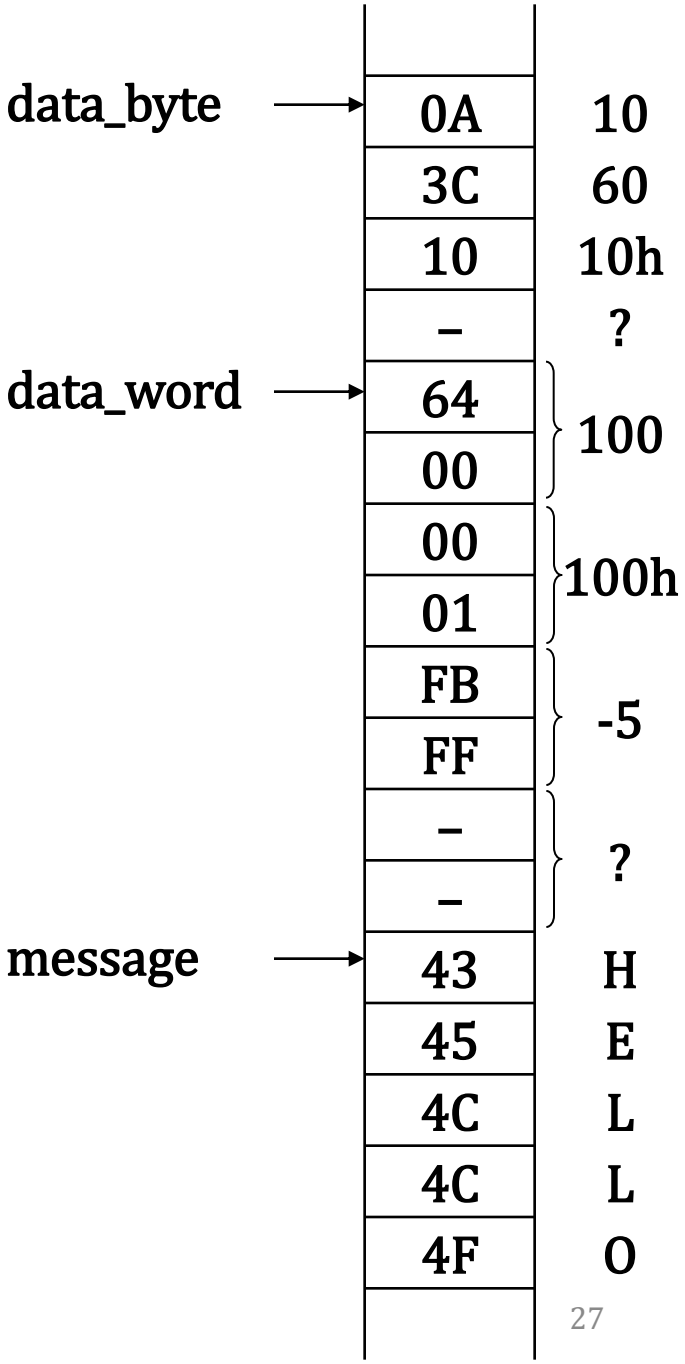
3) Mnemonic: 说明所定义的数据类型

DB/BYTE	字节，1个字节，8位
DW/WORD	字，　2个字节，16位
DD/DWORD	双字，4个字节，32位
DF/FWORD	6个字节，48位，386及后继机型
DQ/QWORD	4字，　8个字节，64位
DT/TBYTE	10个字节

4) Operand: 操作数可以是常数、表达式、字符串，也可以为？，保留存储空间，不存入数据

例4.7-4.9：

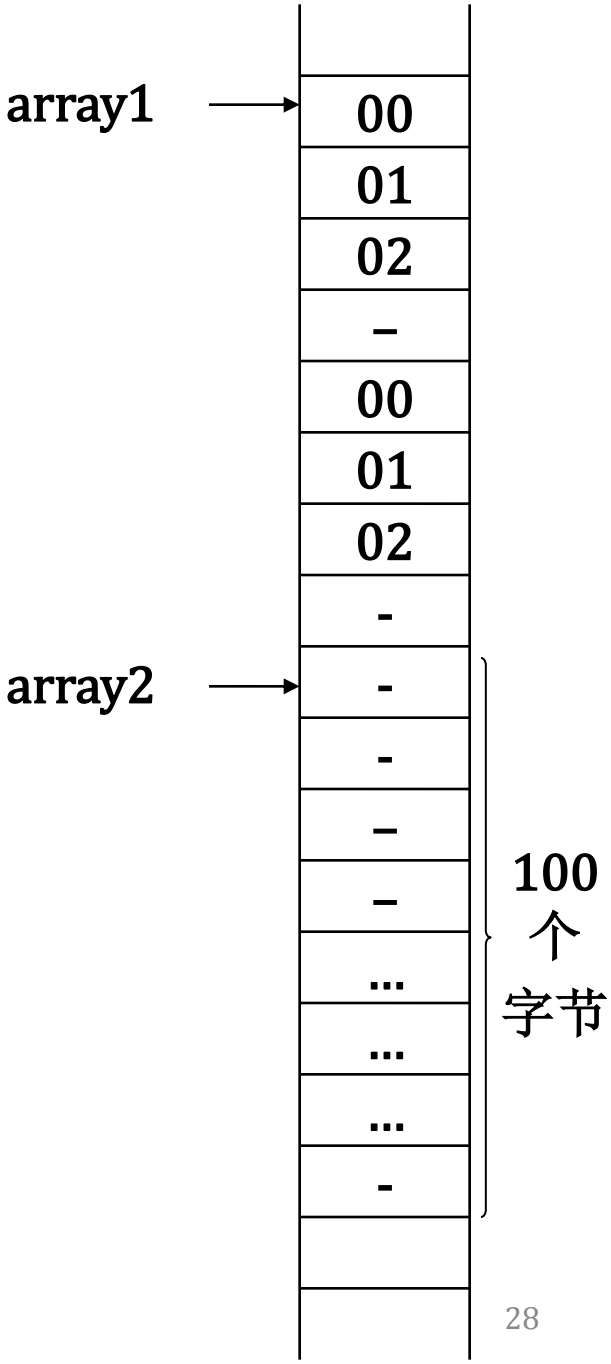
```
data_byte     db     10, 3*20 , 10h, ?
data_word     dw     100, 100h, -5, ?
message       db     'HELLO'
```



- 操作数可以使用复制操作符来复制某个或某些操作数

例4.10：

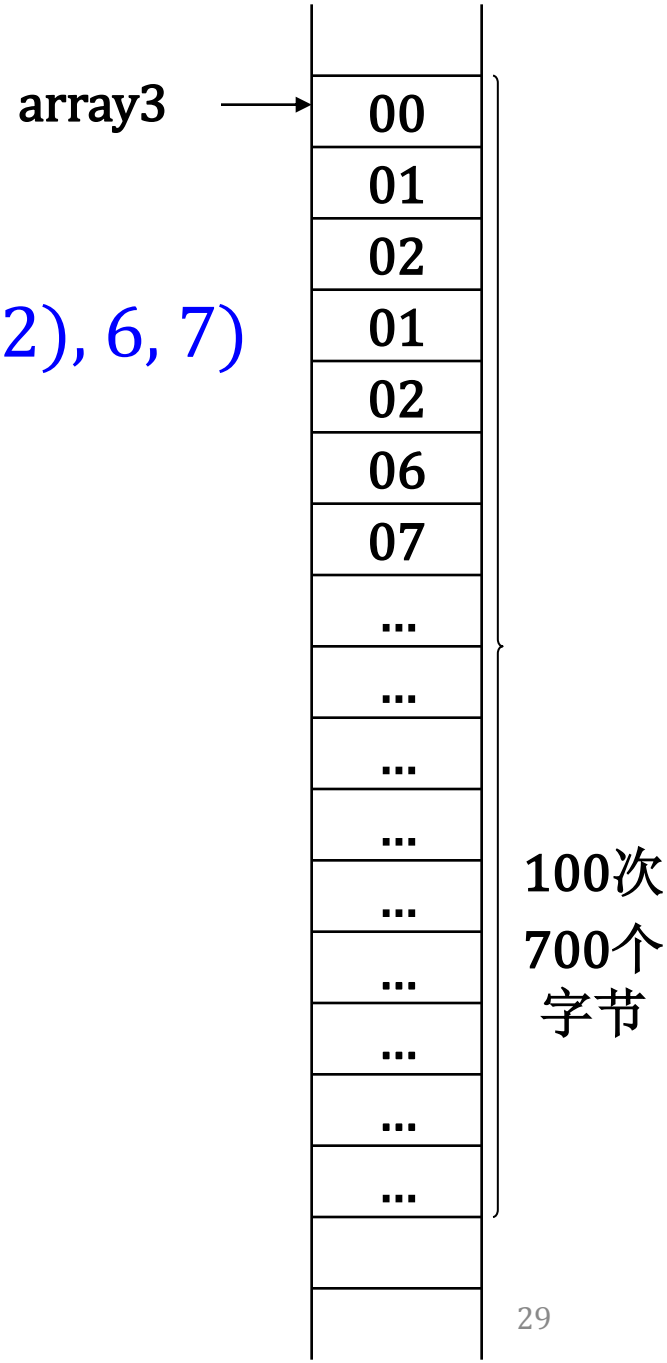
```
array1    db    2 dup(0,1,2,?)
array2    db    100 dup(?)
```



- DUP操作可以嵌套

例4.11：

array3　db 100 dup(0, 2 dup(1, 2), 6, 7)



变量的类型属性

- 变量的值: 是该伪操作中的第1个数据项在当前段内的第1个字节的偏移地址
- 变量的类型属性: 该语句中每一个数据项的长度（以字节为单位）
- 汇编程序用隐含的类型属性确定某些指令的操作类型
- 例4.14:

```
oper1     db     ?,?
oper2     dw     ?,?
          ...
          mov    oper1,0        ;字节指令
          mov    oper2,0        ;字指令
```

例4.15:

```
oper1       db     1 , 2
oper2       dw     1234h , 5678h
            . . .
        mov   ax , oper1+1        ×
        mov   al , oper2          ×
```

error: invalid instruction operands

指定操作数的类型属性

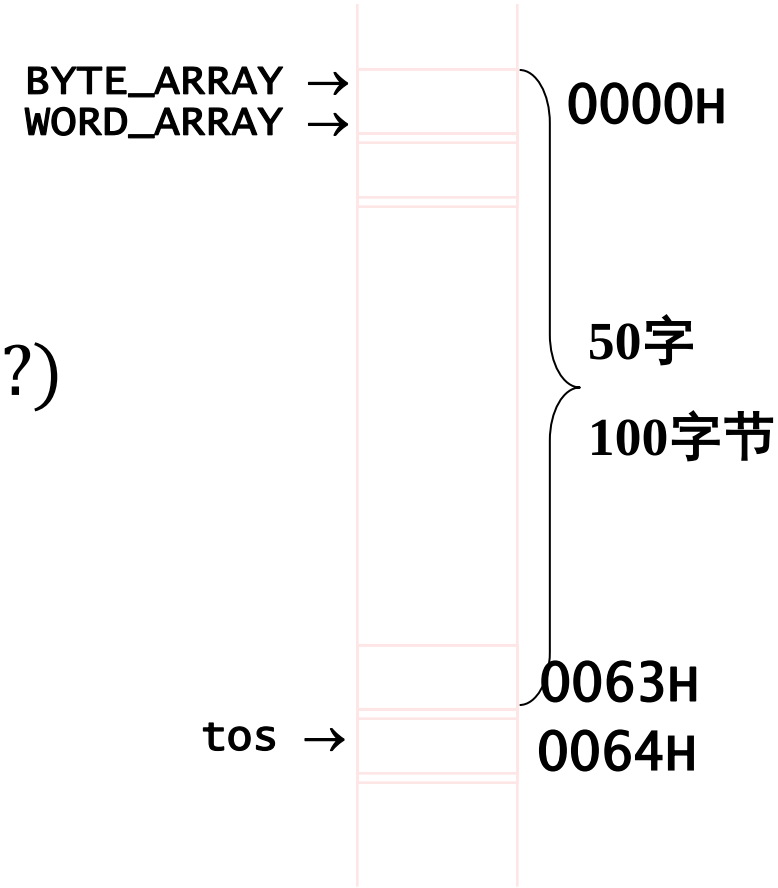
- 格式:　　type　PTR　variable
  其中，type可以是BYTE、WORD、DWORD、FWORD、QWORD、TBYTE
- 例4.15：

```
oper1     db     1,2
oper2     dw     1234h,5678h
               ...
     mov  ax, word   ptr   oper1+1    ;(ax)=3402h
     mov  al, byte   ptr   oper2      ;(al)=34h
```

label伪操作

- 可以使用label伪操作来定义变量的类型属性
- 格式：name　label　type
- 例4.16：

```
byte_array      label  byte
word_array      dw     50 dup(?)
tos             label word
```



4.2.5 表达式赋值伪操作

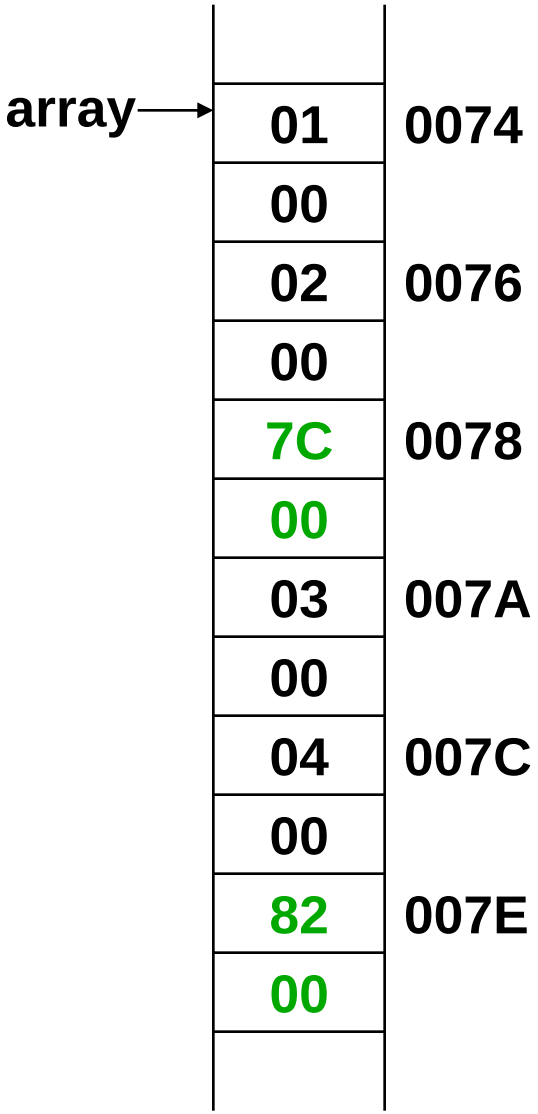
- 程序中多次使用同一个表达式，为方便起见，可以用赋值伪操作给表达式赋予一个名字
- 格式：Expression_name EQU Expression
- 此后，程序中就可以用表达式名代替表达式
- 例如：

```
CONSTANT  EQU     256
DATA      EQU     HEIGHT+12
ADDR      EQU     [ BP + 8 ]
```

4.2.6 地址计数器$伪操作

- 在汇编程序MASM对源程序汇编的过程中，使用地址计数器(location counter)保存当前正在汇编的指令的偏移地址。
- 地址计数器的值可用$表示，汇编语言允许用户用$引用地址计数器的值

- 例2：
  array dw 1,2,$+4,3,4,$+4
- 当$用在伪操作的参数字段时，表示地址计数器的当前值



4.2.7 基数控制伪操作

- 汇编程序默认的数为十进制，当使用其它基数表示的常数时，需要使用标记:

(1)二进制：B

(2)十进制：D

(3)十六进制：H，如果第一个字符是字母时，应在其前面加上数字0

(4)八进制：O Q

(5)字符串可以看成串常数，可以用引号括起来，如‘ABCD’

- .RADIX伪操作可以更改默认的基数
- 格式：.RADIX　　expression
- 例如：

　　.RADIX　　16

第4章 汇编语言程序格式

4.1 汇编程序功能

4.2 伪操作

4.3 汇编语言语句格式

4.4 汇编语言程序的上机过程

4.3 汇编语言程序格式

- 汇编语言源程序中的每个语句可以由4项组成，格式如下：

[name] operation operand [; comment]

标号	指令	寄存器	注释
变量	伪操作	标号	
	宏指令	变量	
		常数	
		表达式	

```
如：    string      db      ‘hello’
        start:      mov     al,string+2
                    jz      match
        match:      lea     si,string
```

操作数项

- 操作数项由一个或多个表达式组成，多个操作数之间用逗号分开
- 对于指令，一般给出操作数地址
  对于伪操作或宏指令，给出所要求的参数
- 可以是常数、寄存器、标号、变量或表达式

表达式

- 表达式是常数、寄存器、标号、变量与一些操作符相结合的序列
- 分为：数字表达式和地址表达式
- 常用的操作符:
  - (1) 算术操作符
  - (2) 逻辑与移位操作符
  - (3) 关系操作符
  - (4) 数值回送操作符
  - (5) 属性操作符

算术操作符

- 包括：**+ - * / mod**
- 可用于数字表达式或地址表达式中
- 当用于地址表达式时，运算结果要与明确的物理意义。经常使用的是：
  地址 + 数字量 或 地址 – 数字量

OFFSET

- 格式：**OFFSET　　variable或label**
- 功能：汇编程序回送变量或标号的偏移地址值
- 例如：

  MOV　　BX , OFFSET oper1

该指令与下面指令是等价的：

  LEA　　BX , oper1

SEG

- 格式：**SEG　　variable 或 label**
- 功能：汇编程序回送变量或标号的段地址值
- 例如：

MOV BX , SEG oper1

属性操作符

- **PTR：MOV　　BYTE PTR [BX] , 5**
- 段操作符：**MOV　AX , ES:[BX+SI]**
- **SHORT: JMP的转向地址在+ / -127字节内**