

# 第四章 汇编语言程序

4.1 汇编语句

4.2 汇编语言数据

4.3 8086指令系统

4.4 汇编语言伪指令

4.5 汇编源程序结构

4.6 上机操作过程

# 4.1 汇编语句 基本单元、语句格式、语法规则

## 4.1.1 语句种类

✓ 3种语句类型，指令语句、伪指令语句、宏指令语句。

✓ 指令语句是可执行语句，在汇编时可产生供机器执行的二进制目标代码。

例4.1 `mov ax, 1000h`

✓ 反汇编代码：0B10:0100 B80010 MOV AX,1000

✓ 伪指令语句是不可执行语句，在汇编时不产生目标代码，汇编程序主要利用它分配存储单元和定义程序段等。例子

例4.2 `X dw 1000h`

✓ 汇编程序在汇编时为变量X分配1个字存储单元，初值是1000h。

✓ 宏指令语句要求先定义后使用，它是一个宏名代替宏体的过程。（后面章节详细介绍）

## 4.1.2 语句格式

- ✓ **指令语句和伪指令语句**具有相似的语句格式，都由4部分组成，一般格式为：  
✓ **[<名字>]** <操作码> [<操作数>] [;<注释>]  
✓ []可选，<>必选，[<>]可选中的必选，|或选，前后内容必选其一
- ✓ 在汇编语言中，允许使用如下语言成分：
  - ✓ 字母：a~z A~Z
  - ✓ 数字：0~9
  - ✓ 字符：? ; : , @ \$ []

标识符?

# 名字

C语言：标识符

- ✓名字是一串字符序列，最多包含31个字符。
- ✓在汇编语言中，名字的应用范围很广，有寄存器名、变量名、常量名、标号、指令名等等。
- ✓在语句格式中作为第一可选成分出现的<名字>，在不同的语句中具有不同的含义。它在指令语句中表示标号，后面必须跟有冒号“:”；而在伪指令语句中表示变量名，常量名，段名，过程名，后面不能有冒号。这是两种语句在格式上的主要不同之处。
- ✓指令语句中的标号和伪指令语句中变量名，段名，过程名是一种符号地址，可作为汇编指令的一个操作数，但常量名不是符号地址，不能用做目的操作数。不同的标号、变量和常量不能同名。

# 操作码/操作数/注释

## 第二章程序例子

✓ **操作码**是汇编语句格式中**唯一不可或缺**的语法成分。它可以是**指令助记符**，如ADD,SUB等，汇编程序将它翻译成**机器指令**；也可以是**伪指令操作助记符**，汇编程序将根据具体要求在程序编译时进行相应处理。

✓ **操作数**可以是**寄存器，常量，变量，标号，表达式**。在指令语句中，**可以没有操作数，最多有两个操作数**。在伪指令语句中，则给出**一系列参数**。使用**两个操作数或多个参数**时，相互间需要用“，”分隔。

✓ **注释**由分号“；”开头，用来对语句的功能加以说明，容易阅读。

## 4.2 汇编语言数据

语句格式中[<操作数>]



✓汇编语言**数据**是组成**指令操作数**或**伪指令参数**的主要成分。数据的形式有**常量**、**变量**、**标号**、**表达式**等（一一详述）。



操作数：寄存器

### 常量

✓在程序运行过程中，**值不发生变化**的量称为常量。常量主要用于伪指令语句中给变量**赋初值**，或用作指令语句中的**立即操作数**，以及相对寻址方式中的**位移量**。

名字：常量不是符号地址

# 常量分类:

数值型常量、符号常量和字符型常量。

## 1. 数值型常量

✓四种：二进制(B)，八进制(Q)，十进制(D或数制 省略)，十六进制(H，A-F前缀0)

## 2. 符号常量

✓对经常引用的数值型常量，可以用等价伪指令EQU或等号伪指令“=”给它定义一个名字，然后在语句中用这个名字来代表该常量。这个名字称为符号常量。

✓例4.3 COUNT EQU 90

✓VALUE = 60

### 3. 字符型常量

- ✓用引号括起来的一个或多个字符称为字符型常量。引号中字符的ASCII码值，即是该字符型常量的值。例如“B”的值是42H，而“BA”的值是4241H。因此字符型常量与数值型常量可以相互通用。

第二程序示例



## 4.2.2 变量

### 1. 变量

符号地址：变量名、段名、过程名  
常量

寄存器

✓在汇编语言中，**变量**是一个数据存储单元的名字，即数据**存放地址**的符号表示。由于主存是**分段**使用的，因而对源程序中所定义的**变量**应体现出以下三方面的**属性**。

# 变量三属性：

## 1) 变量的段属性

- ✓ 变量的段属性（地址）是指定义变量所在段的段首址，当需要访问该变量时，该段首址一定要在某一段寄存器中。

## 2) 变量的偏移属性

- ✓ 变量的偏移属性（地址）是指变量所在段的段首址到该变量定义语句的字节距离。

偏移地址

## 3) 变量的类型

- ✓ 是指存取变量中的数据时所需要的字节数，也是存取变量数据的交换单位。它可以是字节类型、字类型、双字类型、四字类型、十字节类型。这些类型的选择由定义该变量时所使用的数据定义伪指令确定。

## 2. 变量定义

- ✓变量一般在数据段或附加数据段中使用数据定义伪指令DB、DW、DD、DQ和DT来定义。定义变量的一般格式：  
Quartet Word
- ✓[变量名] 数据定义伪指令 表达式 [, ...]
- ✓其中，用DB定义的变量为字节类型的变量；用DW定义的变量为字类型变量；用DD定义的是双字类型的变量；用DQ定义的是四字类型的变量；用DT定义的是十字节类型的变量？
- ✓变量名由用户所取，定义变量时可以省略。表达式确定了变量的初值。

## 例4.4

- ✓ Count db 10
- ✓ Buf dw 1,2,3,4,5
- ✓ Char db 'AB'

### 高级语言的变量定义：

- ✓ Char c1,c2,c3;
- ✓ Int a,b,c,d
- ✓ Float x,y,z

区别？

赋初值；  
格式顺序  
1变量名多初值  
无变量名  
。 。 。

### 3. 变量赋初值

- ✓1) 数值表达式;
- ✓2) ASCII字符串 (通常选用DB类型);
- ✓3) 地址表达式(只适用于DW和DD两个伪指令); DW: 偏移地址, DD: 段首址和EA
- ✓4) ? (表示所定义的变量无确定初值);
- ✓5) 重复子句。
- ✓其格式为: n DUP (表达式)
- ✓6) 可以是以上表达式组成的序列: 逗号隔开, 变量名是首地址, 表达式个数即存储单元个数

## 例4.5 定义Data数据段。

语句格式

✓ Data segment ; 段定义开始

✓ A dw M 和上面5种——对照

✓ B db 'AB', 0DH, 0AH

✓ C dw 0FFAAH 怎么放?

✓ D dd B 地址表达式: DD

✓ Count equ 500H

✓ M db 2 DUP (1, 2)

✓ Data ends

对照

Data: A	0CH	0				
	00H	1				
B	41H	2	0			
	42H	3	1			
	0DH(回车)	4	2			
	0AH(换行)	5	3			
C	AAH	6	4	0		
	FFH	7	5	1		
D	02H	8	6	2	0	
	00H	9	7	3	1	
M		10	8	4	2	
	Data	11	9	5	3	
	01H	12	10	6	4	0
	02H	13	11	7	5	1
	01H	14	12	8	6	2
	.02H	15	13	9	7	3

图 4.1 Data 数据段存储空间图

# 结论

- ✓ **变量**是数据存储单元的**符号地址**，真值为EA
- ✓ 每个变量仅代表第一个数据存储单元，大小由类型而定
- ✓ 多个表达式初值<sup>?</sup>，**实质**是变量名的省略
- ✓ 数据段是**总段**，每个变量是**小段**，线性连续存储
- ✓ A、B、C、D、M的偏移地址
- ✓ 右边第一列0-15偏移量是相对**Data**、**A**而言的
- ✓ 2、3、4、5列偏移量？
- ✓ 数据段和偏移地址均具有**相对性**，变量名可省略

## 结论（续）

✓存取存储区数据时，类型以地址表达式中**某变量名为准**

例如，执行语句 1: MOV AL, B  
执行结果: (AL)=41H  
执行语句 2: MOV AL, B+2  
执行结果: (AL)=0DH  
源操作数类型以变量 B 类型为准。

例如，执行语句 3: MOV AX, C  
执行结果: (AX)= 0FFAAH  
执行语句 4: MOV AX, C+2  
执行结果: (AX)=0002H  
源操作数类型以变量 C 类型为准。



## 4.2.3 标号

单目运算符：一个操作数（源/目的）？

- ✓ 标号是可执行语句的符号地址，也可以是子程序名。
- ✓ 子程序名实际上是子程序入口地址的符号表示，即子程序中第一条机器指令语句的符号地址。
- ✓ 标号可以作为JMP等转移指令和CALL指令的目的操作数。
- ✓ 标号的定义方式为：
  - ✓ NEXT: MOV AL, [SI] ; 定义近标号NEXT 寻址方式
  - ✓ SUB1 PROC FAR ; 定义过程名SUB1为远标号 指令寻址
  - ✓ SUB2 PROC NEAR ; 定义过程名SUB2为近标号
  - ✓ JMP NEXT ; 转移指令的目的操作数 FAR PTR
- ✓ 标号一般只在代码段中定义和引用，作为指令地址的符号表示，标号与变量类似，因而它也有3个属性。

近标号、远标号：本质区别？  
PROC？

# 标号三属性：

- ✓ 标号的**段属性**：标号的段属性是指定义标号所在段的段首址。
- ✓ 标号的**偏移属性**：标号的**偏移地址**是指它所在段的段首址到该标号定义语句的字节距离。
- ✓ 标号的**类型**：标号的类型有两种：**NEAR类型**和**FAR类型**，凡属NEAR类型的标号只能在定义该标号的**段内**引用，**且可以省略**，FAR类型的标号**段间**引用，不可省。

## 4.2.4 表达式

✓在80x86汇编语言中，有数值表达式和地址表达式两种。

### 数值表达式

- ✓由各种常量与数值运算符连接而成的式子，称为数值表达式。  
数值表达式的计算结果是一个数值，它只有大小而没有属性。 ?
- ✓数值运算符：算术、关系、逻辑运算符      Shift left/right??
- ✓算术：+，-，\*，/，MOD(取余)、SHR (左移)、SHL(右移)
- ✓逻辑：AND、OR、NOT、XOR，按位运算，只能是数值型常量
- ? 常量类型      Less/great than
- ✓关系：EQ(=)、NE(≠)、LT (<)、LE (≤)、GT (>)、GE (≥)，  
两个操作数，数值型数据或同一段内的地址，结果是数值型数据；关系假（0），真（0FFFFFFH）

X = 10

MOVE BX, X LT 5

# 地址表达式

?

- ✓由常量、变量、标号、寄存器和数值运算符、地址运算符组合而成的有意义的式子，称为地址表达式，单个变量、标号是地址表达式的最简形式。

变量赋初值

- ✓由于变量和标号具有段（SEG）、偏移（OFFSET）、类型（TYPE）3种属性，这就决定了对它们的访问是多种形式的。

回顾  
类型  
是什么？

# 地址运算符包括以下两种

数值表达式中能否有  
变量、标号？

## 1. 属性分离算符

✓属性分离算符可分离出变量、标号的段地址、  
偏移地址及类型的属性值。使用的格式为：

✓<属性分离算符> <变量>或<标号> 语句格式

✓其运算结果为一数值常量。

寻址方式？

✓SEG(取段地址)：MOVE AX, SEG BUF

第三  
章例子

✓OFFSET(取偏移地址算符)

✓TYPE(取类型算符)

✓近标号：0FFFFH，远：0FFFEH

2020/4/17

表 4.1 变量的类型与类型值对照表

变量类型	类型值
字节	1
字	2
双字	4
四字	8
十字节	10

## 例题4.6：属性分离算符的应用

```
✓DATA    SEGMENT
✓A    DW  50, 100, -70H
✓B    DB  'ABCDEF'
✓DATA    ENDS
✓CODE    SEGMENT
✓        ASSUME CS: CODE, DS: DATA
✓START:  MOV AX, DTAT
✓        MOV DS, AX
✓        MOV AX, SEG B    ; DATA→AX
✓        MOV BX, OFFSET B ; 6→BX
✓        MOV CX, TYPE A   ; 2→CX
✓        MOV DX, TYPE B   ; 1→DX
✓        MOV AH, 4CH
✓        INT 21H
✓CODE    ENDS
✓        END START
```

?

## 2. 属性定义算符 PTR

WORD PTR

FAR PTR

第一章X+2



- ✓(1) 该运算符用来指明某个变量、标号或地址表达式的类型属性，或者使它临时兼有与原来定义所不同的类型属性，但保持它们原来的段地址和偏移地址属性不变。
- ✓格式：<类型>PTR<地址表达式>
- ✓其中，<类型>可以是BYTE、WORD、DWORD、NEAR、FAR。
- ✓例如，语句MOV BYTE PTR [SI], 100
- ✓X DB 5,4,7    MOVE AX, WORD PTR X

AX结果？为什么？

## 2. 属性定义算符 PTR

✓注意：存储器寻址方式，均为简单的地址表达式，具有明确的段属性和偏移属性，但是类型模糊不确定，此时需要用PTR运算符指明它的类型属性。

✓接下来的例子



# PTR的应用：

- ✓(1) 指明某个变量、标号或地址表达式的类型属性，或者使它临时兼有与原定义所不同的类型属性（段和偏移属性不变）
- ✓<类型> PTR <地址表达式>
- ✓类型可以是BYTE，WORD，DWORD，NEAR，FAR。
- ✓X DB 5,4,7
- ✓MOV AX, **WORD PTR X**

# PTR的应用:

## 综合例题 4.7 PTR 运算符的使用。

```
DATA SEGMENT
NUM DB 11H, 22H, 33H
```

✓(2) PTR

临时改变  
址变为-

本句中有效  
C语言强制  
类型转换

```
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START: MOV AX, DAT
    MOV DS, AX
    MOV AX, NUM ; 类型不一致, 错误语句
    MOV AX, WORD PTR NUM ; 临时改变 NUM 为字类型,
                        (NUM)=2211H→AX

    MOV BL, NUM ; (NUM)=11H→BL
    MOV SI, OFFSET NUM ; NUM 的 EA→SI
    ADD 2[SI], 2 ; 类型不明确, 错误语句
    MOV BYTE PTR 2[SI], 2 ; OPD=[SI]+2=NUM+2, 即 2→NUM+2
                        ; PTR 指定该地址表达式类型为字节
    INC [SI] ; 类型不明确, 错误语句
    INC BYTE PTR [SI] ; OPD=[SI]=NUM, 由 PTR 指定为字节类型
    MOV AH, 4CH
    INT 21H
CODE ENDS
    END START
```

# PTR的应用：

✓不带方括号的寄存器符号不是地址表达式，因此，不能用PTR改变寄存器类型。

✓错误语句：

✓MOVE AL, BYTE PTR SI

✓MOVE AX, WORD PTR DL

✓改正：

✓MOVE BX, SI    MOVE AL BL

✓MOVE AL, DL    CBW

# PTR的应用：

- ✓(3) PTR运算符与EQU或等号“=”伪指令连用，将同一存储区地址用不同类型的变量或标号来表示。
- ✓DATA SEGMENT
- ✓A DW 1122H, 3344H
- ✓B EQU BYTE PTR A ;将变量A为首址的字存储区另定义为以变量B为首址的字节存储区
- ✓C DB 10 DUP(0)

书上图

# PTR的应用：

- ✓ `D EQU WORD PTR C` ;将C数据存储区另定义为字类型变量D的数据存储区
- ✓ `DATA ENDS` AX/BL的值  
C/D存储区
- ✓ `CODE SEGMENT`
- ✓ `ASSUME CS: CODE, DS: DATA`
- ✓ `START: MOV AX, DATA MOVE DS, AX`
- ✓ `MOV AX, A MOV BL, B MOV C, BL`
- ✓ `MOV D+2, AX MOV AH, 4CH INT 21H`
- ✓ `CODE ENDS END START`

# PTR的应用：

✓(4) 汇编程序**规定**：**单操作数指令**只有目的操作数地址，类型必须明确；对于**双操作数指令**，如OPS/OPD均明确，必须**一致**；一个明确，一个模糊或没有类型，**取明确的那一个**作为两个操作数地址的共同类型；如两个均模糊，则为**错误语句**。

✓DATA SEGMENT

✓A DW 1122H, 3344H, 0

✓DATA ENDS

# PTR的应用：

- ✓ CODE SEGMENT
- ✓        ASSUME CS: CODE, DS: DATA
- ✓ START: MOV AX, DATA    MOV DS, AX
- ✓ MOV SI, OFFSET A ;字类型
- ✓ MOV AX, A                ;字类型
- ✓ MOV BX, [SI]            ;字类型
- ✓ MOV DL, 2[SI]          ;字节类型
- ✓ MOV AX, DL              ;**错误**，类型不一致

# PTR的应用：

- ✓ MOV BX, AL ;同上
- ✓ INC [BX] ;单操作,OPD类型不明
- ✓ MOV 4[SI], 55H ;错误, 均不明
- ✓ CODE ENDS
- ✓ ENDS START
- ✓ 修改: INC BYTE PTR [BX]
- ✓ MOV WORD PTR 4[SI], 55H
- ✓ MOV 4[SI], WORD PTR 55H



# PTR的应用：

- ✓(5) 标号与属性定义算符和分离算符连用，可以取出标号的段属性和偏移属性。
- ✓JMP NEAR PTR NEXT(标号);偏移地址给IP
- ✓JMP FAR PTR NEXT;偏移地址给IP，段地址给CS
- ✓MOV BX, SEG L2(标号)
- ✓MOV BX, OFFSET L2
- ✓;段地址和偏移地址送至BX

# 表达式的计算

- ✓在汇编过程中，汇编程序按操作运算符**约定**的**优先规则**对表达式进行计算，计算结果是一个**数值**或一个**地址**。
- ✓当表达式中出现多种操作运算符时，汇编程序将按一定的**规则**计算表达式的值：
  - ✓(1)一般先执行优先**级别高**的运算。
  - ✓(2)优先级相同的运算，**自左至右**进行。
  - ✓(3)**括号**可改变运算顺序。

# 表达式的计算

✓ 优先级顺序表：数值、地址运算符

• 58 •

表 4.2 运算符和操作符的优先级顺序

操作运算符	优先级
( ), [ ]	高
段跨越前缀符( : )	
PTR, OFFSET, SEG, TYPE	
*, /, MOD, SHL, SHR	
+, -	
EQ, NE, LT, LE, GT, GE	
NOT	
AND	
OR, XOR	低

C语言

# C语言

## 附录 D 运算符和结合性

优先级	运算符	含 义	要 求 运 算 对象的个数	结合方向
1	( )	圆括号		自左至右
	[ ]	下标运算符		
	->	指向结构体成员运算符		
	.	结构体成员运算符		
2	!	逻辑非运算符	1 (单目运算符)	自右至左
	~	按位取反运算符		
	++	自增运算符		
	--	自减运算符		
	-	负号运算符		
	(类型)	类型转换运算符		
	*	指针运算符		
	&	取地址运算符		
	sizeof	长度运算符		
3	*	乘法运算符	2 (双目运算符)	自左至右
	/	除法运算符		
	%	求余运算符		
4	+	加法运算符	2 (双目运算符)	自左至右
	-	减法运算符		
5	<<	左移运算符	2 (双目运算符)	自左至右
	>>	右移运算符		
6	< <= > >=	关系运算符	2 (双目运算符)	自左至右

## 4.3 8086指令系统

✓ 8086微处理器的指令系统大约具有**100条基本指令**，这些指令可以分为以下**六类**：

- ✓ 数据**传送**指令；                      MOV
- ✓ 算术运算指令；                      ADD
- ✓ **位**操作指令；                      逻辑运算符
- ✓ **串**操作指令；
- ✓ 控制**转移**指令；                      JMP、JNZ
- ✓ **处理机控制**指令。

## 4.3 8086指令系统（强调）

- ✓ **双操作数指令**具有相同的语句格式和操作规定：
- ✓ [标号:]操作符 OPD, OPS [;注释]
- ✓ **操作规定**：OPD和OPS**类型应相同**；目的操作数一定不能是**立即操作数**；操作结束后，运算结果送入目的地址中，而源操作数不改变；二者**不能同时为存储器操作数**(存储器+立即/寄存器)
- ✓ **单操作数指令**：[标号:] 操作符 OPD [;注释]
- ✓ **操作规定**：目的地址中的操作数，结果送入目的地址；操作数不能是立即操作数

## 4.3.1 数据传送指令

- ✓ 数据传送指令负责在寄存器或内存单元之间进行数据、地址或立即数的传输。
- ✓ 主要包括：一般数据传送指令（MOV、XCHG）、堆栈操作指令（PUSH、POP）、标志寄存器传送指令（PUSHF、POPF、LAHF、SAHF）、地址传送指令（LEA、LDS、LES）、输入/输出指令（IN、OUT）。在这些指令中，除了SAHF、POPF两指令外，其它均不影响标志位。
- ✓ 这里主要介绍MOV、XCHG、LEA、LDS、LES指令的语句格式和功能。

# 数据传送指令

✓语句格式: MOV OPD, OPS (OPS)->OPD

✓OPS可以是寄存器、存储器和立即操作数, OPD可以使用寄存器、存储器操作数

寄存器与  
存储器交互

✓说明: 立即数不能作为OPD, 也不能送段寄存器;  
不允许在两个存储单元之间直接传输数据; 不允许在两个段寄存器之间传送数据; 不允许使用CS代码段寄存器; 数据传送过程中注意类型的一致性

✓例: MOV AX, BX ;BX的内容送入AX中

✓MOV AX, DATA\_SEG

模版

✓MOV DS, AX ;段地址必须通过寄存器送到DS



# 数据交换指令

- ✓ 语句格式: **XCHG** OPD, OPS
- ✓ 功能: (OPD)->OPS (OPS)->OPD 内容互换
- ✓ 说明: 2个操作数中**必须有一个寄存器**; **不允许使用 ? 段寄存器**
- ✓ 例: XCHG AX, DI
- ✓ 执行前: (AX)=0001H, (DI)=0FFFFH
- ✓ 执行后: (AX)=0FFFFH, (DI)=0001H

# 偏移地址传送指令

## 回顾

- ✓语句格式：LEA OPD, OPS
- ✓功能：按OPS提供的寻址方式计算偏移地址，将其送入OPD中
- ✓说明：OPD一定是一个十六位的通用寄存器；OPS所提供的一定是一个存储器地址
- ✓例：DATA SEGMENT
- ✓BUF DB 'ABCDEF'
- ✓NUM DW 72H, 55H, 100H
- ✓PIN DW 0
- ✓DATA ENDS

# 偏移地址传送指令

- ✓ CODE SEGMENT
- ✓ ASSUME CS: CODE, DS: DATA
- ✓ START: MOV AX, DATA    MOV DS, AX
- ✓ MOV SI, OFFSET NUM ;EA=6->SI
- ✓ LEA SI, NUM ;同上, 寻址方式
- ✓ MOV AX, [SI] ;([SI])=72H->AX
- ✓ LEA AX, [SI] ;EA=6->AX, 寻址方式
- ✓ LEA DI, 4[SI] ;EA=10->DI, 寻址方式
- ✓ LEA PIN, BUF ;错误语句?
- ✓ MOV PIN, OFFSET BUF ;将BUF的EA->PIN

本质区别

# 偏移地址及(附加)数据段首地址传送指令

✓语句格式: LDS OPD, OPS

✓功能: (OPS)->OPD, (OPS+2)->DS

✓说明: OPD一定是一个十六位的通用寄存器; OPS所提供的一定是一个存储器地址, 且类型为DD

✓语句格式: LES OPD, OPS

✓功能: (OPS)->OPD, (OPS+2)->ES

✓作用: 随时改变DS和ES内容, 更换当前数据段和当前附加数据段

# 偏移地址及(附加)数据段首地址传送指令

✓例：DATA1 SEGMENT

✓T1 DW 5050H

✓T2 DD FF;(T2)为变量FF在DATA2段中的偏移地址6,  
(T2+2)为FF所在段的段首址DATA2

✓DATA1 ENDS

✓DATA2 SETMENT

✓BUF DB 'ABCDEF' FF DW 7070H T3 DD T4

✓DATA2 ENDS

✓DATA3 SEGMENT

✓T4 DW 3050H GG DW 437AH DATA3 ENDS

# 偏移地址及(附加)数据段首地址传送指令

- ✓ STACK SEGMENT
- ✓ DB 200 DUP(0)      STACK ENDS
- ✓ CODE SEGMENT
- ✓      ASSUME DS: DATA1, ES: DATA1,  
         CS:CODE, SS: STACK
- ✓ START: MOV AX, DATA1 MOV DS, AX;当前段
- ✓ MOV ES, AX;当前附加数据段也为DATA1段
- ✓ MOV BX, 2;
- ✓ MOV AX, T1;(T1)=5050H->AX
- ✓ MOV DX, [BX]; (DX)=6    MOV CX, ES:[BX]

# 偏移地址及(附加)数据段首地址传送指令

- ✓ **LDS SI, T2**; (T2)=6-→SI, (T2+2)=DATA2-→DS, 当前数据段变为DATA2
- ✓ **LES DI, DS:T3**; 附加数据段变为DATA3
- ✓ **MOV DX, DS:[BX]**; **(DX)=4443H**
- ✓ **MOV CX, ES:[BX]** ; **(CX)=437AH**
- ✓ **MOV AX, DS:[SI]** ; **([SI])=(FF)=7070H-→AX**

## 4.3.2 算术运算指令

- ✓ 算术运算指令是指对二进制数进行加、减、乘、除运算的指令。
- ✓ 主要包括：加法指令（ADD、ADC、INC），减法指令（DEC、NEG、SUB、SBB、CMP），乘法指令（IMUL、MUL），除法指令（IDIV、DIV）和符号扩展指令（CBW、CWD、CDQ）。
- ✓ 在这些指令中，除符号扩展指令之外，均会不同程度地影响标志寄存器中的标志位。



## 4.3.2 算术运算指令

✓带进位加法指令：ADC OPD, OPS

✓ $(OPD) + (OPS) + CF \rightarrow OPD$

✓求补指令：NEG OPD (NEGate)

✓按位取反（包括符号位）后加1 $\rightarrow$ OPD

✓带借位减指令：SBB OPD, OPS (SuBtract with Borrow)

✓ $(OPD) - (OPS) - CF \rightarrow OPD$

✓比较指令：CMP OPD, OPS

✓ $(OPD) - (OPS)$  根据结果设置标志位，不存入目的地址

## 4.3.2 算术运算指令

✓有符号乘指令：IMUL OPS （无符号乘：MUL OPS）

✓字节乘法：(AL)\*(OPS)->AX

signed MULTiple

✓字乘法：(AX)\*(OPS)->DX、AX

✓无符号数除法指令：DIV OPS

✓字节操作：(AL)<-(AX)/(OPS)商；(AH)<-(AX)/(OPS)余数；  
被除数16位，8位除数为源操作数，商和余数8位

✓带符号数除法指令：IDIV OPS

✓与DIV功能相同，操作数、商和余数带符号数，且余数的符号和被除数的符号相同

## 4.3.2 算术运算指令

- ✓ 类型转换指令（符号扩展指令）
- ✓ 字节转为字指令：CBW
- ✓ AL的内容符号扩展到AH，形成AX中的字；(AL)最高有效位为0（1），则（AH）=0（0FFH）
- ✓ 字转为双字指令：CWD、CWDE
- ✓ AX的内容符号扩展到DX、EAX
- ✓ 双字转为4字指令：CDQ

## 4.3.3 位操作指令

✓ 8086微处理器提供的位操作指令分为**逻辑运算指令**和**移位指令**；它们均可直接对**寄存器或存储器**中的数据位进行操作。

### 一. 逻辑运算指令

✓ AND，OR，NOT（按位取反），

✓ XOR（某些位取反），TEST（**相与，结果不保存，根据其特征设置条件码**）

### 二. 移位指令

移位指令包括算术移位指令、逻辑移位指令和循环移位指令。

(1) 算术左移和逻辑左移指令

✓ 语句格式；SAL OPD, 1 或 SHL OPD, 1

✓ SAL OPD, CL 或 SHL OPD, CL

(2) 算术右移指令（补最高位）

✓ 语句格式：SAR OPD, 1或SAR OPD, CL

(3) 逻辑右移指令（补0）

✓ 语句格式：SHR OPD, 1或SHR OPD, CL

## 2. 循环移位指令

(1) 循环左移指令

✓语句格式：ROL OPD, 1或ROL OPD, CL Rotate left

(2) 循环右移指令

✓语句格式：ROR OPD, 1 或ROR OPD, CL

(3) 带进位的循环左移指令 Rotate left through carry

✓语句格式：RCL OPD, 1或RCL OPD, CL

(4) 带进位的循环右移指令

✓语句格式：RCR OPD, 1或RCR OPD, CL

## 4.3.4 串操作指令

- ✓ 字符串操作指令共有五条，包括：
- ✓ 传送字节(或字)串的指令MOVSB,
- ✓ 搜索字节(或字)串的指令SCAS,
- ✓ 比较字节(或字)串的指令CMPS,
- ✓ 取字节(或字)串的指令LODS,
- ✓ 存储字节(或字)串的指令STOS。

## 4.3.5 控制转移指令

✓控制转移指令包括无条件转移指令和条件转移指令以及循环指令。

### 一、无条件转移指令

✓语句格式：JMP <转向地址>

✓其中，转向地址是标号或地址表达式，寻址方式有直接和间接两种。

✓功 能：无条件地转移到指令指定的转向地址处，去执行从该地址开始的指令。



## 二、条件转移指令

✓8086提供了30条条件转移指令，其基本格式为：

✓J×× <标号>

✓其中，字母J后的“××”是指条件。若条件成立，则程序转移至由标号标识的指令处执行。若条件不满足，则顺序执行下一条指令。

✓所有条件转移指令都是以某些标志位的状态作为依据改变IP当前值，实现目标代码转移的。其目标操作数都是近标号，即只能实现段内转移。

✓所有条件转移指令对状态标志位均无影响。

根据标志位的个数，可分为简单、无符号数和带符号数三类条件转移指令。

### 1. 简单条件转移指令

✓简单条件转移指令只根据标志寄存器中单一标志位的状况做判断，以实现转移。指令有14条，用了5个标志位。

### 2. 无符号数条件转移指令

✓无符号数条件转移指令使用标志寄存器中CF和ZF两标志位的状况做判断，以实现转移。其指令有8条。

### 3. 带符号数条件转移指令

✓带符号数条件转移指令使用标志寄存器中SF、OF和ZF 3个标志位的状况做判断，以实现转移。其指令有8条。

## 四、循环指令

- ✓ 80x86为了简化循环程序的编写，设计了3条循环指令，LOOP,LOOPZ,LOOPNZ
- ✓ 这三条指令的执行步骤是：
  - ✓ ①  $(CX) \leftarrow (CX) - 1$
  - ✓ ② 检查是否满足测试条件，如果条件满足则指令转移到OPD入口地址处。

## 4.3.6 处理机控制指令

### 1. 标志处理指令

✓ 设置和清楚CF、DF、IF三个标志位

### 2. 其他处理机指令

## 4.4 汇编语言伪指令

- ✓用汇编语言设计程序时，经常需要向汇编程序**提供必要的信息**，如数据和名字说明、程序的开始与结束说明、过程说明等。程序中的这些信息**并无对应的机器指令**，因而不产生**机器代码**，仅供汇编程序执行某些特定的任务，故称为**伪操作**，又称为**伪指令**。
- ✓80x86汇编语言有**丰富的伪指令**，如：**符号定义伪指令**、**数据定义伪指令**、**段定义伪指令**、**程序开始与结束伪指令**、**对准与基数控制伪指令**等。

## 4.4.1 符号定义伪指令

✓汇编语言中的所有**名字**（变量名、标号名、过程名、记录名、指令助记符、寄存器名等）都是用**符号表示**的，故又统称为**符号名**。这些符号名可以通过**符号定义伪指令**重新命名，也可以**定义新的类型属性**，因而给程序设计带来很大的灵活性。

✓1. EQU伪指令

✓2. “=”伪指令

课本例题

## 4.4.2 数据定义伪指令

- ✓常用的数据定义伪指令有DB、DW、DD、DQ、DT，主要用来定义变量。
- ✓数据定义语句的格式和功能如下：
  - ✓格式：[变量名] 数据定义伪指令 表达式[, ...]
  - ✓功能：定义一数据存储区，类型由数据定义伪指令指定。
  - ✓DB：数据类型是字节，每个操作数都占有一个字节。
  - ✓DW：数据类型是字，每个操作数占有一个字，其低位字节在低字节地址中，高位字节在高字节地址中。
  - ✓DD：数据类型是双字，每个操作数占有两个字。
  - ✓DQ：数据类型是4个字，每个操作数占有4个字，可用来存放双精度浮点数。
  - ✓DT：数据类型是10个字节，其后的每个操作数均占有10个字节，形成压缩的BCD码。

## 4.4.3 段定义伪指令

### 1. 段定义伪指令

✓SEGMENT / ENDS两条伪指令语句定义一个逻辑段 课本

### 2. ASSUME伪指令

✓由SEGMENT/ENDS伪指令定义一个段后，通常还需要明确段与段寄存器之间的对应关系，ASSUME伪指令可以建立和撤消这种关系。



## 4.4.4 程序开始与结束伪指令

1. NAME伪指令

2. TITLE伪指令

3. PAGE伪指令

4. 源程序结束伪指令

✓ 格式：END [<标号>|<过程名>]

## 4.4.5 对准与基数控制伪指令

1. EVEN伪指令
  2. ORG伪指令
  3. 地址计数器\$
  4. 基数控制伪指令
- ✓应用见相关例题。

## 4.5 汇编源程序结构

- ✓ 一个汇编语言源程序一般由几个段组成，每个段都是可独立寻址的逻辑单位。任何一个源程序至少必须有一个代码段和一条作为源程序结束的伪指令END。若程序中需要用到数据存储区，则要定义数据段，必要时还要定义附加数据段。一个源程序文件可以有多个数据段、多个代码段或多个堆栈段。每个段的排列顺序是任意的。

# 程序结构

- ✓ **NAME** <模块名> ; 可有可无
- ✓ **TITLE** <正文> ; 可有可无
- ✓ **STACK** **PARA** **STACK** **'STACK'**
- ✓ ; 此处输入堆栈段代码
- ✓ **STACK** **ENDS**
- ✓ **DATA** **SEGMENT**
- ✓ ; 此处输入数据段代码
- ✓ **DATA** **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**
- ✓ **SUB\_NAME** **PROC** <TYPE>
- ✓ ; 此处输入子程序指令
- ✓ **RET**
- ✓ **SUB\_NAME** **ENDP**.
- ✓ **CODE** **ENDS**
- ✓ **END** **START**

# 程序结构例题：

- ✓ NAME E456.ASM
- ✓ TITLE DISPLAY—STRING
- ✓ DATA SEGMENT ; 定义数据段
- ✓ STRING DB 'HOW DO YOU DO !',13, 10, '\$' ; 定义输出字符串, \$为串结束符
- ✓ DATA ENDS
- ✓ STACK SEGMENT PARA STACK ; 定义堆栈段
- ✓ DW 20H DUP (0)
- ✓ STACK ENDS
- ✓ CODE SEGMENT ; 定义代码段
- ✓ ASSUME CS: CODE, DS: DATA, SS: STACK
- ✓ START: MOV AX, DATA
- ✓ MOV DS, AX ; 置数据段基地址
- ✓ CALL DISPLAY ; 调子程序DISPLAY
- ✓ MOV AH, 4CH
- ✓ INT 21H ; 并返回DOS
- ✓ DISPLAY PROC
- ✓ MOV DX, OFFSET STRING
- ✓ MOV AH, 9 ; 9号DOS功能调用, 输出字符串
- ✓ INT 21H
- ✓ RET ; 返回子程序调用处
- ✓ DISPLAY ENDP
- ✓ CODE ENDS
- ✓ END START ; 主程序结束

## 4.6 上机操作过程

- ✓ 汇编语言源程序是一个**不可执行的文本文件**，必须用**汇编程序**和**连接程序**对源程序及中间代码进行加工，最后**生成 .EXE**或 .COM文件后，才能在系统环境下直接执行，得到预期的结果。整个上机操作过程可归纳如下：
- ✓ (1) 用文本编辑软件编辑源程序，生成源程序**.ASM文件**；
- ✓ (2) 用**汇编程序**对.ASM文件进行汇编，生成目标程序.OBJ文件；
- ✓ (3) 用**连接程序**对.OBJ文件进行装配连接，生成可执行文件.EXE；
- ✓ (4) 在系统环境下直接输入文件名执行.EXE文件；
- ✓ (5) 如果结果有错，则返回（1）修改或用**DEBUG调试工具**直接对.EXE文件进行修改调试，否则结束。

## 4.6.1 软件环境

✓为运行汇编语言程序，需要用到以下工具软件：

- ✓(1) 编辑程序，如EDIT.COM；
- ✓(2) 汇编程序，如MASM.EXE；
- ✓(3) 连接程序，如LINK.EXE；
- ✓(4) 调试程序，如DEBUG.EXE。

## 4.6.2 生成执行文件

- ✓汇编（编译）
- ✓连接
- ✓执行



## 4.6.3 DEBUG调试

✓各种命令。