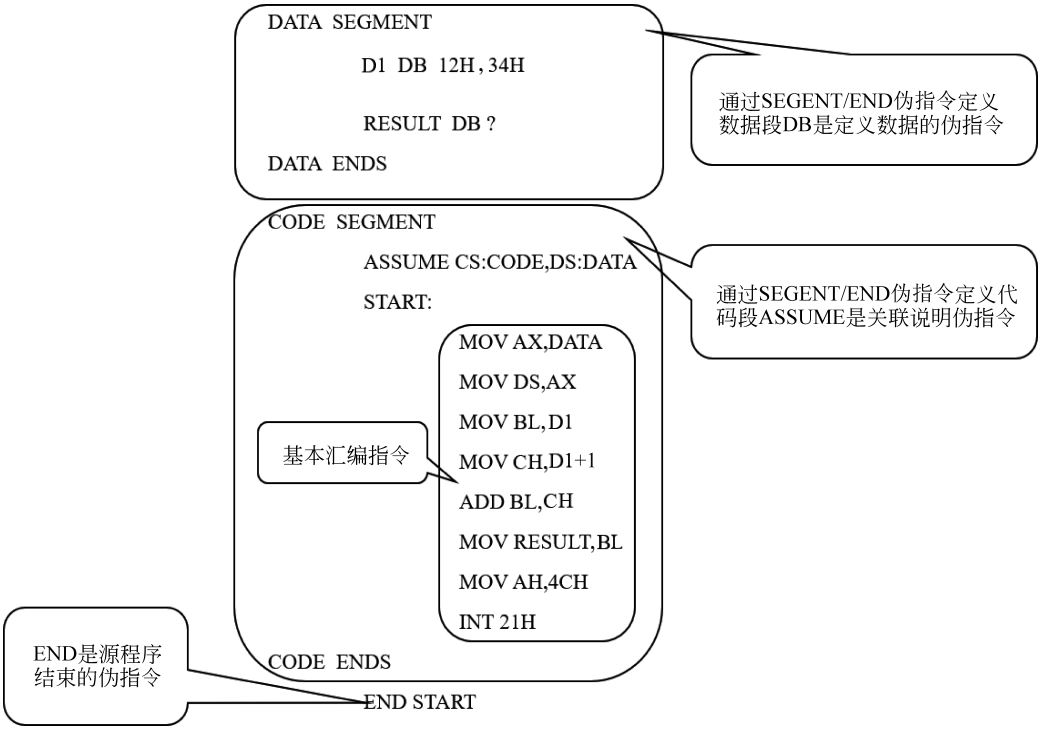
第 5 章 chapter 5

汇编语言程序格式

计算机语言的发展经历了机器语言、汇编语言到高级语言的发展过程。机器语言用 0 和 1 编写代码，这种代码难于编写、纠错和维护；汇编语言是符号语言，通过编写汇编指令对计算机硬件直接发出命令，让内部各个部件直接进行各种运算，相对于机器语言编写、纠错和维护方便。高级语言程序书写更简单，但是各个函数之间的参数传递、指针比较复杂，逻辑结构性强。

汇编语言程序的结构形式有简短模式定义和完整模式定义，不管哪种模式，汇编语言程序由各个段构成，整个程序的书写要用到各种伪指令。下面通过一个实例来介绍汇编语言源程序。

N3（伪指令引言）



以上程序实现任意两个数的和，结果存放在 RESULT 单元。

整个汇编语言源程序基本组成单位是指令。在 8086 宏汇编 MASM 中使用的指令有 3 种类型，分别为汇编指令、伪指令和宏指令。其中，汇编指令和伪指令是最常用的指令。

1. 汇编指令

汇编指令是第 4 章学习的 8086CPU 指令系统中的各种指令。汇编程序在汇编源程序时，每条指令都会产生相应的机器语言目标代码。计算机执行目标代码实现指令的功能。

2. 伪指令

伪指令确定源程序如何分段、逻辑段和哪个段寄存器关联等操作。伪指令在汇编程序汇编源程序时全部完成，不产生机器语言目标代码。

3. 宏指令

宏指令是编程人员按照一定的规则自己来编写的一种指令。一般宏指令中包括伪指令和汇编指令。在实际应用中如需重复执行某些指令序列，为了源程序书写简单、可读性强，可以将这些指令序列定义为一条宏指令。

学习目标：

- 掌握定义段的格式及用法。
- 掌握 ASSUME、END 伪指令的格式及用法。
- 掌握定义数据伪指令的格式及用法。
- 掌握符号伪指令的用法。
- 掌握常用 DOS 系统功能调用。

5.1 段定义伪指令

N4（定义段的伪指令）

段定义伪指令的作用是在汇编语言源程序中定义逻辑段。常用的段定义伪指令有 SEGMENT/ENDS，SEGMENT 代表段的开始，ENDS 代表段的结束。

定义格式：

```
段名  SEGMENT [定位类型] [组合类型] ['类别'] [;注释]
                ⋮
段名  ENDS
```

说明：

📖 段名

段名代表段起始地址，由用户自己定义，定义时做到见名知意，增强可读性。用户定义段名时，必须符合标识符的规则，标识符规则如下：标识符由字母 a～z（不区分大小写）、数字（0～9）、专用字符（ ?、.、_、@ ）构成。但数字不能放在第一位，标识符中如果有“.”必须放在第一位，最大长度为 31。保留字不能用作标识符。例如：

SUM、NAME 和 CLASS_1 是正确标识符。

2ARA 和 DA.是错误标识符。

SEGMENT/ENDS 伪指令

SEGMENT 代表段的开始，ENDS 代表段的结束。SEGMENT/ENDS 必须是成对出现，缺一不可，二者前面的段名必须一致。一个汇编源程序中不同逻辑段的段名应各不相同。

定位类型、组合类型和类别

定位类型、组合类型和类别是段名的属性，用方括号括起来表示是可选项，若不省略，各项之间的顺序不能改变，且它们之间用空格分隔。

定位类型

定位类型是对该段的起始地址提出要求，告诉汇编程序如何确定各个逻辑段在存储器中的位置边界，即各个逻辑段装配在一起，前一个段存放好后，后一个段应该从什么起始边界开始存放。定位类型共有以下 4 种。

(1) BYTE (边界起始地址＝××××　××××　××××　××××B)

BYTE 类型表示本段起始地址可以从任意地址处开始存放。

(2) WORD (边界起始地址＝××××　××××　××××　×××0B)

WORD 类型表示本段起始地址从一个偶数地址处开始存放，即段起始地址的最低一位必须是 0。

(3) PARA (边界起始地址＝××××　××××　××××　0000B)

PARA 类型表示本段起始地址必须能被 16 整除的地址处开始存放，即段起始地址最低 4 位必须是 0。如果省略定位类型，则默认值为 PARA。

(4) PAGE (边界起始地址＝××××　××××　0000　0000B)

PAGE 类型表示本段起始地址必须能被 256 整除的地址处开始存放，即段起始地址最低 8 位必须是 0。

注释字段

注释是一个任选项。注释必须以分号“;”开始，它可对程序或指令加以注解，提高程序的可读性。当需要较多文字说明时，注释可以独占一行或多行，但每行第一个有效字符必须是分号。注释字段的内容不影响程序和指令的功能，它不生成机器目标代码。

【例 5.1】 定位类型应用举例。

```
DATA   SEGMENT              ;DATA 段,定位类型为 PARA
    ⋮
DATA   ENDS
DATA1  SEGMENT WORD         ;DATA1 段,定位类型为 WORD
    ⋮
DATA1  ENDS
EXTRA  SEGMENT PAGE         ;EXTRA 段,定位类型为 PAGE
    ⋮
EXTRA  ENDS
```

本例中共有 3 个逻辑段，段名和定位类型如下。

段名	定位类型
DATA	PARA
DATA1	WORD
EXTRA	PAGE

📖 组合类型

组合类型的作用是告诉连接程序本段和其他段的组合关系。而8086CPU只允许同时访问4个段。利用组合类型将同性质的所有段连接在一起构成一个段，且总长不超过64KB。组合类型共有以下6种。

(1) NONE类型：未指定组合类型，表示本段与其他段无组合关系。装入内存时，各自进行装入即可。如果省略组合类型，默认为NONE类型。

(2) PUBLIC类型：将与本段同名的段组合在一起，形成一个新的逻辑段装入内存，共用一个段的起始地址。

(3) STACK类型：STACK类型的作用同PUBLIC类型基本一样，不同点是STACK类型仅限于堆栈区的逻辑段。

(4) COMMON类型：将与本段同名的段从同一个地址开始装入内存，形成一个覆盖段。段的长度为最长段的长度。

(5) MEMORY类型：当几个逻辑段连接时，本逻辑段定位在地址最高位置。如果连接的逻辑段中有多个段的组合类型为MEMORY，则汇编程序将先遇到的段作为MEMORY类型，其他段作为COMMON类型处理。

(6) AT表达式：本逻辑段根据表达式的值定位段基址。例如，AT 3200H表示本段的段基址为3200H，则本段从内存物理地址为32000H的位置开始装入。

📖 类别

类别必须放在单引号内，类别的作用是在连接时确定各个逻辑段的装入顺序。当几个程序模块进行连接时，将相同类别名的逻辑段按先后顺序装入连续的内存。没有类别名的逻辑段与其他无类别名的逻辑段一起连续装入内存。

5.2 ASSUME、END伪指令和标号

N5（ASSUME伪指令）

5.2.1 ASSUME伪指令

ASSUME是段寻址伪指令，用来说明各个段寄存器分别和哪个逻辑段关联在一起。

格式：

ASSUME 段寄存器名:段名[,段寄存器名:段名…]

说明：

📖 ASSUME伪指令

ASSUME伪指令说明段寄存器和各个逻辑段的关联关系，并没有给段寄存器赋值。一般地，ASSUME伪指令放在代码段中，必须放在第一行。

📖 段寄存器：段名

段寄存器名对于 8086CPU 而言为 CS、ES、SS 和 DS 中选取，段名是用户自己定义的各个逻辑段的段名。

📖 CS:IP

CS 和 IP 中的地址由计算机自动装入，不需要用户手动装入。

📖 DS、ES 段寄存器由用户手动装入

【例 5.2】 定义三个段的构架，实现段寄存器 DS 和 ES 用户手动装入。

```
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
      MOV AX,EXTRA
      MOV ES,AX
       ⋮
CODE   ENDS
   END START
```

1. ASSUME伪指令将段寄存器和各个逻辑段关联说明。
2. 关联后各个段寄存器没有值。
3. 关联时没有顺序要求。

1. DATA代表数据段的段基址，将DATA段的段基址装入DS。

1. EXTRA代表附加段，将EXTRA段的段基址装入ES。
2. 各个段的段基址装入时没有顺序要求。

📖 SS 段寄存器，计算机自动装入

这种方法需要利用定义段时组合类型中 STACK 类型，然后用 ASSUME 伪指令进行关联说明，SS 段寄存器可以实现计算机自动装入。

【例 5.3】 定义三个段的构架，实现段寄存器 SS 计算机自动装入。

```
DATA   SEGMENT              ;定义数据段
    ⋮
DATA   ENDS
STACK1  SEGMENT STACK       ;定义堆栈段
    ⋮
STACK1  ENDS
CODE   SEGMENT              ;定义代码段
    ASSUME  CS:CODE,DS:DATA, SS:STACK1
    START:
        MOV AX,DATA
        MOV DS,AX
            ⋮
CODE   ENDS
    END START
```

定义STACK段，指定组合类型为STACK，SS段寄存器实现计算机自动装入。缺组合类型STACK时必须手动装入。

📖 SS段寄存器，用户手动装入

在定义段时，组合类型没有选用STACK类型，需手动装入。

【例5.4】 定义三个段的构架，实现段寄存器SS用户手动装入。

```
DATA   SEGMENT                      ;定义数据段
    ⋮
DATA   ENDS
STACK1  SEGMENT  省略 STACK 类型    ;定义堆栈段
    ⋮
STACK1  ENDS
CODE   SEGMENT                      ;定义代码段
    ASSUME  CS:CODE,DS:DATA, SS:STACK1
    START:
        MOV AX,DATA
        MOV DS,AX
        MOV AX,STACK1
        MOV SS,AX
          ⋮
CODE   ENDS
    END START
```

定义堆栈时缺组合类型，SS需手动装入。

5.2.2 END伪指令

END伪指令一方面表示汇编源程序结束，是源程序最后一条语句。当汇编程序遇到END伪指令时，END后面的语句汇编程序不再进行处理。另一方面，END伪指令告诉汇编程序在程序装入内存时将程序的启动地址分别提供给CS和IP。

格式：

END [标号]

END的作用是将标号的地址作为程序执行的启动地址（将标号的段基址和偏移地址分别提供给CS和IP寄存器）。标号是可选项，代表程序执行开始地址。

【例5.5】 定义两个段的构架，实现标号START和END伪指令的用法。

```
DATA   SEGMENT          ;定义数据段
    ⋮
DATA   ENDS
CODE   SEGMENT          ;定义代码段
    ASSUME  CS:CODE,DS:DATA
    START :
        MOV AX,DATA
        MOV DS,AX
          ⋮
CODE   ENDS
    END START
```

START是标号

1. START代表程序执行开始的位置，将START标号的段基址和偏移地址提供给CS和IP寄存器。
2. 结束整个源程序。

5.2.3 标号

标号代表一条指令语句的符号地址。在汇编语言源程序中，如果从指令 1 位置跳转到指令 6 位置，指令 6 位置通过定义标号来标识。转移指令或其他指令中可直接引用这个标号，所以标号可作为转移类指令的操作数，即转移地址。

1. 定义标号

标号需用户自己定义，最好做到“见名知意”，标号需符合标识符规则。例如：
NEXT、START、LOOP1 和 SUM 为合法标号。
2NEXT、1_CLASS、&SUM 和%ABC 为不合法标号。

2. 标号使用格式

标号：
标号定义好后，使用标号后加：。

3. 标号的属性

标号有三个属性，分别为段属性、偏移地址属性和类型属性。

📖 段属性

段属性表示标号所在段的段基址，其值通常存放在 CS 段寄存器中。

📖 偏移地址属性

偏移地址属性表示标号所在位置距其所在段首地址的字节数。

📖 类型属性

标号的类型属性有两种：分别为 NEAR 类型和 FAR 类型。NEAR 类型的标号只能在段内使用，而 FAR 类型的标号可在段内使用，也可在段间使用。

【例 5.6】 定义两个段的构架，通过标号实现转移。

```
DATA  SEGMENT          ;定义数据段
   ⋮
DATA  ENDS
CODE  SEGMENT          ;定义代码段
    ASSUME  CS:CODE,DS:DATA
START :
      MOV AX,DATA
      MOV DS,AX
      ⋮
NEXT :
      MOV AL,BL
      ⋮
      ADD AL,BL
JC   NEXT
CODE  ENDS
END START
```

使用START标号，所以加：。START标号代表MOV AX,DATA指令的偏移地址。

使用NEXT标号，所以加：。NEXT标号代表MOV AL,BL指令的偏移地址。

如果CF=1，从该位置跳转到NEXT代表的位置，即MOV AL，BL位置执行。如果CF=0，则不跳转，顺序向下执行。

START代表程序执行开始的位置，将START标号的段基址和偏移地址提供给CS和IP寄存器。

5.3 数据定义伪指令


N7（定义数据伪指令）

数据定义伪指令的作用是定义一个变量，可以给变量赋值，也可以只给变量分配空间。数据定义伪指令有五条，分别为 DB.DW、DD.DQ 和 DT，常用伪指令为 DB、DW 和 DD。

5.3.1 定义变量格式

格式：

[变量名] 伪指令 操作数 1[,操作数 2…] [;注释]

方括号中内容可省略，可有多个操作数，操作数之间用逗号分隔。

说明：

📖 变量名

变量名用户自己定义，最好做到“见名知意”，变量名符合标识符的规则。变量名代表第一个数据的偏移地址。

📖 伪指令

1. DB（Define Byte）


N8（DB-DW-DD 伪指令）

DB 伪指令定义变量类型为字节类型，每一个操作数占 1 个字节。

【例 5.7】 定义字节变量，空间分配，如图 5.1 所示。

```
DATA  SEGMENT
    DAT1  DB  12H,45H,67H
DATA  ENDS
```

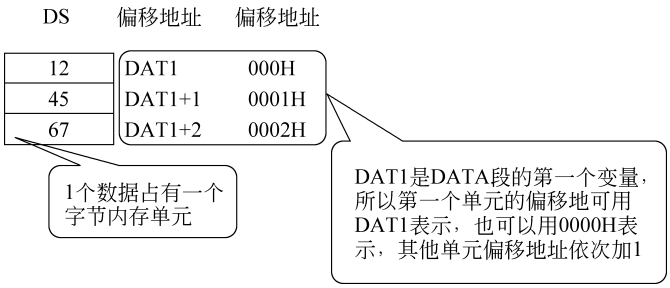


图 5.1 字节类型变量

2. DW（Define Word）

DW 伪指令定义变量类型为字类型，每一个操作数占 2 个字节。在内存存放时，采用“高高低低”规则，低地址存放低字节数据，高地址存放高字节数据。

【例 5.8】 定义字变量，空间分配，如图 5.2 所示。

```
DATA  SEGMENT
    DAT2  DW  90H,5612H,02H,8734H
DATA  ENDS
```

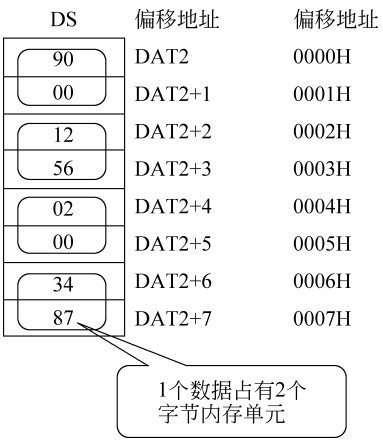


图 5.2 字类型变量

3. DD（Define Double Word）

DD 伪指令定义变量类型为双字类型，每一个操作数占有 4 个字节。在内存存放时，采用“高高低低”规则，低地址存放低字节数据，高地址存放高字节数据。

【例 5.9】 定义双字变量，空间分配，如图 5.3 所示。

```
DATA  SEGMENT
    DAT3  DD  128990H,4512H,06H
DATA  ENDS
```

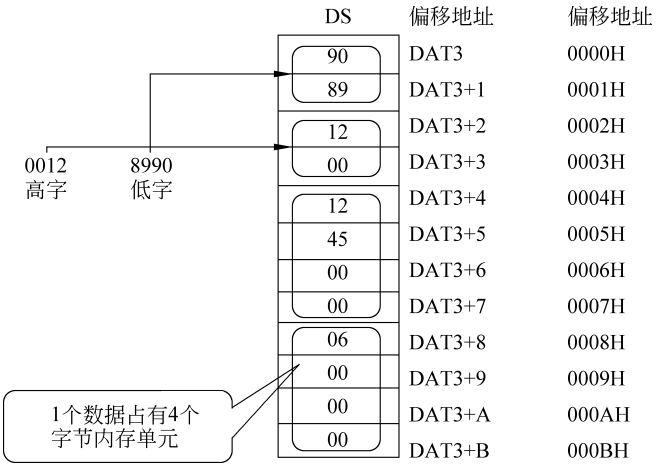


图 5.3 双字类型变量

4. DQ

DQ 伪指令定义变量类型，每一个操作数占 8 个字节。

5. DT

DT 伪指令定义变量类型，每一个操作数占 10 个字节。

5.3.2 变量属性

变量有 3 种属性，分别为段属性、偏移属性和类型属性。

1. 段属性


N9（变量的三种属性）

变量段属性是变量所在段的段基址。变量必须定义在一个段内。编写程序时需要某个变量的段基址，一种方法是用该变量所在段的段名，另一种方法是在变量名前面加上 SEG 运算符。

📖 SEG 运算符格式：

SEG 变量名或标号

功能：提取变量或标号所在段的段基址。

【例 5.10】 定义字变量，提取字变量所在段的段基址。

```
DATA  SEGMENT
    DAT4  DW  90H,2312H
DATA  ENDS
```

第一种方法：

```
MOV AX,DATA              ;段名代表段基址，段基址传送至 AX
MOV DS,AX                ;AX 中数据传送至 DS
```

第二种方法：

```
MOV AX,SEG DAT4          ;通过 SEG 提取 DAT4 变量所在段的段基址传送至 AX
MOV DS,AX                ;AX 中数据传送至 DS
```

2. 偏移属性

变量偏移属性是变量所在本段内偏移地址。偏移地址表示段内某一位置到段基址的距离，偏移地址为 0 表示在段基址处。编写程序时需要某个变量偏移地址，一种方法是用 LEA 指令提取，另一种方法是在变量名前面加上 OFFSET 运算符提取。

📖 OFFSET 运算符格式：

OFFSET 变量名或标号

功能：提取变量或标号的偏移地址。OFFSET 运算符需和 MOV 指令联用。