MCS-86™宏汇编语言

参考手册

北京工业大学
微型计算机研究开发应用中心

再 版 前 言

本书由于是第二次出版，较之原来第一版作了较大修改。在听取有关用户的意见后对于大量系统机中宏语言等内容基本删除，这样便于阅读，也提高了对TP-86单板机的实用价值。关于宏汇编的有关内容我们只保留了一些概念，特此说明。

编 者

84年11月

目　录

第一章 MCS-86 宏汇编语言概述

MCS-86 宏汇编语言是专为编写 16 位的 8086 处理机程序而设计的，本章是它的简介。图 1.1 表示出 MCS-86 宏汇编程序在 8086 软件开发过程中的作用。

MCS-86 宏汇编程序能把 8086 宏汇编语言程序变为目标模块，它是 MCS-86 系列产品中的一种工具，属于 MCS-86 的工具还有：

CONV86 能把没有错误的 8080/8085 源文件变成句法正确的 8086 源文件，并能产生由于变换所引起的出错信息和警告信息。它能告诉程序员哪些地方需要重新编辑。

PLM86 能对PL/M-86 高级语言编写的程序进行编译，产生目标模块。

LINK86 能把若干目标模块连接成浮动地址的模块。

LOC86 能把浮动地址的模块变为绝对地址的模块。

ICE-86 能对 8086 进行联机的仿真。

和 MCS-86 有关的软件手册请参阅图 1.1 的说明

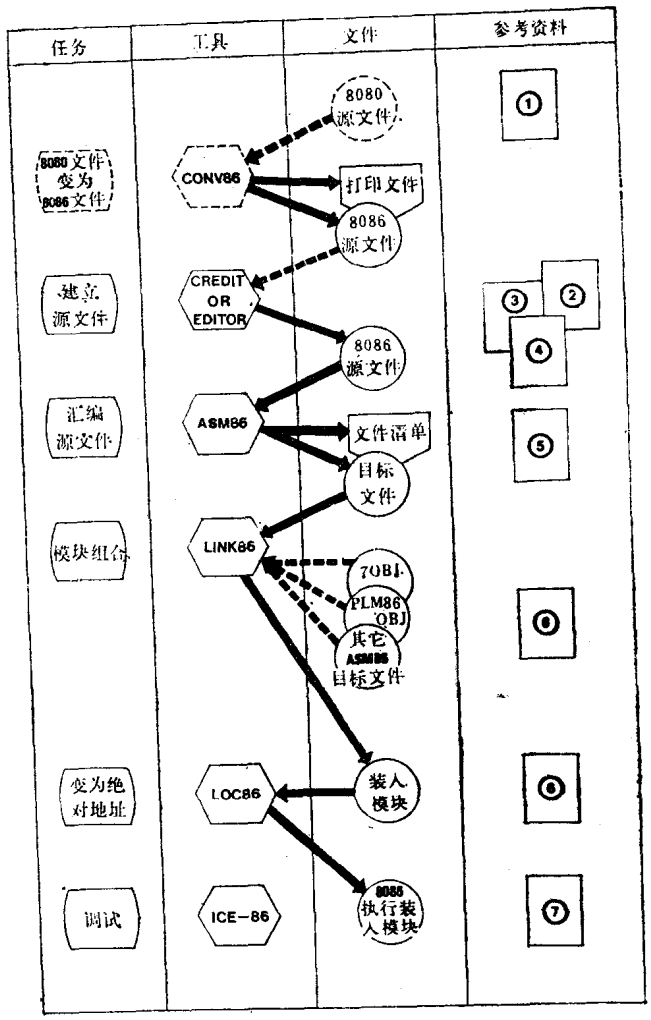
MCS-86 宏汇编语言提供了一些功能很强而使用简单的编程方法。这种汇编语言程序经过汇编、连接、定位之后，可以用 8086 处理机来执行。如果读者还不熟悉 8086 的设计，可以先阅读 8086 系列产品的用户手册。

1.1 为什么要用汇編語言編写程序？

虽然采用高级语言（PL/M，BASIC，FORTRAN，PASCAL）能够节省软件的开发时间，但它们不允许程序员直接使用集成电路芯片的许多特性，例如寄存器、处理机的标志和一些特征的指令。另外，高级语言的编译程序和解析程序所产生的代码比较冗长。虽然汇编语言程序的编程和调式需要较长的时间，但是它比与它等效的高级语言程序执行起来快得多，而且所占内存容量也少得多。在每一个具体的应用场合，因为软件的开发时间相当昂贵，所以，应当对软件的开发时间和软件的质量进行抉择。最佳的方案往往是在某些对执行时间和内存容量要求较高的程序采用汇编程序编写，而其它的程序则采用高级语言编写。

1.2 宏处理器是ASM86的一个組成部份嗎？

调用 ASM86 宏处理器时就把控制数交给 MCS-86 汇编程序，而宏汇编程序的最前面就是宏处理器。宏处理器首先扫视源文件，查找用宏处理器语言（MPL）编写的宏



① MCS 汇编语言变换器操作说明书

② CREDIT 操作说明书

③ ISIS-11 用户指南

④ MCS-86 宏汇编语言参考手册

⑤ MCS-86 宏汇编程序操作说明书

⑥ MCS-86 软件包操作说明

⑦ ICE-86 在线仿真器操作说明书

图 1.1 软件开发和 MCS-86 宏汇编语言

定义和宏调用。根据宏定义对宏调用进行扩展。MCS-86 的宏汇编程序，就是这样把宏汇编语言程序变成汇编语言程序的。利用第七章中所讲的宏处理器语言，可以用一串汇编语言指令来建立某些特殊用途和功能很强的算法程序库。实际上，混合使用 MPL 和 MCS-86 汇编语言，既可以缩短汇编语言的开发时间，也可以提高运行的时间效率。

在本手册中 MCS-86 宏汇编语言是指汇编语言本身，而 MCS-86 宏汇编程序是指能处理 MPL 和汇编语言的软件。

1.3 汇編語言的特点

每种语言都有自己的特点。本章所介绍的汇编语言有以下特点：

(1) 功能很强的指令系统。

(2) 只有在一般高级语言中才能找到的代码和数据结构技巧。

(3) 它的汇编程序能对数据和数据说明的一致性进行校验。它所产生的指令目标代码是比较简短有效的。

(4) 它的宏语言能与任何字符串处理语言比美。

1.4 指令系統的設計

绝大多数的汇编语言对指令助记符（例如MOVE，ADDI）和操作代码（二进制代码）采用一一对应的方式。MCS-86宏汇编语言却采用一对若干的方式，即根据操作数类型的说明，一个单指令助记符可以汇编成若干操作代码中的一种。也就是汇编程序能根据操作数的类型选择相应的操作代码。

MCS 宏汇编语言是一种类型化（TYPED）的语言，因为：可以把数据说明为不同的类型（例如字节，字，双字）；在同一操作中不允许两种操作数类型混合使用（例如把说明为字节的数据传送给字的寄存器）。

类型化的结果可以避免下列几种差错：

(1) 把一个字传送给一个字节的目标单元，而冲掉了与它相邻的字节。

(2) 把一个字节传送给一个字的目标单元，而产生一个无意义的相邻字节。

但是，类型化并不妨碍不同类型的数据之间的传送，例如 DATA8 被说明是字节类型，而 DATA16 被说明是字类型，那么它俩之间的传送可以用下列的方法来实现：

(1) 如果要把 16 位寄存器 AX中的内容传送给DATA8，对DATA8的内容进行重写，则可写：

```
MOV WORD PTR DATA8, AX; 把 AX 的内容传送给 DATA8
```

(2) 如果要把 8 位寄存器AL的内容传送给DATA16的低字节，而且使DATA16的高字节保留不变，则可写：

```
MOV BYTE PTR DATA16, AL; 把AL传送给DATA16
```

在上面两例中，利用了指示器操作符 PTR，它在执行一条指令的期间暂时对数据的类型作了规定。如果没有这种 PTR 的暂时规定，指令就按原始的数据类型说明执行。

有好几种方法可以访问类型化的数据。如果要既按字节又按字访问数组，可利用上述的 PTR，或者对数据作如下说明：

```
DATA8—ARRAY LABEL BYTE; 说明以下按字节类型访问数组
DATA16—ARRAY DW100 DUP (0); 说明数组是 100 个填零的双字节现在如果要把 8 位寄存器 CH 的内容传送给数组的第 10 个字节，可写成：
MOV DATA8—ARRAY[9], CH; 第一字节为 0 字节
```

如果要把16位寄存器的内容传送给数组的第10和第11个字节，可写为：

MOV DATA16—ARRAY[9], CX

有关类型的修改方法将在第4章中作完整的介绍。虽然在上例中是采用MOV指令，但方法上也适用于其它两个操作数的指令（例如ADD, SUB, AND, XOR等）。

一、一般指令的助记符和代码宏

数千种不同的操作可以用大约100个汇编语言的助记符来表达，例如不论是字的传送，字节的传送或者立即数的传送都是用MOV助记符。所以，不需要去背诵上千条不同操作的指令，只要记住一些能用于一组操作的助记符，和如何去用操作数（数据项）进行编程，MCS-86宏汇编程序就能把它变换成相应的机器代码。

上述的一般指令的这种功能是靠代码宏来实现的。代码宏主要是用于确定指令的汇编格式（在第六章中将详细说明），不应当把它和第七章的宏互相混淆，宏只是用于规定源程序如何变换为汇编语言程序。

汇编程序通过访问代码宏的助记符（其中也包括寄存器、存贮器、位移量、操作码和类型信息），把汇编语言指令变换成机器语言指令。汇编程序把助记符的操作数与代码宏为那个助记符规定的操作数进行比较，找出相同的操作数类型，然后把代码宏扩展为机器指令，这就是基本的汇编过程，虽然也可以对代码宏重新定义，设计出自己的指令系统，但如果只是为了编写程序，并不需要知道代码宏的内容。附录1中给出了MCS-86汇编语言的全部代码宏。在第五章中表示出汇编后的指令格式。

二、指令分析

在一条简单易写的指令中包含了许多信息、我们以下列的指令为例来进行剖析：

ADD[BP][SI].STRUCFIELD3, DX

即使还不知道这里所讲的汇编语言，也可以推断出：

1. 位寄存器DX的内容是总数的一部份；
2. 这条指令的操作内容是一个字的加法运算；
3. 寄存器BP和BI是以某种方式用于地址的计算；
4. 标识符STRUCFIELD3和句点“.”也参加了地址的计算。

图1.2表示出地址计算的全过程，说明如下：

(1) DX是第二个操作数，所以它是源操作数，它的内容与目标操作数相加，目标操作数是按表达式[BP][SI].STRUCFIELD3寻址的一个字。

(2) 寄存器BP（有时也称为堆栈标记）在表达式中表示第一操作数的基址。当BP用于表示基址，如果没有加上段的说明，操作数就应当是在当前的堆栈段中。因为堆栈段是用SS段寄存器中16位数值来表示节（PARAGRAPH）的编号，所以，表达式的段的值可认为是已知的。

(3) SI寄存器的16位内容在这里是作为目标操作数的变址指示器，即把BP和SI的内容相加起来，形成第一个操作数的有效地址部份。

(4) 句点放在STRUCFIELD3的前面，表示这是一个结构的标识符，STRUCFIELD3

是结构的字段，其中的数值 3 表示距离这个结构起点的距离（以字节数计算）。结构这种形式可以用作一种存贮结构或者相对位移量数值的图形，参阅第三章的详细说明。

（5）因此第一操作数的地址就是：

16位的节的编号：SS

16位的位移量：BP + SI + （STRUCFIELD3的相对位移量）

20位的机器地址：16*SS + BP + SI + （STRUCFIELD$_3$的相对位移量）

这样，ADD[BP][SI].STRUCFIELD3,DX 这条指令的作用就是把 16 位的 DX 寄存器的内容，和堆栈中一个 16 位的数值相加，这个数值是根据上述 20 位机器地址在存贮器中找到的。这条指令包括指令操作码、基址寄存机、变址寄存器、结构位移量和相对位移量、类型信息和源寄存器等，总共才占用三个字节。

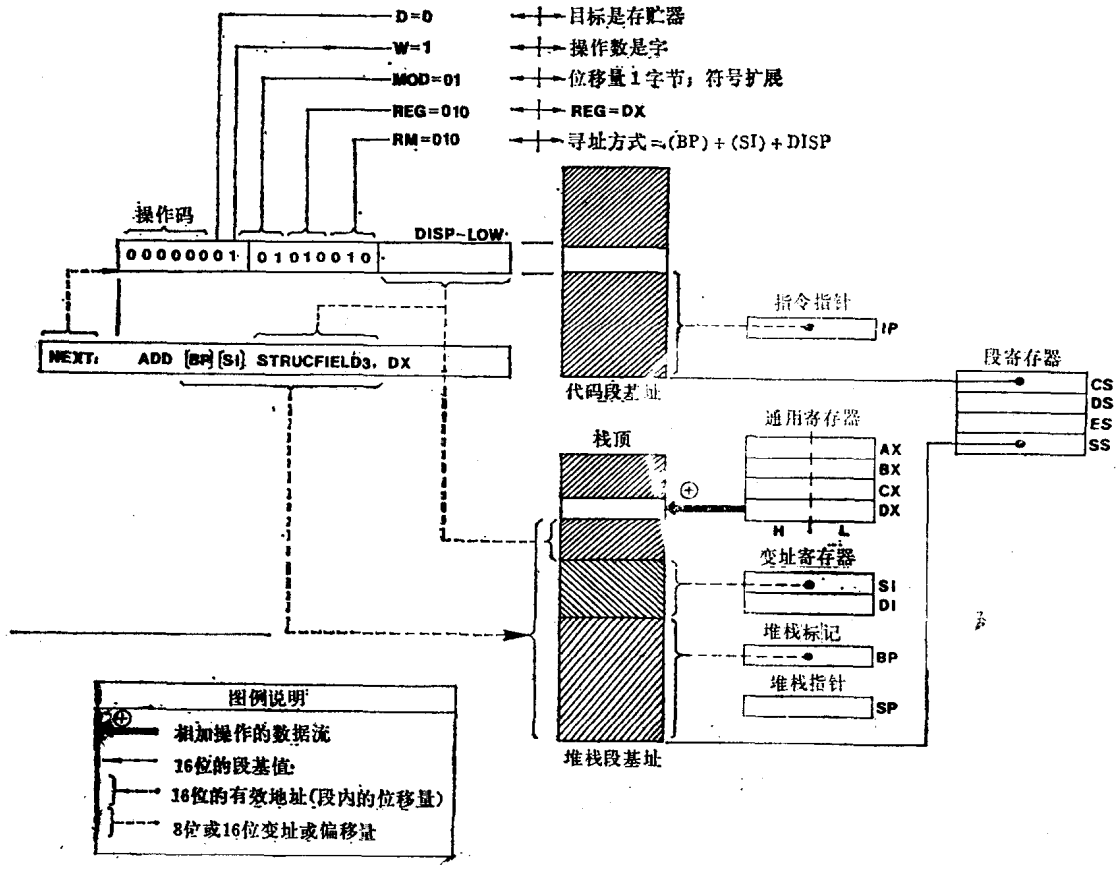


图 1.2　指令分析举例

1.5 代碼和数据的結构

一、结构（STRUCTURES）

首先说明一下，为了访问变量（不论是否放在堆栈中），怎样利用结构去对存贮结构

进行定义，可以把逻辑上互相关连的数据项聚集起来并定义为结构，如下列所示：

假设有一个用户命名的结构 PHYS，允许程序员用符号按各个字段的字节数容量为1，2，2，1和2的顺序进行定义（它们的相对位移量是汇编程序所产生的）。

```
PHYS    STRUC       ; 不保留存贮单元，
AGE     DB0         ; 访问.AGE 所产生的位移量为0.
EYES    DW0         ; 访问.EYES 所产生的位移量为1.
HAIR    DW0         ; 访问.HAIR 所产生的位移量为3.
HTFT    DB0         ; 访问.HTFT 所产生的位移量为5.
HTIN    DW0         ; 访问.HTIN 所产生的位移量为6.
PHYS    ENDS
```

程序员可以用 PHYS 作为操作符对这个结构进行预置和分配实际的存贮内容如下：

```
JONESPHYS<22, 'BI', 'BR', '5''10>; 分配8个字节，预置。
```

其中，用户命名的 PHYS 是一个操作符，结构的预置采用了整数22和字符数据（'BL'，'5'）。

利用操作符 PHYS 进行预置的结果与执行下列指令的结果是一样的：

```
JONES DB8 DUP(? )          ; 分配8个字节。
MOV  JONES.AGE,22          ; AGE 字段的预置。
MOV  JONES.EYES,'BL'       ; EYES 字段的预置。
MOV  JONES.HAIR,'BR'       ; HAIR 字段的预置。
MOV  JONES.HTFT,'5'        ; HTFT 字段的预置。
MOV  JONES.HTIN,'10'       ; HTIN 字段的预置。
```

在第二章中还提供了一些能用于定义结构的其它手段，例如在定义结构时如何对未定义的数值进行预置，然后在分配存贮时又如何对未定义的数值进行重写，这些都是在汇编时进行的。

结构是一般汇编语言中所没有的很有价值的编程工具。在第二章将讨论结构的定义，存贮分配和预置。

二、数组

可以用 DB，DW，DD，结构名称，记录名称等伪指令分别对字节，字，双字，结构和记录（在下面介绍）进行定义和预置：

```
BYTE—ARRAY     DB 100 DUP(1)        ; 分配100个字节，并把每一字节预置为1
WORD—ARRAY     DW 256 DUP(0)        ; 分配265个字，并把每一个预置为零
DWORD—ARRAY    DD 200 DUP(? )       ; 分配200个双字，不管预置
PHYS—ARRAY     PHYS1000 DUP(<>)     ; 分配1000个经过预置的PHYS
```

当访问数组中的元素时，必需注意以下两点：

1. 数组元素从零算起，例如数组 F00 的第一字节是 F00[0] 而不是 F00[1]，第N

个字节是 FOO[N-1]。

2. 不论数组元素是字节，字或双字，下标可以解释为从数组起始端算起的字节数。

访问数组中的元素有好几种方法，例如：

```
ADD AH, BYTE_ARRAY          ; 数组的第一节和 AH 相加
ADD AH, BYTE_ARRAY[0]       ;          〃
ADD AH, BYTE_ARRAY[7]       ; 数组的第8字节和 AH 相加
ADD AH, BYTE_ARRAY+7        ;          〃
ADD AH, BYTE_ARRAY[SI]      ; 数组的第(SI+1)字节和 AH 相加
MOV BX, WORD_ARRAY+14       ; 数组中的第8个字传送给BX,
MOV BX, WORD_ARRAY[14]      ;          〃
MOV WORD_ARRAY[BX][SI], 7 ; 把7传送给数组中第(BX+SI+1)字节中的字
MOV WORD_ARRAY[BX+SI], 7    ;          〃
```

三、记录

记录和结构是相似的，不同的地方在于记录是用于处理按位计算的信息组的位移量数值，而结构是用于处理按字节计算时信息组的字节位移量数值。一个记录可用于规定一个字节或字的格式，可以对记录中的每一字段进行命名，并分配给它一个长度（按位计算）；然后，当访问某一字段名称时，汇编程序可以通过向右移位来对它进行识别。但也可以不用移位的方法，而用记录操作符 MASK 来进行识别。例如：假设有一个数据共占16位（一个字），它的内容如下：

1. 三个1位的触发器（FF1，FF2，FF3）；
2. 1位是无关紧要的（DNC）（允许为任意值）；
3. 三个十进制数（DIG1，DIG2，DIG3），每个十进制数各占4位。

于是可以把这个数据定义为一个记录如下：

```
GCNZO RECORD FF1:1, FF2:1, FF3:1, DNC:1, DIG1:4, DIG2:4, DIG3:4
```

虽然对记录的定义本身不用分配存贮单元，但是可以利用记录的名称作为汇编时的操作符去分配经过定义的记录，例如：

```
BUFFALO GONZO 100 DUP (<>) ; 分配100个预置后的副本
```

现在如果需要从数组 BUFFALO 的第2个字分离出相当于 DIG2 的字段，并把它保存在寄存器AX中的0～3位，就可以这样写：

```
MOV AX, BUFFALO[2]      ; 把第2个字送入AX
AND AX, MASK DIG2       ; 分离出 DIG2 字段，其余各位变为零.
MOV CL, DIG2            ; 记录的名称所提供的移位次数装入 CL
SHR AX, CL              ; 把 DIG2 字段向右移位，使它和 AX 的0～3位对齐，
```

如果需要按照 GONZO 的格式，从 BUFFALO 中分离出别的字段，可以用别的字段名称取代上面的 DIG2 就可以了。（注：通常字的数组 BUFFALO 中的第 N 个字，应写

为BUEEALO[2*(N-1)]。

利用记录可以对字节或字中的位信息组进行定义和处理，这可以使编程和调试更为方便，只有少数的高级语言才具有这种特点。

四、组合

以上对结构、数组和记录分别都作了说明，我们还可以把这几种数据结构组合起来，例如可以定义出记录的数组和数组的结构。如果引用上述的结构名称PHYS，我们可以为PHYS分别和预置1000个副本如下：

```
LOTSA_PHYS PHYS 1000 DUP (<>) ; 可以看作一个8000字节的数组
```

其中LOTSA_PHYS是新的结构名称，PHYS是上面已经分配和预置了的结构名称。在第三章还有许多例子。

五、程序分段的概念

8086CPU访问存贮器时，是把存贮器分成若干段的，每一段最多可容纳64K字节，因此8086的宏汇编语言也把程序分段存入存贮器。段和程序块的概念不同，段可以看作是一个浮动地址的存贮区，不同的段可以分配在存贮器中的不同存贮区，它们可以是重叠的，也可以是分散的（不连续的）。而程序块则不能重叠，而且是连续的。可以为每一个程序块定义一个段，如果某一程序块没有段的定义说明，汇编程序就用？？SEG作为它的段的名称。用户编的程序中的每一条指令和每一数据项都应当放在某些段中。虽然代码和数据也可以混杂地放在某些段中，但这种做法在实践中是不可取的。下面是分段的实际例子：

用于存放全局数据的段
用于存放局部数据的段
作为堆栈的段
用于存放主程序的段
用于存放公用（可重入的）子程序的段
用于存放串行可重复使用的子程序的段
用于存放中断矢量的段
用于存放中断服务子程序的段

在8086的存贮器中，一个实际的段最多可容纳65535字节，而且必需从一个能被16除尽的绝对地址开始。这样的一种地址称为（PARAGRAPH）节的边界。在8086的存贮器中能作为节的边界的是0，16，32 …,16*65635=1048560。

逻辑的段不一定需要从节的边界开始，因为它不一定要与实际的段相对应。

如上所述，段是从节的边界开始的，所以CS，DS，ES和SS等四个16位的段寄存器中所存放的内容就是节的边界的值（即段的起始地址）。这些边界值表示4个“当前”运行的段的基址。这4个段寄存器的用法如下：

CS 寄存器—当前运行的代码段

DS 寄存器—当前运行的数据段

SS 寄存器—当前的堆栈段

ES 寄存器—辅助的数据段

在运行的时候，8086每次访问存储器的实际地址是由两部分组成的。

(1) 16位的段的基址，它必须是四个段寄存器的内容之一。

(2) 16位的有效地址，它表示访问存储器时离开段的基址的位移量。

有效的地址通常是在汇编时计算出来的，而段的基址可以在汇编、定位或运行时加以说明。程序员应当把段的基址妥善地保存在段寄存器中。

当从存储器取出一个数据项时，8086CPU 的硬件把那个数据项的16 位段的基址和16位的有效地址，按下面的等式计算出20位的实际地址：

20 位的实际地址 = 16*（段的基址）+有效地址，例如 GONZO 在 数据段中的位移量是 1200H，段寄存器 DS 中的值是 5D00H，那么 GONZO 的绝对地址（实际 地址）就是：

$$16*5D00H + 1200H = 5E200H$$

在用汇编语言编程时，实际上不需要考虑 20 位 的绝对 地址。一般的习惯是用符号去代替段的基址和有效地址，不应当对段的基址另外再作算术运算。

六、过程

汇编语言中用过程的定义来实现子程序的功能。通常在一般的汇编语言中都用一个起始标号和一条返回指令作为定义一个子程序的工具，而在 MCS-86 宏汇编语言中是把过程看作一个程序块（也可能是数据块），并用 PROC 和 ENDP语句来定义这个程 序块的范围。使过程的定义十分明确。例如：

```
READFILEPROC
    0
    0
    0
    RET
    0
    0
    0
    RET
READFILEENDP
```

过程是可以嵌套的（即一个被另一个完全套住），但不能重叠。下面 是 过 程 的嵌套例子：