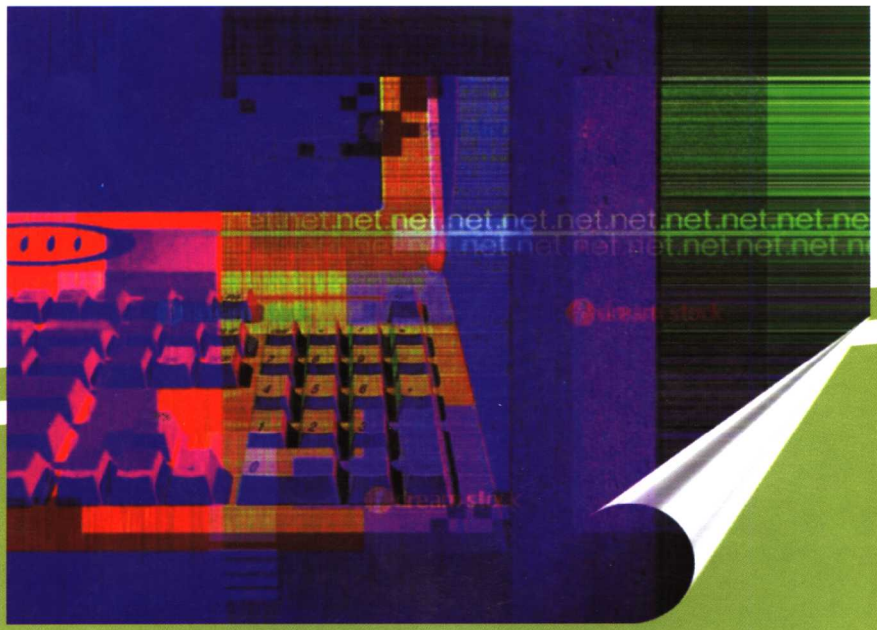
理工科电子信息类 DIY 系列丛书

微型计算机原理及应用
实验指导

邹丽新 陈 蕾
邱国平 周 江 编著



苏州大学出版社

理工科电子信息类 DIY 系列丛书

微型计算机原理及应用实验指导

邹丽新　陈　蕾　邱国平　周　江　编著

苏州大学出版社

图书在版编目(CIP)数据

微型计算机原理及应用实验指导/邹丽新等编著.
苏州：苏州大学出版社，2006.11
（理工科电子信息类 DIY 系列丛书）
ISBN 978-7-81090-779-8

Ⅰ.微… Ⅱ.邹… Ⅲ.微型计算机-高等学校-教学参考资料 Ⅳ.TP36

中国版本图书馆 CIP 数据核字(2006)第 149307 号

内 容 简 介

本书是理工科电子信息类 DIY 系列丛书中的一本。全书以 80X 系列微机系统和 MCS-51 系列单片微型计算机的教学内容为主线，分软件和硬件两大部分，共八章，介绍了 28 个实验。每个实验都分为基础篇和提高篇，相对独立，内容由浅入深，涉及的面较宽，且相对独立，所以不仅适用于本科、大专、职教、成教等各种层次的学生，还可作为从事微机技术工作的工程技术人员的参考书。

微型计算机原理及应用实验指导

邹丽新 陈 蕾 邱国平 周 江 编著

责任编辑 苏 秦

苏州大学出版社出版发行
（地址：苏州市干将东路 200 号 邮编：215021）
常熟高专印刷有限公司印装
（地址：常熟市元和路 98 号 邮编：215500）

开本 787mm×1 092mm 1/16 印张 12 字数 300 千
2006 年 11 月第 1 版 2006 年 11 月第 1 次印刷
ISBN 978-7-81090-779-8 定价：22.00 元

苏州大学版图书若有印装错误，本社负责调换
苏州大学出版社营销部 电话：0512-67258835

电工电子实验教材编委会

序　言

为了进一步加强高等学校综合实力，整合和共享教学资源，充分发挥理工科专业实验实践教学环节对于创新型人才培养的重要作用，从根本上提高本科教学的质量，提高大学生继续深造及创业就业的竞争力，近年来，各大学纷纷建立面向全校甚至所在城市的校级实验教学中心。实验中心的建立，统一和规范了实验仪器配备、实验教学要求，加强了实验教学的人才和梯队建设。

实验教学质量的提高，也离不开实验教材的建设。在苏州大学出版社的大力支持下，苏州大学电工电子实验教学中心学术委员会聘请校内外专家，成立了电工电子实验教材编委会，将陆续组织出版一套理工科电子信息类 DIY 系列丛书。这套丛书包括《数字电子技术实验指导》、《电子线路实验指导》、《硬件描述语言实验指导》、《电路与信号系统实验指导》、《微型计算机原理及应用实验指导》、《电工电子技术基础实验指导》等共六本。丛书将着力考虑符合实验教学大纲的要求，同时对不同专业留有充分的选择余地，可以灵活组合。希望丛书的出版能够改变实验教学不够规范，随意性较大，相同的课程分散在多处做实验，要求不同、过程不同、设备不一的现状。

编委会将在丛书出版之后，广泛听取各方面的反映和意见，不断完善丛书的内容，提高丛书的学术水平和质量，更好地满足高等学校电工电子类实验教学的需求。

电工电子实验教材编委会

2004.6

前　言

微型计算机原理及应用课程是高等院校电子信息、通信工程、自动控制、机电一体化、测控技术与仪器等专业的重要专业基础课。近年来随着计算机技术运用进程的不断加快，微型计算机原理及应用课程更加受到重视。本课程包括两个方面内容，一是微型计算机原理课程，它主要学习以 80X 系列 CPU 组成的微机系统；二是单片微型计算机原理与接口技术，目前国内多数院校采用 MCS-51 系列单片微型计算机作为分析讨论的对象。前者主要侧重于微机系统的结构与组成，以及汇编语言的特点与编程方法等内容；而后者除了介绍单片微型计算机的结构与组成，更侧重于介绍用单片微型计算机组成的一个实际应用系统。两部分内容融会贯通，相互补充。通过这两部分内容的学习使学生较好地掌握了微型计算机原理、微型计算机的硬件结构及汇编程序设计方法。本课程在进行课堂教学的同时必须结合实验教学，这样才能取得较好的教学效果。基于这样的教学思想，我们在总结了多年教学经验，特别是实验教学经验的基础上编写了这本书。

本教材分为软件和硬件两大部分，共八章和五个附录。第一章为 8086/8088 微型计算机汇编语言程序设计，要求学生掌握 8086/8088 系列微型计算机汇编语言源程序的编写和编辑，以及汇编、调试等上机过程，通过实验掌握算术运算、逻辑运算、条件转移、循环操作、子程序调用等编程方法。第二章为 MCS-51 单片微型计算机汇编语言程序设计，要求学生掌握 MCS-51 单片微型计算机汇编语言程序的结构、设计与调试过程，熟悉集成开发环境，并掌握用 MCS-51 单片微型计算机汇编语言程序设计的方法进行程序设计。第三章为 MCS-51 单片微型计算机基本输入/输出接口实验，要求学生掌握包括 LED 发光管的显示接口、开关量的读取方法，用扫描法进行键识别的方法。第四章为 MCS-51 单片微型计算机的显示接口实验，要求学生掌握 LED 数码块显示接口的应用、点阵 LED 显示器的应用、笔段式 LCD 显示接口的应用、点阵字符式 LCD 显示接口的应用、点阵图形式 LCD 显示接口的应用。第五章为 MCS-51 单片微型计算机的驱动接口实验，要求学生掌握混合式步进电机驱动的硬件接口和编程方法、“H”型驱动接口驱动直流电机的硬件接口和编程方法、单片微机打印机的硬件接口和字符式打印的编程方法。第六章为 MCS-51 单片微型计算机的 A/D 和 D/A 接口实验，要求学生掌握逐次逼近型 A/D 转换器 0809 与单片微型计算机的接口技术、双积分型 A/D 转换器 MC14433 与单片微型计算机的硬件接口方式及编程方法、D/A 转换器 0832 与单片微型计算机的硬件接口技术及编程方法。第七章为 MCS-51 单片微型计算机的定时器/计数器应用实验，要求学生掌握单片微型计算机的定时器/计数器工作原理，并运用定时器/计数器设计一个电脑时钟。掌握时钟芯片 DS12C887 的基本原理、与单片微机

的硬件接口以及编程方法。第八章为单片微型计算机的综合实验，共介绍了三个综合性实验，这些实验基本上已构成了一个完整的单片微机应用系统，具有一定的综合性和复杂程度，完成这些实验需要较多的时间，要求学生较为全面地掌握单片微机的知识。这些实验培养了学生综合应用单片微机技术的能力。附录介绍了常用的编程工具及指令表。

本书具有如下特点：

1. 将 80X 系列微机与 MCS-51 系列单片微机的实验合在一起介绍，有利于学生能更好地掌握这两个微机系统的区别与联系。

2. 针对目前微型计算机技术实际使用的状况，即以 80X 系列为代表的微机主要偏重于软件设计应用，而单片微机则侧重于系统的硬件设计，本书的实验也作了相应的安排，80X 系列微机系统全部是软件实验，而 MCS-51 单片微机则主要介绍了硬件实验。

3. 考虑到目前的实验仿真系统种类繁多，特别是 MCS-51 单片微机的实验系统，本书所介绍的实验对实验系统没作要求，因此适用面较宽，这样有利于实验的灵活安排。

4. 为了培养学生分析问题和解决问题的能力，使学生的程序设计能力得到提高，部分实验并没有提供完整的软件，而只是提供了程序流程图和关键的程序段。

5. 本书中的每一个实验都独立成篇，与课堂理论教学采用何种教材无关。

在本书编写期间各兄弟院校的老师提出了不少宝贵意见和建议，在此一并致以衷心的谢意。

由于编者水平有限，时间仓促，错误、遗漏和不妥之处在所难免，敬请各位读者批评指正。

编　者

2006.7

Contents 目录

第一篇　软　　件

第一章　8086/8088 微型计算机汇编语言程序设计

实验 1.1　汇编语言程序的编辑、调试

【基础篇】

一、实验目的

1. 学习并掌握 8086/8088 系列微型计算机汇编语言源程序的编写和编辑过程。
2. 学习并掌握对汇编语言源程序进行汇编和连接，最后产生可执行文件的操作过程。
3. 学习 DEBUG 调试程序的各种命令和使用，掌握对可执行程序进行调试的基本方法。

二、实验原理

1. 汇编语言源程序的编辑和修改。

汇编语言是一种符号语言，它用助记符来表示指令的操作码，用符号或符号地址来表示指令的操作数或操作数地址。汇编语言指令与由二进制码构成的机器语言指令是一一对应的，它们都是面向机器的语言。汇编语言指令比机器语言指令使用起来要方便得多，比较容易记忆和书写；但汇编语言程序必须翻译成机器语言后才能在机器上运行，我们把这种翻译称为“汇编”，使用的汇编工具是汇编程序。

用户根据实际问题用汇编语言编写的程序称为汇编语言源程序，汇编语言除了各种操作指令以外，还有许多伪指令和宏指令。伪指令在汇编时不能被翻译成机器指令，它不像一般指令那样是在程序运行期间由计算机来执行的，而是在汇编程序对源程序汇编期间由汇编程序处理的，它们主要用于数据定义、分配存储区、指示程序的开始与结束等。

先看一个乘法运算的程序：设有两个无符号数双字变量 DATAX 和 DATAY，编写程序计算这两个数相乘后的结果。

解题方案：

这是一个双字相乘的问题，但是 8086/8088 CPU 的字长为 16 位，内部所有寄存器都是 16 位的，因而单条指令能处理的操作数最大长度为 16 位。所以要实现双字相乘，必须分步

进行，也就是根据十进制多位数相乘的原理，用先将低位字和高位字分别相乘，再将分部积相加的方法实现运算过程。

例如，两个2位十进制数A和B相乘，其中$A = A_1A_0$，$B = B_1B_0$，乘积为C，运算过程如下：

$$\begin{array}{rrrrr} & & & A_1 & A_0 \\ & & \times & B_1 & B_0 \\ \hline & & & C_{11} & C_{10} \\ & & C_{21} & C_{20} & \\ & & C_{31} & C_{30} & \\ + & C_{41} & C_{40} & & \\ \hline & C_3 & C_2 & C_1 & C_0 \end{array}$$

其中：

$A_0 \times B_0 = C_{11}C_{10}$

$A_1 \times B_0 = C_{21}C_{20}$

$A_0 \times B_1 = C_{31}C_{30}$

$A_1 \times B_1 = C_{41}C_{40}$

$C_3C_2C_1C_0$为4位数的积。

从上面十进制数相乘结果可知，两个2位数相乘，积可能是4位数，$C = C_3C_2C_1C_0$。同理，在计算机内部，两个双字相乘，积可能是4字长的。

根据这一原理，我们将两个双字按它们存放的地址分别表示为DATAX、DATAX+2和DATAY、DATAY+2，乘积的高位字至低位字依次为S3、S2、S1、S0。编制流程图如图1-1所示。

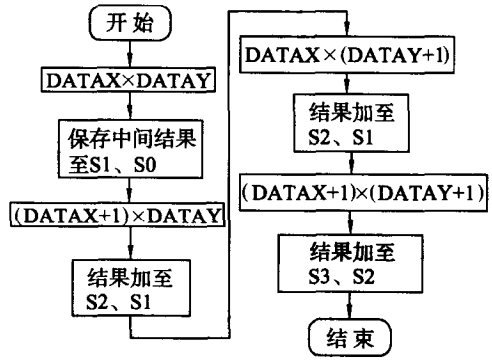


图1-1　两个双字相乘的流程图

根据乘法运算指令的使用规则，字操作的乘法运算时，其中一个操作数必须在AX寄存器中，乘积为双字，且积的高位字和低位字分别在DX和AX寄存器中。在分部积的相加过程中，还必须要考虑低位向高位的进位问题。

根据流程图，编制完整的汇编语言源程序如下：

```
DATA    SEGMENT
        DATAX   DW   0148H
                DW   2316H
        DATAY   DW   0237H
                DW   4052H
        S0      DW   0          ;定义4个用于存放结果的字变量，并且这4个变量的值均为0
        S1      DW   0
        S2      DW   0
        S3      DW   0
DATA    ENDS                    ;数据段结束
CODE    SEGMENT                 ;定义代码段
    ASSUME    CS: CODE,DS: DATA
START: MOV    AX,DATA           ;设定数据段首地址
        MOV     DS,AX
        MOV     AX,DATAX        ;第一个操作数低位字送累加器AX
```

```
        MUL     DATAY           ;(AX)与第二个字的低位字相乘
        MOV     S0,AX           ;积的低位字保存至 S0 单元
        MOV     S1,DX           ;积的高位字保存至 S1 单元
        MOV     AX,DATAX+2      ;第一个操作数高位字送累加器 AX
        MUL     DATAY           ;(AX)与第二个字的低位字相乘
        ADD     S1,AX           ;积的低位字加至 S1 单元
        ADC     S2,DX           ;积的高位字加至 S2 单元(带进位加)
        MOV     AX,DATAX        ;第一个操作数低位字送累加器 AX
        MUL     DATAY+2         ;(AX)与第二个字的高位字相乘
        ADD     S1,AX           ;积的低位字加至 S1 单元
        ADC     S2,DX           ;积的高位字加至 S2 单元(带进位加)
        MOV     AX,DATAX+2      ;第一个操作数高位字送累加器 AX
        MUL     DATAY+2         ;(AX)与第二个字的高位字相乘
        ADD     S2,AX           ;积的低位字加至 S2 单元
        ADC     S3,DX           ;积的高位字加至 S3 单元(带进位加)
        HLT                     ;停机
CODE    ENDS                    ;代码段结束
        END     START
```

汇编语言源程序的建立、编辑和修改：

建立和编辑汇编语言源程序，可以用各种行编辑软件和文本编辑软件，如 Wordstar、Edlin、Editor、记事本、写字板等。无论用哪种编辑软件，所建立的汇编语言源程序文件的扩展名必须是 ASM，大小写不限。下面以 MS-DOS Editor 文本编辑器为例，创建、编辑汇编语言源程序的大致过程如下：

先使系统运行于命令方式，选择好要建立的源程序文件名，在命令提示符下启动 Editor，如

E:\> edit <文件名>.asm ↙

假如本题的源程序文件名为 syt1.asm，则启动命令为

E:\> edit syt1.asm ↙

系统进入 Editor，并处于对 syt1.asm 文件的编辑状态，此时可在 Editor 的编辑窗口输入汇编语言源程序。

输入程序时，每输完一条指令后以回车键换行，程序输入完毕后通过文件菜单命令进行保存，然后退出 Editor 完成编辑任务。

有关 Editor 的详细操作功能，可参阅相关书籍。

2. 对汇编语言源程序的汇编原理。

源文件建立后，要用汇编程序对源文件进行汇编，汇编后产生二进制的目标文件(扩展名为 OBJ)、列表文件(扩展名为 LST)和交叉索引文件(扩展名为 CRF)。其中目标文件是最主要的文件，另两个文件可根据用户需要进行选择。目前使用的汇编工具主要是宏汇编程序 masm.exe。例如以如下格式输入命令：

E:\> masm syt1 ↙

该命令在启动 MASM 汇编时直接输入了需要汇编的源程序名 syt1(不必加扩展名)，回车后屏幕上出现第一条提示信息：

Object filename [SYT1.OBJ]:

提示用户输入目标文件名,因为在启动 MASM 时已输入了源文件名,所以在提示信息后面的方括号中给出了默认的目标文件名,如不想改变此文件名,可直接按回车键;如果想另起文件名或者在启动 MASM 时未输入源文件名,此时应输入目标文件名,但不必输入文件的扩展名,因为系统会自动生成扩展名 OBJ。

回车后,屏幕上出现第二条提示信息:

Source listing [NUL.LST]:

提示用户输入列表文件名。输入的文件名可以和前面的文件一致,如 syt1,但不必输入扩展名(扩展名自动生成为 LST);假如不需要生成列表文件则不要输入文件名,直接回车。

回车后,屏幕上出现第三条提示信息:

Cross reference [NUL.CRF]:

提示用户输入交叉索引文件名。输入的文件名可以和前面的文件一致,如 syt1,但不必输入扩展名(扩展名自动生成为 CRF);假如不需要生成交叉索引文件则不要输入文件名,直接回车。

也可以用下面的形式来启动汇编命令:

```
E:\> masm  syt1;↙      (比前面第一种启动方式多了一个分号)
```

这种方式下屏幕不会出现以上三条提示信息,默认为三次均选择直接回车。

若汇编过程中未发现错误,则出现如下信息:

```
Warning Severe
Errors  Errors
0       0
```

表示汇编结束无错误,此时已产生了相应的 OBJ 文件,如果在前面选择的话,还将产生 LST 文件和 CRF 文件。

如果源程序中有错误,汇编结束后会在屏幕上出现错误提示信息,并且将不会产生以上 3 个文件。错误信息包括出错的地址、指令代码、汇编语言指令、错误的类型编号及错误的简要英文提示。例如,对某一源程序 exsam1.asm 进行汇编:

```
E:\> masm exsam1;↙

0002  B2 35                          mov dl,7f35h
Error  ---        50: Value is out of range
000B  05 0002                        add ax
Error  ---        10: Syntax error

Warning Severe
Errors  Errors
0       2
```

以上信息提示在汇编过程中发现源程序中有两处错误,第一处在“mov dl,7f35h”语句上,错误类型号为 50,后面的英文提示说明数值超出使用范围(目的操作数 DL 是字节,但立即数 7f35h 已超出字节取值范围);第二处错误在“add ax”语句上,错误类型号为 10,后面的英文提示说明是语法错误(指令中少了一个操作数)。此时应回到前一步骤,再用 Editor 打

开源程序文件，进行编辑和修改，修改完毕后保存文件，然后重新汇编。

根据错误的不同类型和情况，汇编程序可有 80 多条相应的错误信息提示，详细的错误信息表见附录二。

假如在汇编程序运行后出现错误提示信息

```
Error  ---          110:
in: EXSAM1.ASM
```

则表示汇编程序找不到要汇编的源程序 exsam1.asm，这种情况可能是程序名输入有误，或者在编辑源程序时建立的源文件名没有取扩展名 ASM，此时只需对原来的源文件名进行重命名，使之带有扩展名 ASM，然后再重新进行汇编。

3. 目标程序模块的连接。

在前一步正常完成对源程序的汇编后，已产生出二进制的目标文件（扩展名为 OBJ），但 OBJ 文件不是可执行的文件，还必须用连接程序 LINK 对目标程序模块进行连接，产生可执行文件（扩展名为 EXE）。如前面用 MASM 汇编后产生的目标程序模块文件名为 syt1.obj，此时可输入命令：

```
E: \> link syt1 ↙
```

对其进行连接。屏幕出现的提示和用户的回答为：

```
Microsoft (R) Personal Computer Linker  Version 2.40
Copyright (C) Microsoft Corp 1983, 1984, 1985.  All rights reserved.

Run File [SYT1.EXE]: ↙
List File [NUL.MAP]: ↙
Libraries [.LIB]: ↙
Warning: no stack segment
```

LINK 程序有两个输入文件，即 OBJ 文件和 LIB 文件。OBJ 文件是需要连接的目标文件；LIB 文件则是程序中需要用到的库文件，如无此特殊需要，则应对“[.LIB]:”提示直接回车。LINK 程序有两个输出文件，一个是 EXE 文件，也就是我们所需要的可执行文件，由于在提示信息中已给出了默认的文件名 syt1.exe，故可对“[SYT1.EXE]:”提示直接回车，这样就在磁盘上建立了名为 syt1.exe 的可执行文件。LINK 的另一个输出文件为 MAP 文件，它是连接程序的列表文件，又称为连接映象（Link Map），它给出每个段在存储器中的分配情况，如无此需要可直接回车。连接命令后面也可加上分号以跳过这三条屏幕提示信息。

最后一行“Warning: no stack segment”是连接程序给出的无堆栈段的警告信息，它并不影响程序的执行。到此为止，连接过程已经结束，我们已得到了可执行文件 syt1.exe，原则上说可以直接运行该程序了。

4. 程序的运行和调试。

可执行文件的运行方法是双击该程序的图标或在命令状态下直接键入文件名（不需要键入扩展名）并回车。但实际上，大部分程序还是必须经过调试阶段才能达到我们预期的目标。调试的工具是 DEBUG 调试程序。

对程序进行调试的基本步骤是：

（1）启动 DEBUG 并把要调试的程序（如 syt1.exe）调入内存。

命令格式为：

```
E: \ > debug syt1. exe ↙
```

系统进入 DEBUG 调试状态后(出现提示符“ - ”),要调试的程序已被调入内存,我们可以用反汇编命令 U 调看反汇编后程序的指令序列,例如:

```
-u ↙
139D: 0000 B8A213        MOV     AX,13A2
139D: 0003 8ED8          MOV     DS,AX
139D: 0005 A10000        MOV     AX,[0000]
139D: 0008 F7260400      MUL     WORD PTR [0004]
139D: 000C A30800        MOV     [0008],AX
139D: 000F 89160A00      MOV     [000A],DX
139D: 0013 A10200        MOV     AX,[0002]
139D: 0016 F7260400      MUL     WORD PTR [0004]
139D: 001A 01060A00      ADD     [000A],AX
139D: 001E 11160C00      ADC     [000C],DX
-u ↙
139D: 0022 A10000        MOV     AX,[0000]
139D: 0025 F7260600      MUL     WORD PTR [0006]
139D: 0029 01060A00      ADD     [000A],AX
139D: 002D 11160C00      ADC     [000C],DX
139D: 0031 A10200        MOV     AX,[0002]
139D: 0034 F7260600      MUL     WORD PTR [0006]
139D: 0038 01060C00      ADD     [000C],AX
139D: 003C 11160E00      ADC     [000E],DX
139D: 0040 F4            HLT
139D: 0041 0000          ADD     [BX + SI],AL
-
```

从上面显示的内容可看出,指令序列中已没有了源程序中所使用的所有伪指令,从而说明源程序中的伪指令在汇编后是不产生机器代码的,所以它们不能被反汇编。伪指令只是告诉汇编程序在对源程序进行汇编时如何定义变量及其单元的地址,如何分配各段以及各段的起始地址,如何计算有关转移的地址等。从上面的指令序列中也可看到,第一条指令是“MOV AX,13A2”,它替代了原来源程序中的“MOV AX,DATA”指令,说明系统分配给数据段的段起始地址为 13A2H;第三条指令是“MOV AX,[0000]”,它替代了源程序中的“MOV AX,DATAX”指令,说明系统分配给变量 DATAX 的起始偏移地址为 0000H;…。

(2) 用 DEBUG 命令对程序进行调试。

通过在 DEBUG 状态下键入各种命令,如 R 命令、D 命令、G 命令、T 命令等分别可执行查看寄存器的内容、查看内存的内容、运行程序、跟踪调试等操作,从而观察程序的运行流程是否正常以及各指令在执行后相关的中间结果是否正确。

例如,上面的 syt1. exe 程序在进入调试状态后,可先用 R 命令看一下各寄存器的内容,然后用 T 命令跟踪运行,观察寄存器内容的变化,如:

```
E: \ > debug syt1. exe ↙
-r ↙
```

```
AX=0000  BX=0000  CX=0060  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=138D  ES=138D  SS=139D  CS=139D  IP=0000  NV UP EI PL NZ NA PO NC
139D:0000 B8A213      MOV     AX,13A2
-t↙
AX=13A2  BX=0000  CX=0060  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=138D  ES=138D  SS=139D  CS=139D  IP=0003  NV UP EI PL NZ NA PO NC
139D:0003 8ED8        MOV     DS,AX
-t↙
AX=13A2  BX=0000  CX=0060  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13A2  ES=138D  SS=139D  CS=139D  IP=0005  NV UP EI PL NZ NA PO NC
139D:0005 A10000      MOV     AX,[0000]                    DS:0000=0148
-t↙
AX=0148  BX=0000  CX=0060  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13A2  ES=138D  SS=139D  CS=139D  IP=0008  NV UP EI PL NZ NA PO NC
139D:0008 F7260400    MUL     WORD PTR [0004]              DS:0004=0237
-
```

上面第一条 R 命令，屏幕显示了程序刚调入内存时各 CPU 的初值和下面将要执行的指令。在连续输入两次 T 命令后，程序运行了“MOV AX,13A2”和“MOV DS,AX”两条指令，从此时寄存器的值可知(AX)=13A2H,(DS)=13A2H，说明数据段起始地址值已赋给了段寄存器 DS。

再输入一次 T 命令，可看到(AX)=0148H，说明程序执行了第三条指令“MOV AX,[0000]”后，变量 DATAX 的第一个字的内容已传入 AX 寄存器中。

以此类推，每输入一次 T 命令，程序会按顺序运行一条指令，并显示该指令运行后各寄存器的值，从而跟踪程序在运行过程中是否符合设计要求。

另外，还可用 D 命令来显示内存的内容，如：

```
-d ds:0↙
13A2:0000 48 01 16 23 37 02 52 40-78  D6 CC 1E AC BD D0 08   H..#7.R@x.......
13A2:0010 3E 43 04 00 75 0D F6 06-21  04 FF 75 06 E8 0B 00   >C..u...!..u....
13A2:0020 E8 59 00 5F 5E 59 5B 58-5A  1F C3 2E 80 3E F1 E3   .Y._^Y[XZ....>..
13A2:0030 00 74 F7 1E 0E 1F BE F1-E3  E8 B9 02 2E A1 4B E8   .t..........K.
13A2:0040 BB 40 00 BA 01 00 33 FF-CD 21  1F 72 0B 8B D8 B0   .@....3..!.r....
13A2:0050 FF 86 47 18 A2 18 00 C3-0E  1F E8 D2 00 3D 41 00   ..G.........=A.
13A2:0060 74 07 0B FF 74 06 BA 54-83  E9 B5 FC 2E C6 06 83   t...t..T........
13A2:0070 E2 01 BA 85 E2 2E A3 85-E2  E9 A5 FC 80 3E E7 04   ............>..
-
```

也可用 G 命令来连续运行程序直至结束或至断点地址。

有关 DEBUG 的详细功能及使用方法见附录一。

5. 程序编辑、汇编、连接、调试的流程。

汇编语言源程序经汇编和连接，如无误才能产生可执行文件，但此时还并不能认为程序设计就没有问题了，因为这两步只能说明程序中没有语法或指令使用方面的错误，而程序设计流程是否合理、寄存器及变量之间数据的存取是否正确，必须经过调试才能知道。一个程

序从源文件的建立一直到调试完毕，需经过各个环节，缺一不可，整个操作流程可用图 1-2 来表示。

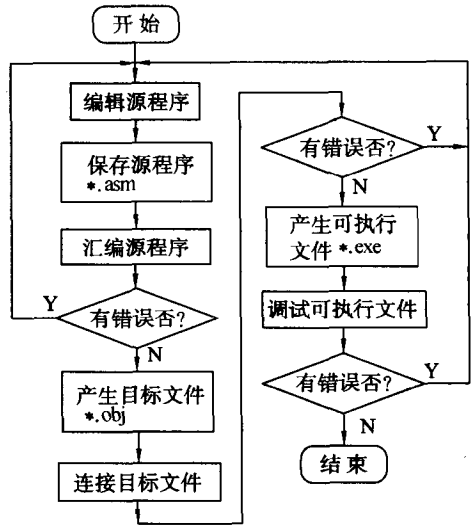


图 1-2　汇编语言程序的编辑、汇编、连接、调试流程

三、实验内容

1. 用 Editor 建立和编辑汇编语言源程序。

先使系统运行于命令状态，在命令提示符下启动 Editor，并设要建立的源程序文件取名为 syt1. asm，程序输入完毕后通过文件菜单命令进行保存，然后退出 Editor 完成编辑任务。

2. 对源程序进行汇编，产生目标程序模块。

启动宏汇编程序 MASM，根据提示信息作相应的回答并输入。如果出现错误信息提示，应记录这些信息，然后重新启动 Editor，对源程序进行修改。修改完毕后保存文件，再进行汇编，直到无错误信息出现。

3. 连接目标程序，形成可执行程序文件。

启动连接程序 LINK，根据提示信息作相应的回答并输入。如果出现错误信息提示，应重新对源程序进行编辑和汇编，然后再进行连接。直到没有错误信息出现，使其产生可执行文件 syt1. exe。

4. 用调试程序 DEBUG 对程序进行调试。

启动 DEBUG 调试程序并装入可执行程序 syt1. exe，即输入如下命令：

E: \ > debug syt1. exe ↙

输入时注意不可忽略“. exe。”进入调试状态后进行以下操作：

(1) 用 R 命令调看各寄存器的内容，记录 AX、BX、CX、DX、IP 寄存器的初始值。

(2) 用 U 命令进行反汇编，观察反汇编显示的指令序列及其每条指令的地址和代码，再和源程序的指令序列进行比较，看看有何异同。

(3) 用 T 命令跟踪各条指令的执行，记录每条指令执行后 AX、BX、CX、DX、IP 寄存器的内容，直到 HLT 指令为止。