汇编语言实验教程

张晋东 秦贵和 刘萍萍 黄永平 赵宏伟

吉林大学

2020 年 5 月

内 容 简 介

本书是《微机系统》课程的配套实验教材。共包括九个实验项目，每个实验项目包含基础性实验和加强性实验两个部分。实验涉及 DEBUG 环境的使用、汇编语言程序结构应用、分支程序设计、循环程序设计、子程序设计、字符串处理程序设计、BIOS 调用、中断处理等。

本书可作为高等院校与职业技术学校计算机、软件工程、工业自动化等相关专业的微型计算机技术与应用课程的实验教材，也适用于工程技术人员作参考。

前　言

汇编语言是面向机器的底层程序设计语言，其指令可以直接控制硬件最底层，能充分发挥机器的硬件性能，具有其他高级语言不可替代的作用。对于很多实时性强、需要直接控制硬件的应用只有汇编语言可以胜任。汇编语言程序设计是高等院校计算机专业学生必修的专业基础课程内容之一。本书配合科学出版社的《微型计算机原理与汇编语言程序设计》和《微型计算机原理与接口技术》教材，同步《微机系统》课程的教学，进行汇编语言程序设计的实践教学规范与指导。

《汇编语言程序设计》实验是必修实验课程，这一实践环节是学生深入理解和掌握理论教学知识的重要途径。通过本实验课程的学习，加深对相关理论知识的理解，熟悉汇编语言指令，熟练掌握汇编语言程序设计的基本方法、编程风格和调试技术，为后续课程和微机应用打下坚实的基础。

本书包括九个实验项目。实验一涉及 DEBUG 环境的使用；实验二涉及汇编语言程序结构的应用；实验三涉及分支程序设计；实验四涉及循环程序设计；实验五和实验六涉及子程序设计；实验七涉及字符串处理程序设计；实验八涉及 BIOS 调用；实验九涉及中断处理。其中每个实验均包含基础性实验和加强性实验两部分。

本书编写过程中，微机系统课程教学梯队的教师们做了很多工作，尤其是智能控制与嵌入式系统研究室的一些研究生同学做了大量的实验验证工作，为本书的顺利出版提供了强有力的支持和帮助，在此，向这些同学表示真心感谢。

本书编写过程中，参考了有关的优秀教材、专著、以及优秀的网络站点，恕不一一列出。能够领略众多优秀的观点和技术，是原创者的无私奉献，是读者的集萃之想。本书编者在此向提供各种观点和技术的各位编者表示最真诚的谢意。

我们对在本书编写、出版过程中给予支持和帮助的所有领导及朋友们，一并表示衷心的感谢。

由于编者水平和经验所限，书中不足之处在所难免，恳请指正。

编者

2020 年 5 月于长春

目　　录

实验要求

1．上机前要作好充分准备，包括程序框图、源程序清单、调试步骤、测试方法、运行结果的分析。

2．上机时要遵守实验室的规章制度，爱护实验设备。要熟悉与实验有关的系统软件（包括编辑程序、汇编程序、连接程序和调试程序）的使用方法。在程序的调试过程中，有意识地学习及掌握 debug 程序的各种操作命令，学习程序的调试方法及技巧。

3．程序调试完后，须由实验辅导教师在机器上检查运行结果，经教师认可后的源程序可在实验报告中写出。

4．每个实验完成后，应写出实验报告。实验报告的要求如下：

（1）设计说明：用来说明程序的功能、结构。它包括程序名、功能、原理及算法说明、程序及数据结构、主要符号名的说明等。

（2）调试及体会：它包括调试情况，如上机时遇到的问题及解决办法，观察到的现象及其分析，对程序设计技巧的总结及分析等，程序的输出结果、对结果的分析和实验的心得体会，以及诸如调试日期等需要记录的信息。

（3）所完成程序使用说明：为用户提供完整的使用说明。包括程序的功能，使用的资源说明，输入信息的类型及格式、输出信息的类型及格式、出错信息的含义及程序的适用范围等。

（4）程序结构框图以及数据结构：包括程序的结构说明，程序流程图，以及使用的数据结构和存储器分配。

（5）自编程序原稿及调试后修正稿，写出原稿错在那里。经辅导教师认可后的程序清单。

5．上交实验报告。

实验一　实验环境及 DEBUG 使用

一、 基础性实验

1. 实验目的

(1) 熟悉 DEBUG 的功能，运行并掌握其常用命令。

(2) 掌握用 DEBUG 调试程序的方法。

2. 实验内容

DEBUG 的加载及其常用命令（D、E、F、R、G、A、T、U、N、L、W、Q）的使用情况。

3. 实验仪器设备

(1) 硬件环境：IBM-PC及其兼容机。

(2) 软件环境：操作系统DOS 3.0以上。

(3) 调试程序：DEBUG.COM。

4. 实验原理

DEBUG 调试程序是为调试目标程序而设计的。它有很强的功能，可以访问系统所配置的内存每一个地址、每一个 I/O 接口、CPU 的每一个寄存器；可以对 EXE 文件进行各种方法的执行和停止；可以随时了解中间结果的情况，以及程序执行流程的情况。

DEBUG 程序以“－”作提示符等待输入命令。DEBUG 的命令都是单字符命令，其后跟着一个或多个参数，命令和参数可用大写、小写或大小写混杂的输入方式；命令与参数可用分界号（空格或逗号）隔开，而两个连续的 16 进制值之间必须用分界号隔开。键入回车键表示输入结束。执行命令时，如果键入格式有错，则显示出错提示∧error。

(1) 启动DEBUG

为了启动调试程序 DEBUG，在 DOS 下应输入命令：

DEBUG[d：][path][filename[．exe]][parm1][parm2]

如果输入文件名 filename，那么 DEBUG 就把所指定的文件装入内存，然后可以输入命令来修改、显示或执行指定文件的内容。如果不输入文件名 filename，那么或使用当前存储器的内容进行工作，或使用命名命令 Name 和装入命令 Load 把需要的文件装入存储器中，然后可以输入命令来修改、显示或执行存储器的内容。

可选择的参数 parm1 和 parm2 表示对命名的文件说明 filespec 来说，是按需要而选择使用的参数。

例如：C>DEBUG　DISKCOMP.COM　A：　B：↓

在本命令中，A：和 B：是 DEBUG 为 DISKCOMP 程序准备的参数。

当 DEBUG 启动时，把寄存器和标志位置成适合于程序调试而用的下列值：

① 段寄存器(CS，DS，ES 和 SS)置成空闲存储器底部，即 DEBUG 程序末尾之后的第一个段。

② 指令指针 IP 置成 0100H。

③ 堆栈指针 SP 设置在该分段的末尾。

④ 其余的寄存器(AX，BX，CX，DX，BP，SI 和 D1)置成零。但是，如果使用文件说明启动 DEBUG，则 CX 寄存器包含有该文件的长度(以字节计)。如果文件大于 64K，则其长度包含在 BX 和 CX 中(高部分在 BX 中)。

⑤ 标志位置成它们的清除值。

(2) DEBUG的命令及其参数

DEBUG 的命令是具有规定格式的特殊字符串。命令是单个字母，通常后面还有 1 个或 1 个以上的参数。可以用大写或小写字母，或二者组合形式输入命令和参数。命令和参数用分隔符（空格或逗号）把它们分开，然而只有在两个相邻的 16 进制值之间才必须要求分隔符。因此下列这些命令是等价的：

dcs：100　110

d　cs：100　110

d，cs：100，110

只有在按 ENTER 键之后，输入的命令才有效。如果遇到语法出错，那么就显示“error”字样，如：

d　cs：100　cs：110

∧

error

错误原因是，命令 D 要求第二个地址中只包含 16 进制的偏移值。按 ctrl—Break 键可以终止命令的执行。DEBUG 程序的提示符是连字符“－”，DEBUG 的命令就是在该提示符下输入进行操作的。DEBUG 下的数都是十六进制数，并且不带后缀“H”。

(3) DEBUG命令的使用

DEBUG 的命令共有 18 条，现在具体介绍它们的使用。为便于查阅，按命令字头英文字母顺序逐条介绍。

① 汇编命令(Assemble)

用途：键入汇编指令，并把它们汇编成机器代码，相继存放在从指定地址开始的存储区中。

格式：A[address]

说明：输入给本命令的所有数字都是十六进制的。将输入的指令从指定的地址 address 开始装入内存的连续单元。如果不指定地址，则把语句装入 CS：0100 确定的区域中；如果已用过 A 命令，那么就在前一个 A 命令所装入的最后一条指令的后继单元中，接着输入指令。当所有要求的指令已输入完成，在提示继续输入下一条指令时，直接按 ENTER 就退回 DEBUG 提示符下。

DEBUG 对不可用语句的响应是显示信息：

∧

error

并显示当前的装入地址。

例如：

C>debug

a200

08B4：0200 xor ax，ax

08B4：0202 mov [bx]，ax

08B4：0204 ret

08B4：0205

② 比较命令(Compare)

用途：比较存储器中两个数据块的内容。

格式：C range address

说明：比较存储器两个数据块的内容，range 给出第一个数据块的首址及长度，address 给出第二个数据块的首址。如果找到不相等的字节，则用下列形式显示地址和内容：

addr1 byte1 byte2 addr2

此处，前一半(即 addr1 byte1)说明 range 内不相等单元的位置和内容，后面的一半指的是在 address 中找到的不相等的内容和位置。

如果只输入地址的偏移值，则 C 命令认为是 DS 寄存器中的段。例如：

C 100L20 200

把 DS：100 开始的存储器的 32 个字节同从 DS：200 开始的 32 个字节进行比较。

③ 显示内存命令(Dump)

用途：显示存储器的内容。

格式：D[address]

或者 D[range]

说明：用两个部分来显示内存内容：

- 十六进制部分。用十六进制显示每个字节。

- ASCII 部分。把字节显示成 ASCII 字符。句点“.”表示非显示打印的字符。

按照 40 列系统显示格式，每行在 8 字节边界上开始，并显示出 8 个字节。按照 80 列系统显示格式，每行在 16 字节边界上开始，并显示出 16 个字节。在第 8 至第 9 字节之间有一个连字符“—”。

显示内存命令有两种格式可供选用。

形式 1——用于显示 40H 个字节（40 列方式）或 80H 个字节（80 列方式）的内容。

例如：

D address

或者

D

显示的内容从指定的地址或当前的地址开始。

形式 2——用于显示指定地址范围内的内容。

这时应输入命令为：

D range

例如

D CS：100 10C

显示地址从 CS：100 到 10C 范围的内存单元的内容。

④ 修改内存命令（Enter）

用途：该命令有两种工作方式：用清单中包含的值替换从指定地址开始的一个或一个以上的字节内容（参看形式 1）；显示和允许顺序方式修改字节（参看形式 2）。

格式：E address （list）

说明：如果只输入偏移值作为地址，那么 E 命令认为是包含在 DS 寄存器内的那个段。

修改内存命令有两种格式供选用：

形式 1——用于把清单的内容存放在指定地址开始的内存中。

命令为

E address list

例如：

E ds：100 F3‘XYZ’8D

用清单中指定的 5 个字节填满 ds:100 到 ds:104 存储单元。

形式 2——用于显示地址和单元的内容，然后系统等待输入。

例如：

E address

现在可以开始下列操作：

- 输入一个或两个十六进制值的字符去替换字节的内容，然后采取下面操作之一。
- 按空格步进到下个地址，并显示其内容。如果希望改变其内容就采取上述第一条操作。
- 输入连字符“—”退回到前一个地址，并显示其内容。如果希望改变其内容则采取上述第一条操作。
- 为了退回一个以上的字节且不改变当前字节，应输入另一个连字符。
- 为了结束 E 命令，按 ENTER 键。

例如：

E cs：100

可以得到如下显示内容：

04BA：0100 EB．_

为了把 04BA：0100 内容由 EBH 改变成 41H，则输入 41：

04BA：0100 EB．41

为了查看下三个单元的内容，按三次空格，屏幕显示出：

04BA：0100 EB．41 10．00．BC．_

为了把当前单元(04BA：0103)由 BCH 改成 42H，应输入 42：

04BA：0100 EB．41 10．00．BC．42

如果要退回并把 10H 改成 6FH，则按两次连字符和输入置换的字节之后，屏幕上显示：

04BA：0100 EB．41 10．00．BC．42

04BA：0102 00．_

04BA：0101 10．6F

按 ENTER 键结束输入命令，将会看到提示符为连字符“—”。

⑤ 填写命令（Fill）

用途：用清单中的值填写 range 范围内的存储单元。

格式：F range list

说明：如果清单中包含的字节数小于地址范围，则重复地使用该清单，直到把所指定范围内的存储器单元填满为止。如果清单中包含的字节数大于地址范围，就忽略不计超过的部分。

例如：

F4BA：100 L 5 F3‘XYZ’8D

用指定的 5 个字节填写到内存的 04BA：100 到 04BA：104 单元中。注意，存储清单字符是 ASCII 字符码，因此 100—104 单元中为 F3 58 59 5A 8D。

⑥ 执行命令（GO）

用途：执行正在调试的程序。当达到所指定的断点地址处时，就停止执行，并显示寄存器、标志位以及下一条要执行的指令。

格式：G[=address][address[address…]]

说明：如果不采用=address 参数（必须输入=号），就从 CS 和 IP 寄存器内容决定的地址开始执行程序。如果指定=address，则程序就从 CS：address 处开始执行。

执行命令（GO）有两种形式供选用：

形式 1——在无断点时，利用此形式执行正在调试的程序。

例如：

G[=address]

如果不采用=address 时，在发出 G 命令之前，务必要把 CS：IP 值设置正确。

形式 2——完成与形式 1 相同的功能，此外还允许在指定的地址上设置断点。

例如：

G[=address]address[address…]

此形式使执行停在指定的单元上，因此可以检查系统 / 程序环境。可以按任一种顺序指定十个断点。DEBUG 程序在断点地址上用一个中断码 CCH 置换该指令码。在执行时，无论到达哪一个断点都停止执行，显示寄存器和标志位，并把所有断点地址的内容都恢复成它们原来的指令码。

例如：

G 102 1EF 208

从当前的指令开始执行，当前指令的地址是 CS:IP 的当前值。没有使用=address 参数。指定了三个断点，假设到达第二个断点，指令执行到 CS:1EF 单元之前停止执行，恢复原来的指令码，取消所有三个断点，产生显示并结束 GO 命令。

⑦ 十六进制算术运算命令（Hexarithmetic）

用途：先把两个十六进制的值相加，然后第一个值减去第二个值。在一行上显示和与差。

格式：H value value

例如：

H 0F 8

17 07

000F 和 0008 的 16 进制和是 0017，而其差是 0007。

⑧ 输入命令（Input）

用途：从指定的端口输入并显示（用十六进制）一个字节。

格式：I portaddress

例如：

I 2F8

6B

显示由 02F8 端口读入的一个十六进制字节数(6BH)。

⑨ 装入命令（Load）

用途：把文件或绝对盘扇区装入到内存中。

格式：L[address[drive sector sector]]

说明：用单个 Load 命令可装入的最大扇区数是 80H。如果出现读盘错，则 DEBUG 显示出错信息。

装入命令有两种形式：

形式 1——用于从指定的 drive 驱动器中的盘上装入数据，并把数据存放在存储器中指定的 address 开始的单元中。例如：

L address drive sector sector

从指定的起始相对扇区（第一个 sector）读入数据，并连续地读，直到读到所指定的扇区数（第二个 sector)为止。

例如：

L 4BA：100 1 OF 6D

从驱动器 B 的盘上装入数据，并把数据存放在以 4BA：100 开始的内存中。从相对扇区 0FH（15），即该软盘的第 16 扇区开始，传送 6DH（109）个连续扇区的数据。

形式 2——用于在没有参数或只用一个地址参数时，装入文件。

例如：

L

或者

L address

当启动 DEBUG 程序时，由指定的 filespec 或使用 N 命令满足此要求。把文件装入在以 CS:100 单元或由 address 指定的单元开始的内存中。该文件按文件说明 filespec 读入。BX 和 CX 寄存器置成读入的字节数目。

例如：

C>DEBUG

—N myprog

—L

—

从默认驱动器的盘上，装入名叫 myprog 的文件，并把它存放在 CS：0100 单元开始的存储器中。

⑩ 传送命令（Move）

用途：把 range 指定的内存单元内容，传送到以 address 指定开始的一些单元中。

格式：M range address

说明：源区域的数据仍保持不变，除非由传送命令重写过。

例如：

M CS：100 110 500

把 CS:100 到 CS:110 的 17 个字节的数据，传送到以 DS:500 开始的内存区中。

⑪ 命名命令（Name）

用途：为读写磁盘文件定义文件名。

格式：N filespec[filespec. .]

说明：如果在无文件说明时启动 DEBUG，在用 L 命令装入文件之前，必须使用 Name 命令。

例如：

C>DEBUG

—N myprog

—L

—

⑫ 输出命令（Output）

用途：把字节发送到指定的端口。

格式：O Portaddress byte

例如：为了把字节值 4F 发送到输出端口 2F8，输入：

O 2F8 4F

⑬ 退出命令（Quit）

用途：结束 DEBUG 程序。

格式：Q

说明：Q 命令不保存内存中正工作的文件，需要时可用 Write 命令保存文件。

例如：

一 Q

A>

⑭ 寄存器命令（Register）

用途：寄存器命令有三种功能：

- 显示单个寄存器的十六进制内容，并带有修改这些内容的选择。
- 显示全部寄存器、标志位、以及将被执行的下条指令。
- 显示 8 个标志位状态，并带有修改它们之中任一个或全部的选择。

格式：R[registename]

使用说明：

其一：显示单个寄存器

有效寄存器是：AX、BX、CX、DX、SP、BP、SI、DI、DS、ES、SS、CS 和 IP。

例如，为了显示 AX 寄存器的内容，可输入：

R　AX

系统显示如下：

AX　　F1E4

：_

现在可以采用下列两个操作中的某一个：

- 按 ENTER 键保留未修改的内容。
- 输入 1—4 字符的十六进制值来修改 AX 寄存器的内容，例如 FFFH。

AX　　F1F4

：FFF_

现在按ENTER键把AX寄存器内容改变成0FFFH。

其二：显示所有的寄存器和标志位

为了显示所有寄存器的内容和标志位（还有下条要执行的指令），输入：

R

则系统可能显示如下：

AX=0E00　BX=00FF　CX=0007　DX=01FF

SP=039D　BP=0000　SI=005C　DI=0000

DS=04BA　ES=04BA　SS=04BA　CS=04BA

IP=011A NV UP DI NG NZ AC PE NC

04BA：011A CD21 1NT 21

头4行显示寄存器十六进制内容和8个标志位状态。最后一行指出下一条要执行的指令地址和它的16进制机器码以及反汇编形式，这是当前CS：IP指向的指令。

③显示标志位

有8个标志位，每位用2个字母表示是置“1”状态还是清除状态，详细说明见下表：

表1-1 标志位的符号表示

标志位　名　称	Set(置位)	clear(清除)
溢出(是／否)	OV	NV
方向(减／增)	DN	UP
中断(允许／禁止)	EI	DI
符号(负／正)	NG	PL
零　(是／否)	ZR	NZ
辅助进位(是／否)	AC	NA
奇偶(偶／奇)	PE	PO
进位(是／否)	CY	NC

为了显示所有的标志位，输入命令：

R　F

如果所有标志位处于置1状态，就显示：

OV DN EI NGZR AC PE CY—

⑮ 检索命令（Search）

用途：按照 list 清单中的字符检索 range 范围。

格式：S range list

说明：以显示被查找到的匹配字符所在的地址来指明全部的匹配。显示没有地址的提示符“—”，表示未找到任何匹配字符。

例如：要想从 CS:100 到 CS:110 地址范围内检索 41H，那么输入：

S CS：100 110 41

如果找到两个匹配，则显示也许是：

04BA:0104

04BA:010D

⑯ 跟踪命令(Trace)

用途：从 CS:IP 或者=address（如果指定的话）单元中的指令开始单步执行一条或多条指令。这里的=号必须输入。可以用 value 指定跟踪多条指令。每条指令执行后，显示所有寄存器的内容、标志位的状态以及下一条要执行的指令。

格式：T[=address)[value]

例如，输入命令：

T

如果 IP 寄存器内容为 011A，而该地址指向的指令是 MOV AH，0EH，这可能显示：

AX=0E00　BX=00FF　CX=0007　DX=01FF

SP=039D　BP=0000　SI=005C　D1=0000

DS=04BA　ES=04BA　SS=04BA　CS=04BA

IP=011C　NV　UP　DI　NG　NZ　NC　PE　NC

04BA:011C　　CD21　INT　21

这是执行 011A 中指令之后显示的结果，并且指出下条要执行的指令是 04BA:011C 单元中的 INT 21。

⑰ 反汇编命令（Unassemble）

用途：反汇编命令可把内存中的机器码转换成汇编语句，并显示指令的地址、机器码以及汇编语句。例如：

04BA:0100　206472　　AND　[SI+72],AH

04BA:0103　　FC　　CLD

04BA:0104　　7665　　JBE　　016B

格式：U　[address]

或者　U　[Range]

反汇编命令有两种选择形式：

形式 1——用于反汇编不带指定地址的指令，或反汇编从指定地址开始的指令

例如：

U

或者　U address

用 40 列显示时，一次把 16 个字节反汇编；用 80 列显示时，一次把 32 个字节反汇编。若给出地址 address，则从指定的地址开始把指令反汇编。如果不指定地址，则 U 命令认为起始地址是原先 U 命令反汇编过的最后指令的下一个单元。因此，连续地输入无参数的 U 命令能够对连续的内存单元进行反汇编，并产生连续的反汇编的显示。如果原先没有输入 U 命令，则起始单元是由 DEBUG 初始化在代码中的段内偏移值 0100H。

形式 2——用于把指定地址范围内的指令进行反汇编。

例如：

U range

不管系统显示格式如何，把指定地址范围内的所有指令都进行反汇编。

例如，输入命令

U 04ba：0100 108

显示可能是：　　•

04BA:0100　　206472　AND[SI+72],AH

04BA:0103　　FC　　CLD

04BA:0104　　7665　　JBE 016B

04BA:0106　　207370　AND[BP+DI+70],DH

⑱ 写命令（Write）

用途：把调试过的信息写到磁盘上。

格式：W[address[drive sector sector]]

说明：用单条写命令可以写的最大扇区数是 80H。如果出现写盘错，DEBUG 就显示出错信息。

写命令有两种选择形式：

形式 1——用于从指定地址开始把信息写到磁盘上。

例如：

W address drive sector sector

从指定的开始相对扇区（第一个 sector）开始写数据，并且连续写，直到写完所要求的扇区数（第二个 sector）为止。

例如，命令：

W 1FD 1 100 A

把从 CS：01FD 单元开始的数据写到驱动器 B 的软盘中，由相对扇区 100H（256）开始连续写 0AH（10）个扇区。

形式 2——允许在没有指定参数或只指定地址参数的情况下使用写命令。

例如：

W

或者 W address

此时，写命令把文件写到磁盘上，把从 CS:100 或由参数 address 指定的单元开始的文件按文件说明 filespec 写到磁盘上。此文件的文件说明被格式化在 CS:5C 文件控制块中。当启动 DEBUG 时指定文件说明，或者使用 N 命令，采用这两种方法都可满足这个条件。

另外，必须把寄存器 BX 和 CX 置成被写的字节数目，它们可能已由 DEBUG 或 Load 命令设置过，也可能被修改过。必须确保 BX 和 CX 寄存器包含正确的值。

(4) 注意事项。

DEBUG支持基本的8086／8088汇编语言语法，DEBUG下的汇编语言有下面一些特点和规则：

① 数值都是十六进制数，并且不加尾缀“H”。

② 不能使用符号常量和符号地址。

③ 不能使用绝大部分伪指令，但两个最常用的伪指令 DB 和 DW 能被使用，用于直接把字节和字数据置入相应存储单元。如：

DB 1，2，3，4，“ABCD”

DW 1234，5678

④ 可以使用属性操作符“PTR”对 DEBUG 不能明确类型的操作数进行说明。如：INC BYTE

PTR [BX]。

⑤ DEBUG 的 A 命令汇编程序能根据转移目标地址的距离自动地汇编出短、近或远的转移或调用指令。当然，这也能由“SHORT”、“NEAR PTR”或"FAR PTR”对转移目标地址进行说明来实现。

⑥ 远返回指令的助记符在 DEBUG 中为 RETF。

⑦ 指令前缀助记符必须在相关的指令之前输入，也可以分别放在不同的行。

⑧ 串操作指令只能用其字节型或字型的助记符形式，如：MOVSB、CMPSW 等。

⑨ 可以使用段超越前缀助记符 CS：、DS：、ES：和 SS：。如：

DS：

MOV AL，[BP]

5. 实验步骤

(1) 在DOS或Windows的命令行窗口执行命令Debug.exe，进入Debug程序环境。

例如：

开机进入c:\

进入MASM子目录c:\cd MASM

显示：c:\MASM>DEBUG↙

显示 －

输入程序命令—A↙

18E4:0100 mov AX, 1234↙

18E4:0104 mov BX, 1111↙

18E4:0106 ADD AX, BX↙

18E4:0108 ↙

(2) 运行程序命令

- 单步执行 显示—T↙

查看并记录相关寄存器的内容,及有关条件码的状态。例 AX BX CX,CS IP DS 等

－t↙

同上步

－t↙

同上步

－t=100↙

同上步

－t=100,n↙ （实际－T=100,3）

同上步

- 第二种执行命令 G

－G=100, 108↙

记录各寄存器及各条件码的状态

- 存盘操作

－N　C：CC

－R　BX↙

显示：　BX　0034

：0000↙

－R　CX↙

CX　0000↙

：0008↙

－W　[指定存贮区]

注：如不指定，则数据从 CS:0100 开始写入（BX，CX）个字节数到指定的文件中。

记录屏幕显示的内容

－Q↙退出到 DOS 状态

C：\MASM>dir c:cc↙

查看是否存盘，文件字节数。

(3) 装入文件命令　>DEBUG↙

－N　c:cc↙

－L(指定装入的存贮区)如未指定，则装入 cs:100 开始的存贮区

(4) 反汇编　－U（指定存贮区）

要同 L 中的存贮区相同，屏幕显示的内容是否是刚才装入的文件内容？

(5) 显示存贮单元命令－D（指定存贮区）

例－D 100 132↙

观察结果。

(6) 其它命令：

- 修改存贮单元的命令 E
- 用给定的内容来替代指定范围的存贮单元的内容

－D　DS：100↙

－E　DS：100　F3　‘X　Y　Z’8D↙

－D　DS：100↙

查看是否修改

- 用逐个单元相继修改

－E　DS：100↙

可能显示 18E4：0100　89　78

再按“空格”键可接着显示下一个单元的内容，可继续修改直到用↙结束

- 填写命令 F

－F　4234：0100　L　5　F3　‘X　Y　Z’8 D

记录显示内容

6. 实验提示

DEBUG 支持基本的 8086 / 8088 汇编语言语法，DEBUG 下的汇编语言有下面一些特点和规则：

(1) 在 DEBUG 中数据一律默认为 16 进制数，当 A～F 为数据最高位时，它们前面也不用加数字 0，数后面也不用跟字母 H。

(2) 在 DEBUG 下，源程序中不能使用“标号”、“变量”和一些伪指令。

(3) 在 DEBUG 下，大小写不敏感。

7. 思考题

(1) DEBUG 中 T=100 与单独的 T 指令有何不同？100 的含义是什么？T=100,3 指令中的 3 有何含义？应如何应用？

(2) DEBUG 中 G=100，108 命令的 100，108 的含义是什么？说明 T 与 G 命令的异同。

(3) 说明 D 与 U 命令的区别。

二、 加强性实验

1. 实验目的

(1) 利用 DEBUG 掌握有关的命令。

(2) 利用 DEBUG 运行简单的程序段。

2. 实验内容

用命令 A 汇编下列程序段，并将程序写到硬盘上，然后调试程序段，观察程序段能否完成指定功能，若不能，说明原因，并修改程序。

(1) 内存操作数及各种寻址方式使用

```
MOV     AX，1000
```

```
        MOV     [1200]，AX
        MOV     BX，1202
        MOV     BYTE PTR[BX]，20
        MOV     DL，39
        INC     BX
        MOV     [BX]，DL
        DEC     DL
        MOV     SI，3
        MOV     [BX+SI]，DL
        MOV     [BX+SI+1]，DL
        MOV     WORD PTR[BX+SI+2]，1234
```

(2) 多字节加法程序

```
        MOV     DI，1234
        MOV     CX，8
        MOV     SI，2234
        CLC
Again：  MOV     AL，[SI]
        ADC     [DI]，AL
        INC     SI
        INC     DI
        LOOP    Again
```

(3) 移位操作

```
        MOV     AL,06
        SHL     AL,1
        MOV     BL,AL
        MOV     CL,2
        SHL     AL,CL
        ADD     AL,BL
```

(4) 数据串移动

用A命令DB伪指令在1000H键入下列字符串： ‘This’s a string.'。

用A命令键入下列程序：

```
        MOV     SI，1000
        MOV     DI，2000
        MOV     CX，0FH
LOP：    MOV     AL，[SI]
        MOV     [DI]，AL
        INC     SI
        INC     DI
        LOOP    LOP
        INT     20
```

3. 实验仪器设备

(1) 硬件环境：IBM-PC 及其兼容机。

(2) 软件环境：操作系统 DOS3.0 以上。

(3) 调试程序：DEBUG.COM。

4. 实验步骤

(1) 在 DOS 提示符下进入 DEBUG。

(2) 分别调试上面程序段。

(3) 详细记录每一步所使用的命令，以及查看结果的方法和具体位置。

(4) 记录每一段程序查看结果的方法和具体结果。

(5) 分析上面程序段实现的功能。

5. 实验提示

(1) 汇编语言的指令 7 种寻址方式的具体实现。

(2) 算术运算指令执行数据的加减乘除运算。进行一次数据运算除需将运算结果保存为目的操作数外，通常还会涉及或影响到状态标志。

(3) 程序段是多条指令的集合，指令按顺序执行，所以前面指令的执行结果会影响到后面指令的执行结果。

6. 思考题

(1) 假设某内存单元的内容为 CCH，试问该内容如看作一条指令、看作无符号数、看作带符号数，那将分别表示什么？不查看教材中的指令系统代码，如何在 DEBUG 下知道 CCH 表示一条什么指令？

(2) 在指令 MOV　[BX]，AX 中，操作数[BX]的寻址方式是什么？在 DEBUG 下执行完该指令后，如何查看执行结果？

(3) 可否随意在 DEBUG 提示符“－”后不带参数发出命令 G？什么情况下使用命令 G 时，可不用“=”给出执行的首地址？

实验二　汇编语言程序结构

一、基础性实验

1. 实验目的

(1) 掌握汇编语言程序上机过程。

(2) 掌握汇编语言结构。

(3) 学习汇编语言的伪操作。

(4) 了解汇编程序、连接程序、DOS系统装入和返回功能。

(5) 掌握用DEBUG调试汇编语言程序的方法。

2. 实验内容

编写完整的汇编语言程序：从变量 DD1 开始存放 3 个无符号字节数据，试编制一个找出其中最大者存入 DD2 单元的源程序。

3. 实验仪器设备

(1) PC微机。

(2) DOS操作系统或Windows操作系统。

(3) MASM.EXE，LINK.EXE，DEBUG.COM或宏汇编集成环境。

4. 实验原理

(1) 汇编语言源程序的上机环境

硬件环境：目前 8086 汇编语言程序一般多在 IBMPC/XT 及其兼容机上运行，因此要求机器具有一些基本配置就可以，汇编语言对机器硬件环境没有特殊要求。

软件环境：软件环境是指支持汇编语言程序运行和帮助建立汇编语言源程序的一些软件，主要包括以下几个方面：

DOS 操作系统：汇编语言程序的建立和运行都是在 DOS 操作系统的支持下进行的。

目前 IBMPC/XT 上流行的是 MS-DOS，因此，要首先进入 MS-DOS 状态，然后开始汇编语言的操作。

编辑程序：编辑程序是用来输入和建立汇编语言源程序的一种通用的系统软件，通常源程序的修改也是在编辑状态进行的。

常用的编辑程序有：

- 行编辑程序：EDLIN.COM。
- 全屏幕编辑程序：EDIT.COM、WORDSTAR、NE.COM、TC.COM 等。

汇编程序：8086 的汇编程序有基本汇编 ASM.EXE 和宏汇编 MASM.EXE 两种。基本汇编不支持宏操作，因此，一般选用宏汇编 MASM.EXE。

连接程序：8086 汇编语言使用的连接程序是 LINK.EXE。

调试程序：这类程序作为一种辅助工具，帮助编程者进行程序的调试，通常用动态调试程序 DEBUG.COM。

(2) 运行汇编语言程序的步骤

一般情况下，在计算机上运行汇编语言程序的步骤如下：

① 用编辑程序(例如 EDIT.COM)建立扩展名为.ASM 的汇编语言源程序文件。

② 用汇编程序(例如 MASM.EXE)将汇编语言源程序文件汇编成用机器码表示的目标程序文件，其扩展名为．OBJ。

③ 如果在汇编过程中出现语法错误，根据错误的信息提示(如错误位置、错误类型、错误说明)，用编辑软件重新调入源程序进行修改。没有错误时采用连接程序(例如 LINK.EXE)

④ 把目标文件转化成可执行文件，其扩展名为.EXE。

⑤ 生成可执行文件后，在 DOS 命令状态下直接键入文件名就可执行该文件。

(3) 汇编语言源程序的建立

当启动系统后，进入 DOS 状态，发出下列命令，就可以进入 EDIT 屏幕编辑软件，然后输入汇编语言源程序。

C：＼>EDIT

当不指定具体文件名称时，进入 EDIT 状态，用<ALT>键激活命令选项，选择 NEW 命令建立一个新文件，进入编辑状态，开始输入源程序并编辑。程序输入完毕后一定要执行存盘，将源程序文件存入盘中，以便进行汇编及连接，也可以再次调出源程序进行修改。

(4) 将源程序文件汇编成目标程序文件

一般情况下，MASM 汇编程序的主要功能有以下 3 点：

① 检查源程序中存在的语法错误，并给出错误信息。

② 源程序经汇编后没有错误，则产生目标程序文件，扩展名为．OBJ。

③ 若程序中使用了宏指令，则汇编程序将展开宏指令。

源程序建立以后，在 DOS 状态下，采用宏汇编程序 MASM 对源程序文件进行汇编，其操作过程为键入命令：

C：＼>MASM　ABC．ASM↓

汇编程序调入后，首先显示软件版本号，然后出现三个提示行：

第 1 个提示行是询问目标程序文件名，方括号内为机器规定的默认文件名，通常直接键入回车，表示采用默认的文件名，也可以键入指定文件名。

第 2 个提示行是询问是否建立列表文件，若不建立，可直接键入回车；若要建立，则输入文件名再键入回车。列表文件中同时列出源程序和机器语言程序清单，并给出符号表，有利于程序的调试。

第 3 个提示行是询问是否要建立交叉索引文件，若不要建立，直接键入回车；如果要建立，则输入文件名，就建立了扩展名为.CRF 的文件。为了建立交叉索引文件，必须调用 CREF．EXE 程序。

调入汇编程序以后，当逐条回答了上述各提示行的询问之后，汇编程序就对源程序进行汇编。如果汇编过程中发现源程序有语法错误，则列出有错误的语句和错误代码。

汇编过程的错误分警告错误(Warning Errors)和严重错误（Severe Errors）两种。其中警告错误是指汇编程序认为的一般性错误；严重错误是指汇编程序认为无法进行正确汇编的错误，并给出错误的个数、错误的性质。这时，就要对错误进行分析，找出原因和问题，然后再调用屏幕编辑程序加以修改，修改以后再重新汇编，一直到汇编无错误为止。

(5) 用连接程序生成可执行程序文件

经汇编以后产生的目标程序文件（.OBJ 文件）并不是可执行程序文件，必须经过连接以后，才能成为可执行文件（即扩展名为.EXE）。连接过程为键入命令：

C：＼>LINK　ABC．OBJ↓

在连接程序调入后，首先显示版本号，然后出现三个提示行。

第 1 个提示行是询问要产生的可执行文件的文件名，一般直接键入回车，采用方括号内规定的隐含文件名就可以了。

第 2 个提示行是询问是否要建立连接映像文件。若不建立，则直接回车；如果要建立，则键入文件名再回车。

第 3 个提示行是询问是否用到库文件，若无特殊需要，则直接键入回车就可以了。

上述提示行回答以后，连接程序开始连接，如果连接过程中出现错误，则显示出错误信息，根据提示的错误原因，要重新调入编辑程序加以修改，然后重新汇编，再经过连接，直到没有错误为止。连接以后，便可以产生可执行程序文件（.EXE 文件）。

通常情况下，汇编程序连接以后，可以产生以下三个文件：

① EXE 文件：这是可以直接在 DOS 操作系统下运行的文件。

② MAP 文件：这是连接程序的列表文件，又称为连接映像文件。它给出每个段在存储器中的分配情况。

③ LIB 文件：这是指明程序在运行时所需要的库文件。

(6) 程序的执行

当我们建立了正确的可执行文件以后，就可以直接在 DOS 状态下执行该程序。键入命令：

C：＼>ABC↓

我们可以采用调试程序 DEBUG 来进行检查。

5. 实验步骤

(1) 建立汇编语言程序源文件。可使用任何一个文本编辑器（如EDIT.com）编辑源文件，源文件的扩展名通常取为.ASM。

(2) 汇编文件，形成目标模块。

(3) 若文件有语法错误，根据错误提示，返回（1），修改源文件。若无错，则形成.OBJ目标文件和.LST列表文件。

(4) 连接目标模块，形成可执行文件。若有错，根据错误提示返回（1），修改错误。若无错，则形成.EXE可执行文件和.MAP映像文件。

(5) DOS下直接键入可执行文件名，即可执行该程序。

(6) 运行结果不对，需调试程序，查找原因。

(7) 返回步骤（1）查看源文件，查找错误，修改文件。

(8) 若无法通过查看源文件找出错误，可在DEBUG下调试.EXE文件，找出错误之处，再返回步骤(1)。

6. 实验要求

(1) 了解在DEBUG下装入.exe文件的方法，利用在实验一中掌握的调试方法对程序进行调试。

(2) 在汇编过程中，除形成.OBJ目标文件外，还要形成源文件的.LST列表文件，通过查看列表文件，掌握汇编程序的基本功能。

(3) 在连接过程中，除形成.EXE执行文件外，还要形成其.MAP映像文件，通过查看映像文件，掌握连接程序的功能。

(4) 由于程序中无结果的显示，故需在DEBUG下运行并查看结果。

(5) 在DEBUG下查看程序，对照源程序文件．ASM,掌握汇编程序伪操作的功能。

(6) 利用DEBUG，学习伪操作的方法。

7. 实验提示

在 DEBUG 下查看源程序时，发现.ASM 源文件指令中用符号表示的变量、标号、过程名等符号地址以及一些由汇编提供的操作符（如 SEG，OFFSET 等）均被一些确定的数值代替，从而说明，这些用符号表示的地址，经过汇编、连接及装入内存后，都有了具体的选择物理地址与之对应，其它的一些伪操作也都在程序执行前已经完成，通过对在 DEBUG 下查看源程序，对比．AS 源文件，可以掌握伪操作的功能。

8. 流程图

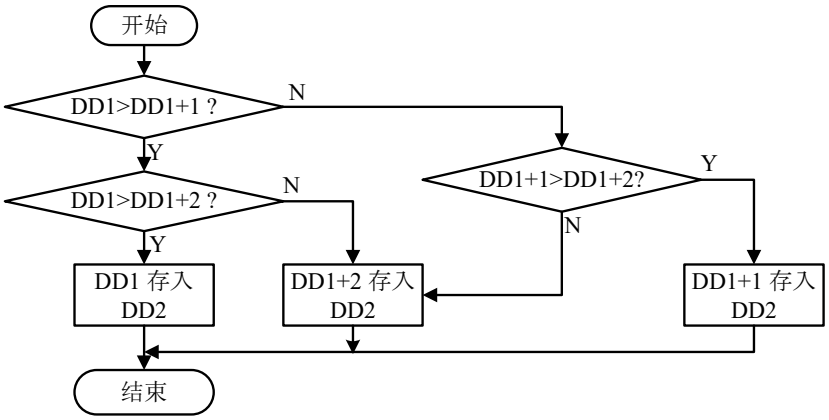


图 2.1 程序流程图

9. 参考代码

```
DATA    SEGMENT
DD1     DB  35H，78H，85H
DD2     DB  ?
DATA    ENDS
CODE    SEGMENT
MAIN    PROC    FAR
        ASSUME      CS：CODE，DS：DATA
START：  PUSH    DS
        SUB     AX，AX
        PUSH    AX
        MOV     AX，DATA
        MOV     DS，AX
        MOV     AL，DD1
        CMP     AL，DD1+1
        JA      AAA1
        MOV     AL，DD1+1
AAA1：   CMP     AL，DD1+2
        JA      AAA2
        MOV     AL，DD1+2
AAA2：   MOV     DD2，AL
        RET
MAIN    ENDP
CODE    ENDS
        END     START
```

10. 思考题

(1) 汇编语言程序中语句END后的标号作用是什么？

(2) 一个段的大小为多少？一定是64KB吗？如何在DEBUG下查看程序在内存的存放情况？画出编程任务4各段在内存的存放情况。

二、加强性实验

1. 实验目的

(1) 进一步熟悉DEBUG的使用方法。

(2) 掌握汇编语言程序的编辑，汇编,连接以及调试过程.实验内容。

2. 实验内容

试编写一个汇编语言程序，要求对键盘输入的小写字母用大写字母显示出来。

3. 实验仪器设备

(1) 硬件环境：IBM-PC及其兼容机；软件环境：操作系统DOS3.0以上。

(2) 编辑程序：EDIT或其它编辑程序。

(3) 汇编程序：MASM.EXE。

(4) 连接程序：LINK.EXE。

(5) 调试程序：DEBUG.COM。

4. 实验步骤

(1) 用文字编辑工具（记事本或EDIT）将源程序输入，其扩展名为.ASM。

(2) 用MASM对源文件进行汇编，产生.OBJ文件和.LST文件。若汇编时提示有错，用文字编辑工具修改源程序后重新汇编，直至通过。

(3) 用TYPE命令显示1产生的.LST文件。

(4) 用LINK将.OBJ文件连接成可执行的.EXE文件。

(5) 在DOS状态下运行LINK产生的.EXE文件。即在屏幕上显示标题并提示按键。每按一键在屏幕上显示二个相同的字符，但小写字母被改成大写。按ESC键可返回DOS。若未出现预期结果，用DEBUG检查程序。

5. 流程图

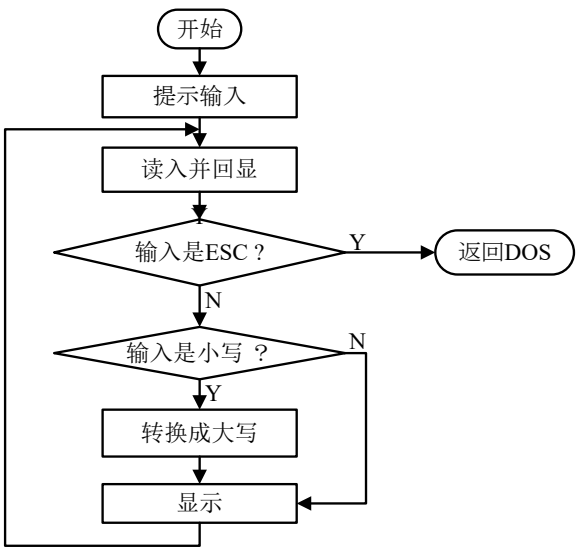


图 2.2 程序流程图

6. 参考代码

```
DATA    SEGMENT
NOTIC   DB  "Please input the word !",0AH,0DH
DATA    ENDS
CODE    SEGMENT
        ASSUME      CS:CODE,DS:DATA
START:  MOV     AX,DATA
        MOV     DS,AX
        MOV     CX,19H
        LEA     BX,[NOTIC]
 AA0:   MOV     DL,[BX]                 ;输出提示
        MOV     AH,2
        INT     21H
        INC     BX
        LOOP    AA0
AA1:    MOV     AH,1                    ;读入回显
        INT     21H
        CMP     AL,1BH                  ;是 ESC  跳转
        JZ      AA3                     ;判断是小写就转换  否则不变
        CMP     AL,61H                  ;判断是不是小写
        JS      AA2
        CMP     AL,7AH
        JNS     AA2
        SUB     AL,20H                  ;转换成大写
 AA2:   MOV     DL,AL
        MOV     AH,2
        INT     21H
        LOOP    AA1
AA3:    MOV     AH,4CH
        INT     21H
CODE    ENDS
        END     START
```

7. 思考题

(1) 若在源程序中把 INT 21H的 'H' 省去，会产生什么现象?

(2) 把 INT 21H 4CH号功能改为 INT 20H，行不行?

实验三　分支程序设计

一、 基础性实验

1. 实验目的

(1) 掌握多路分支 If…Else IF…Else 程序结构和设计方法。

(2) 进一步熟悉DEBUG的使用方法。

(3) 掌握汇编语言程序的编辑,汇编,连接以及调试过程.实验内容。

2. 实验内容

编写程序,将一个包含有 20 个数据的数组 M 分成两个数组：正数数组 P 和负数数组 N,并分别把这两个数组中数据的个数显示出来。

3. 实验仪器设备

(1) 硬件环境：IBM-PC及其兼容机；软件环境：操作系统DOS 3.0以上。

(2) 编辑程序：EDIT或其它编辑程序。

(3) 汇编程序：MASM.EXE。

(4) 连接程序：LINK.EXE。

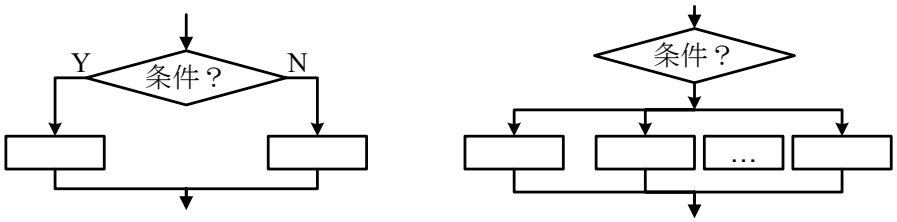
(5) 调试程序：DEBUG.COM

4. 实验原理

分支程序就是根据不同的情况或条件执行不同功能的程序，它具有判断和转移功能，在程序中利用条件转移指令对运算结果的状态标志进行判断实现转移功能。

(1) 分支程序的结构形式

分支程序结构有 2 种基本形式，即二路分支结构和多路分支结构，如图 3.1 所示。两种结构的共同特点是：在某一特定条件下，只执行多个分支中的一个分支。



(A)二路分支结构　　(B)多路分支结构

图 3.1　分支程序的结构形式

(2) 分支程序的设计方法

① 二路分支的设计方法

二路分支可以用简单的转移指令来实现。其流程结构为 If…Else 形式。

例如有一符号函数，任意给定 X 值（－128≤X≤127）,当 X>0 时 Y=1、当 X<0 时 Y=-1、当 X=0 时 Y=0；设给定的 X 值存入 XX1 单元，函数 Y 值存入 YY1 单元。编制按 X 的不同值，求符号函数 Y 值的程序如下所示。

```
DATA    SEGMENT
XX1     DB X
YY1     DB ?
DATA    ENDS
CODE    SEGMENT
        ASSUME      CS:CODE,DS:DATA
START:  MOV     AX,DATA
        MOV     DS,AX
        MOV     AL,XX1
        CMP     AL,0
        JZ      AA1
        JNS     AA2
        MOV     AL,0FFH
        JMP     AA1
AA2:    MOV     AL,1
AA1:    MOV     YY1,AL
        MOV     AH,4CH
        INT     21H
CODE    ENDS
        END     START
```

② 多路分支的设计方法

多路分支程序结构设计的实现过程相当于 CASE 语句的形式。最直接的方法就是使用简单的分支结构进行组合，使用条件转移指令结合指令结合无条件转移指令来实现。在实际应用中，多路分支在是实现时通常有三种方法：即逻辑分解法、地址表法和转移表法。

例如：某工厂有 5 种产品的加工程序 1 到 5 分别存放在 WORK1,WORK2…WORK5 为首地址的内存区域中，分别键入 1，键入 2…键入 5 转入选择其对应的加工程序。下面用不同方法编制其程序。

其一：逻辑分解法

```
CODE    SEGMENT
        ASSUME      CS:CODE
START:  MOV     AH,1
        INT     21H
        CMP     AL,31H
        JZ      WORK1
        CMP     AL,32H
        JZ      WORK2
        CMP     AL,33H
        JZ      WORK3
```

```
                CMP     AL,34H
                JZ      WORK4
                CMP     AL,35H
                JZ      WORK5
                JMP     WORK0
WORK1:          ┆
WORK2:          ┆
WORK3:          ┆
WORK4:          ┆
WORK5:          ┆
WORK0:  MOV     AH,4CH
        INT     21H
CODE    ENDS
        END     START
```

其二：地址表法

表地址＝表首址＋（键号－1）×2

```
DATA    SEGMENT
TABLE   DW WORK1,WORK2,WORK3,WORK4,WORK5    ；地址表
DATA    ENDS
CODE    SEGMENT
        ASSUME      CS:CODE,DS:DATA
START:  MOV     AX,DATA
        MOV     DS,AX
        LEA     BX,TABLE
        MOV     AH,1
        INT     21H
        AND     AL,0FH
        DEC     AL              ；键号减一
        ADD     AL,AL           ；键号乘 2
        SUB     AH,AH
        ADD     BX,AX           ；形成表地址
        JMP     WORD PTR[BX]    ；转移
WORK1:  ┆
WORK2:  ┆
WORK3:  ┆
WORK4:  ┆
WORK5:  ┆
        MOV     AH,4CH
        INT     21H
CODE    ENDS
        END     START
```

其三：段内转移表法

段内短转移表中每条段内短转移指令占用 2 个字节，其表地址为：

表地址＝表首址＋（键号－1）×2

段内近转移表中每条段内近转移指令占用 3 个字节，其表地址为：

表地址＝表首址＋（键号－1）×3

段内近转移表法程序如下所示：

```
CODE    SEGMENT
        ASSUME    CS:CODE
START:  LEA     BX,WORK          ；转移表首址送 BX
        MOV     AH,1
        INT     21H
        AND     AL,0FH
        DEC     AL               ；键号减一
        MOV     AH,AL
        ADD     AL,AL            ；键号乘 2
        ADD     AL,AH            ；键号乘 3
        SUB     AH,AH
        ADD     BX,AX
        JMP     BX
WORK:   JMP     NEAT PTR WORK1   ；转移表
        JMP     NEAT PTR WORK2
        JMP     NEAT PTR WORK3
        JMP     NEAT PTR WORK4
        JMP     NEAT PTR WORK5
WORK1:          ┆
WORK2:          ┆
WORK3:          ┆
WORK4:          ┆
WORK5:          ┆
        MOV     AH,4CH
        INT     21H
CODE    ENDS
        END     START
```

5. 实验步骤

(1) 进入调试软件环境，输入源程序。

(2) 汇编源程序。

(3) 用单步方式运行程序。

(4) 检查并记录各寄存器和存储单元内容的变化。

6. 实验提示

字符在计算机内以 ASCII 码值来表示，大小写字母也不例外，其中大写字母（A~Z），小写字母（a~z）在 ASCII 码字符集中连续存放，其值的范围分别为 65~90，97~122。这样我们根据字符的 ASCII 码值，自然能判定字符是否为大、小写字母，同时对小写字母，用减 32 的方法转化为其对应的大写字母。键盘输入和字母显示可用 DOS 或 BIOS 调用实现。

7. 流程图

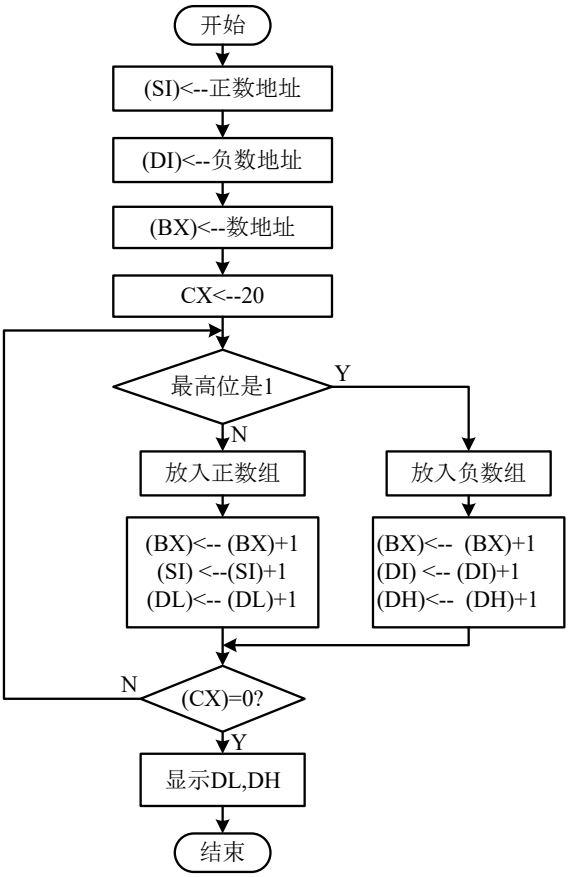


图 3.2 程序流程图

8. 参考代码

```
DSEG    SEGMENT    PARA    PUBLIC'DSEG'
M       DB         X1,X2,…X20        ;填写实际需要分组的数值
P       DB         20 DUP(?)
N       DB         20 DUP(?)
DSEG    ENDS
CODE    SEGMENT
        ASSUME     CS:CODE,DS:DSEG
START:  MOV        AX,DSEG
        MOV        DS,AX
        LEA        SI,P          ; 正数数组首地址
        LEA        DI,N          ; 负数数组首地址
        LEA        BX,M          ; 输入数组首地址
        XOR        AX,AX
        XOR        DX,DX
        MOV        CX,20
L1:     MOV        AL,[BX]
        TEST       AL,80H
        JZ         L2
        MOV        [DI],AL       ; 放入负数
        INC        BX
        INC        DI
        INC        DH            ; 负数个数增加
```

```
        JMP     L3
L2:     MOV     [SI],AL         ；放入正数
        INC     BX
        INC     SI
        INC     DL              ；正数个数增加
L3:     LOOP    L1
        MOV     CX,2
L5:     MOV     BL,DL           ；DL 的内容先复制到 BL，然后每 4 位输出
        SHR     DL,1
        SHR     DL,1
        SHR     DL,1
        SHR     DL,1
        AND     DL,0FH
        CMP     DL,10
        JB      L4
        ADD     DL,7            ；若待输出的数字大于 9，加 7 调整后输出
L4:     ADD     DL,30H          ；若待输出的数字小于 10，直接加 30H 输出
        MOV     AH,2
        INT     21H
        MOV     DL,BL
        AND     DL,0FH
        CMP     DL,10
        JB      L6
        ADD     DL,7
L6:     ADD     DL,30H
        MOV     AH,2
        INT     21H
        MOV     DL,0AH          ；换行
        MOV     AH,2
        INT     21H
        MOV     DL,0DH          ；回车
        MOV     AH,2
        INT     21H
        MOV     DL,DH           ；再输出负数个数
        LOOP    L5
        MOV     AH,4CH
        INT     21H
CODE    ENDS
        END     START
```

9. 思考题

示例题中是否一定要用 CMP 指令来实现条件转移？如果不是，请设计出替代程序。

二、 加强性实验

1. 实验目的

(1) 进一步掌握多路分支 If…Else IF…Else 程序结构和设计方法。

(2) 进一步熟悉DEBUG的使用方法。

2. 实验内容

接收键盘输入字符，判别该字符是否为字母、数字或非字母也非数字类的字符。若是数字，则显示“DIGIT”；若是字母，显示“LETTER”；若是非数字也非字母类的字符，则显示“OTHER”。

3. 实验仪器设备

(1) 硬件环境：IBM-PC 及其兼容机 软件环境：操作系统 DOS3.0 以上

(2) 编辑程序：EDIT 或其它编辑程序

(3) 汇编程序：MASM.EXE 6.1X

(4) 连接程序：LINK.EXE

(5) 调试程序：DEBUG.COM

4. 实验步骤

(1) 进入调试软件环境，输入源程序。

(2) 汇编源程序。

(3) 用单步方式运行程序。

(4) 检查并记录各寄存器和存储单元内容的变化。

5. 实验提示

采用逻辑分解法。

6. 流程图

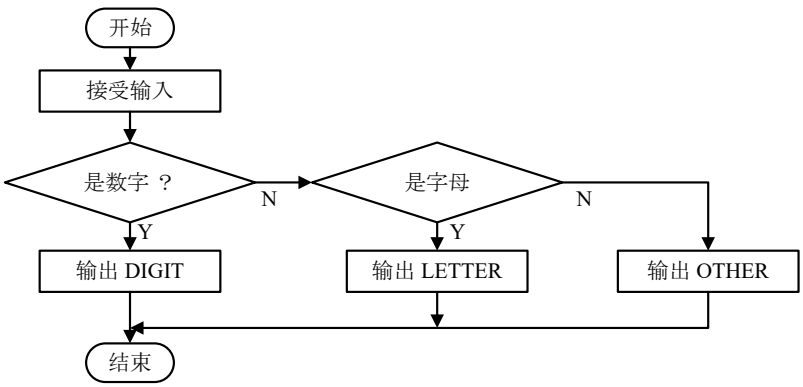


图 3.3 程序流程图

7. 参考代码

```
DATA    SEGMENT
DIGIT       DB  "DIGIT",0AH,0DH
LETTER      DB  "LETTER",0AH,0DH
OTHER       DB  "OTHER",0AH,0DH
DATA    ENDS
CODE    SEGMENT
        ASSUME      CS:CODE,DS:DATA
START:  MOV     AX,DATA
        MOV     DS,AX
        MOV     AH,07               ;接受输入不回显
```

```
        INT     21H
        CMP     AL,30H          ;判断是否是数字
        JB      AA1
        CMP     AL,39H
        JA      AA1
        MOV     CX,7            ;显示 digit
        LEA     SI,DIGIT
AA0:    MOV     DL,[SI]
        MOV     AH,2
        INT     21H
        INC     SI
        LOOP    AA0
        JMP     BB
AA1:    CMP     AL,41H          ;判断是否为字母
        JB      AA3
        CMP     AL,5AH
        JA      AA2
        LEA     SI,LETTER
        MOV     CX,8
AA4:    MOV     DL,[SI]
        MOV     AH,2
        INT     21H
        INC     SI
        LOOP    AA4
        JMP     BB
AA2:    CMP     AL,61H
        JB      AA3
        CMP     AL,7AH
        JA      AA3
        LEA     SI,LETTER
        MOV     CX,8
AA5:    MOV     DL,[SI]
        MOV     AH,2
        INT     21H
        INC     SI
        LOOP    AA5
        JMP     BB
AA3:    LEA     SI,OTHER        ;其他
        MOV     CX,7
AA6:    MOV     DL,[SI]
        MOV     AH,2
        INT     21H
        INC     SI
        LOOP    AA6
BB:     MOV     AH,4CH
        INT     21H
CODE    ENDS
        ENDSTART
```

8. 思考题

如果实验题中还需进一步对大、小写字母进行区分，如何编写程序？

实验四　循环程序设计

一、 基础性实验

1. 实验目的

循环结构程序设计是使用循环控制类指令，实现对某程序段重复执行一定的次数。通过本次实验，使学生初步掌握单重、多重循环程序的设计方法，为编制较大的程序和阅读较复杂程序打下基础。

2. 实验内容

设有5个学生参加4门课的考试，其中4门课的成绩以压缩BCD码方式存放在字组COURSE1中（设每门课的成绩满分为 99 分）；试编制计算每个学生总分（总分存入字组 NUM1 中）的源程序。

3. 实验仪器设备

(1) 硬件环境：IBM-PC及其兼容机；软件环境：操作系统DOS3.0以上。

(2) 编辑程序：EDIT或其它编辑程序。

(3) 汇编程序：MASM.EXE。

(4) 连接程序：LINK.EXE。

(5) 调试程序：DEBUG.COM。

4. 实验原理

循环程序是强制 CPU 重复执行某一指令序列（程序段）的一种程序结构形式；凡是要重复执行的程序段都采用循环结构设计。循环结构程序设计方法简化了程序清单书写形式，而且减少了内存空间占用。但循环程序并不简化程序执行过程，相反，它增加了一些循环控制等环节，总的程序执行语句和时间有所增加。

(1) 循环程序的组成

在程序中，凡是有重复执行某一段程序的程序结构称为循环程序。在程序设计时，循环程序是经常采用的一种结构形式，它不仅能缩短代码长度，而且能简化设计。

循环程序由五部分组成：

① 初始化部分：它是循环的准备部分，为程序操作、地址指针、循环计数、结束条件等设置初始值。

② 循环工作部分：这是循环程序的核心部分，它以初始化部分设置的初值为基础，动态地执行功能相同的操作。

③ 循环修改部分：与工作部分协调配合，修改循环工作部分的变量指针等，为下次重复操作准备。

④ 循环控制部分：修改控制量用以判断和控制循环的走向，即根据控制量的状态标志条件，决定是继续循环还是退出循环。

⑤ 循环结束部分：退出循环后，对循环结果进行处理。

循环程序五部分中的 1、5 部分，分别在程序的头和尾，而 2、3、4 部分在循环程序中间的不同排序就使循环程序有多种结构形式，但常用的基本结构形式有两种，如图 4.1 和图 4.2 所示。

(2) 循环程序的结构

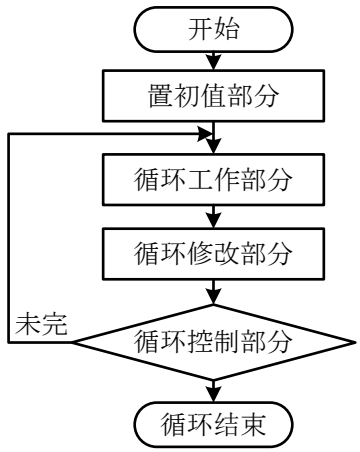


图 4.1 先执行后判断结构

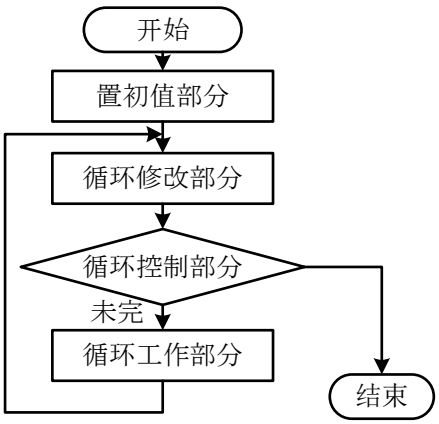


图 4.2 先判断后执行结构

图 4.1 先执行后判断结构形式的特点是先执行循环工作部分，然后再判断循环是否结束。所以这种循环结构称为不允许零次循环结构，即不论循环是否满足，至少要执行一次循环体工作部分。

图 4.2 先判断后执行结构形式的特点是先判循环是否结束，若未结束就执行循环工作部分，否则退出循环。因为这种循环结构有可能循环工作部分一次也不执行，所以这种循环结构称为允许零次循环结构。

(3) 循环程序的设计方法

下面以实例来说明循环程序的设计方法及不同循环结构的编程特点。

设从内存偏移地址 1000H 单元开始连续存放无符号整数 1、2、3…254、255。试编制将其和存入 SUM1 单元中的源程序。

自然数 1～255 的和一定是一个大于 8 位二进制数，而小于 16 位二进制数的数。为避免 8 位累加求和应处理进位问题，我们采取 16 位累加求和方式，即先 0 送 DH，AX 为累加和，每次将内存的数取入 DL 中后进行 AX 与 DX 相加。下面分别使用不同的循环控制条件完成循环控制部分的设计。

其一：计数控制循环

取 255 为计数控制循环条件。

① 先执行后判断结构形式的计数控制循环

```
DATA        SEGMENT
ORG         1000H
NUMBER1     DB  1,2,3 … 255
SUM1        DW ?
DATA        ENDS
CODE        SEGMENT
            ASSUME      CS:CODE,DS:DATA
START:  MOV     AX,DATA
        MOV     DS,AX
        LEA     BX,NUMBER1
        MOV     AX,0
        MOV     DH,0
        MOV     CL,255          ；计数初值
AA1:    MOV     DL,[BX]         ；取数
        ADD     AX,DX           ；求和
        INC     BX              ；修改地址指针
        SUB     CL,1            ；计数
        JNZ     AA1             ；判转
        MOV     SUM1,AX         ；存和
        MOV     AH,4CH
        INT     21H
CODE    ENDS
        END     START
```

② 先判断后执行结构形式的计数控制循环

```
DATA        SEGMENT
ORG         1000H
NUMBER1     DB 1,2,3 … 255
SUM1        DW ?
DATA        ENDS
CODE        SEGMENT
            ASSUME      CS:CODE,DS:DATA
START:  MOV     AX,DATA
        MOV     DS,AX
        LEA     BX,NUMBER1－1
        MOV     AX,0
        MOV     DH,0
        MOV     CL,0            ；计数初值 256
AA1:    INC     BX              ；修改地址指针
        SUB     CL,1            ；第一次（CL）－1=255,即 0FFH
        JZ      AA2             ；判转
        MOV     DL,[BX]         ；取数
        ADD     AX,DX           ；求和
        JMP     AA1
AA2:    MOV     SUM1,AX         ；存和
        MOV     AH,4CH
```

```
        INT     21H
CODE    ENDS
        END     START
```

需要说明的是：计数控制循环程序可分为正计数法和倒计数法；上述程序采用的是倒计数法。当编制计数控制循环程序时，这两种结构的计数初始值差 1。上述程序的循环控制是通过判断 ZF 标志位来实现的；若将各计数初值减 1，则循环控制可通过判断 CF 标志位来实现，这是用 SUB 实现 CL 减 1，而不用 DEC 的原因。

其二：条件控制循环

因该例中的数均不相同，且以 255 为结尾，则取 255 为结束循环的控制条件。

① 先执行后判断结构形式的条件控制循环

```
DATA        SEGMENT
ORG         1000H
NUMBER1     DB  1,2,3 … 255
SUM1        DW ?
DATA        ENDS
CODE        SEGMENT
        ASSUME CS:CODE,DS:DATA
START:  MOV     AX,DATA
        MOV     DS,AX
        LEA     BX,NUMBER1
        MOV     AX,0
        MOV     DH,0
AA1:    MOV     DL,[BX]         ；取数
        ADD     AX,DX           ；求和
        INC     BX              ；修改地址指针
        CMP     DL,255
        JNZ     AA1             ；判转
        MOV     SUM1,AX         ；存和
        MOV     AH,4CH
        INT     21H
CODE    ENDS
        END     START
```

② 先判断后执行结构形式的条件控制循环

```
DATA        SEGMENT
ORG         1000H
NUMBER1     DB  1,2,3 … 255
SUM1        DW ?
DATA        ENDS
CODE        SEGMENT
            ASSUME      CS:CODE,DS:DATA
START:  MOV     AX,DATA
        MOV     DS,AX
        LEA     BX,NUMBER1－1
        MOV     AX,255
        MOV     DH,0
AA1:    INC     BX              ；修改地址指针
```

```
        MOV     DL,[BX]
        CMP     DL,255
        JZ      AA2             ；判转
        ADD     AX,DX           ；求和
        JMP     AA1
AA2:    MOV     SUM1,AX         ；存和
        MOV     AH,4CH
        INT     21H
CODE    ENDS
        END     START
```

需要说明的是：本例结束循环的控制条件是 255，对于先判断后执行结构形式的条件控制循环程序未进行相加 255（所以 AX 的初值为 255）；而对于先执行后判断结构形式的条件控制循环程序已进行相加 255。可见，对于先判断后执行结构形式的结束循环条件，是不被循环工作部分处理的条件；而对于先执行后判断结构形式的结束循环条件，是被循环工作部分处理的条件。

其三：变量控制循环

因该例数据偏移地址 1000H 为首地址，那么，尾数 255 的偏移地址应为 10FEH，则取变量偏移地址 10FFH 为结束循环的控制条件。

① 先执行后判断结构形式的变量控制循环

```
DATA        SEGMENT
ORG         1000H
NUMBER1     DB  1,2,3 … 255
SUM1        DW ?
DATA        ENDS
CODE        SEGMENT
            ASSUME      CS:CODE,DS:DATA
START:  MOV     AX,DATA
        MOV     DS,AX
        LEA     BX,NUMBER1
        MOV     AX,0
        MOV     DH,0
AA1:    MOV     DL,[BX]         ；取数
        ADD     AX,DX           ；求和
        INC     BX              ；修改地址指针
        CMP     BX,10FFH
        JNZ     AA1             ；判转
        MOV     SUM1,AX         ；存和
        MOV     AH,4CH
        INT     21H
CODE    ENDS
        END     START
```

② 先判断后执行结构形式的变量控制循环

```
DATA        SEGMENT
ORG         1000H
NUMBER1     DB  1,2,3 … 255
SUM1        DW ?
```

```
DATA        ENDS
CODE        SEGMENT
            ASSUME    CS:CODE,DS:DATA
START:  MOV     AX,DATA
        MOV     DS,AX
        LEA     BX,NUMBER1－1
        MOV     AX,0
        MOV     DH,0
AA1:    INC     BX              ；修改地址指针
        CMP     BX,10FFH
        JZ      AA2
        MOV     DL,[BX]         ；判转
        ADD     AX,DX           ；求和
        JMP     AA1
AA2:    MOV     SUM1,AX         ；存和
        MOV     AH,4CH
        INT     21H
CODE    ENDS
        END     START
```

需要说明的是：本例变量控制循环结束的条件是尾数 255 的偏移地址 10FEH 的下一个地址 10FFH。对于两种循环结构形式均是不被循环工作部分处理的条件。

通过循环程序设计过程，可以看出，同一问题可采用不同的循环结构；同一循环结构可采用不同的控制方法；同一控制方法可采用不同的状态标志作为控制条件。在上例中，我们分别编制了计数控制循环、条件控制循环和变量控制循环程序；在编制不同问题的循环程序过程中，还会有其它控制方法，但各种控制方法实质上均可归结为一种，即条件控制循环法，只是条件的特点不同而已。

(4) 循环的嵌套

按循环的嵌套层次，循环程序又可分为单循环程序和多重循环程序。单循环程序的循环体内只是一些简单的分支程序，多重循环程序的循环体内还包含一个循环程序。

单重循环的结构常常难以解决实际应用问题，所以人们引入了多重循环。这些循环是一层套一层的。因此又称为循环的嵌套。对于多重循环，在设计时应该注意：

①内循环必须完整地包含在外循环内，内外循环不能相互交叉；

②内循环在外循环中的位置可根据需要任意设置，但应避免出现混乱；

③当通过外循环再次进入内循环时，内循环中的初始条件必须重新设置；

④多个内循环可以拥有一个外循环，这些内循环间的关系可以是嵌套的，也可以是并列的。

(5) 循环程序的设计步骤

设计循环程序时，我们建议一般采用以下几个步骤：

①根据问题，确定是选取先执行后判断结构形式，还是选取先判断后执行结构形式。

②必须选择恰当的循环条件来控制循环的运行和结束。其中，计数器控制法适用于循环次数

已知的情况，条件控制法适用于循环次数未知的情况。

③根据循环变化的规律，确定适合于循环的工作部分。

④可根据循环程序控制方法的不同，来设置循环参数的初值。

⑤考虑循环参数的修改部分，分析并确定参数的每次修改方法。

总之，读者主要靠自己结合语句功能并根据具体情况多实践编程上机，并逐步总结积累经验，就可编制出高质量的循环结构程序。

5. 实验步骤

(1) 进入调试软件环境，输入源程序。

(2) 汇编源程序。

(3) 用单步方式运行程序。

(4) 检查并记录各寄存器和存储单元内容的变化。

6. 实验提示

用多重循环实现，内层循环用来计算某个学生的四门成绩之和，外层循环用来控制所有学生的总分计算。注意内层循环获取某个学生的某门课程成绩需要将课程地址加 5。

7. 流程图

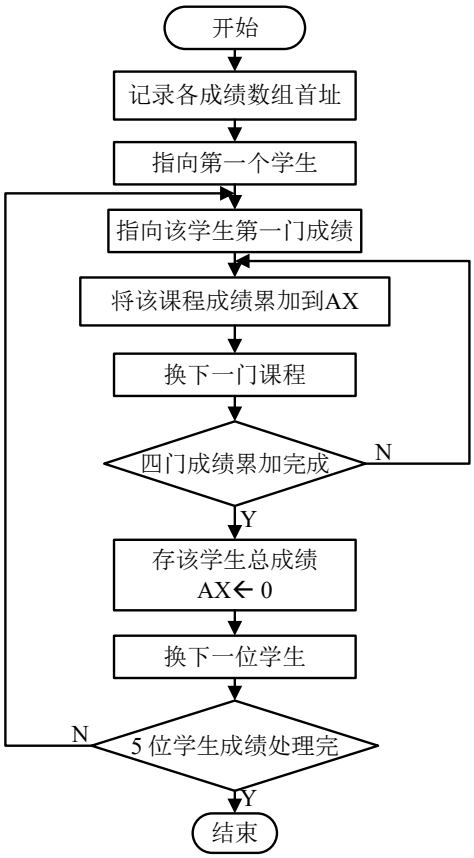


图 4.3 程序流程图

8. 参考代码

```
DATA      SEGMENT
COURSE1   DB    70H,88H,92H,90H,99H      ；第一门课成绩
          DB    67H,77H,88H,76H,69H      ；第二门课成绩
          DB    74H,87H,77H,74H,70H
          DB    99H,97H,94H,98H,96H
NUM1      DW    5 DUP(0)
DATA      ENDS
CODE      SEGMENT
          ASSUME    CS:CODE,DS:DATA
START:    MOV   AX,DATA
          MOV   DS,AX
          LEA   SI,COURSE1
          LEA   DI, NUM1
          SUB   SI,5          ；COURSE1 首址减 5 以便循环
          MOV   CL,5          ；外层循环计数初值
AA1:      MOV   BX,SI         ；形成某个学生第一门课成绩的首址减 5
          SUB   AX,AX         ；清和寄存器
          MOV   CH,4          ；内循环计数初值
AA2:      ADD   BX,5          ；形成某个学生 1、2…4 门课成绩地址
          ADD   AL,[BX]       ；按 BCD 码求和
          DAA                 ；压缩 BCD 码加调整
          ADC   AH,0          ；累加进位
          DEC   CH            ；内循环计数控制
          JNZ   AA2
          MOV   [DI],AX       ；存总分
          INC   SI            ；形成下个学生第一门课成绩的首址减 5
          ADD   DI,2          ；形成下个学生的总分地址
          DEC   CL            ；外循环计数控制
          JNZ   AA1
          MOV   AH,4CH
          INT   21H
CODE      ENDS
          END   START
```

9. 思考题：

(1) 在示例题中，JMP指令的作用是什么？

(2) 本次实验中，出现了哪些错误？如何排除？

二、 加强性实验

1. 实验目的

进一步掌握循环结构程序设计是使用循环控制类指令。

2. 实验内容

将一个 4 ×4 的矩阵与一个 4 ×1 的矩阵相乘，并将所得的矩阵存入单元中。

3. 实验仪器设备

(1) 硬件环境：IBM-PC及其兼容机；软件环境：操作系统DOS3.0以上。

(2) 编辑程序：EDIT或其它编辑程序。

(3) 汇编程序：MASM.EXE。

(4) 连接程序：LINK.EXE。

(5) 调试程序：DEBUG.COM。

4. 实验步骤

(1) 进入调试软件环境，输入源程序。

(2) 汇编源程序。

(3) 用单步方式运行程序。

(4) 检查并记录各寄存器和存储单元内容的变化。

5. 流程图

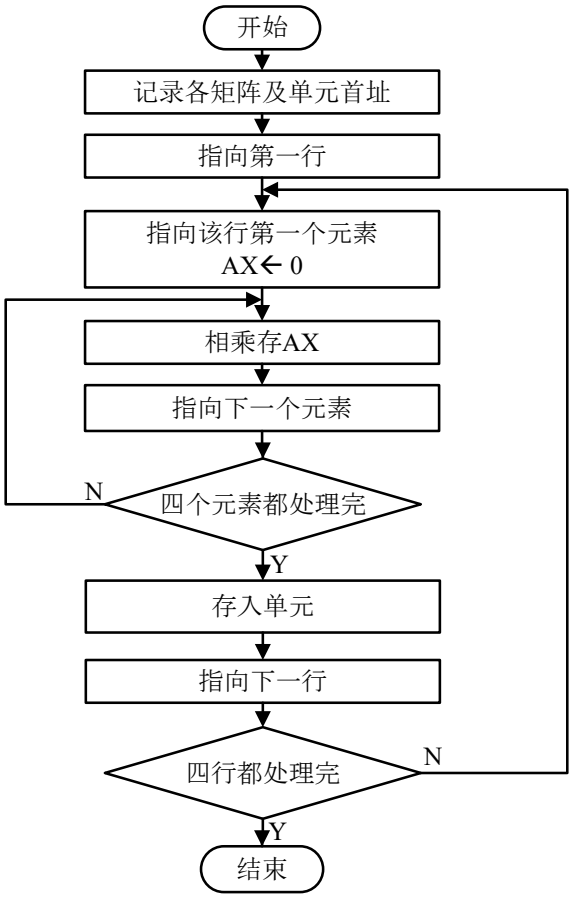


图 4.4 程序流程图

6. 参考代码

```
DATA    SEGMENT
```

```
MATRIX1   DB  3H,4H,15H,8H
          DB  4H,5H,6H,17H
          DB  8H,9H,3H,2H
          DB  1H,1H,4H,6H
MATRIX2   DB  0F3H,0F9H,8H,0E6H
UNIT      DW 4 DUP(0)
DATA      ENDS
CODE   SEGMENT
       ASSUME     CS:CODE,DS:DATA
START: MOV    AX,DATA
       MOV    DS,AX
       LEA    SI,MATRIX1
       LEA    DI,MATRIX2
       LEA    BX,UNIT
       MOV    CL,4H
AA0:   SUB    AX,AX
       MOV    CH,4H
AA1:   MOV    DX,AX
       MOV    AL,[DI]
       MUL    BYTE PTR[SI]
       ADD    AX,DX
       INC    SI
       INC    DI
       DEC    CH
       JNZ    AA1
       LEA    DI,MATRIX2
       MOV    [BX],AX
       INC    BX
       INC    BX
       DEC    CL
       JNZ    AA0
       MOV    AH,4CH
       INT    21H
CODE   ENDS
       END    START
```

实验五　子程序设计（一）

一、 基础性实验

1. 实验目的

(1) 掌握主程序和子程序之间的调用关系及其调用方法。

(2) 掌握子程序调用过程中近程调用与远程调用的区别。

(3) 掌握子程序设计方法，能合理划分子程序。

(4) 掌握汇编子程序的定义、调用、返回、参数传递等有关问题的实现，以及运行过程中的堆栈、标志位变化情况。

2. 实验内容

用堆栈传送参数和参数表地址方式，编制键入 8 位的非压缩 BCD 码加法并显示的程序。

3. 实验仪器设备

(1) 硬件环境：IBM-PC及其兼容机；软件环境：操作系统DOS3.0以上。

(2) 编辑程序：EDIT或其它编辑程序。

(3) 汇编程序：MASM.EXE。

(4) 连接程序：LINK.EXE。

(5) 调试程序：DEBUG.COM。

4. 实验原理

(1) 子程序的结构形式

子程序由子程序定义（即子程序入口）、保护现场、取入口参数、子程序体、存输出参数、恢复现场和返回 7 部分组成，其结构形式如图 5.1 所示。

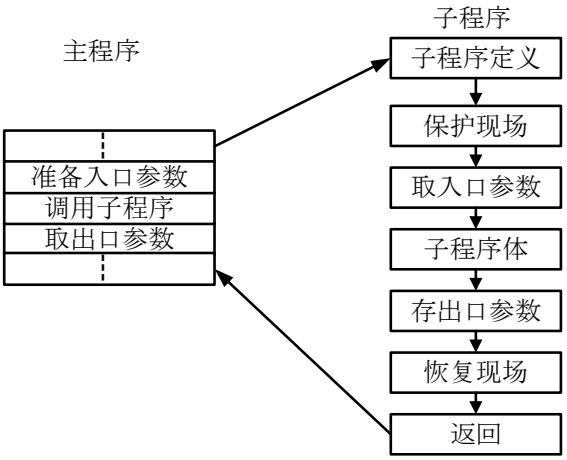


图 5.1 子程序的结构形式

(2) 子程序的定义

子程序是通过过程定义伪指令 PROC～ENDP 来定义的。过程定义伪指令用在过程(子程序)的前后，使整个过程形成清晰的、具有特定功能的代码块。其格式为：

```
过程名    PROC    属性
          ┆
过程名    ENDP
```

其中过程名为标识符，它又是子程序入口的符号地址。它的写法和标号的写法相同。属性是指类型属性，它可以是 NEAR 或 FAR。段内调用时用 NEAR 属性，段间调用时用 FAR 属性。汇编程序对源程序进行汇编时，根据 PROC 伪指令的类型属性来确定 CALL 和 RET 指令的属性。即若过程是 FAR 属性的，则 CALL 和 RET 指令就确定为 FAR 属性；若过程是 NEAR 属性的，则 CALL 和 RET 指令就确定为 NEAR 属性的。属性缺省时默认为 NEAR。这样，用户只需在定义过程时考虑它的属性，而 CALL 和 RET 的属性可以由汇编程序来确定。用户对过程属性的确定原则是：

① 主程序和过程在同一个代码段中，则用 NEAR 属性。

② 主程序和过程不在同一个代码段中，则用 FAR 属性。

③ 一般可将主程序定义为 FAR 属性，因为我们将程序的主过程看作为 DOS 调用的一个子过程，而 DOS 对主过程的调用和返回都是 FAR 属性。

(3) 子程序的调用和返回

子程序的正确执行是由子程序的正确调用和正确返回保证的。我们在第 3 章中已经介绍过的 CALL 和 RET 指令就是完成调用和返回的功能。为保证其正确性，除 PROC 的属性要正确选择外，还应该注意子程序运行期间的堆栈状态，由于执行 CALL 指令时已把返回地址压入堆栈，所以执行 RET 指令时应该将返回地址弹出堆栈，如果在子程序运行过程中不能正确使用堆栈而造成执行 RET 前 SP 并未指向进入子程序时的返回地址，则必然会导致运行出错，因此子程序中对堆栈的使用应该特别小心，以免发生错误。

由于主程序和子程序经常是分别编制的，所以它们所使用的寄存器往往会发生冲突。如果主程序在调用子程序之前的某个寄存器内容在从子程序返回后还有用，而子程序又恰好使用了同一个寄存器，这就破坏了该寄存器的原有内容，因而会造成程序运行错误，为避免这种错误的发生，这就需要对这些寄存器的内容加以保护。这就称为保护现场。子程序执行完毕后再恢复这些被保护的寄存器的内容，称为恢复现场。保护现场和恢复现场的工作原则上既可在主程序中完成，也可在子程序中完成。但通常采用在子程序中完成方式。在子程序中保护现场则一定在子程序中恢复。这样安排程序结构清楚，使用方便，不易出错。如果子程序设计时，未考虑保护主程序的现

场，则可在主程序调用子程序前再保护现场，从子程序返回后再恢复现场。

在子程序中进行现场保护和现场恢复通常采用下述方法：

① 利用栈指令进行现场的保存与恢复

用进栈指令 PUSH 将寄存器的内容保存在堆栈中，恢复时再用出栈指令 POP 从堆栈中取出。这种方法较为方便，尤其在设计嵌套子程序和递归子程序时，由于进栈和出栈指令会自动修改堆栈指针，保护和恢复现场层次清晰，只要注意堆栈操作的先进后出的特点，就不会引起出错，故这是经常采用的一种方法。

② 利用数据传送指令进行现场的保存与恢复

利用数据传送指令 MOV 将寄存器的内容保存在指定的内存单元中，恢复现场时再用数据传送指令，从指定的内存单元取回到对应的寄存器中。这种方法使用时不方便，故较少使用。

③ 用 PUSHA 和 POPA 指令进行现场的保存与恢复

从 286 开始可用的 PUSHA 指令将寄存器的内容保存在堆栈中：恢复时再用 POPA 指令从堆栈中取出。

```
SUB6    PROC      NEAR
        PUSHA
        ┆
        POPA
        RET
SUB6    ENDP
```

④ 用 PUSHAD 和 POPAD 指令进行现场的保存与恢复

从 386 开始可用的 PUSHAD 指令将寄存器的内容保存在堆栈中：恢复时再用 POPAD 指令从堆栈中取出。

```
SUB7    PROC    NEAR
        PUSHAD
        ┆
        POPAD
        RET
SUB7    ENDP
```

需要说明的是，在子程序设计时，有时并非所有寄存器都需要保存。程序设计者应考虑哪些寄存器是必须保存的，哪些寄存器是不需要保存的。其保存原则是，子程序中用到的寄存器是应该保存的。但是，在主程序和子程序之间传送参数的寄存器，就不一定需要保存，特别是用来向主程序回送结果的寄存器，就更不应该因保存和恢复寄存器而破坏了向主程序传送的信息。

(4) 子程序的参数传送

主程序在调用子程序前，必须把需要子程序加工处理的数据传送给子程序，这些加工处理的数据就称为输入参数。当子程序执行完返回主程序时，应该把最终结果传送给主程序，这些加工处理所得的结果称为输出参数。这种主程序和子程序之间的数据传送称为参数传送。通常进行主

程序和子程序间参数传送的方法有三种。

① 通过寄存器传送参数

这种方法是在主程序里预先将输入参数置于规定的寄存器中，进入子程序后，可直接从规定的寄存器中取出数据进行加工处理。子程序执行后又将结果存到指定的寄存器中，返主后就可从指定的寄存器中取出输出参数。这种方法最简单方便，实际上也是用得最多的一种方法。但由于CPU 中的寄存器数量有限，故该方法仅适合于传送参数个数较少的情况。

② 通过存储单元传送参数

采用存储器的某个区域传递参数，可以预先定义从内存的某个指定地址开始，按一定的顺序存放输入参数和输出参数。主程序可以先将输入参数利用数据传送指令送到指定内存单元保存，进入子程序后由子程序按约定取出这些参数。子程序加工完毕，又将这些结果送回存储区域保存，由主程序再取出处理。这种方法适合参数较多的情况。

③ 通过堆栈传送参数或参数表地址

通过堆栈传送参数或参数表地址方法是在主程序里将输入参数或参数表地址压入堆栈内，在子程序里从堆栈中取出参数或参数表地址。子程序执行后又将结果或结果地址压入堆栈内。返回主程序后再由主程序从堆栈中取出结果或结果地址。如果传送参数较多，或在子程序内部多次访问参数时，该方法是行之有效的手段。但其缺点是执行 CALL 指令时也要压栈保存返回地址，因此使用堆栈中的参数应注意避免破坏返回地址。

5. 实验步骤

(1) 进入调试软件环境，输入源程序。

(2) 汇编源程序。

(3) 用单步方式运行程序。

(4) 检查并记录各寄存器和存储单元内容的变化。

6. 实验提示

(1) NUMBER1、NUMBER2分别是两个8位的非压缩BCD码数的最低位的地址, 最高位地址分别为NUMBER1＋7、NUMBER2＋7。NUMBER3是和的最低位的地址, 和的最高位地址为NUMBER3＋8。

(2) SUB1、SUB2子程序和主程序在同一程序模块中；主程序完成8位的非压缩BCD码加法运算。

(3) 为简化程序，显示时不考虑最高位为0的情况。

(4) SUB1子程序采用堆栈传送参数方式与主程序传送参数，把键入8位以内的非压缩BCD码及其个数压入堆栈，键入非数字键返主。

(5) SUB2子程序在CRT上显示8位非压缩BCD码加法运算结果。

7. 流程图

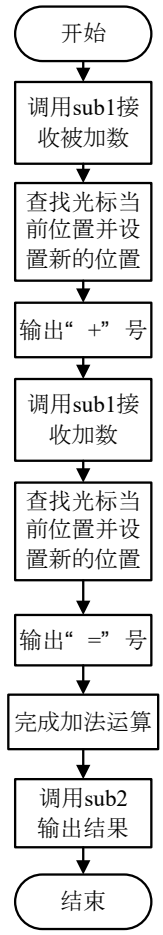


图 5.2 主程序流程图

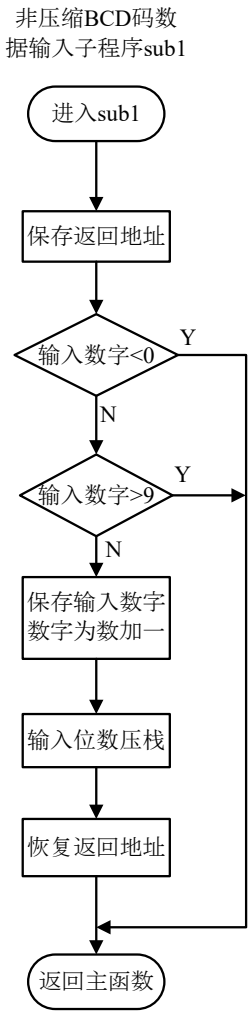


图 5.3 输入子程序流程图

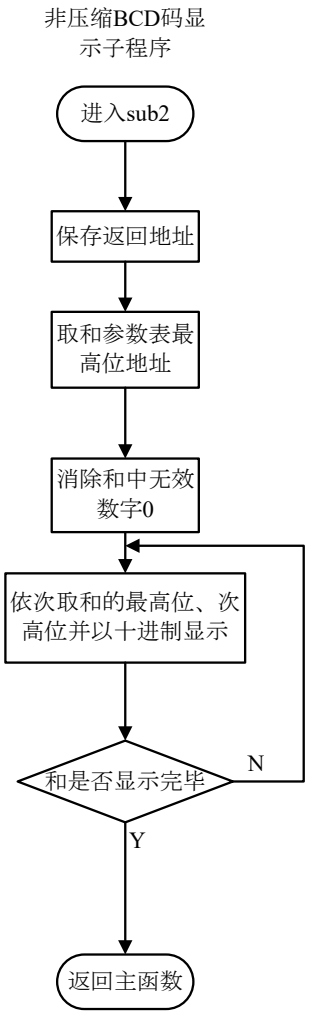


图 5.4 显示子程序流程图

8. 参考代码

```
DATA    SEGMENT
NUMBER1   DB 8 DUP(0)          ;被加数参数列表
NUMBER2   DB 8 DUP(0)          ;加数参数列表
NUMBER3   DB 9 DUP(0           ;和参数列表
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE,DS:DATA
MAIN    PROC    FAR            ;主程序定义为 FAR 类型
        MOV     AX,DATA
        MOV     DS,AX
        CALL    SUB1           ;调 SUB1 非压缩 BCD 码接收子程序
        POP     CX             ;取被加数位数
        LEA     BX,NUMBER1     ;取被加数参数表地址
AA3:    POP     AX             ;取被加数个位、十位、百位…
        MOV     [BX],AL        ;被加数存入参数表
```

```
        INC     BX              ;形成下位地址
        LOOP    AA3
        MOV     AH,3            ;查找当前光标位置
        INT     10H
        MOV     AH,2            ;设置光标位置
        MOV     DL,8
        INT     10H
        MOV     DL,2BH          ;输出+号
        MOV     AH,2
        INT     21H
        CALL    SUB1            ;调 SUB1 非压缩 BCD 码接收子程序
        POP     CX              ;取加数位数
        LEA     BX,NUMBER2      ;取加数参数表地址
AA4:    POP     AX              ;取加数个位、十位、百位…
        MOV     [BX],AL         ;加数存入参数表
        INC     BX              ;形成下位地址
        LOOP    AA4
        MOV     AH,3            ;查找当前光标位置
        INT     10H
        MOV     AH,2            ;设置光标位置
        MOV     DL,17
        INT     10H
        MOV     DL,3DH          ;输出=号
        MOV     AH,2
        INT     21H
        LEA     SI,NUMBER1      ;取被加数参数表地址
        LEA     DI,NUMBER2      ;取加数参数表地址
        LEA     BX,NUMBER3      ;取和参数表地址
        SUB     CX,CX           ;0→CF
        MOV     CX,8            ;加位数计数器初值
AA5:    MOV     AL,[SI]         ;取被加数
        ADC     AL,[DI]         ;非压缩 BCD 码加
        AAA
        MOV     [BX],AX         ;将和存到 NUMBER3 中
        INC     SI              ;形成下一地址
        INC     DI              ;形成下一地址
        INC     BX              ;形成下一地址
        LOOP    AA5
        ADC     CL,CL           ;最高位送 CL
        MOV     [BX],CL         ;存最高位
        LEA     AX,NUMBER3+8    ;取和参数表最高位地址
        PUSH    AX              ;向子程序提供和参数表最高位地址
        CALL    SUB2            ;非压缩 BCD 码显示子程序
        MOV     CX,16
        LEA     BX,NUMBER1      ;取被加数参数表地址
        XOR     AL,AL
QQQ2:   MOV     [BX],AL
        INC     BX
        LOOP    QQQ2
```

```
        MOV     AH,4CH          ;返回 DOS
        INT     21H
SUB1    PROC    NEAR            ;非压缩 BCD 码数据输入子程序
        POP     BX              ;保存返回地址
        SUB     CX,CX           ;键入位数计数器清 0
AA1:    MOV     AH,1
        INT     21H
        CMP     AL,30H          ;判小于 0 键返主程序
        JC      AA2
        CMP     AL,3AH          ;判大于 9 键返主程序
        JNC     AA2
        INC     CX              ;输入数据位数
        PUSH    AX              ;非压缩 BCD 码压栈
        JMP     AA1
AA2:    PUSH    CX              ;输入数据位数压栈
        PUSH    BX              ;返回地址压栈
        RET                     ;返回主函数
SUB1    ENDP
SUB2    PROC    NEAR            ;非压缩 BCD 码显示子程序
        POP     AX              ;保存返回地址
        POP     BX              ;取和参数表最高位地址
        PUSH    AX              ;返回地址压栈
        MOV     CX,9
AA7:    MOV     AL,[BX]         ;消除头部的无效数字 0
        CMP     AL,0
        JNZ     AA6
        DEC     CX
        DEC     BX
        JMP     AA7
AA6:    MOV     DL,[BX]         ;取和最高位、次高位…个位
        ADD     DL,30H          ;形成 ASCII 码
        MOV     AH,2
        INT     21H
        DEC     BX
        LOOP    AA6
        RET
SUB2    ENDP
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

9. 思考题

说明调用指令 CALL 和无条件转移指令的区别及应用。

二、 加强性实验

1. 实验目的

进一步掌握主程序和子程序之间的调用关系及其调用方法。

2. 实验内容

编写程序求数据区中 10 个无符号数中的最大值和最小值，结果分别存入 MAX 和 MIN 单元中，要求最大值和最小值分别用子程序完成计算，主程序和子程序之间通过寄存器传递参数。

3. 实验仪器设备

(1) PC微机。

(2) DOS操作系统或Windows操作系统。

(3) MASM.EXE，LINK.EXE，DEBUG.COM或宏汇编集成环境。

4. 实验步骤

(1) 进入调试软件环境，输入源程序。

(2) 汇编源程序。

(3) 用单步方式运行程序。

(4) 检查并记录各寄存器和存储单元内容的变化。

5. 流程图

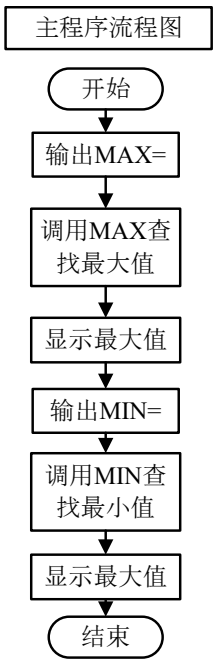


图 5.5 主程序流程图

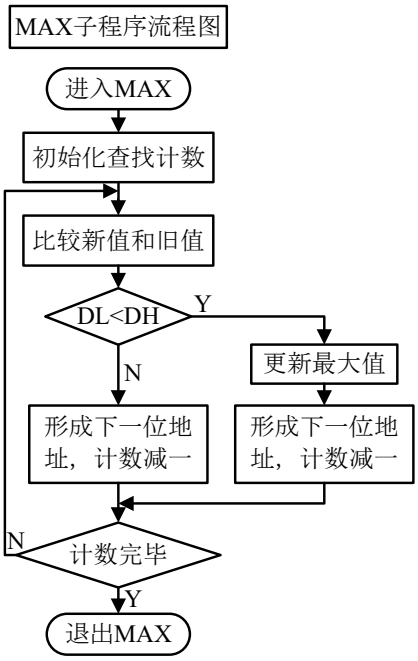


图 5.6 MAX 子程序流程图

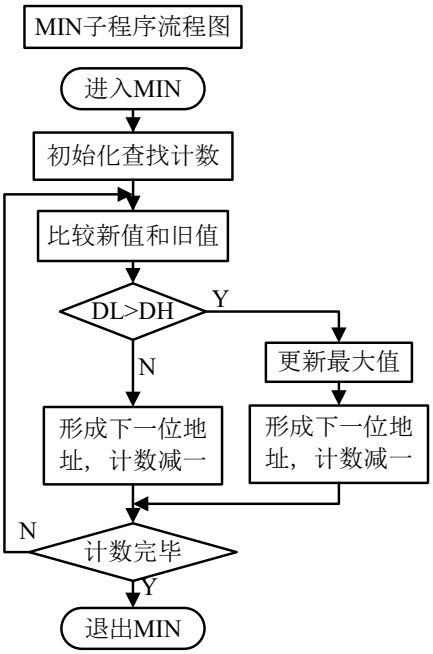


图 5.7 MIN 子程序流程图

6. 参考代码

```
DATA    SEGMENT
ARRAY DB  13,24,92,42,25,46,75,81,53,10                    ;
MA      DB  ' MAX = ','$'
MI      DB  0DH,0AH,' MIN = $'
DATA    ENDS
CODE    SEGMENT
        ASSUME      CS:CODE,DS:DATA
MAIN    PROC    FAR                     ;主程序定义为 FAR 类型
```

```
        MOV     AX,DATA
        MOV     DS,AX
        LEA     DX,MA
        MOV     AH,9                ;输出 MAX=
        INT     21H
        LEA     BX,ARRAY
        CALL    MAX                 ;调用 MAX 字程序查找最大数值
        CMP     DL,10               ;是否为两位数
        JC      AA6
        MOV     DH,30H
AA5:    INC     DH                  ;十位数加一
        SUB     DL,10               ;模十
        CMP     DL,10
        JNC     AA5
        PUSH    DX                  ;保存两位数
        MOV     DL,DH               ;显示十位数
        MOV     AH,2
        INT     21H
        POP     DX                  ;恢复两位数
AA6:    ADD     DL,30H              ;显示个位数
        MOV     AH,2
        INT     21H
        LEA     DX,MI               ;输出“回车  MIN=”
        MOV     AH,9
        INT     21H
        LEA     BX,ARRAY
        CALL    MIN                 ;调用 MIN 查找最小数值
        CMP     DL,10
        JC      AA8
        MOV     DH,30H
AA7:    INC     DH
        SUB     DL,10
        CMP     DL,10
        JNC     AA7
        PUSH    DX
        MOV     DL,DH
        MOV     AH,2                ;输出十位数
        INT     21H
        POP     DX
AA8:    ADD     DL,30H              ;输出个位数
        MOV     AH,2
        INT     1H
        MOV     AH,4CH              ;返回 DOS
        INT     21H
MAX     PROC    NEAR                ;最大值查找子程序
        MOV     CX,10               ;查找计数
        MOV     DL,[BX]             ;初始化 DL，以便后面冒泡查找
AA1:    MOV     DH,[BX]
        CMP     DL,DH
        JC      AA2
        INC     BX                  ;形成下一查找位置
```

```
        DEC     CX              ;查找计数减一
        CMP     CX,0
        JZ      OUT1
        JMP     AA1
AA2:    MOV     DL,DH
        INC     BX
        DEC     CX
        CMP     CX,0
        JZ      OUT1
        JMP     AA1
OUT1:   RET                     ;返回主函数
MAX     ENDP
MIN     PROC    NEAR            ;最小值查找子程序，查找过程与 MAX 函数类似
        MOV     CX,10
        MOV     DL,[BX]
AA3:    MOV     DH,[BX]
        CMP     DL,DH
        JNC     AA4
        INC     BX
        DEC     CX
        CMP     CX,0
        JZ      OUT2
        JMP     AA3
AA4:    MOV     DL,DH
        INC     BX
        DEC     CX
        CMP     CX,0
        JZ      OUT1
        JMP     AA3
OUT2:   RET
MIN     ENDP
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

实验六　子程序设计（二）

一、 基础性实验

1. 实验目的

(1) 强化主程序和子程序之间的调用关系及其调用方法。

(2) 掌握嵌套子程序的设计以及调用方法。

(3) 掌握递归调用子程序的方法。

2. 实验内容

试编制在 ARRAY1 无符号数组中，选出最大值及其所在的位置，然后以 10 进制数形式显示在 CRT 上的源程序。

3. 实验仪器设备

(1) PC微机。

(2) DOS操作系统或Windows操作系统。

(3) MASM．EXE，LINK．EXE，DEBUG．COM或宏汇编集成环境。

4. 实验原理

(1) 子程序的嵌套

子程序的嵌套调用就是一个子程序作为调用程序调用另一个子程序，此时需要注意：

- 正确使用CALL和RET
- 寄存器的保存与恢复
- 堆栈的平衡使用

(2) 子程序递归

在子程序嵌套的情况下，如果一个子程序调用的子程序就是它本身，称为子程序的递归调用。要注意：

- 保证每次调用都不破坏以前调用时所用的参数；
- 堆栈的正确使用；
- 基数的设置；
- 条件转移指令实现递归退出。

5. 实验步骤

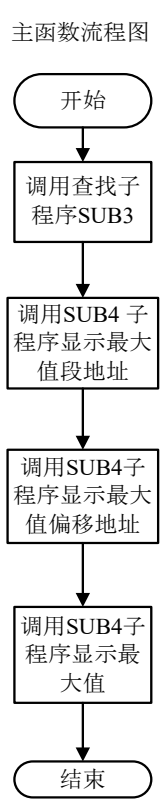
(1) 进入调试软件环境，输入源程序。

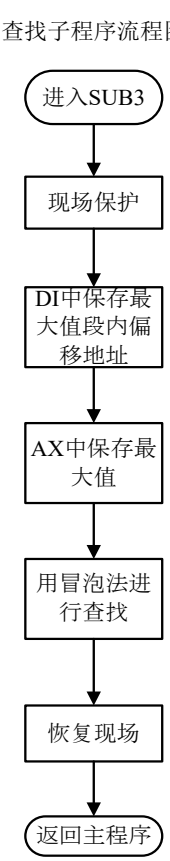
(2) 汇编源程序。

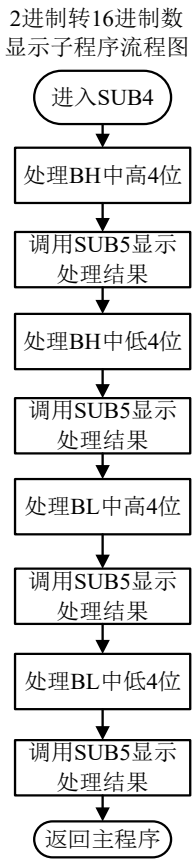
(3) 用单步方式运行程序。

(4) 检查并记录各寄存器和存储单元内容的变化。

6. 流程图







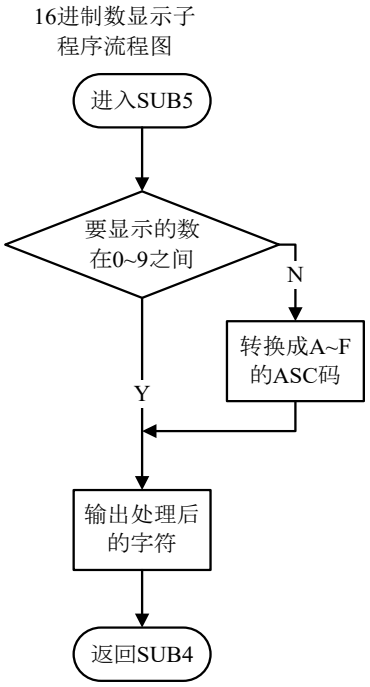


图 6.1 主程序流程图　图 6.2 SUB3 流程图　图 6.3 SUB4 流程图　图 6.4 SUB5 流程图

7. 参考代码

```
DATA    SEGMENT
ARRAY1 DW 5    DUP(5    DUP(212FH,5A5DH,1234H,7865H))        ;待查找数组
EOD         DW 0
DATA    ENDS
CODE    SEGMENT
        ASSUME     CS:CODE,DS:DATA
MAIN    PROC    FAR                    ;主函数定义为远程调用
        MOV     AX,DATA
        MOV     DS,AX
        LEA     SI,ARRAY1
        MOV     CX,(EOD-ARRAY1)/2  ;计算数组个数
        CALL    SUB3                   ;调用查找子程序 SUB3
        MOV     DI,CX                  ;最大元素所在的段地址
        MOV     BX,DATA
        CALL    SUB4                   ;调用显示子程序，显示最大元素的段地址
        MOV     DL,':'
        MOV     AH,2
```

```
        INT     21H
        MOV     BX,SI           ;最大元素的段内偏移
        CALL    SUB4            ;调用显示子程序，显示最大元素的段内偏移
        MOV     DL,20H          ;显示空格
        MOV     AH,2
        INT     21H
        MOV     BX,DI           ;传出数组中的最大元素的值
        CALL    SUB4            ;显示查找到最大值
        MOV     AH,4CH
        INT     21H
SUB3    PROC    NEAR            ;查找子程序，查找数组中最大元素及其位置
        PUSH    AX              ;相关寄存器保护
        PUSH    DI
        MOV     DI,SI           ;DI 中保存最大值的段内地址
        MOV     AX,[SI]         ;AX 中保存最大值
AA1:    CMP     AX,[SI]
        JNC     AA2
        MOV     AX,[SI]
        MOV     DI,SI
AA2:    ADD     SI,2
        LOOP    AA1
        MOV     SI,DI           ;最大元素段内地址通过 SI 传出
        MOV     CX,AX           ;最大元素的值通过 CX 传出
        POP     DI
        POP     AX
        RET
SUB3    ENDP
SUB4    PROC    NEAR            ;十六进制数显示子程序
        MOV     DL,BH           ;处理 BX 中高 8 位
        MOV     CL,4
        SHR     DL,CL
        CALL    SUB5            ;调用十六进制数显示子程序
        MOV     DL,BH
        AND     DL,0FH
        CALL    SUB5
        MOV     DL,BL           ;处理 BX 中低 8 位
        SHR     DL,CL
        CALL    SUB5
        MOV     DL,BL
        AND     DL,0FH
        CALL    SUB5
        RET
SUB4    ENDP
SUB5    PROC    NEAR            ;十六进制数显示子程序
        OR      DL,30H
        CMP     DL,3AH
        JC      AA3
        ADD     DL,7
AA3:    MOV     AH,2
        INT     21H
        RET
```

```
SUB5    ENDP
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

二、 加强性实验

1. 实验目的

进一步掌握主程序和子程序之间的调用关系及其调用方法。

2. 实验内容

用子程序结构编程计算：S=1!+2!+3!+……+5!

3. 实验仪器设备

(1) PC 微机。

(2) DOS 操作系统或 Windows 操作系统。

(3) MASM.EXE，LINK.EXE，DEBUG.COM 或宏汇编集成环境。

4. 实验步骤

(1) 进入调试软件环境，输入源程序。

(2) 汇编源程序。

(3) 用单步方式运行程序。

(4) 检查并记录各寄存器和存储单元内容的变化。

5. 流程图

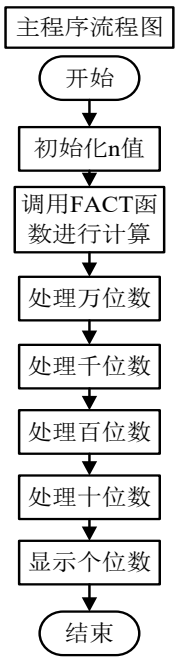


图 6.5 主程序流程图

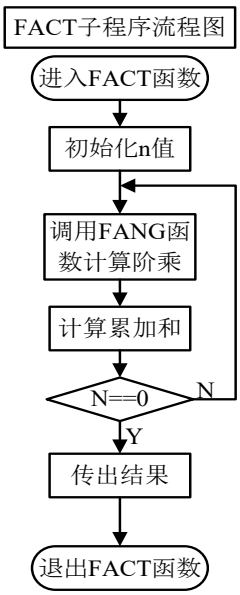


图 6.6 FACT 子程序流程图

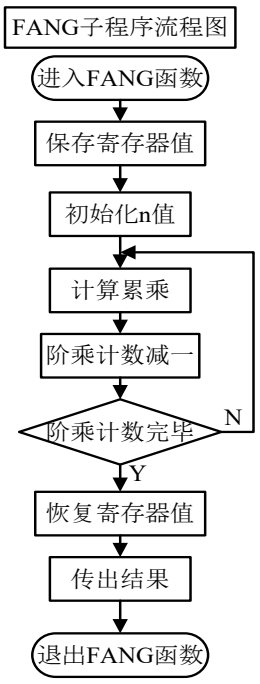


图 6.7 FANG 子程序流程图

6. 参考代码

```
DATA    SEGMENT
RESULT DW 0
DATA    ENDS
CODE    SEGMENT
        ASSUME      DS:DATA,CS:CODE
MAIN    PROC    FAR
        MOV     AX,DATA
        MOV     DS,AX
        MOV     CX,5
        CALL    FACT                ;调用 FACT 函数计算
        CMP     CX,10000            ;显示结果,处理万位,结果小于 1 万则跳过
        JC      BB1
        MOV     DL,30H
AA1:    INC     DL                  ;万位计数
        SUB     CX,10000            ;模 10000
        CMP     CX,10000
        JNC     AA1
        MOV     AH,2
        INT     21H
BB1:    CMP     CX,1000
        JC      BB2
        MOV     DL,30H
AA2:    INC     DL                  ;千位计数
        SUB     CX,1000             ;模 1000
        CMP     CX,1000
        JNC     AA2
        MOV     AH,2
        INT     21H
BB2:    CMP     CX,100
        JC      BB3
        MOV     DL,30H
AA3:    INC     DL                  ;百位计数
        SUB     CX,100              ;模 100
        CMP     CX,100
        JNC     AA3
        MOV     AH,2
        INT     21H
BB3:    CMP     CX,10
        JC      BB4
        MOV     DL,30H
AA4:    INC     DL                  ;十位计数
        SUB     CX,10               ;模 10
        CMP     CX,10
        JNC     AA4
        MOV     AH,2
        INT     21H
BB4:    MOV     DL,CL               ;个位
        ADD     DL,30H
        MOV     AH,2
        INT     21H
```

```
        MOV     AH,4CH
        INT     21H
FACT    PROC    NEAR                    ;处理 1!+2!+3!+4!+5!子程序
        AND     CX,0FFH
        MOV     AX,0
LOP2:   CALL    FANG
        ADD     AX,DX
        DEC     CX
        CMP     CX,0
        JNZ     LOP2
        MOV     CX,AX
        RET
FACT    ENDP
FANG    PROC    NEAR
        PUSH    CX
        PUSH    AX
        AND     CX,0FFH
        MOV     AX,1
LOP1:   MUL     CX
        DEC     CX
        CMP     CX,0
        JNZ     LOP1
        MOV     DX,AX
        POP     AX
        POP     CX
        RET
FANG    ENDP
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

实验七　字符串处理程序设计

一、 基础性实验

1. 实验目的

(1) 掌握串操作指令的寻址方式及使用方法。

(2) 编写常用的字符串处理程序。

2. 实验内容：

用 1 号系统调用从键盘键入一个字符串，然后统计其中小写字母的个数。

3. 实验仪器设备

(1) 硬件环境：IBM-PC及其兼容机；软件环境：操作系统DOS3.0以上。

(2) 编辑程序：EDIT或其它编辑程序。

(3) 汇编程序：MASM.EXE。

(4) 连接程序：LINK.EXE。

(5) 调试程序：DEBUG.COM。

4. 实验步骤

(1) 建立汇编语言程序源文件。可使用任何一个文本编辑器（如EDIT.com）编辑源文件，源文件的扩展名通常取为.ASM。

(2) 汇编文件，形成目标模块。

(3) 若文件有语法错误，据错误提示，返回（1），修改源文件。若无错，则形成.OBJ目标文件和.LST列表文件。

(4) 连接目标模块，形成可执行文件。若有错，据错误提示，返回（1），修改错误。若无错，则形成.EXE可执行文件和.MAP映像文件。

(5) DOS下直接键入可执行文件名，即可执行该程序。

5. 实验提示：

输入输出设备以 ASCII 码表示字符，数通常是用十进制数表示，而机器内部以二进制表示。所示，在 CPU 与 I/O 设备之间必须要进行码的转换。下面给出常用的码制处理子程序集。

(1) 二进制的输入(0-1111111111111111)

```
PUTINBIN    PROC
            PUSH    CX
            MOV     BX,0
        P1: MOV     AH,1
            INT     21H
```

```
            CMP     AL,0DH
            JE      BINEXIT
            SUB     AL,30H
            MOV     AH,0
            XCHG    AX,BX
            MOV     CX,2
            MUL     CX
            XCHG    AX,BX
            ADD     BX,AX
            JMP P1
BINEXIT:POP     CX
            RET
PUTINBIN        ENDP
```

(2) 十进制的输入(0-65535)

```
PUTINDEC    PROC
            PUSH    CX
            MOV     BX,0
        P1: MOV     AH,1
            INT     21H
            CMP     AL,0DH
            JE      DECEXIT
            SUB     AL,30H
            MOV     AH,0
            XCHG    AX,BX
            MOV     CX,10
            MUL     CX
            XCHG    AX,BX
            ADD     BX,AX
            JMP P1
DECEXIT:POP     CX
            RET
PUTINDEC        ENDP
```

(3) 十六进制的输入(0-ffff)

```
PUTINHEX    PROC
            PUSH    CX
            MOV     BX,0
        P1: MOV     AH,1
            INT     21H
            CMP     AL,0DH
            JE      HEXEXIT
            SUB     AL,30H
            CMP     AL,10
            JB      OOO1
            SUB     AL,27H
OOO1:       MOV     CL,4
            SHL     BX,CL
            MOV     AH,0
            ADD     BX,AX
            JMP P1
HEXEXIT:POP     CX
```

```
        RET
PUTINHEX    ENDP
```

(4) 二进制的输出(0-1111111111111111)

```
PUTOUTBIN   PROC
        PUSH    CX
        MOV     NOZERO,0
        MOV     CL,16
    PP2:MOV     DL,0
        TEST    BX,8000H
        JE      PP
        MOV     NOZERO,1
        MOV     DL,1
    PP: CMP     NOZERO,0
        JE      PP1
        ADD     DL,30H
        MOV     AH,2
        INT     21H
    PP1:SHL     BX,1
        LOOP    PP2
        POP     CX
        RET
PUTOUTBIN   ENDP
```

(5) 十进制的输出(0-65535)

```
PUTOUTDEC   PROC
        PUSH    CX
        MOV     NOZERO,0
        MOV     CX,10000
        CALL    MAIN
        MOV     CX,1000
        CALL    MAIN
        MOV     CX,100
        CALL    MAIN
        MOV     CX,10
        CALL    MAIN
        MOV     CX,1
        CALL    MAIN
        CMP     NOZERO,0
        JNE     LL
        MOV     DL,30H
        MOV     AH,2
        INT     21H
    LL: POP     CX
        RET
        MAIN    PROC
        MOV     AX,BX
        MOV     DX,0
        DIV     CX
        MOV     BX,DX
        MOV     DL,AL
        CMP     DL,0
```

```
        JNE     LL1
        CMP     NOZERO,0
        JNE     LL1
        JMP     LL2
    LL1:MOV     NOZERO,1
        ADD     DL,30H
        MOV     AH,2
        INT     21H
    LL2:RET
        MAIN    ENDP
PUTOUTDECENDP
```

(6) 十六进制的输出(0-ffff)

```
PUTOUTHEXPROC
        PUSH    CX
        MOV     TEMP,BX
        MOV     TEMP1,BX
        MOV     CL,8
        SHR     TEMP,CL
        MOV     BX,TEMP
        CALL    MAIN1
        AND     TEMP1,00FFH
        MOV     BX,TEMP1
        CALL    MAIN1
        POP     CX
        RET
        MAIN1   PROC
        MOV     AL,BL
        MOV     CL,4
        SHR     AL,CL
        CMP     AL,10
        JB      KK
        ADD     AL,7
    KK: MOV     DL,AL
        CMP     DL,0
        JE      KK2
        ADD     DL,30H
        MOV     AH,2
        INT     21H
        AND     BL,0FH
    KK2:CMP     BL,10
        JB      KK1
        ADD     BL,7
    KK1:MOV     DL,BL
        ADD     DL,30H
        MOV     AH,2
        INT     21H
        RET
        MAIN1   ENDP
PUTOUTHEX       ENDP
```

(7) ASCII 码转二进制码程序

```
DATA    SEGMENT
KEY_BUF    DB  10
           DB  ?
           DB  10   DUP(?)
DATA1      DW ?
DATA       ENDS
CODE    SEGMENT
           ASSUME      CS:CODE,DS:DATA
START:  MOV     AX,DATA
        MOV     DS,AX
        MOV     DX,OFFSET   KEY_BUF
        MOV     AH,0AH
        INT     21H                     ;接收键盘输入
        MOV     SI,OFFSET   KEY_BUF+1
        MOV     BL,[SI]                 ;输入字符个数
        INC     SI                      ;SI 指向输入字符的数据区
        MOV     AX,0                    ;设置初值 0
TRANS:  MOV     DX,10
        MUL     DX                      ;乘 10
        MOV     DH,0
        MOV     DL,[SI]
        AND     DL,0FH                  ;ASCⅡ码转换成二进制码
        ADD     AX,DX                   ;相加
        INC     SI
        DEC     BL
        JNZ     TRANS                   ;还有数字?
        MOV     DATA1,AX                ;转换后的结果
        CALL    OUTPUT
        MOV     AX,4C00H
        INT     21H                     ;返回 DOS
;.....    输出子程序   .....
OUTPUTPROC    NEAR
        PUSH    AX
        MOV     DL,0AH
        MOV     AH,2
        INT     21H
        MOV     DL,0DH
        INT     21H
        POP     AX
        MOV     BH,10H
        MOV     CX,AX
OUT1:   MOV     DL,CH
        AND     DL,80H
        ROL     DL,1
        ADD     DL,30H
        MOV     AH,2
        INT     21H
        DEC     BH
        CMP     BH,0
        JZ      EDED
        ROL     CX,1
        JMP     OUT1
```

```
EDED:   RET
OUTPUT      ENDP
;输出子程序结束  .....
CODE    ENDS
        END     START
```

6. 流程图

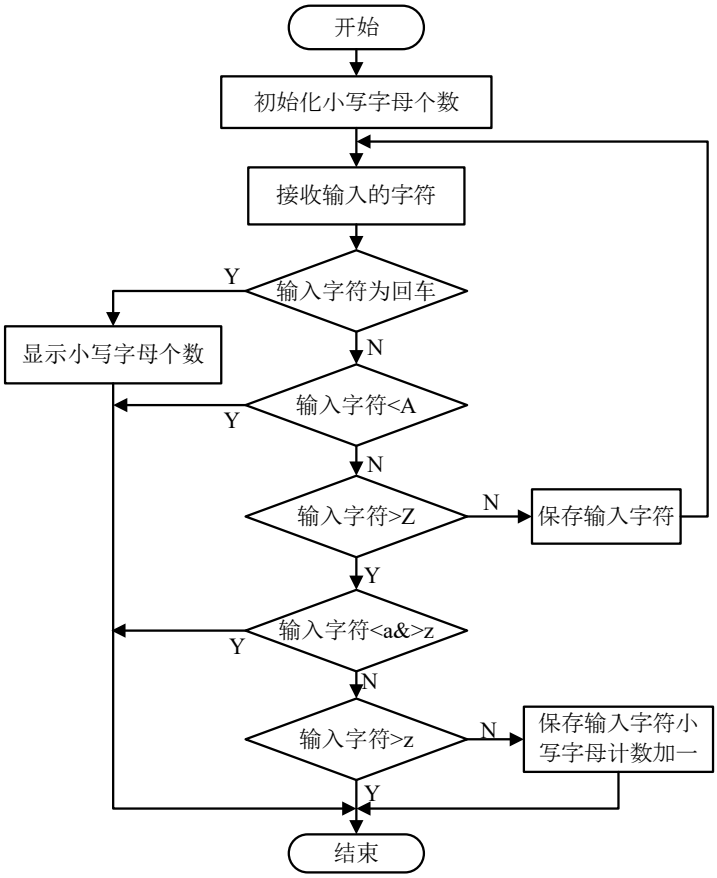


图 7.1 程序流程图

7. 参考代码

```
DATA    SEGMENT
STRING DB  100  DUP(0)              ;用来存放输入的字符串
DATA    ENDS
CODE    SEGMENT
        ASSUME      CS:CODE,DS:DATA
MAIN    PROC    FAR
        MOV     AX,DATA
        MOV     DS,AX
        LEA     BX,STRING
        MOV     CL,0                ;小写字母数
AA1:    MOV     AH,1                ;输入字符串
        INT     21H
        CMP     AL,0DH              ;当输入回车时，计算小写字母个数并输出
        JZ      AA3
        CMP     AL,41H              ;小于 A 的退出
```

```
        JC      AA4
        CMP     AL,5BH          ;大于 Z 的
        JNC     AA2
        MOV     [BX],AL
        INC     BX
        JMP     AA1
AA2:    CMP     AL,61H          ;小于 a 而大于 Z 的退出
        JC      AA4
        CMP     AL,7BH          ;大于 z 的退出
        JNC     AA4
        MOV     [BX],AL         ;将输入合法字符存入内存中
        INC     BX              ;形成下一存放地址
        INC     CL              ;小写字母数量加一
        JMP     AA1             ;继续输入字符
AA3:    PUSH    CX              ;保存小写字母个数值
        MOV     AH,3            ;查找当前光标位置
        INT     10H
        MOV     AH,2            ;设置光标位置
        LEA     DI,STRING
        SUB     BX,DI           ;计算光标列位置
        MOV     DL,BL           ;设置光标列位置
        INC     DL              ;光标向后移一位
        INT     10H
        POP     CX
        MOV     DL,CL
        ADD     DL,30H          ;将字母个数从十进制转换成十六进制
        MOV     AH,2
        INT     21H
AA4:    MOV     AH,4CH
        INT     21H
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

8. 思考题

本次实验中，出现了哪些错误？如何排除？

二、 加强性实验

1. 实验目的

进一步掌握串操作指令的寻址方式及编程方法。

2. 实验内容：

编写程序，比较两个字符串 STRING1 和 STRING2 所含字符是否完全相同，若相同则显示“YES”，若不同则显示“NO”。

3. 实验仪器设备

(1) PC 微机。

(2) DOS 操作系统或 Windows 操作系统。

(3) MASM.EXE，LINK.EXE，DEBUG.COM 或宏汇编集成环境。

4. 实验步骤

(1) 用文字编辑工具（记事本或 EDIT）将源程序输入，其扩展名为.ASM。

(2) 用 MASM 对源文件进行汇编，产生.OBJ 文件和.LST 文件。若汇编时提示有错，用文字编辑工具修改源程序后重新汇编，直至通过。

(3) 用 TYPE 命令显示 1 产生的.LST 文件。

(4) 用 LINK 将．OBJ 文件连接成可执行的.EXE 文件。

(5) 在 DOS 状态下运行 LINK 产生的.EXE 文件。

5. 流程图

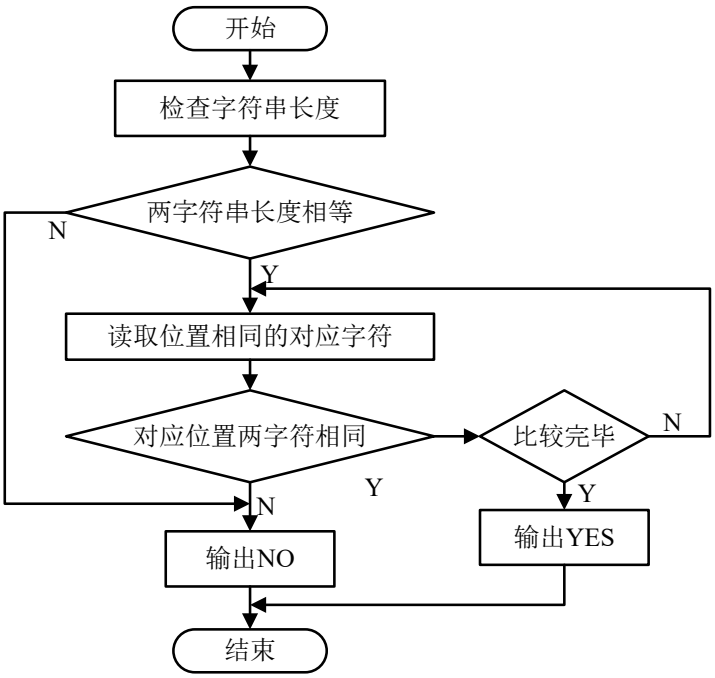


图 7.2 程序流程图

6. 参考代码

```
DATA    SEGMENT
STRING1     DB  'ABCDEFG'                 ;用来存放输入的字符串
STRING2     DB  'ABCDEFG'
YES         DB  'YES$'
NO          DB  'NO$'
DATA    ENDS
CODE    SEGMENT
        ASSUME      CS:CODE,DS:DATA
MAIN    PROC    FAR
        MOV     AX,DATA
        MOV     DS,AX
```

```
        MOV     CX,STRING2-STRING1
        MOV     BX,YES-STRING2
        CMP     CX,BX
        JNZ     AA2                     ;输出 NO
        LEA     BX,STRING1
        LEA     SI,STRING2
AA1:    MOV     AL,[BX]
        MOV     AH,[SI]
        CMP     AL,AH
        JNZ     AA2
        INC     BX
        INC     SI
        LOOP    AA1
        MOV     AH,9
        LEA     DX,YES
        INT     21H
        MOV     AH,4CH
        INT     21H
AA2:    MOV     AH,9
        LEA     DX,NO
        INT     21H
        MOV     AH,4CH
        INT     21H
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

7. 思考题

编写程序，在一个字符串中查找匹配字符。

实验八　BIOS 调用程序设计

一、基础性实验

1.　实验目的

(1)　掌握在 DOS 中对屏幕和键盘中断的调用方法。

(2)　掌握在 BIOS 中对屏幕和键盘中断的调用方法。

2.　实验内容

在屏幕上显示背景色为白色，前景色不同的 16 个字符“A”。

3.　实验仪器设备

(1)　硬件环境：IBM-PC及其兼容机；软件环境：操作系统DOS3.0以上。

(2)　编辑程序：EDIT或其它编辑程序。

(3)　汇编程序：MASM.EXE 6.1X。

(4)　连接程序：LINK.EXE。

(5)　调试程序：DEBUG.COM。

4.　实验原理

(1)　BIOS/DOS 简介

固化在 ROM 中的基本输入输出系统 BIOS 包含了主要 I/O 设备的处理程序和许多常用例行程序，它们一般以中断处理程序的形式存在。

磁盘操作系统 DOS 建立在 BIOS 的基础上，通过 BIOS 操纵控制硬件。DOS 调用 BIOS 显示输出，调用 BIOS 打印输出程序完成打印输出，调用 BIOS 键盘输入程序完成键盘输入。DOS、BIOS 和硬件接口都为应用程序提供完成输入输出的功能，而且随着层次的加深访问外部设备的能力越强；从应用程序的编写角度出发，随着层次的加深，应用程序的编写难度和复杂度大大增加。

通常 I/O 程序应该由 DOS 提供的系统功能，完成输入输出，这样实现容易，而且对硬件的依赖性最少。如果 DOS 不提供某种服务或者不能使用 DOS 的场合可考虑 BIOS 调用。应用程序可以直接操纵外设接口来控制外设，从而获得速度上的最要效率。

(2)　DOS 调用方法

DOS 功能调用的子程序已按顺序编号——功能号（00H～68H），其调用方法是：

- 功能号→AH
- 入口参数→指定寄存器

- INT 21H

用户只须给出以上三方面信息，DOS 就可根据所给信息自动转入相关子程序执行。

(3) DOS 调用注意事项

①INT 21H，不要写成 INT 21

② 可以简单地将“INT 21H”看成一个过程（函数），它提供了多个子功能，每个子功能都有一个编号，分别是 0 号功能、1 号功能、2 号功能、…

③ 调用“INT 21H”过程时，需要在 AH 中设定子功能号。

④ 每次执行“INT 21H”调用，只能执行一个子功能。

```
MOV   DX, OFFSET STRING
MOV   AX, 4C00H
INT   21H
```

这个语句序列只能返回到操作系统，而不能在显示字符串的同时返回。

⑤ 每个子功能又都分别有自己的调用约定，调用时应该按照该子功能的约定设定调用参数。

⑥常用的几个子功能

- 4C 号子功能：程序退出，返回到操作系统

```
MOV   AL, 程序返回代码（常常为 0，表示正常结束）
MOV   AH, 4CH
INT   21H
```

- 1 号子功能：键盘输入带回显

调用约定：

```
MOV   AH, 1
INT   21H
```

输出：AL 中存储了从键盘输入字符的 ASCII 码

- 8 号子功能：键盘输入无回显

调用约定：同 1 号子功能。

- 2 号子功能：显示一个字符

调用约定：输出字符的 ASCII 码预先放置到 DL 中

```
MOV   DL, XX; XX 是预输出字符的 ASCII 码
MOV   AH, 2
INT   21H
```

- 9 号子功能：显示一个以字符'$'结尾的字符串

调用约定：字符串的首地址预先放置到 DS:DX 中

```
LEA   DX, STRING;
```

或者：MOV　DX, OFFSET STRING

MOV　AH,　9

INT　　21H

(4)　BIOS 中断调用方法

使用 BIOS 中断调用与 DOS 系统功能调用类似，用户也无须了解相关设备的结构与组成细节，直接调用即可。

用户在汇编语言程序中可使用软中断指令“INT n”调用 BIOS 程序，其中 n 是中断类型码。常用的 BIOS 程序的功能与其中断类型码对应关系如表 8-1。

表 8-1 中断类型码表

中断类型码	BIOS 中断调用功能
10H	显示器 I/O 中断调用（即显示器驱动程序）
16H	键盘驱动程序
17H	打印机驱动程序
13H	磁盘驱动程序
14H	通信驱动程序

例如，用指令“INT 16H”可调用键盘驱动程序。当某个 BIOS 程序中具有多种不同功能时，用不同的编号——功能号加以区分，并约定功能号存放在寄存器 AH 中。其调用方法与 DOS 功能调用类似：

①　功能号→AH。

②　入口参数→指定寄存器。

③　指令“INT n”实现对 BIOS 子程序的调用

5.　实验步骤

(1)　进入调试软件环境，输入源程序。

(2)　汇编源程序。

(3)　用单步方式运行程序。

(4)　检查并记录各寄存器和存储单元内容的变化。

6.　实验提示

(1)　256 个 ASCII 字符中的大多数是通过符号来表示的，这些符号能在视频屏幕上显示，而对于某些值，比如 00H 和 FFH 没有可显示的符号，所以显现为空白，虽然实际上的 ASCII 空白符是 20H。

(2)　为在屏幕上显示字符，可使用 INT 21H 中的 2 号调用和 INT 10H 中的调用实现。

(3)　用 INT 10h 调用 2 号功能设置光标位置，入口参数有 bh,dh,dl。显示字符“A”可以用 INT 21h

调用中的 2 号功能，入口参数有 dl。

(4) 要显示带背景色的字符，可使用 INT 10h 中断调用中的 9 号调用，要显示的字符放在 al 中，字符的属性设置在 bl 中。显示字符的个数在 dx 中设置。

7. 流程图：

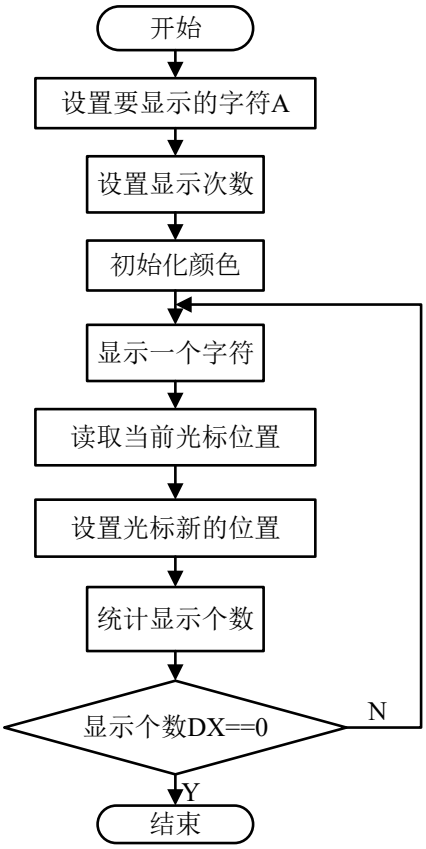


图 8.1 程序流程图

8. 参考代码

```
DATA    SEGMENT
DATA    ENDS
CODE    SEGMENT
        ASSUMECS:CODE,DS:DATA
START:  MOV     AL,'A'              ;要显示的字符
        MOV     DX,10H              ;显示个数为 16
        MOV     BL, 0F0H            ;初始化颜色，前景为黑色，背景为白色
A1:     PUSH    DX
        MOV     AH,9                ;显示带有属性的字符
        MOV     BH,0                ;页号为 0
        MOV     CX,1                ;显示一个字符
        INT     10H
        MOV     AH,3                ;读取光标位置
        MOV     BH,0                ;页数为 0
        INT     10H
```

```
        INC     DL                  ;改变光标的列号
        MOV     AH,2                ;设置光标
        INT     10H
        ADD     BL,1                ;字符属性加 1，改变前景颜色
        POP     DX
        DEC     DX                  ;显示字符个数统计
        JNZ     A1
        MOV     AH,4CH
        INT     21H
CODE    ENDS
        END     START
```

9. 思考题

若要将程序中的字符以斜线方式显示，应该如何修改指令？

二、加强性实验

1. 实验目的

进一步掌握在 BIOS 中对屏幕和键盘中断的调用方法。

2. 实验内容

利用视频显示调用，实现简单的动画,在屏幕上显示一个水平开动的汽车。

3. 实验仪器设备

(1) 硬件环境：IBM-PC及其兼容机；软件环境：操作系统DOS3.0以上。

(2) 编辑程序：EDIT或其它编辑程序。

(3) 汇编程序：MASM.EXE。

(4) 连接程序：LINK.EXE。

(5) 调试程序：DEBUG.COM。

4. 实验步骤

(1) 进入调试软件环境，输入源程序。

(2) 汇编源程序。

(3) 用单步方式运行程序。

(4) 检查并记录各寄存器和存储单元内容的变化。

5. 实验提示

了解动画实现的基本步骤。屏幕上出现动画，实际上是通过显示图形，清除图形的交替过程来实现的。所以，一般动画实现的步骤为下面几步；

(1) 显示图形并延时一段时间。

(2) 清除图形。

(3) 改变新图形显示坐标位置。

(4) 重复(1)。

6. 流程图

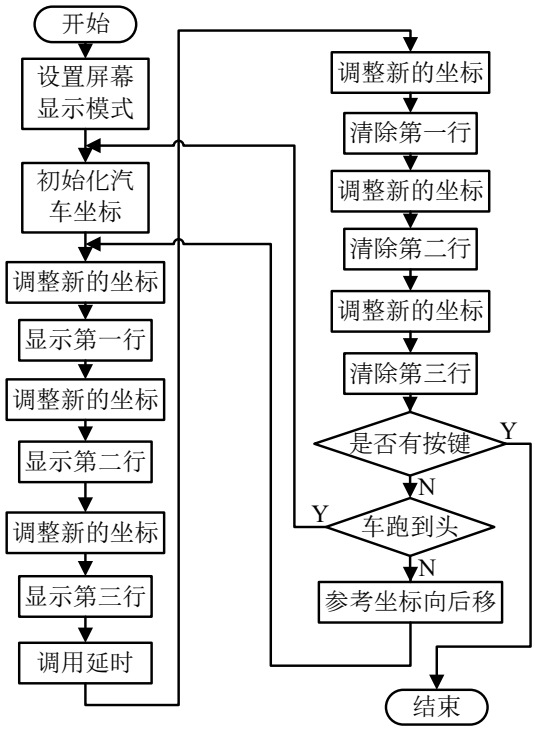


图 8.2 程序流程图

7. 参考代码

```
WR      MACRO CHA,ATR,NUM       ;宏定义,显示处理
        MOV     AH,2            ;设置光标位置
        INT     10H
        MOV     AL,CHA          ;要显示的字符送 AL
        MOV     CX,NUM          ;要显示字符的个数送 CX
        MOV     BL,ATR          ;要显示字符的颜色属性
        MOV     AH,9
        INT     10H
        ENDM
CODE    SEGMENT
        ASSUME      CS:CODE
MAIN    PROC    FAR
CAR:    MOV     AH,0
        MOV     AL,3            ;设置屏幕显示模式为 80*25 彩色字符方式
        INT     10H
LOP1:   MOV     SI,0A0AH        ;定义显示位置(10,10)
LOP2:   MOV     DX,SI
        WR      0DBH,0CH,5      ;显示第一行
        INC     DH              ;调整行坐标
        SUB     DL,2            ;调整列坐标
```

```
        WR      0DBH,4,9            ;显示第二行
        INC     DH                  ;调整行坐标
        INC     DL                  ;调整列坐标
        WR      09H,8EH,1           ;显示前轮
        ADD     DL,6                ;调整列坐标
        WR      9,8EH,1             ;显示后轮
        CALL    DELAY               ;调用延时
        MOV     DX,SI
        WR      0,0,5               ;清除第一行
        INC     DH
        SUB     DL,2
        WR      0,0,9               ;清除第二行
        INC     DH
        WR      0,0,8               ;清除第三行
        MOV     AH,1                ;判断是否有按键
        INT     16H
        JZ      CONU
        MOV     AH,4CH              ;返回 DOS
        INT     21H
JLOP2:  JMP     LOP2
CONU:   INC     SI
        CMP     SI,0A50H            ;判断车是否跑到头
        JB      JLOP2
        JMP     LOP1
MAIN    ENDP
DELAY   PROC    NEAR                ;延时子程序
        MOV     CX,0FFFFH
        LOOP    $
        MOV     CX,0FFFFH
        LOOP    $
        MOV     CX,0FFFFH
        LOOP    $
        MOV     CX,0FFFFH
        LOOP    $
        MOV     CX,0FFFFH
        LOOP    $
        MOV     CX,0FFFFH
        LOOP    $
        MOV     CX,0FFFFH
        LOOP    $
        RET
DELAY   ENDP
CODE    ENDS
        END     MAIN
```

实验九　中断处理程序设计

一、 基础性实验

1. 实验目的

(1) 学习和了解汇编语言程序设计中有关中断处理的相关知识。

(2) 了解汇编语言程序设计中软中断的基本编写步骤。

2. 实验内容

(1) 编写一个中断处理程序，当主程序中除法指令除数为0时，显示字符串“ERROR!DIVIDE BY ZERO!”，并将除数置为3。

(2) 为了验证该中断处理程序是否有效，编写一个主程序，做10次9/BL，在屏幕上显示商，BL中初始数据为6，每除一次，BL中除数减1。

3. 实验仪器设备

(1) PC微机。

(2) DOS操作系统或Windows操作系统。

(3) MASM.EXE，LINK.EXE，DEBUG.COM或宏汇编集成环境。

4. 实验原理

(1) 先为本中断处理程序选择一个中断类型号。中断类型号一共有256个，一般选择系统未使用的中断类型号。但是除数为0是系统中的0号中断，也就是遇到除数为0的指令的时候，就会去执行该系统中已经有的中断处理子程序，所以，为了使我们编写的子程序被执行，也要选择中断类型号为0。

(2) 编写中断处理子程序。若主程序中出现除数为0的情况，则显示提示字符串，并将BL置为3。编写主程序。主程序中需要设置中断向量表，然后编写10次除法运算的程序。

5. 实验步骤

(1) 进入调试软件环境，输入源程序。

(2) 汇编源程序。

(3) 用单步方式运行程序。

(4) 检查并记录各寄存器和存储单元内容的变化。

(5) 将上述主程序与子程序编辑为一个文件，生成.EXE文件后执行，察看结果。是否实现了题目所要求的功能。

6. 流程图

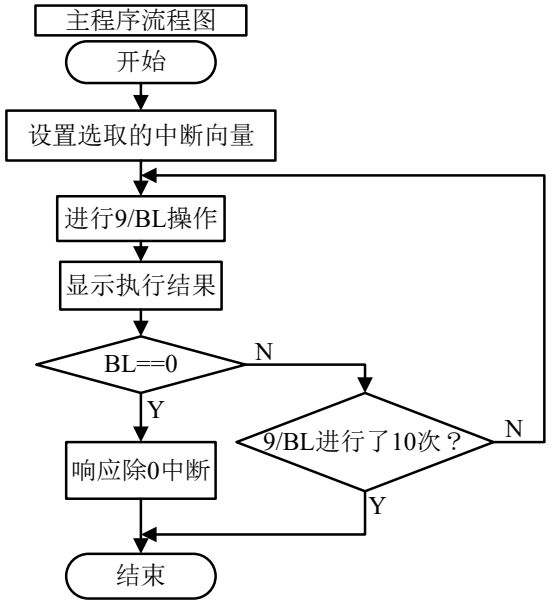


图 9.1 主程序流程图

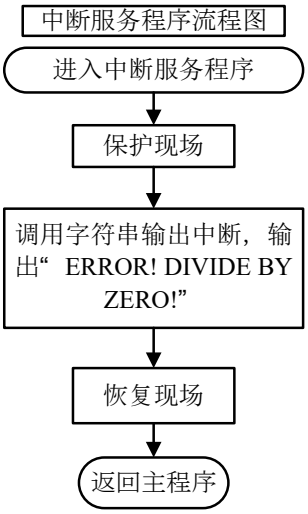


图 9.2 中断服务程序流程图

7. 参考代码

```
DATA    SEGMENT
STRING DB  'ERROR! DIVIDE  BY  ZERO!',0AH,0DH,'$'
CODE    SEGMENT
        ASSUME      CS:CODE,DS:DATA
MAIN    PROC    FAR
        LEA     DX,INT0             ;中断服务程序偏移地址
        MOV     AX,CS
        MOV     DS,AX
        MOV     AL,32               ;选用 32 号软中断
        MOV     AH,25H              ;调用设置中断向量中断
        INT     21H
        MOV     AX,DATA
        MOV     DS,AX
        MOV     CX,10               ;9/BL 做 10 次
        MOV     BL,6                ;BL 初值设为 6
A1:     MOV     AX,9
        DIV     BL
        ADD     AL,30H              ;将十进制数转换成十六进制数
        MOV     DL,AL               ;显示除法结果
        MOV     AH,2
        INT     21H
        MOV     DL,0DH
        MOV     AH,2
        INT     21H
        MOV     DL,0AH
        MOV     AH,2
        INT     21H
        DEC     BL
        CMP     BL,0                ;当除数为 0 时
        JZ      A2
```

```
        LOOP    A1
A3:     MOV     AH,4CH
        INT     21H
A2:     INT     32              ;除数为 0 时进入 32 号软中断
        MOV     AH,4CH
        INT     21H
MAIN    ENDP
INT0    PROC    FAR             ;中断服务程序
        LEA     DX,STRING
        MOV     AH,9            ;调用字符串输出中断
        INT     21H
        MOV     BL,3            ;BL 赋为 3
        IRET                    ;返回主程序
INT0    ENDP
CODE    ENDS
        END     MAIN
```

8. 思考题

(1) 对中断处理子程序做修改，实现当出现除数为0时，显示错误提示，然后结束程序。这时，应该修改程序中的什么地方？

(2) 本次实验中，出现了哪些错误？如何排除？

二、 加强性实验

1. 实验目的

(1) 中断服务子程序的编写。

(2) BIOS 和DOS中断调用。

(3) 实时时钟显示的实现。

2. 实验内容

采用软中断处理程序实现实时时钟显示，每隔 1 秒更新实时显示时间。

3. 实验仪器设备

(1) PC微机。

(2) DOS操作系统或Windows操作系统。

(3) MASM.EXE，LINK.EXE，DEBUG.COM或宏汇编集成环境。

4. 实验原理

(1) 计算机在系统加电期间，把系统定时器初始化为每隔55毫秒发出一次中断请求。CPU在相应定时中断请求后转入08H号中断处理程序。

(2) BIOS提供的08H号中断处理程序中有一条中断指令”INT 1CH”，所以每秒要调用约18．2次1CH号中断处理程序。而BIOS的1CH号中断处理程序实际上并没有作任何工作，只有一条中

断返回指令（IRET）。

(3) 这样安排的目的是为应用程序留下一个软接口，应用程序只要提供新的1CH号中断处理程序，就能实现某些周期性的工作。本实验就是利用这个软接口实现“实时时钟显示”。

5. 实验提示

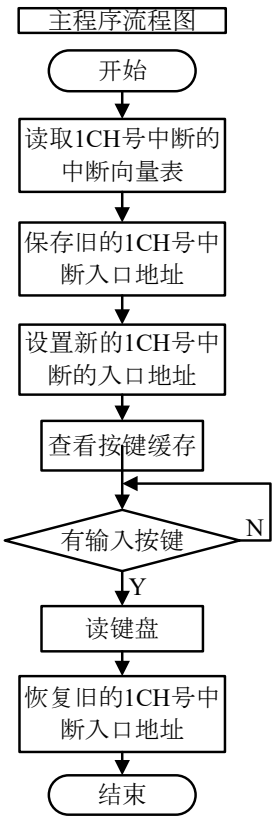
(1) 主程序的功能：

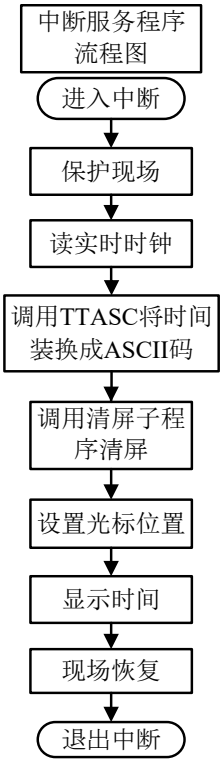
- 保存原 1CH 号的中断向量；
- 设置新的 1CH 号的中断向量；
- 在主程序完成其他工作后，恢复原 1CH 号的中断向量；

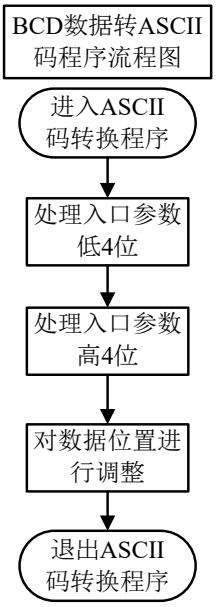
(2) 新的1CH号中断处理程序功能如下：

- 清屏，利用 1AH 号中断处理程序的 2 号功能获取当前时间；
- 在屏幕的右上角现实当前时间；
- 记录调用该中断处理程序的次数，当计数满 18 次后（1 秒到），更新显示时间。

6. 流程图







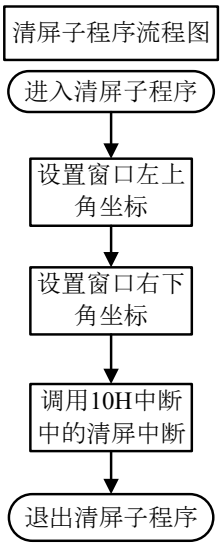


图 9.3 主程序流程图　　图 9.4 中断服务程序流程图　图 9.5 转换程序流程图　图 9.6 清屏程序流程图

7. 参考代码

```
;定义代码段
CSEG    SEGMENT
```

```
        ASSUME    CS:CSEG
START PROC    FAR
        PUSH    CS
        POP     DS
        MOV     AX,351CH
        INT     21H                             ;获取原 1CH 号的中断向量
        MOV     CS:WORD    PTR OLD1C,BX
        MOV     CS:WORD    PTR OLD1C+2,ES ;保存原 1CH 号的中断向量
        MOV     DX,OFFSET  INT1C           ;设置新的 1CH 号的中断向量
        MOV     AX,251CH
        INT     21H                             ;此后，每 55 毫秒就进入一次新的
;1CH 号的中断处理程序
WAITN:MOV     AH,1
        INT     16H                             ;查有无键按下
        JZ      WAITN                           ;转等待键按下
        MOV     AH,0
        INT     16H                             ;读键盘
        LDS     DX,CS:OLD1C
        MOV     AX,251CH
        INT     21H                             ;恢复原 1CH 中断向量
        MOV     AH,4CH
        INT     21H                             ;返回 DOS
START ENDP
;定义数据空间
        OLD1C  DD       ?                       ;保存原中断向量
        COUNT  DW       0                       ;调用 1CH 中断程序的次数
         HHH    DB      ?,?,':'                 ;”时”
         MMM    DB      ?,?,':'                 ;”分”
         SSS    DB      ?,?,'$'                 ;”秒”
;定义新的 1CH 号的中断处理程序
INT1C  PROC    FAR
        CMP     COUNT,0                         ;调用次数为”0”时（1 秒到）
        JZ      NEXT
        DEC     COUNT                           ;显示次数递减（第一次 18-1）
        IRET
NEXT:  MOV     COUNT,18                        ;置显示次数初值
        STI
        PUSH    DS                              ;保护现场
        PUSH    ES
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    DI
        MOV     AH,2
        INT     1AH                             ;读实时时钟
        MOV     AL,CH                           ;时送 AL
        CALL    TTASC                           ;转换成 ASCII 码
        MOV     WORD   PTR HHH,AX               ;保存时
```

```
        MOV     AL,CL                           ;分转换
        CALL    TTASC
        MOV     WORD   PTR MMM,AX
        MOV     AL,DH                           ;秒转换
        CALL    TTASC
        MOV     WORD   PTR SSS,AX
        CALL    CLS                             ;清屏
        MOV     BH,0
        MOV     DX,0140H
        MOV     AH,2
        INT     10H                             ;设置光标（1，65）
        PUSH    CS
        POP     DS
        MOV     DX,OFFSET   HHH
        MOV     AH,9
        INT     21H                             ;显示实时时钟
        POP     DI
        POP     SI
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        POP     ES
        POP     DS
        IRET                                    ;中断返回
INT1C   ENDP
;将 AL 中的 BCD 数据转换成 ASCII 码存入 AX 中
TTASC   PROC
        MOV     AH,AL
        AND     AL,0FH
        SHR     AH,1
        SHR     AH,1
        SHR     AH,1
        SHR     AH,1
        ADD     AX,3030H
        XCHG    AH,AL
        RET
TTASC   ENDP
;清屏子程序
CLS     PROC
        MOV     AX,0600H
        MOV     CX,0
        MOV     DX,184FH
        MOV     BH,7
        INT     10H
        RET
CLS     ENDP
CSEG    ENDS
        END     START
```

8. 思考题

(1) 对程序做修改，读取年月日时间，实现计数 1 分钟，更新显示时间，这时，应该修改程序中

的什么地方？

(2) 本次实验中，出现了哪些错误？如何排除？

附录一　汇编程序出错信息

1. Block nesting error　嵌套过程、段、结构、宏指令、IRP、IRPC 或 REPT 不是正确结束，如嵌套的外层已终止，而内层还是打开状态。
2. Extra characters on line　在一行上已接受了定义指令的足够信息，而又出现了多余的字符。
3. Register already defined　寄存器已经定义过。
4. Unknown symbol type　有不能识别的符号。
5. Redefinition of symbol　符号重新定义。
6. Symbol is multi-defined　符号多次定义。
7. Phase error between passes　指令全书写有误。
8. Already had ELSE CLAUSE　在 ELSE 从句中试图再定义 ELSE 从句。
9. Not in conditional block　在没有提供条件汇编指令的情况下，指定了 ENDIF 或 ELSE。
10. Symbol notdefined　符号没有定义。
11. Syntax error　语句的语法出错。
12. Type illegal in context　指定的类型非法。
13. Should have been group name　给出的组合不符合要求。
14. Mast be declared in pass 1　得到的不是汇编程序所要求的常数值。如向前引用的向量长度。
15. Symbol type usage illegal　PABLIC 符号的使用不合法。
16. Symbol already different kind　企图定义与以前定义不同的符号。
17. Symbol is reserved word　企图非法使用一个汇编程序的保留字。(例如，宣布 MOV 为一个变量)
18. Forward reference is illegal　向前引用非法。
19. Mast be register　操作数必须是寄存器。
20. Wrong type of register　指定的寄存器类型出错。
21. Must be segment or group　必须给出段或组。
22. Symbol has no segment　想使用带有 SEG 的变量，而这个变量不能识别段。
23. Must be symbol type　必须是 WORD、DW、QW、BYTE 或 TB，但接收的是其它内容。
24. Already defined locally　试图定义一个符号作为 EXTERNAL，但这个符号已经在局部定义过了。
25. Segment parameters are changed　SEGMENT 的自变量表已被改变。

26. Not proper align/combine type　SEGMENT 参数不正确。
27. Reference to mult defined　指令引用的内容已多次定义。
28. Operand was expected　指令中缺少操作数。
29. Operator was expected　指令中缺少操作符。
30. Division by 0 or overflow　除数为零错误。
31. Shift count is negative　移位表达式产生的移位计数值为负数。
32. Operand type must match　指令中的操作数类型不匹配。
33. Illegal use of external　外部使用非法。
34. Must be record field name　需要外部字段名。
35. Must be record or field name　需要记录名或字段名。
36. Operand must have size　需要操作数的长度。
37. Must be var, label or constant　需要变量、标号或常数。
38. Must be structure field name　需要结构字段名。
39. Left operand must have segment　右操作数所用的某些东西要求左操作数必须有一个段。
40. One operand must be const　这是加法指令的非法使用。
41. Operand must be same or 1 abs　这是减法指令的非法使用。
42. Normal type operand expect　当需要变量、标号时，得到的却是 STRUCT、FIELDS、NAMES、BYTE、WORD 或 DW。
43. Constant was expected　需要一个常量。
44. Operand must have segment　SEG 伪操作使用不合法。
45. Must be associated with data　有关项用的是代码，而这里需要的是数据，例如一个过程的 DS 取代。
46. Must be associated with code　有关项用的是数据，而这里需要的是代码。
47. Already have base register　已经有基址寄存器。
48. Already have index register　已经有变址寄存器。
49. Must be index or base register　指令需要基址或变址寄存器。
50. Illegal use of register　在指令中使用了不存在的寄存器。
51. Value is out of range　数值超出使用范围，例如将 DW 传送到寄存器中。
52. Operand not in IP segment　操作数不在当前 IP 段中，因此不能存取。
53. Improper operand type　使用的操作数不能产生操作码。
54. Relative jump out of range　指定的转移超出了允许的范围(-128~+127)。

55. Index disp1 must be constant　试图使用脱离变址寄存器的变量位移量。位移量必须是常数。
56. Illegal register value　指定的寄存器值不能放入“reg”字段中(即“reg”字段大于 7)。
57. No immediate mode　指定的立即方式或操作码都不能接收立即数。
58. Illegal size for item　引用的项长度非法。例如：双字移位。
59. Byte register is illegal　在上下文中，使用一个字节寄存器是非法的。例如：PUSH AL
60. CS register illegal usage　试图非法使用 CS 寄存器。例如：XCHG CS，AX
61. Must be AX or AL　某些指令只能用 AX 或 AL。例如：IN 指令
62. Improper use of segment reg　段寄存器使用不合法。
63. No or unreachable CS　试图转移到不可达到的标号。
64. Operand combination illegal　在双操作数指令中，两个操作数的组合不合法。
65. Near JMP／CALL to different CS　企图在不同的代码段内执行 NEAR 转移或调用。
66. Label can't have seg Override　非法使用段取代。
67. Must have opcode after prefix　使用前缀指令之后，没有正确的操作码说明。
68. Can't override ES segment　企图非法地在一条指令中取代 ES 寄存器。例如：存储字符串。
69. Can't reach with segment reg　没有使变量可达到的 ASSUME 语句。
70. Must be in segment block　企图在段外产生代码。
71. Can't use EVEN on BYTE segment　被提出的是一个字节段，但试图使用 EVEN。
72. Forward needs override　目前不使用这个信息。
73. Illegal value for DUP count　DUP 计数必须是常数，不能是 0 或负数。
74. Symbol already external　企图定义一个局部符号，但此符号已经是外部符号了。
75. DUP is too large for linker　DUP 嵌套太长，以至于从连接程序不能得到所要的记录。
76. Usage of ?(indeterminate)bad　“?”使用不合适。例如：?+5
77. More values than defined with　在定义结构变量或记录变量时,初始值太多。
78. Angle brackets requored around initialized list　定义结构体变量时,初始值未用尖括号括起来。
79. Directive illegal in Structure　在结构体定义中的伪指令使用不当。定义中的伪指令语句仅二种：分号(；)开始的注释语句和用 DB，DW 等数据定义伪指令语句。
80. Override with DUP is illegal　在结构变量初始值表中使用 DUP 操作符出错。
81. Field cannot be overridden　在定义结构变量语句中试图对一个不允许修改的字段设置初值。
82. Override is of wrong type　在定义结构变量语句中设置初值时类型出错。
83. Register can’t be farward ref　寄存器不能被向前引用。

84. Circular chain of EQU aliases　用等值语句定义的符号名，最后又返回指向它自己。

例如

A EQU B

B EQU A

85. End of file,not END directive　源程序文件无 END 文件。

附录二　ASCII 码字符表

ASCII（美国标准信息交换码）字符表

低位＼高位		0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	、	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	US	/	?	O	←	o	DEL

表中符号说明：

NUL	空	DLE	数据链换码
SOH	标题开始	DC1	设备控制1
STX	正文结束	DC2	设备控制2
ETX	本文结束	DC3	设备控制3
EOT	传输结束	DC4	设备控制4
ENQ	询问	NAK	否定
ACK	承认	SYN	空转同步
BEL	报警符	ETB	信息组传送结束
BS	退一格	CAN	作废
HT	横向列表	EM	纸尽
LF	换行	SUB	减
VT	垂直制表	ESC	换码
FF	走纸控制	FS	文字分隔符
CR	回车	GS	组分隔符
SO	移位输出	RS	记录分隔符
SI	移位输入	US	单元分隔符
SP	空格	DEL	作废

完整 ASCII 码字符表

代码	字符	代码	字符	代码	字符	代码	字符
0		32	SP	64	@	96	`
1		33	!	65	A	97	a
2		34	"	66	B	98	b
3		35	#	67	C	99	c
4		36	$	68	D	100	d
5		37	%	69	E	101	e
6		38	&	70	F	102	f
7		39	'	71	G	103	g
8	**	40	(	72	H	104	h
9	**	41	)	73	I	105	i
10	**	42	*	74	J	106	j
11		43	+	75	K	107	k
12		44	,	76	L	108	l
13	**	45	-	77	M	109	m
14		46	.	78	N	110	n
15		47	/	79	O	111	o
16		48	0	80	P	112	p
17		49	1	81	Q	113	q
18		50	2	82	R	114	r
19		51	3	83	S	115	s
20		52	4	84	T	116	t
21		53	5	85	U	117	u
22		54	6	86	V	118	v
23		55	7	87	W	119	w
24		56	8	88	X	120	x
25		57	9	89	Y	121	y
26		58	:	90	Z	122	z
27		59	;	91	[	123	{
28		60	<	92	\	124	|
29		61	=	93	]	125	}
30	-	62	>	94	^	126	~
31		63	?	95	_	127	

代码	字符	代码	字符	代码	字符	代码	字符
128	€	160	SP	192	À	224	à
129		161	¡	193	Á	225	á
130	‚	162	¢	194	Â	226	â
131	ƒ	163	£	195	Ã	227	ã
132	„	164	¤	196	Ä	228	ä
133	…	165	¥	197	Å	229	å
134	†	166	¦	198	Æ	230	æ
135	‡	167	§	199	Ç	231	ç
136	ˆ	168	¨	200	È	231	ç
137	‰	169	©	201	É	232	è
138	Š	170	ª	202	Ê	233	é
139	‹	171	«	203	Ë	234	ê
140	Œ	172	¬	204	Ì	235	ë
141		173		205	Í	236	ì
142	Ž	174	®	206	Î	237	í
143		175	¯	207	Ï	238	î
144		176	°	208	Ð	239	ï
145	‘	177	±	209	Ñ	240	ð
146	’	178	²	210	Ò	241	ñ
147	“	179	³	211	Ó	242	ò
148	”	180	´	212	Ô	243	ó
149	•	181	µ	213	Õ	244	ô
150	–	182	¶	214	Ö	245	õ
151	—	183	·	215	×	246	ö
152	˜	184	¸	216	Ø	247	÷
153	™	185	¹	217	Ù	248	ø
154	š	186	º	218	Ú	249	ù
155	›	187	»	219	Û	250	ú
156	œ	188	¼	220	Ü	251	û
157		189	½	221	Ý	252	ü
158	ž	190	¾	222	Þ	253	ý
159	Ÿ	191	¿	223	ß	254	þ

附录三　常用子程序集

1.　输出回车换行

```
CRLF    PROC
        MOV     DL,0DH
        MOV     AH,2
        INT     21H
        MOV     DL,0AH
        MOV     AH,2
        INT     21H
        RET
CRLF    ENDP
```

2.　输出空格

```
WHITESPACE    PROC
        MOV     DL,20H
        MOV     AH,02H
        INT     21H
        RET
WHITESPACE    ENDP
```

3.　字符串的输出

```
PROMPT      PROC
        MOV     AH,09H
        INT     21H
        RET
PROMPT      ENDP
```

4.　二进制的输入(0-1111111111111111)

```
PUTINBIN    PROC
        PUSH    CX
        MOV     BX,0
P1:     MOV     AH,1
        INT     21H
        CMP     AL,0DH
        JE      BINEXIT
        SUB     AL,30H
        MOV     AH,0
        XCHG    AX,BX
        MOV     CX,2
        MUL     CX
        XCHG    AX,BX
        ADD     BX,AX
        JMP     P1
BINEXIT:POP     CX
        RET
```

```
PUTINBIN    ENDP
```

5. 十进制的输入(0-65535)

```
PUTINDEC    PROC
            PUSH    CX
            MOV     BX,0
P1:         MOV     AH,1
            INT     21H
            CMP     AL,0DH
            JE      DECEXIT
            SUB     AL,30H
            MOV     AH,0
            XCHG    AX,BX
            MOV     CX,10
            MUL     CX
            XCHG    AX,BX
            ADD     BX,AX
            JMP     P1
DECEXIT:POP     CX
            RET
PUTINDEC    ENDP
```

6. 十六进制的输入(0-FFFF)

```
PUTINHEX    PROC
            PUSH    CX
            MOV     BX,0
P1:         MOV     AH,1
            INT     21H
            CMP     AL,0DH
            JE      HEXEXIT
            SUB     AL,30H
            CMP     AL,10
            JB      OOO1
            SUB     AL,27H
OOO1:       MOV     CL,4
            SHL     BX,CL
            MOV     AH,0
            ADD     BX,AX
            JMP     P1
HEXEXIT:POP     CX
            RET
PUTINHEX    ENDP
```

7. 二进制的输出(0-1111111111111111)

```
PUTOUTBIN PROC
            PUSH    CX
            MOV     NOZERO,0
            MOV     CL,16
PP2:        MOV     DL,0
            TEST    BX,8000H
```

```
        JE      PP
        MOV     NOZERO,1
        MOV     DL,1
PP:     CMP     NOZERO,0
        JE      PP1
        ADD     DL,30H
        MOV     AH,2
        INT     21H
PP1:    SHL     BX,1
        LOOP    PP2
        POP     CX
        RET
PUTOUTBIN ENDP
```

8. 十进制的输出(0-65535)

```
PUTOUTDECPROC
        PUSH    CX
        MOV     NOZERO,0
        MOV     CX,10000
        CALL    MAIN
        MOV     CX,1000
        CALL    MAIN
        MOV     CX,100
        CALL    MAIN
        MOV     CX,10
        CALL    MAIN
        MOV     CX,1
        CALL    MAIN
        CMP     NOZERO,0
        JNE     LL
        MOV     DL,30H
        MOV     AH,2
        INT     21H
LL:     POP     CX
        RET
        MAIN    PROC
        MOV     AX,BX
        MOV     DX,0
        DIV     CX
        MOV     BX,DX
        MOV     DL,AL
        CMP     DL,0
        JNE     LL1
        CMP     NOZERO,0
        JNE     LL1
        JMP     LL2
LL1:    MOV     NOZERO,1
        ADD     DL,30H
        MOV     AH,2
        INT     21H
LL2:    RET
        MAIN    ENDP
```

```
PUTOUTDEC       ENDP
```

9. 十六进制的输出(0-FFFF)

```
PUTOUTHEX       PROC
        PUSH    CX
        MOV     TEMP,BX
        MOV     TEMP1,BX
        MOV     CL,8
        SHR     TEMP,CL
        MOV     BX,TEMP
        CALL    MAIN1
        AND     TEMP1,00FFH
        MOV     BX,TEMP1
        CALL    MAIN1
        POP     CX
        RET
        MAIN1   PROC
        MOV     AL,BL
        MOV     CL,4
        SHR     AL,CL
        CMP     AL,10
        JB      KK
        ADD     AL,7
KK:     MOV     DL,AL
        CMP     DL,0
        JE      KK2
        ADD     DL,30H
        MOV     AH,2
        INT     21H
        AND     BL,0FH
KK2:CMP BL,10
        JB      KK1
        ADD     BL,7
        KK1:    MOV     DL,BL
        ADD     DL,30H
        MOV     AH,2
        INT     21H
        RET
        MAIN1   ENDP
PUTOUTHEX       ENDP
```

10. 逆序排序模块

```
SORT    PROC    NEAR
        PUSH    CX
        MOV     CL,ACTLEN
        MOV     CH,0
        SORTLOOP1:
        MOV     DI,  CX
        MOV     SI,0
        SORTLOOP2:
        MOV     AL,STRING[SI]
```

```
        CMP     AL,STRING[SI+1]
        JAE     SORTCONTINUE
        XCHG    AL,NUM[SI+1]
        MOV     NUM[SI],AL
        SORTCONTINUE:
        ADD     SI,1
        LOOP    SORTLOOP2
        MOV     CX,DI
        LOOP    SORTLOOP1
        POP     CX
        RET
SORT    ENDP
```

11. 统计非字符和非数字的字符个数

```
        MOV     CL,NUM+1
        LEA     SI,NUM+2
PP1:    MOV     AL,[SI]
        CMP     AL,'0'
        JB      PP
        CMP     AL,'9'
        JB      PP2
        CMP     AL,'A'
        JB      PP
        CMP     AL,'Z'
        JB      PP2
        CMP     AL,'A'
        JB      PP
        CMP     AL,'Z'
        JB      PP2
        JMP     PP2
PP:     INC     COU
PP2:    INC     SI
        LOOP    PP1
        MOV     BL,COU
```

12. 显示字符和数字的字符

```
        MOV     CL,NUM+1
        LEA     SI,NUM+2
P1:     MOV     AL,[SI]
        CMP     AL,'0'
        JB      P2
        CMP     AL,'9'
        JB      P
        CMP     AL,'A'
        JB      P2
        CMP     AL,'Z'
        JB      P
        CMP     AL,'A'
        JB      P2
        CMP     AL,'Z'
```

```
        JB      P
        JMP     P
P:      MOV     DL,AL
        MOV     AH,2
        INT     21H
P2:     INC     SI
        LOOP    P1
```

13. 大写转换小写并显示

```
CONVERT     PROC
QQ: MOV     AL,[SI]
    CMP     AL,'A'
    JB      QQ2
    CMP     AL,'Z'
    JA      QQ2
    ADD     AL,20H
QQ2:MOV     DL,AL
    MOV     AH,02H
    INT     21H
    INC     SI
    LOOP    QQ
    CALL    CRLF
    RET
    CONVERT     ENDP
```

14. 找到输入的串中的字符中最大的 ASSCII 码

```
MAXASC      PROC    NEAR
    LEA     SI,NUM+2
    MOV     CL,NUM+1
    MOV     BL,[SI]
LL: INC     SI
    CMP     BL,[SI]
    JA      GG
    MOV     BL,[SI]
GG: MOV     MAXA,BL
    MM:     LOOP    LL
    RET
MAXASC      ENDP
```

15. 十进制显示百分比(方法，先拿其中一个数乘 100，然后再除以总和，AL 即所得)

```
    LEA     DX,OUT3
    MOV     AH,09H
    INT     21H
    MOV     AL,NUM1
    MOV     BH,100
    MUL     BH
    MOV     BL,SUM
    DIV     BL
    MOV     BL,AL
    CALL    DISP10
```

```
        MOV     DL,'%'
        MOV     AH,02
        INT     21H
```

16. 十进制转换为二进制子程序

```
; PROCEDURE: GET DECIMAL STRING FROM KEYBOARD AND THEN CONVERT
;TO BINARY
; ENTER PARA    : NONE
; OUTPUT PARA : BX=BINARY CONVERED
; LIMITATION    : 0<=DECIMAL<32768
 D_BP   PROC
        PUSH    AX
 PUSH   DX
 MOV    BX,0
 NEWCHAR:MOV    AH,1
        INT     21H
        CMP     AL,0DH
        JZ      EXIT1
        SUB     AL,30H
        JL      EXIT1
        CMP     AL,9
        JG      EXIT1
        CBW
        XCHG    AX,BX
        MOV     CX,10
        MUL     CX
        XCHG    AX,BX
        ADD     BX,AX
        JMP     NEWCHAR
 EXIT1: MOV     DL,0DH
        MOV     AH,2
        INT     21H
        MOV     DL,0AH
        MOV     AH,2
        INT     21H
        POP     DX
        POP     AX
        RET
        D_BP    ENDP
```

17. 二进制转换为十进制子程序

```
;PROCEDURE    :BINARY COVERT TO DECIMAL STRING AND OUPUT
;ENTRY PARA :AX=BINARY,DS:DX=DEC STRING HEAD ADDRESS
;EXIT PARA    :NONE
 B_DP   PROC                                    ; CONVERT TO DECCIMAL STRING
 PUSH   BX
 PUSH   CX
 PUSH   DX
 PUSH   SI
 JMP    SSTART
 DECSTR     DB  5     DUP(0),0DH,0AH
```

```
SSTART:CMP     AX,0
       JNZ     SDONE
       MOV     DL,30H
       MOV     AH,2
       INT     21H
       JMP     EXITT
SDONE:MOV      BX,10
       MOV     CX,5
       MOV     SI,OFFSET    DECSTR
CIR0:  MOV     DX,0
       DIV     BX                          ;EXTRA IS IN DL,AX IS QUOTIENT
       OR      DL,30H
       MOV     BYTE    PTR CS:[SI+4],DL
       DEC     SI
       LOOP    CIR0                        ;OUTPUT      STRING
       MOV     CX,7
       MOV     BL,0
CIR1:  INC     SI
       MOV     DL,CS:[SI+4]
       CMP     DL,30H
       JNZ     NONZERO
       CMP     BL,0                        ;HIGH    BIT IS    ZERO,NON    DISPLAY
       JZ      NEXT0
NONZERO:MOV    BL,1
       MOV     AH,2
       INT     21H
NEXT0:LOOP     CIR1
EXITT: POP     SI
       POP     DX
       POP     CX
       POP     BX
       RET
B_DP   ENDP
```

18. 多字节的二进制加法程序

```
DATA   SEGMENT
DATA1 DB  1,   2                           ;被加数
DATA2 DB  5,   6                           ;加数
DATA3 DB  0,   0                           ;和
DATA   ENDS
CODE   SEGMENT
       ASSUME      CS:CODE,     DS:DATA
START: MOV     AX, DATA
       MOV     DS,  AX
       MOV     SI,  0
       CLC                                 ;清进位标志
       MOV     AL,  DATA1[SI]
       ADC     AL,  DATA2[SI]              ;第一个字节相加
       MOV     DATA3[SI],    AL
       INC     SI
       MOV     AL,  DATA1[SI]
```

```
        ADC     AL, DATA2[SI]           ;第二个字节相加
        MOV     DATA3[SI],  AL
;.....  输出显示部分      .....
        MOV     BX, OFFSET DATA3
        MOV     DL, [BX]
        ADD     DL, 30H
        MOV     AH, 2
        INT     21H
        INC     BX
        MOV     DL, [BX]
        ADD     DL, 30H
        MOV     AH, 2
        INT     21H
;  .....输出显示结束      .....
        MOV     AH, 4CH
        INT     21H                     ;返回     DOS
CODE    ENDS
        END     START
```

19. 小写字母转换成大写字母程序

```
DATA    SEGMENT
MSG     DB  'ERROR!',    0DH,    0AH,'$'
DATA    ENDS
CODE    SEGMENT
        ASSUME      CS:CODE,    DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     AH, 1
        INT     21H                     ;读入一字符送入 AL
        CMP     AL, 'A'
        JB      ERR                     ;不是小写字母转出错处理
        CMP     AL, 'Z'
        JA      ERR                     ;不是小写字母转出错处理
        SUB     AL, 32                  ;转换成大写字母后送到 AL
        MOV     DL, AL
        MOV     AH, 02
        INT     21H                     ;输出转换后的字符
OVER:   MOV     AH, 4CH
        INT     21H                     ;返回 DOS
ERR:    MOV     DX, OFFSET MSG
        MOV     AH, 09
        INT     21H
        JMP     OVER
CODE    ENDS
        END     START
```

20. 判别字母、数字程序

```
DATA    SEGMENT
MSG1    DB  'DIGIT', 0DH,    0AH,'$'
MSG2    DB  'LETTER',    0DH,    0AH,    '$'
```

```
MSG3  DB  'OTHER',    0DH,    0AH,    '$'
DATA  ENDS
CODE  SEGMENT
      ASSUME    CS:CODE,    DS:DATA
START: MOV    AX, DATA
       MOV    DS, AX
       MOV    AH, 1
       INT    21H                    ;读取一字符送入 AL
       CMP    AL, '0'
       JB     OTHER                  ;转非字母、数字显示
       CMP    AL, '9'
       JBE    DIGIT                  ;转数字显示
       CMP    AL, 'A'
       JB     OTHER                  ;转非字母、数字显示
       CMP    AL, 'Z'
       JBE    LETTER                 ;转字母显示
       CMP    AL, 'A'
       JB     OTHER                  ;转非字母、数字显示
       CMP    AL,'Z'
       JBE    LETTER                 ;转字母显示
       JMP    OTHER                  ;转非字母、数字显示
DIGIT: MOV    DX, OFFSET MSG1
       MOV    AH, 09H
       INT    21H                    ;显示'DIGIT'
       JMP    OVER
LETTER:MOV    DX, OFFSET MSG2
       MOV    AH, 09H
       INT    21H                    ;显示'LETTER'
       JMP    OVER
OTHER:MOV     DX, OFFSET MSG3
       MOV    AH, 09H
       INT    21H                    ;显示'OTHER'
OVER:  MOV    AH, 4CH
       INT    21H                    ;返回 DOS
CODE  ENDS
       END    START
```

21. 偶校验程序

```
DATA  SEGMENT
BUFOUT    DB  10                     ;10 个数据
          DB  0, 1, 2, 3, 4, 5, 6, 7, 8, 9
MSG       DB  ' ..... $'
DATA  ENDS
STACK     SEGMENT
   DW     256     DUP(?)
   TOP    LABEL   WORD
STACK     ENDS
CODE  SEGMENT
      ASSUME    CS:CODE, DS:DATA, SS:STACK
START: MOV    AX, DATA
```

```
        MOV     DS, AX
        MOV     AX, STACK
        MOV     SS, AX
        MOV     SP, OFFSET TOP
        LEA     DI, BUFOUT              ;BUFOUT 首址送 DI
        CALL    STEV
        MOV     AX, 4C00H
        INT     21H                     ;返回 DOS
STEV    PROC    NEAR
        PUSH    DS
        PUSH    DI
        MOV     CL, [DI]                ;取缓冲区长度送 CL
        MOV     CH, 0
        INC     DI                      ;指向缓冲区字符
LI:     MOV     AL, [DI]                ;缓冲区中取一个字符
;.........输出显示部分  .........
        PUSH    AX
        CALL    OUTPUT
        POP     AX
;.........输出显示结束  .........
        OR      AL, 80H                 ;最高位置 1
        JNP     L2                      ;是奇数个 1 吗?
        MOV     [DI],AL                 ;不是，则(AL)送[DI]
;.........  输出显示部分       .........
   L2:  MOV     DX, OFFSET MSG
        MOV     AH, 9
        INT     21H
        MOV     AL, [DI]
        CALL    OUTPUT
        MOV     DL, 0AH
        MOV     AH, 2
        INT     21H
        MOV     DL, 0DH
        MOV     AH, 2
        INT     21H
;.........  输出显示结束       .........
        INC     DI
        LOOP    L1
        POP     DI
        POP     DS
        RET
STEV    ENDP
;.........  输出子程序   ..........
OUTPUT    PROC    NEAR
        MOV     BH, 8
        MOV     BL, AL
OUTI:   MOV     DL, BL
        AND     DL, 80H
        ROL     DL, 1
        ADD     DL, 30H
        MOV     AH, 2
```

```
        INT     21H
        DEC     BH
        CMP     BH, 0
        JZ      EDED
        ROL     BL, 1
        JMP     OUT1
EDED:   RET
OUTPUT  ENDP
;.....输出子程序结束 .....
CODE    ENDS
        END     START
```

22. 计算 N!程序

```
STACK   SEGMENT
        STPN        DW 100 DUP(?)   ;建立一个堆栈区
        TOP EQU     $-STPN          ;TOP    指针初始化
STACK   ENDS
DATA    SEGMENT
        N   DW 6                    ;数 N=6
        RESULT DW ?                 ;N!存放单元
DATA    ENDS
CODE1 SEGMENT
        ASSUME      CS:CODE1,DS:DATA,SS:STACK
MAIN:   MOV     AX, DATA
        MOV     DS, AX
        MOV     AX, STACK
        MOV     SS, AX
        MOV     SP, TOP
        PUSH    N
        CALL    FAR PTR FACT        ;调用求阶乘子程序
M1:     POP     RESULT
;  .....输出显示部分    .....
        CALL    OUTPUT
;  .....输出显示结束    .....
        MOV     AX, 4C00H
        INT     21H
;  .....输出子程序 .....
OUTPUT  PROC    NEAR
        MOV     CH, 2
        MOV     BX, OFFSET RESULT
        MOV     DX, [BX]
        PUSH    DX
        MOV     DL, DH
OUT1:   PUSH    DX
        AND     DL, 0FOH
        MOV     CL, 4
        ROR     DL, CL
        ADD     DL, 30H
        CMP     DL, 39H
        JBE     DD1
        ADD     DL, 7
```

```
DD1:    MOV     AH, 2
        INT     21H
        POP     DX
        AND     DL, 0FH
        ADD     DL, 30H
        CMP     DL, 39H
        JBE     DD2
        ADD     DL, 07H
DD2:    MOV     AH, 2
        INT     21H
        DEC     CH
        CMP     CH, 0
        JZ      EDED
        POP     DX
        JMP     OUT1
EDED:   MOV     AH, 2
        MOV     DL, 'H'
        INT     21H
        MOV     DL, 0AH
        INT     21H
        MOV     DL, 0DH
        INT     21H
        RET
OUTPUT  ENDP
;.....输出子程序结束  .....
        CODE1   ENDS
CODE2   SEGMENT
        ASSUME  CS:CODE2,DS:DATA,SS:STACK
FACT    PROC    FAR
        PUSH    AX
        PUSH    BP
        MOV     BP, SP
        ADD     BP, 8
        MOV     AX, [BP]
        CMP     AX, 0                   ;判断阶乘是否结束
        JNZ     FACTAGAIN
        MOV     AX, 1
        JMP     EXIT
FACTAGAIN:SUB   SP, 2
        DEC     AX
        PUSH    AX
        CALL    FAR PTR FACT            ;求(N-1)!
FL:     POP     AX
        MUL     WORD PTR[BP]
EXIT:   MOV     [BP+2], AX
        POP     BP
        POP     AX
        RET
FACT    ENDP
CODE2   ENDS
        END     MAIN
```

23. 数据块移动程序

```
DATA    SEGMENT
DATA1   DB      0, 1, 2, 3, 4, 5, 6, 7, 8, 9        ;数据块 1
DATA_LEN EQU    $-DATA1
DATA2   DB      10   DUP(0)                         ;数据块 2
DISP1   DB  0DH, 0AH, 'SOURCE   DATA:', '$'
DISP2   DB  0DH, 0AH, 'TARGET   DATA:', '$'
DATA    ENDS
CODE    SEGMENT
        ASSUME      CS:CODE,    DS:DATA
START:  MOV     AX, DATA
        MOV     DS,  AX
        MOV     ES,  AX
        MOV     CX, DATA_LEN
        CLD
        MOV     SI,  OFFSET DATA1
        MOV     DI,  OFFSET DATA2
        REP     MOVSB                           ;块移动
;.........输出显示部分   .....
        MOV     DX, OFFSET DISP1
        MOV     AH, 9
        INT     21H
        MOV     BX, OFFSET DATA1
        MOV     CX, WORD   PTR 10
LOOP1:  PUSH    CX
        CALL    OUTPUT
        INC     BX
        POP     CX
        LOOP    LOOP1
        MOV     DX, OFFSET DISP2
        MOV     AH, 9
        INT     21H
        MOV     BX, OFFSET DATA2
        MOV     CX, WORD   PTR 10
LOOP2:  PUSH    CX
        CALL    OUTPUT
        INC     BX
        POP     CX
        LOOP    LOOP2
;  .....输出显示结束 .....
        MOV     AX, 4C00H
        INT     21H
;  .....输出子程序   .....
OUTPUT  PROC    NEAR
        MOV     DL, [BX]
        ADD     DL, 30H
        CMP     DL, 39H
        JBE     DD1
        ADD     DL, 7
   DD1: MOV     AH, 2
        INT     21H
        RET
```

```
    OUTPUTENDP
;.....输出子程序结束.....
CODE  ENDS
      END     START
```

24. ASCⅡ码转二进制码程序

```
DATA  SEGMENT
KEY_BUF  DB  10
         DB  ?
         DB  10   DUP(?)
DATA1    DW ?
DATA  ENDS
CODE  SEGMENT
      ASSUME     CS:CODE,     DS:DATA
START: MOV     AX, DATA
       MOV     DS, AX
       MOV     DX, OFFSET KEY_BUF
       MOV     AH, 0AH
       INT     21H                      ;接收键盘输入
       MOV     SI,  OFFSET KEY_BUF+1
       MOV     BL, [SI]                 ;输入字符个数
       INC     SI                       ;SI 指向输入字符的数据区
       MOV     AX, 0                    ;设置初值 0
TRANS:MOV      DX, 10
       MUL     DX                       ;乘 10
       MOV     DH, 0
       MOV     DL, [SI]
       AND     DL, 0FH                  ;ASCⅡ码转换成二进制码
       ADD     AX, DX                   ;相加
       INC     SI
       DEC     BL
       JNZ     TRANS                    ;还有数字?
       MOV     DATA1,  AX               ;转换后的结果
       CALL    OUTPUT
       MOV     AX, 4C00H
       INT     21H                      ;返回 DOS
;.....  输出子程序  .....
OUTPUT    PROC    NEAR
       PUSH    AX
       MOV     DL, 0AH
       MOV     AH, 2
       INT     21H
       MOV     DL, 0DH
       INT     21H
       POP     AX
       MOV     BH, 10H
       MOV     CX, AX
OUT1:  MOV     DL, CH
       AND     DL, 80H
       ROL     DL, 1
       ADD     DL, 30H
```

```
        MOV     AH, 2
        INT     21H
        DEC     BH
        CMP     BH, 0
        JZ      EDED
        ROL     CX, 1
        JMP     OUT1
EDED:   RET
OUTPUT  ENDP
;.....  输出子程序结束  .....
CODE    ENDS
        END     START
```

参考文献

[1] 沈美明、温冬婵编著，80X86 汇编语言程序设计，清华大学出版社，2002.3

[2] 戴梅萼、史嘉权编著，微型计算机技术及应用，清华大学出版社，1997.3

[3] 张昆藏编著，IBM PC/XT 微型计算机接口技术，清华大学出版社，1993.7

[4] 刘均、周苏、金海溶等编著，汇编语言程序设计实验教程，科学出版社，2006.9

[5] 赵国相、赵大鹏、张健、徐长青编著，微型计算机原理与汇编语言程序设计，科学出版社，2005.7

[6] 赵国相、于秀峰编著，微型计算机原理与接口技术，科学出版社，2005.3